OraGiST - How to Make User-Defined Indexing Become Usable and Useful

Carsten Kleiner, Udo W. Lipeck
Universität Hannover
Institut für Informationssysteme
FG Datenbanksysteme
Welfengarten 1
30167 Hannover
{ck | ul}@dbs.uni-hannover.de

Abstract: In this article we present a concept for simplification of user-defined indexing for user-defined data types in object-relational database systems. The concept is based on a detailed analysis of user-defined indexing in ORDBS on one hand, and features of generalized search trees (GiST) as an extensible indexing framework on the other hand. It defines a minimal interface to be implemented in order to use GiST within ORDBS; this greatly simplifies the process of implementing user-defined indexes. The effectiveness of the approach is illustrated by performance experiments carried out on a prototypical implementation of our concept. For the experiments we have used new specialized spatial data types, that store spatial as well as thematic information within a single attribute. These data types facilitate advanced spatial analysis operators. The experiments show great performance improvements on these operators by using multidimensional user-defined index structures based on R-trees when compared to system-provided indexes.

Keywords: object-relational databases, user-defined indexing, data cartridge, user-defined datatype, advanced spatial analysis operators

1 Introduction

1.1 Motivation

In recent years object-relational database systems (ORDBS) have advanced past research prototypes and are becoming commercially available. One of the key advantages over traditional relational databases is the possibility to use user-defined datatypes (UDTs). It is a very important feature in many advanced applications, because adequate modeling of non-standard domains requires such types. This can be inferred from the increased popularity of object-oriented modeling which implicitly facilitates the use of arbitrary datatypes.

Also in the spirit of object-oriented modeling these new UDTs will have specific functions operating on them, usually called methods in object-oriented technology. The most important of these methods from a database perspective are methods that can be used in queries to select objects from tables or join objects from different tables. These methods are also called *operators* in object-relational terminology. In commercial settings these new features will only be used, if they achieve a performance comparable to standard datatypes and functions in traditional relational databases.

Since UDTs can be defined arbitrarily by the user, it is impossible for the ORDBS vendor to provide efficient support of all such operators in selection and/or join queries. Consequently they provide *extensible* indexing and query optimization features. The author of the UDT has to implement these features using domain-specific knowledge. The complete package of UDT, operators, indexing and query optimization is also called a *cartridge* (or, depending on the particular vendor, *extender* or *data blade*), since it can be plugged into the database server similar to cartridges in hardware. The enhanced database server then efficiently supports UDTs transparent to the end-user.

A big obstacle, however, is that the implementation of user-defined indexing is pretty complex and time-consuming. Moreover as it is, it has to be carried out completely from scratch for every UDT that requires efficient query support. In order to overcome this and make user-defined indexing more usable, a *generic* indexing framework such as generalized search trees (GiST; [HNP95]) should be used. This has the advantage of already providing most of the required functionality for several different index structures. These structures can easily be specialized to obtain a specific index for a particular datatype.

Therefore it would be desirable to have an easy to use tool that combines a particular UDT from an ORDBS with a concrete extension of an index framework automatically and leaves only the (very few as will be shown in this article) type and operator specific implementation parts to the user. In this article we will first investigate what the type and operator specific details of such a definition under some reasonable assumptions are. The result is a concept for greatly simplifying the usage of an indexing framework in ORDBS. We will then present the design of a prototypical implementation of such a tool, called *OraGiST*, which facilitates the use of GiST as user-defined index structures in the ORDBS Oracle 9*i*. The tool which was developed in our group does as much work as possible automatically, leaving the programmer with just very few tasks[1] to be solved, before being able to use GiST inside Oracle. While OraGiST is specifically designed for Oracle, similar extensions for other ORDBS based on the previous concept would also be possible.

The effectiveness of this approach is illustrated by some sample results from spatial data. In advanced applications one would like to combine spatial selection criteria with other thematic attributes in a single operator. Queries such as *select all cities in a given query window where the population density is higher than a given value and the percentage of male inhabitants is more than 55 percent* could be answered efficiently by such complex operators. We will show that using high-dimensional R-tree-like GiST extensions developed by using OraGiST is more efficient than working with classical indexes. It is even more efficient than using the DBMS provided spatial indexes on the spatial component.

[1]If using one of the predefined GiST extensions it takes in the order of minutes to implement the required parts.

1.2 Related Work

The concept of object-relational database systems (ORDBS) has been described in [SM96] and [SB99]. Most commercial DBMS that were relational can be called object-relational to a certain degree today. Probably the most commonly used systems are Oracle 9*i* and IBM DB2. In the open-source area e. g. PostgreSQL is also object-relational; it is based on the oldest ORDBS Postgres (cf. [SR86]). Requirements specific to user-defined indexing in ORDBS are explained in [SB99].

A good overview of advanced spatial applications can be found in [RSV01]. In particular indexing in spatial databases is reviewed in [GG98]. RSS-Trees were introduced in detail in [KL00]. The introduction of user-defined types in ORDBS requires easily extensible indexes in order to facilitate efficient querying of UDTs. Therefore generic index structures such as generalized search trees have to be used. A similar idea to ours has been described in [RKS99] for the CONCERT DBS. It allows indexing based on concepts but is more focused on spatio-temporal data and is restricted in the indexes to be used since it requires a hierarchical partitioning of space. The basic insight that only very few features are actually specific to a data type in indexing is also used in our approach. More advanced research on query optimization for user-defined types is described in [Hel98]. Another implementation for advanced index structures was presented in [BDS00]. It is written in Java and focuses on main memory index structures. When Java has become more efficient and systems even more powerful in terms of main memory this will probably also be a very interesting approach. The combination of an extensible indexing framework with a commercial ORDBS has to the best of our knowledge never been researched before. Also the idea of using higher-dimensional indexing for advanced spatial analysis has never been analyzed in such detail. Finally no experimental results on the efficiency of real user-defined indexing in ORDBS have been reported yet.

2 Current Situation

Generalized search trees (GiST; [HNP95]) are search trees that combine and implement the common features of tree-based access methods such as insertion and searching but still allow to adjust the classical operations to a particular data type and indexing structure. This is achieved by an implementation using certain extensible methods in the general algorithms that have to be implemented by the user for the particular data type and indexing structure by object-oriented specialization later; these methods are consistent, penalty, pickSplit, union and optionally compress and decompress. Due to space constraints details on GiST are omitted; they can be found in the full version of this paper ([KL02]).

A file-based implementation of generalized search trees called `libgist` has been made available under `http://gist.cs.berkeley.edu`. It is written in C++ and provides the general tree along with several extensions for most of the trees proposed in the literature such as B-Tree, R*-Tree and SS-Tree. Its completion was reported in [Kor99]. Since it operates on index files, it cannot be used directly in conjunction with a commercial ORDBS, where index files should be under control of the DBS as well, e. g. for transactional correctness reasons. Moreover the interfaces need to be linked together in order to use a GiST specialization as an index inside the ORDBS. In addition ORDBS users want to use the given and other GiST-based index structures for their own user-defined types. This requires a mapping of database types and operators to GiST objects and predicates.

In order to implement **extensible user-defined indexing** - for example in Oracle 9*i* - certain methods have to be programmed: IndexCreate, IndexInsert, IndexDrop, IndexStart, IndexFetch, IndexClose, IndexAlter and IndexUpdate. These are later automatically invoked by the database server when executing regular SQL commands using the UDTs. Again due to space constraints details are omitted here, but can be found in the full version of the paper.

User-defined data types in ORDBS need data type specific **query optimization** to be able to use them efficiently. In Oracle this is implemented by providing special interfaces for extensible optimization. Functions defined in these interfaces are called automatically by the database server, if an implementation of the interface for the particular data type is registered with the server. This way extensible optimization is as directly integrated into the database server as possible for arbitrary data types. The interfaces of the extensible optimizer are described in detail in system documentation.

3 Concept of OraGiST

In order to connect the file-based implementation of generalized search trees with the indexing interface of the ORDBS we have firstly analyzed which components of the mapping between a generic index and an ORDBS are generic themselves and which are dependent on the particular data type in question. Consequently, the implementation of a connector component between ORDBS and generic index consists of two main parts. We have called our prototypical implementation `OraGiST`, since it operates based on `libgist` and Oracle 9*i*. An overview of the architecture of `OraGiST` is given in figure 1.

As explained before, the tool `OraGiST` mainly consists of two components. The first component (called OraGiST library) is independent of the particular data type, user and GiST extension. It provides functionality for calling the appropriate generic index methods inside functions of the ORDBS extensible indexing interface (cf. section 2) on the one hand. Also it facilitates storing of index information of a file-based tree index structure inside an ORDBS on the other hand. These are the upper two associations in figure 1.

The second component (OraGiST toolbox in figure 1) is dependent on the data type and thus index structure specialization. Consequently, it cannot work completely without programmer interaction. It can rather be a support tool providing the user with method prototypes, and taking over all generic tasks of this part of the user-defined index development process. In particular, only four methods have to be implemented for the particular data type by the database programmer. Firstly, it has to be defined which GiST specialization is to be used for a particular ORDBS user-defined data type[2]. This is done by implement-

[2]This specialization has to be written separately in advance; it is not a big restriction, though, since many important index structures for different domains are already provided with `libgist`. Also because of the

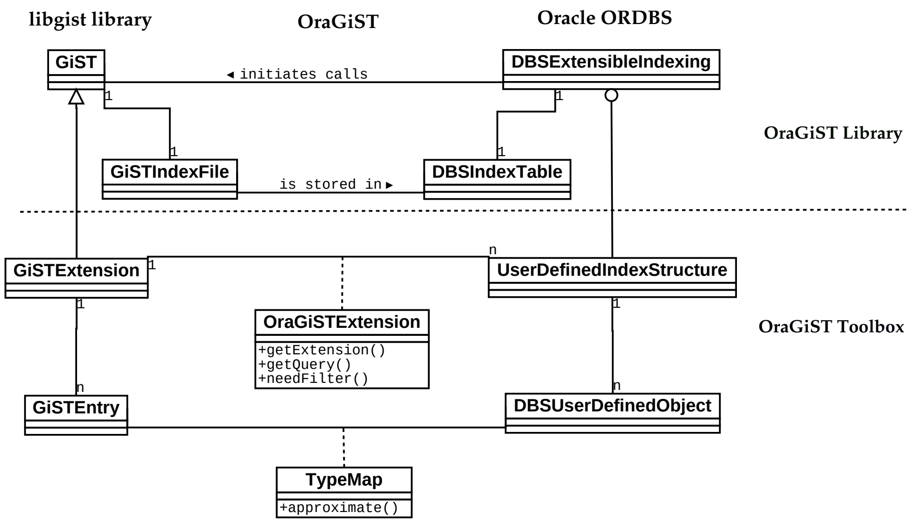

Figure 1: Architecture and Functionality of `OraGiST`

ing the method `getExtension`. Secondly, the mapping of database operators to index structure methods implementing the desired predicate has to be declared by implementing the method `getQuery` by the developer. The implementation of these two methods typically consist of only a single line returning a pointer to the particular object required.

Also the mapping between ORDBS data type and index structure entry has to be implemented. Since often not the exact objects but rather approximations are inserted into an index, this method is called `approximate` (cf. MBRs in spatial indexing). More generally speaking this method implements the compress method of the original GiST concept. This method is the only one that may incur considerable programming work, since it must implement the mapping of a database object to an index object; in the case of spatial data in Oracle 9*i*, for instance, the rather complex definition of the database geometry type has to be investigated in C to compute the MBR of the object.

Finally, results from an index lookup may not be results to the original query, if the index entries are approximations of the original objects. For spatial data e. g. not all objects whose MBR intersect a given object do intersect the object themselves; thus usually the so-called filter-and-refine strategy of query execution is used for spatial queries. This is reflected in `OraGiST` by means of a method `needFilter`. It has to return `true`, if results from the index scan still have to be tested, before the query result is determined, and `false` else. For the programmer this is only one additional line of code in the current version of `OraGiST`, since as testing method the functional implementation of the operator which is required by the ORDBS anyway may be used. If such testing is required, after registering the index with `OraGiST`, the toolbox will automatically take care, that results of an index scan are additionally filtered by the functional implementation of the operator

framework character of `libgist` such extensions are supported very well.

```
CREATE TYPE twoIntegerGeometry AS OBJECT (
   geom OGCGeometry,
   theme1 INTEGER,
   theme2 INTEGER,
   -- type-specific methods);
```

Figure 2: Data type definition for combining spatial and thematic information

in the ORDBS.

As stated earlier based on this concept, we have developed a prototypical implementation of a connector between GiST implementation `libgist` and Oracle 9*i* as ORDBS. This simplifies user-defined indexing to such a degree that it is really usable with acceptable implementation effort. If one of the predefined GiST extensions is to be used as index only the small `OraGiST` toolbox classes have to be implemented. But even if a new GiST extension is required as index development time is greatly reduced. This is due to the extensibility features of GiST which facilitate simple definition of new index structures as long as they can be expressed in the GiST framework. We will illustrate that user-defined indexing is also very useful for UDTs by presenting performance experiments of using such indexes for advanced spatial analysis operators in the next section.

4 Case Study: Advanced Spatial Analysis

4.1 Data Types and Operators

Spatial index structures have been researched in great detail in the past. Operators supported by most of these indexes are spatial selections such as overlap or window queries. Some also support spatial join or nearest neighbor queries. Common for all these queries is that they only use the spatial information of objects for determining the query results.

Recent spatial applications on the other hand tend to use queries that combine spatial and thematical information in finding the desired objects. One could e. g. be interested in all counties in a given window where the median rent is below a given value. Or using even more thematic information, one could ask for all those counties where, in addition, the population is higher than another fixed value. For these queries traditional spatial indexes are only partly helpful, since they only support the spatial selection part of the queries. Especially in cases where the stronger restrictions are on the thematic attributes (e. g. a very low threshold value for median rent) the spatial index does not help at all. Since in many cases it is difficult or impossible to predict which queries will be used on a given table[3], it would be desirable to have a combined index that supports spatial-thematic queries of different selectivities equally well, regardless of the particular query parameters.

[3]Also any type of query could be posed equally often.

Consequently we suggest to define user-defined data types in ORDBS which combine spatial and thematic attributes in a single type[4]. A sample definition for `twoIntegerGeometry` combining a spatial and two numerical thematic attributes is given in figure 2. We use a type `OGCGeometry` as an implementation of the OpenGIS Consortium simple features specification here. Operators on these newly defined types can be conjunctions of spatial operators and operators on the types of the thematic attributes. In the sequel we will investigate the frequently used operator `twoBetweenOverlap` on the aforementioned type, which is a conjunction of spatial overlap operator and between operator on the thematic attributes. Since `twoIntegerGeometry` combines information from orthogonal domains in a single type, it is straightforward to use higher dimensional spatial indexes for the newly introduced operator. In particular, we obtain a four-dimensional domain for `twoIntegerGeometry`.

4.2 Performance Evaluation

An extensive performance evaluation for different datatypes can be found in the full version. If we use one thematic attribute in addition to the spatial attribute and thus use type `integerGeometry` and operator `betweenOverlap`, we obtain the results depicted in figure 3. Indexing options compared in this work are user-defined three-dimensional versions of R*- and RSS-Tree ([KL00]), as well as Oracle spatial indexes on the spatial component of such objects[5]. For the latter indexes the thematic attribute has to be checked separately in order to compute a correct query result. Figure 3 shows that this separate processing incurs a big overhead and thus leads to a pretty bad performance. User-defined indexes perform much better with the R*-tree outperforming the RSS-tree by a small margin. The unstable performance of the predefined indexes is due to them only considering the spatial component. Consequently, when the stronger restriction in the query parameters is on the spatial component they perform better, but worse if the stronger restriction is on the thematic attribute. Figure 3 clearly shows the usefulness of user-defined indexing, since only by using it, an efficient and predictable query performance can be achieved for the user-defined type considered.

Since the built-in indexes already performed bad on `integerGeometry` we do not consider them an alternative[6] for the more complex type `twoIntegerGeometry`. Figure 4 illustrates the performance of different user-defined indexes on queries using the operator `twoBetweenOverlap`. In particular the variants of the user-defined RSS-tree for different dimensionalities are compared; also the four-dimensional R*-tree can be compared with the RSS-tree. The figure clearly shows the benefits of using a multidimensional index structure as compared to a lower dimensional one, since, at least for low to medium dimensional data, the more dimensions are indexed the better the performance is. Also,

[4]Using a user-defined index on the attribute combination without defining a new type was not evaluated due to current technical restrictions that do not allow such a definition. Actually the introduction of such combined types should be subject to some automatic generation based on a specification of the attributes.

[5]Spatial indexes provided with ORDBMS currently do not support more than two dimensions.

[6]In fact, experiments have shown that they perform by far worse than on `integerGeometry`. Therefore we omit these results here.

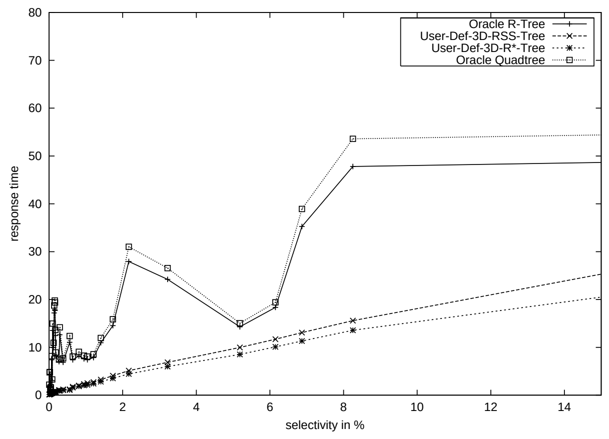

Figure 3: Performance Evaluation on 2D spatial data plus one thematic dimension

different from the indexes on `integerGeometry`, this time the full-dimensional RSS-tree is faster than the R*-tree for small selectivities. Consequently it is difficult to say which of the two variants should be preferred in real applications. But, since both perform almost equally well, the RSS-tree has the advantage of a faster index creation time due to its simpler, namely linear, insertion algorithm. More details can be found in the full version. Nevertheless we can already conclude that the performance gain from using a more appropriate index for a particular data type illustrates, that an easily adaptable indexing framework is required. Only by doing so one can obtain optimal performance with acceptable effort for each of the many possible UDTs.

5 Conclusion and Future Work

5.1 Conclusion

In this article we have presented a concept on how to integrate a flexible extensible indexing framework into recent commercial ORDBS. The need for this integration was motivated by the new features of ORDBS, namely the possibility to use user-defined data types. Since these types also have type-specific operators, type-specific indexes will be required in order to process user-defined operators efficiently.

The concept of integrating generalized search trees into an ORDBS was then applied to specialized spatial data types to prove its feasibility and also its effectiveness. In particular, spatial data enriched with one or two thematic attributes was used as data type. Operators on these types answer queries such as *retrieve all objects that lie in a given area where the median rent is below a given threshold value*. These operators are very frequent in advanced spatial analysis and therefore need index assistance.

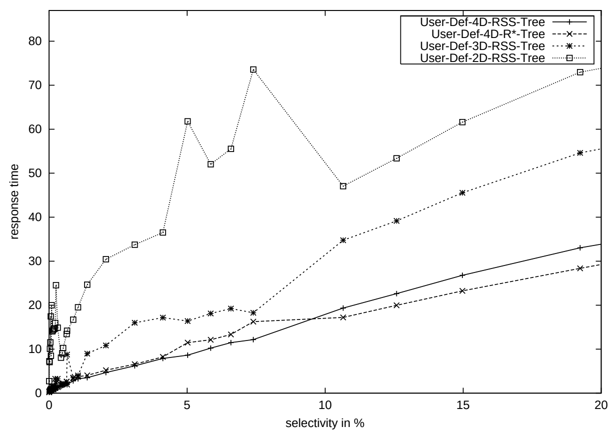

Figure 4: Performance Evaluation on 2D spatial data plus two thematic dimensions (standard query)

Our experiments showed that viewing the objects as elements of a three- (for one thematic attribute) or four-dimensional (for two thematic attributes) space and using spatial indexes extended to three or four dimensions shows best query performance. Overall the experiments showed the huge performance gain achieved by using user-defined indexing as compared to using system-provided indexes. The concept of OraGiST uses object-oriented techniques to simplify the development in two ways: firstly, if an existing index structure is used only the implementation of the interface defined by OraGiST is required. If on the other hand a different index is desired, its development is supported by using an extensible indexing framework such that only a new spezilization has to be developed.

5.2 Future Work

The concept has so far only been implemented in a prototype fashion. Firstly the implementation is currently restricted to GiST and Oracle 9*i*. An extension to other indexing frameworks and more importantly other ORDBS should be evaluated. Also the very good performance results shown previously only hold for rather simple objects. If we consider complex polygons in the spatial component, we currently obtain good results only for very small selectivities. These results could probably become better, if the OraGiST concept is extended to allow for a direct implementation of the refine step during two step query processing in the connector class. Currently the *functional* implementation (fallback) of an operator inside the DBS is used for refinement.

Our concept and prototype implementation should be tested with different, especially more complex, user-defined data types and operators. We expect the results to be even more impressive with such types, since no system-provided indexing is available at all. Also, tests with different operators on the given types, especially different spatial operators, could give an interesting overview of overall performance. Moreover to provide a complete data cartridge fine tuning of user-defined indexing by implementing user-defined cost and selectivity estimation is also missing so far. Conceptually we think, that an extensible framework-like approach could greatly simplify implementation for a particular UDT, similar to user-defined indexing. To be able to use such techniques a parametrized approach to cost and selectivity estimation in ORDBS has to be developed. In particular the processing of data type specific join queries could benefit significantly from optimized query execution, both by indexing and by cost and selectivity estimation. Research on data type specific joins in ORDBS is currently only in its early stages.

Bibliography

[BDS00] J. van den Bercken, J. P. Dittrich, and B. Seeger. javax.XXL: A Prototype for a Library of Query Processing Algorithms. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, May 16-18, 2000*. ACM Press.

[GG98] Volker Gaede and Oliver Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[Hel98] Joseph M. Hellerstein. Optimization Techniques for Queries with Expensive Methods. *ACM Transactions on Database Systems*, 23(2):113–157, June 1998.

[HNP95] J.M. Hellerstein, J.F. Naughton, and A. Pfeffer. Generalized Search Trees for Database Systems. In: *Proceedings of the 21th International Conference on Very Large Data Bases, Zurich, Sept. 11-15, 1995*. Morgan Kaufmann Publishers.

[KL00] Carsten Kleiner and Udo W. Lipeck. Efficient Index Structures for Spatio-Temporal Objects. In: *Eleventh International Workshop on Database and Expert Systems Applications (DEXA 2000; 4-8 September 2000, Greenwich, UK)*. IEEE Computer Society Press.

[KL02] Carsten Kleiner and Udo W. Lipeck. OraGiST - How to Make User-Defined Indexing Become Usable and Useful. Technical Report DBS 01-2002, Institut für Informationssysteme, Universität Hannover, September 2002.

[Kor99] Marcel Kornacker. High-Performance Extensible Indexing. In: *Proceedings of the 25th International Conference on Very Large Data Bases, Edinburgh, Sept 7-10, 1999*. Morgan Kaufmann Publishers.

[RKS99] Lukas Relly, Alexander Kuckelberg, and Hans-Jörg Schek. A Framework of a Generic Index for Spatio-Temporal Data in CONCERT. In: *Spatio-Temporal Database Management – Int. Workshop STDBM'99, Edinburgh, Sept. 10-11, 1999*. Springer-Verlag.

[RSV01] Philippe Rigaux, Michel Scholl, and Agnes Voisard. *Spatial Databases: With Application to GIS*. Morgan Kaufmann Publishers, 2001.

[SB99] Michael Stonebraker and Paul Brown. *Object-Relational DBMSs – Tracking the Next Great Wave (Second Edition)*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, 2^{nd} edition, 1999.

[SM96] Michael Stonebraker and Dorothy Moore. *Object-Relational DBMSs – The Next Great Wave – First Edition*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, 1^{st} edition, 1996.

[SR86] Michael Stonebraker and Lawrence A. Rowe. The Design of POSTGRES. In: *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*. ACM Press.