# Performance of Querying Temporal Attributes in Object-Relational Databases

Carsten Kleiner, Udo W. Lipeck
University of Hannover
Institute for Information Systems
30159 Hannover, Germany
{ck|ul}@informatik.uni-hannover.de

## Abstract

*In this paper we evaluate a model for temporal data utilizing the benefits of object-relational database systems (ORDBS). In particular we show how attribute time-stamping can be efficiently implemented in state-of-the-art ORDBS. The attribute timestamping concept is based on introducing user-specific types for temporal versions of datatypes. Moreover on the physical level we make use of user-defined index structures; in particular adapted spatial indexes based on generalized search trees are applied. These index structures greatly improve performance of important operators on both valid and bitemporal datatypes and show the effectiveness of this approach.*

**Keywords:** ORDBS, temporal datatypes, attribute time-stamping, physical design, user-defined index structures

## 1. Introduction and Related Work

Many recent complex applications involve time-varying information. Dependency on time is in most cases not a feature of a real-world object alone but also of its attributes. Temporal data is treated differently from spatial data; whereas the spatial position or extent of an object is usually perceived as a *regular* attribute of an object, time is more fundamental. All information (i. e. all attributes, standard as well as spatial) may be subject to change over time. Time seems to be the governing dimension behind all information.

In the spirit of object-oriented modeling using time on the attribute level as opposed to the class level should be preferred. This facilitates one-to-one modeling of real-world objects. It essentially means using attribute-timestamping as opposed to the widely used tuple-timestamping. We even propose to go one step further in modeling and suggest that combinations of attribute values and time should be seen as basic value units; thus we say that the *datatypes* to be used in temporal applications should be temporal versions of basic datatypes, e. g. `vt_integer`. The reason for using 'datatype-timestamping' is the more adequate modeling capability.

Due to space constraints we only give a brief overview of the new concepts; details can be found in [6] as well as [4]. The theoretical foundation of the conceptual model for attribute-timestamping is derived from the bitemporal conceptual data model ([3]). ORDBS were introduced and described in detail in [8]. Temporal extensions of ORDBS based on tuple timestamping have been presented recently by [1] and [12]. In contrast to those, our approach is rather based on attribute timestamping; such models were e. g. described in [9], but not considered in recent years since they are difficult to implement efficiently in purely relational database systems[1]. The connection of user-defined datatypes, attribute-timestamping and ORDBS has to the best of our knowledge never been researched before. Generalized search trees (GiST) which we will use as foundation were introduced in [2] as extensible indexing structures for databases. The idea of using spatial index structures for indexing pure temporal data has been used in several papers and is described in [7]. In this work we extend that idea to the newly defined temporal datatypes and embed it into an extensible indexing framework.

## 2. Physical Design

The physical model is developed by defining temporal versions of datatypes (conceptual and logical model are explained in [6]).

### 2.1. Operators

For temporal datatypes as combinations of temporal and standard datatypes (e. g. `vt_integer` for valid time dependent integer values or `bt_integer` for bitemporal integer values), queries can be differentiated by which dimen-

[1]or internally modeled by tuple timestamping

sions the desired results are restricted to, and whether they query for a *range* in a dimension or a *point*. For these query types we can use the notational conventions of [10], e. g. '*range//point/range*' means querying for a range in the standard attribute domain, a certain point in valid time and a range in transaction time. In addition, our experiments showed that the size of the range in a *range* query is an important measure. Therefore experiments on range queries were performed with several different selectivities.

The sample results for temporal queries in section 3 are presented with the objective of finding the optimal index structure for **all** possible query types. This is important, since in most cases only one index can be built on a single column, and it should provide for good query performance of all temporal, thematic and combined operators.

## 2.2. Indexing

By using the extensibility features of ORDBS like Oracle9*i* as described in [4] together with the generalized search tree approach ([2]) one can add appropriate index structures. We use attribute timestamping here and implement the index in an extensible framework (GiST). A similar approach for tuple timestamping without using the GiST framework has been described for Informix in [1]. Similarly no extensible framework was used for other temporal index structures such as RI-Tree, GR-Tree and 4R-Tree. Also these indexes only use the temporal information but not the thematic information. Thus only temporal queries will be efficiently supported. Thematic and combined queries will perform poorly (cf. spatial indexes on temporal queries evaluated in section 3, such as 2D R*-Tree on valid time intervals).

Since temporal datatypes combine values from different domains or dimensions into a single item to be indexed, the use of index structures from spatial databases seems straightforward. One can view the standard value (such as salary) as one dimension and the valid (as well as the transaction) time information as another dimension[2]. Thus in the case where the standard datum is from a one-dimensional domain (such as `integer`) we obtain two- or three-dimensional domains for temporal datatypes which may be indexed by spatial indexes. E. g. the temporal integer value (3000,[10,30)) is treated like the two-dimensional interval [(3000,10),(3000,30)]. Since the standard datum may not remain one-dimensional (e. g. spatio-temporal data) it is not recommended to use the 2D (or at best 3D) spatial indexes shipped with current ORDBS, but rather to use user-defined extensible indexes which may be adapted to as many dimensions as required.

[2]Special properties of the valid and transaction time dimension such as *now* or *UC* should be simple to add due to the extensibility features of the index structures used.

## 3. Performance Evaluation

### 3.1. Efficiency for Valid Time Data

For the temporal datatype `vt_integer` performance tests with different index structures and query types were conducted on synthetically generated data as well as on data generated using the SPYTIME-benchmark. Comprehensive results can be found in [6], figure 1 shows sample results on *range//range* queries of different selectivities.

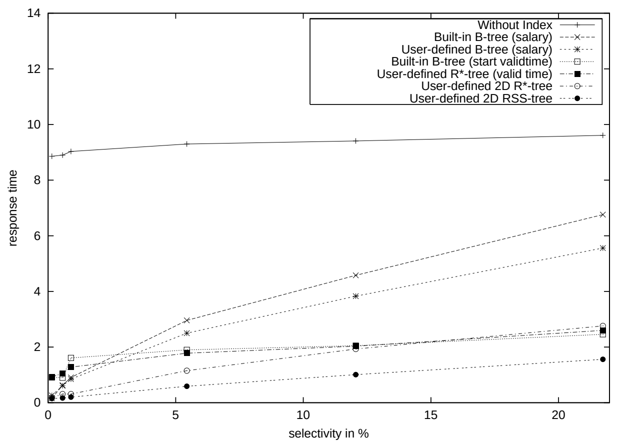

**Figure 1. Index Performance on `vt_integer`**

Using no index at all or simply a B*-tree on salary (built-in or user-defined) led to inacceptable results as expected. The creation of a B*-tree on start point of the valid time interval showed acceptable performance but requires a complex recoding of queries to be able to use the index. Also this approach would not be scalable to more complex datatypes.

The 2D RSS-Tree[3] outperforms all other indexes for all selectivities, since it provides good clustering and subdivision of space in each dimension, not just in one as several others. The distance based clustering leads to outperforming the 2D R*-Tree. A little bit surprising is the good performance of the R*-Tree on valid time intervals which almost outperforms its 2D counterpart. This is due to the low execution times for the operator on the integer component and would not scale well for more complex datatypes. The idea of using two-dimensional indexes significantly improves query performance, especially for the 2D RSS-Tree by factors of between 2 and 6.

For *range//point* queries we obtained a similar picture but this time the R*-Tree on valid time was slightly faster than 2D RSS-Tree and 2D R*-Tree, since it can optimally exploit the query point information in valid time. *Point//range* queries showed exactly reversed results with

[3]A combination of R*- and SS-Tree, see [5, 11] for details.

the 2D RSS-Tree being faster by a much bigger difference. Thus the 2D-RSS-Tree should be used, since it provides a good performance over all possible query types.

### 3.2. Efficiency for Bitemporal Data

The bitemporal datatype `bt_integer` can be interpreted as multidimensional data in the form of rectangles in three-dimensional space. Figure 2 summarizes performance of the different indexing options exemplarily for *range//range/range* queries.

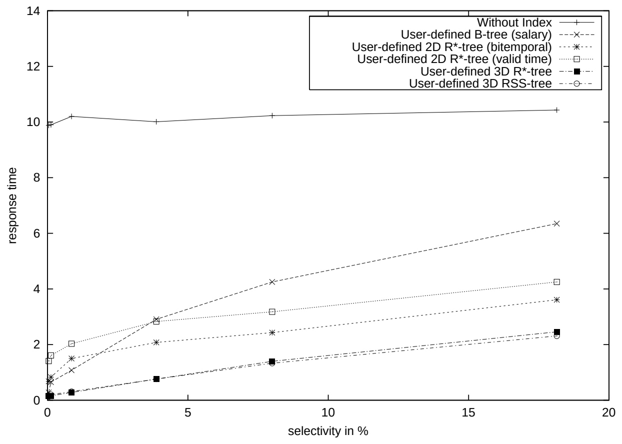

**Figure 2. Index Performance on `bt_integer`**

Options that already performed poorly on datatype `vt_integer`, such as no index, built-in or user-defined B*-tree on salary, again performed poorly and are not considered any further. The three-dimensional indexes outperform all other indexes significantly by factors between 2 and 6 where results for 3D R*-Tree and 3D RSS-Tree were almost the same this time. Among the better indexes are user-defined 2D R*-Trees (either on valid time intervals or on bitemporal rectangles). If *point//range/range* queries are also considered the pure temporal indexes perform extremely bad since they do not support the strongest restriction on salary. In this case the B*-Tree index on salary is the most efficient index as expected, but the three-dimensional indexes also perform pretty good.

To support all kinds of operators on `bt_integer` the only choice can be the 3D indexes; differences among R*-Tree and RSS-Tree are not significant. Coupled with the results for `vt_integer` the RSS-tree should be preferred.

Similarly to `vt_integer` the index entries are not approximations of real objects but rather the objects themselves. Thus query results can be directly taken from the index-based filter step and no refinement is necessary. This greatly improves index performance since the call overhead associated with exact operators is absent. Thus no user-defined selectivity estimation is required since an index-based execution is always faster than a full table scan.

## 4. Future Work

Our work focused on the physical implementation of temporal information. In order to use it for a complete temporal database it needs to be embedded into a temporal query language and user-friendly environment.

The experiments with spatial indexes need to be extended to different temporal datatypes. In particular the case of complex base types, leading to higher dimensional temporal types would be interesting. Results for spatio-temporal data (valid time only) will be reported in [4] but more work is required. Also, details about user-defined selectivity estimation will have to be investigated. Choosing the appropriate execution plan will become more important with more complex datatypes. Finally the special characteristics of *now* and *UC* (*until changed*) in valid and transaction time, respectively, have to be taken into account in more detail in future work.

## References

[1] R. Bliujute, S. Saltenis, G. Slivinskas, and C. S. Jensen. Developing a datablade for new index. In *Proceedings of $15^{th}$ International Conference on Data Engineering (ICDE'99)*

[2] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of the $21^{th}$ International Conference on Very Large Data Bases 1995*

[3] C. Jensen, M. Soo, and R. Snodgrass. Unifying temporal data models via a conceptual model. *Information Systems*, 19(7):513–547, 1994.

[4] C. Kleiner. *Modeling Spatial, Temporal and Spatio-Temporal Data in Object-Relational Database Systems*. PhD thesis, Universität Hannover, 2002. in preparation.

[5] C. Kleiner and U. W. Lipeck. Efficient index structures for spatio-temporal objects. In *Proceedings of $11^{th}$ International Workshop DEXA 2000*

[6] C. Kleiner and U. W. Lipeck. *Natural and efficient modeling of temporal information with object-relational databases*. Technical Report 01-2002, Universität Hannover, Apr. 2002.

[7] Y. Manolopoulos, Y. Theodoridis, V. J. Tsotras. *Advanced Database Indexing*. Kluwer Academic Publishers, 2000.

[8] M. Stonebraker and P. Brown. *Object-Relational DBMSs – Tracking the Next Great Wave (Second Edition)*. Morgan Kaufmann Publishers, $2^{nd}$ edition, 1999.

[9] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993.

[10] V. J. Tsotras, C. S. Jensen, and R. T. Snodgrass. An extensible notation for spatiotemporal index queries. *SIGMOD Record*, 27(1):47–53, Mar. 1998.

[11] S. Wang, J. M. Hellerstein, and I. Lipkind. *Near-neighbor query performance in search trees*. Technical Report CSD-98-1012, University of California, Berkeley, Sept. 1998.

[12] J. Yang, H. C. Ying, and J. Widom. Tip: A temporal extension to informix. In *Proceedings of the ACM SIGMOD International Conference on Management of Data 2000*