# SECONDO: A Platform for Moving Objects Database Research and for Publishing and Integrating Research Implementations

Ralf Hartmut Güting Thomas Behr
Christian Düntgen
Faculty of Mathematics and Computer Science
University of Hagen, 58084 Hagen, Germany
{rhg, thomas.behr, christian.duentgen}@fernuni-hagen.de

**Abstract**

*Databases supporting time dependent and continuously changing geometries, called* moving objects databases*, have been studied for about 15 years. The field has been flourishing and there exist many hundreds, more likely thousands, of publications. However, very few of these results have made it into systems (research prototypes or commercial) and are available for practical use today.*

*It is not that the publications are purely theoretical. In most cases data structures and algorithms have been proposed, implemented, and experimentally evaluated. However, whereas there exists a well established infrastructure for publishing research papers through journals and conferences, no such facilities exist for the publication of the related prototypical implementations. Hence implementations are just done for experiments in the paper and then usually abandoned. This is highly unfortunate even for research, as future proposals of improved algorithms most often have to reimplement the previous techniques they need to compare to.*

*In this paper we describe an infrastructure for research in moving objects databases that addresses some of these problems. Essentially it allows researchers to implement their new techniques within a system context and to make them available for practical use to all readers of their papers and users of the system. The infrastructure consists of* SECONDO*, an extensible database system into which a lot of moving object technology has been built already. It offers BerlinMOD, a benchmark to generate large sets of realistic moving object trajectories together with a comprehensive set of queries. Finally, it offers* SECONDO *Plugins as a facility to publish new research implementations that anyone can merge with a standard* SECONDO *distribution to have them run in a complete system context.*

## 1 Introduction

Moving objects databases allow one to represent and query in a database the time dependent location of mobile entities such as vehicles or animals, either online for current movement, or offline storing large sets of trajectories, or histories of movement. This has been a very active field of research since about the mid-nineties. There exist probably several thousand related publications. A large fraction of them addresses practical issues


**Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**

such as indexing and query processing algorithms and the respective paper describes an implementation and an experimental evaluation. However, only very few of these implementations have ever been integrated into some larger system environment, and only a few of them are still available today for practical use or comparison with other solutions. The basic reason is that the current methodology and infrastructure for experimental research in databases is geared to publishing papers, not software. As a consequence, the field has grown much more in terms of papers than of systems.

This is unfortunate as the transfer of research results to practical applications is quite difficult. Reimplementing published solutions may require several months of work. Combining many such solutions into an application will often be not feasible. It is also unfortunate for research itself as the quality of research deteriorates (as explained below) and a lot of unnecessary double work arises.

The current methodology can roughly be described as follows. A new data structure or algorithm is proposed, say, a new type of index structure, or a query processing algorithm. The authors describe their proposal and implement it. To prove the new proposal is worth publishing, they have to provide an experimental evaluation which needs to include a comparison with the strongest competing proposals from the literature. Unfortunately in most cases the competing algorithms need to be reimplemented, because the original implementations of their authors are not available or suitable for comparison. Hence the authors take the effort to also reimplement the competitors, perform their experiments and report them in the paper. Assuming the paper is accepted, this is the end of the story. The implemented software is abandoned. Anyway, it is only suitable to be used in a very specialized system context for performing experiments and doing measurements. There is no way to use it in a practical system or in real applications.

Original implementations of competing proposals may be available on request from the authors in a few lucky cases, but most often they are not. There are many reasons: The Ph.D. student who did it, left. The software was written some years ago and not maintained. It was buggy in the first place and barely able to execute the experiments of the original paper (omitting those that did not work). It was undocumented. It was written in a different programming language. And so forth.

That these algorithms need to be reimplemented by the authors of another proposal is bad for several reasons. First, it is a waste of resources. Second, there is a great danger that in the reimplementation errors are made. Even with the best effort, it is easily possible that subtle points in the descriptions in the respective papers have been misunderstood. Possibly some issues have not even been described clearly or at all. Third, the authors of the new proposal are of course interested in demonstrating that their new algorithm is better than the competitors. It exercises a lot of discipline in them to make sure that within the competing implementations everywhere the most efficient technique is used and minor details, that however might severely deteriorate performance, are treated right.

The lack of the software being published with the paper also has a negative impact on the scientific quality of the publication. Authors design certain experiments with certain data sets, varying some parameters. Although referees try to make sure that this has been done carefully, in many cases questions remain. How would this algorithm behave for this other parameter combination? What were the exact properties of the data set? Could they have had a special impact on this algorithm? If the competing algorithms were available with the publication and could easily be run with other parameters or data sets, all such questions could be clarified. Definitely results would be more reliable. Moreover, even years after the publication, issues could be reexamined.

The research community is aware of some of these issues. For example, there is a trend to encourage experimental repeatability, as shown at the last SIGMOD conferences. VLDB has an "Experiments and Analyses Track" that aims at providing a prestigious forum for careful experimental investigation of known techniques.

In this paper, we propose an infrastructure that allows authors to publish their research implementations together with the papers in such a way that readers of the paper can directly run the software, repeat the experiments, and even extend the experiments using other parameters and data sets. The infrastructure is especially attractive for research implementations for moving objects databases. This is because large parts of the MOD data models of [12, 13] have been implemented, benchmark data and queries [7] are available, and spatial as

well as moving objects can be visualized and animated.

The infrastructure consists of the SECONDO system, a DBMS prototype that has been designed as an extensible system from the beginning. SECONDO is in particular extensible by so-called algebra modules. Furthermore, it includes the plugin facility which allows one to describe and pack extensions.

The basic idea is that the author $A$ of a paper wraps her implementation as a SECONDO algebra module, packs it into a plugin, and publishes it on her web site. The reader can get a SECONDO system from the standard distribution, get the plugin from the author's web site, and merge them by a simple command. The reader can then try algorithms in SECONDO and redo and extend experiments.

A future researcher who wishes to propose an improvement to $A's$ algorithm can implement the new algorithm in SECONDO and compare to $A's$ original implementation by just running it.

In the following sections we describe in a bit more detail the architecture of SECONDO, the moving objects implementations that are available, the BerlinMOD benchmark, and the plugin facility.

# 2 SECONDO

Research implementations for papers, as discussed in the previous section, might be integrated into various DBMS environments. For moving objects related implementations, PostGIS or OracleSpatial might be interesting candidates. In this section we argue that SECONDO is a particularly suitable environment. SECONDO is a prototype DBMS developed at University of Hagen since about 1995. It runs on Windows, Linux, and MacOS X platforms and is freely available open source software [4]. The main design goals were a clean extensible architecture and support for spatial and spatio-temporal applications. In the sequel we address the following questions:

- Why is SECONDO suitable for publishing research extensions?
- Why is SECONDO suitable for publishing implementations related to moving objects?

We also provide an example of an extension and explain how it benefits from the environment.

## 2.1 Extensible Architecture

SECONDOs architecture is shown in Figure 1. It consists of three major components: the kernel, the optimizer, and the GUI. The kernel does not implement a fixed data model but is open for implementation of a wide

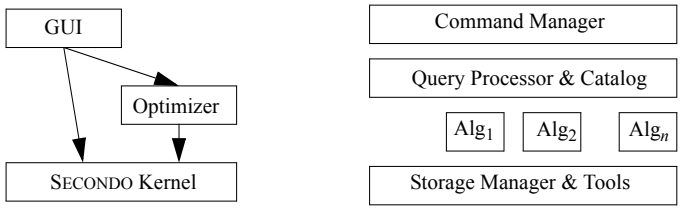


Figure 1: SECONDO components (left), architecture of kernel system (right)

variety of DBMS data models. The kernel is extensible by algebra modules. In fact, the entire implementation of a particular data model is done within algebra modules. An algebra module generally offers a set of type constructors and a set of operators. A type constructor is a (parameterized) data type and is defined via a signature.

| | | |
|---|---|---|
| $\underline{int}$, $\underline{real}$, $\underline{bool}$, $\underline{string}$: | | $\rightarrow$ DATA |
| $\underline{tuple}$: | (IDENT $\times$ DATA)+ | $\rightarrow$ TUPLE |
| $\underline{rel}$: | TUPLE | $\rightarrow$ REL |

For example, the above signatures define type constructors for the relational model. Here $\underline{rel}$ is a type constructor applicable to all types in the kind TUPLE, returning a type in kind REL. The terms generated by such signatures define the available data types. Over the data types, operations can be defined, for example

| | | |
|---|---|---|
| + : | $\underline{int} \times \underline{int}$ | $\rightarrow \underline{int}$ |
| $feed$: | $\underline{rel}(tuple)$ | $\rightarrow \underline{stream}(tuple)$ |
| $filter$: | $\underline{stream}(tuple) \times (tuple \rightarrow \underline{bool})$ | $\rightarrow \underline{stream}(tuple)$ |
| $consume$: | $\underline{stream}(tuple)$ | $\rightarrow \underline{rel}(tuple)$ |

An algebra module implements its type constructors and operators. Essentially for a type constructor a data structure and for an operator an evaluation function need to be provided. Some more support functions must be written, e.g. type checking functions for type constructors and for operators.

Algebra modules encapsulate everything needed to implement a DBMS data model, hence there are algebras in SECONDO for basic data types, for relations and tuples including operations such as hashjoin, for B-Trees and R-trees with their access operations, for spatial and spatio-temporal data types, and many more. There are also algebras beyond the scope of a relational model such as for nested relations or networks.

The kernel is an engine to evaluate terms over the existing objects and operations. For example, it evaluates expressions:

```
query 3 + 5
query Trains feed filter[.Trip present sixthirty]
  filter[length(trajectory(.Trip)) > 2000.0] count
```

Here *Trains* is a relation containing trajectories represented in an attribute *Trip* (of type $\underline{mpoint}$ = moving point in SECONDO). Syntax for operations can be freely chosen and it is often practical to use postfix notation for query processing operations. For example, the first argument to $filter$ is *Trains feed*. Stream processing is built into the engine. The commands and queries processed directly by the kernel are called the *executable language*. The kernel is written in C++ and uses BerkeleyDB as the underlying storage manager.

The optimizer is not as data model independent as the kernel. It assumes an object relational model and supports an SQL-like language. It maps SQL to the executable language shown above. The optimizer is extensible by registering types and operations of the executable level, by translation rules, and cost functions. It allows for extension by new index types, providing concepts to distinguish between logical and physical indexes (a physical index is a particular index structure available in the kernel, a logical index is a strategy to use it, which may be complex). The optimizer determines predicate selectivities by a sampling strategy which is the only feasible way to support predicates with arbitrary data type operations. The optimizer is written in Prolog.

The GUI allows one to send commands and queries to a kernel and visualize the results. It supports both the executable level language and the SQL level. In the latter case it interacts with the optimizer to get a plan (executable query) which it then sends to the kernel. The GUI is extensible by so-called viewers which can offer their own methods to display data types. One of the available viewers (the so-called Hoese-Viewer) allows for a sophisticated representation of spatial data and for animation of spatio-temporal data types. This viewer is itself extensible to support further data types. The GUI is written in Java.

SECONDO is suitable for publishing research implementations for the following reasons:

- It offers clean concepts and interfaces for adding data structures and algorithms as type constructors and operators.
- A complete DBMS interface for data manipulation and querying is available at the executable level. Queries at this level are completely type checked. This is crucial for experiments as one can call query processing operators directly without any need to play tricks with a query optimizer. Very often it is also not clear how the new query operations could even be expressed in SQL. SECONDO allows one to focus on the query processing level, as many research papers do. We are not aware of any other DBMS that allows one to type in query plans directly.

- SECONDO provides simple and efficient concepts to deal with data types of widely varying size, from very small to very large. Basically small value representations are embedded into tuples, large representations are stored in separate records. This is transparent for the implementor of a data type. Such capabilities are crucial to deal with spatial or moving object types for which sizes are unpredictable.
- Query optimizer and GUI also support extensions as described above.
- Commands and queries of the executable level can be written into SECONDO scripts (text files with commands). Such scripts can be used to set up experimental data and to run experiments.

## 2.2 Support for Moving Objects

SECONDO is suitable for publishing implementations related to moving objects for the following reasons.

- Several algebras implement large parts of the data model of [12, 8], the fundamental data model for unconstrained movement (i.e. free space movement described by $(x, y)$ coordinates). It provides representations for complete trajectories, but also for elements of trajectories (units, short linear pieces), as data types $\underline{mpoint}$ and $\underline{upoint}$. One can freely convert between a relation containing $\underline{mpoint}$ attributes (called the compact representation) and one having $\underline{upoint}$ attributes (unit representation).
- Beyond the trajectory types, the type system of [12] has a rich set of related standard, spatial, and time dependent types such as $\underline{mint}$, $\underline{mreal}$, and $\underline{mbool}$, representing time dependent integer, real, and boolean functions, respectively. SECONDO has implementations for all these types and for a large set of the related operations. They are crucial for formulating queries on moving objects.
- Furthermore, also a large part of the model for network-constrained movement [13] has been implemented. There are algebras offering the network data type (with a graph-based representation) and types for network based static and moving objects. At least all operations to execute the BerlinMOD benchmark on a network-based representation are available.
- R-trees and TB-trees are available for spatio-temporal indexing.
- The GUI provides visualizations and animation for all implemented data types of [12].
- A simple data set for moving objects is included in the SECONDO distribution, the *Trains* relation in database *berlintest*. By simple commands, this data set can be scaled up to arbitrary size. A further big and realistic data set *Cars* can be created by running the BerlinMOD benchmark (Section 3).

Hence there is a rich infrastructure into which new indexing or query processing methods can be integrated.

## 2.3 An Example: Nearest Neighbor Extension and Plugin

As an example, we consider a recent research paper whose implementation has been published as a SECONDO plugin [11]. The paper addresses the problem of finding the continuous $k$ nearest neighbors to a query trajectory in a large set of stored trajectories. The result is a set of parts of trajectories.

Stored trajectories are given in unit representation and indexed in a 3D R-tree. For each node $p$ of the R-tree, in preprocessing its coverage function is computed, which is the time dependent number of trajectories represented in $p$'s subtree.

The solution uses a filter and refine strategy. In the filter step, the R-tree is traversed. Based on the coverage numbers of nodes, some nodes can be pruned. The filter step returns a stream of units ordered by start time which are guaranteed to contain the $k$ nearest neighbors. The refinement step then determines the precise pieces of units forming the $k$ nearest neighbors based on a plane sweep determining the $k$ lowest distance curves.

The nearest neighbor extension benefits from the environment as follows:

- It uses the existing data types for moving point (trajectory) and for units.
- It uses R-trees and especially the bulkload facility to build its own index structure. It uses TB-trees to implement competing solutions.
- Coverage functions for R-tree nodes can be directly represented in a data type $\underline{mint}$ (moving integer). Hence an operation can be written in SECONDO that traverses an R-tree and returns a stream of tuples with coverage numbers, to be stored in a relation.
- Visualization of coverage functions and of R-tree nodes, available in SECONDO, has been crucial to determine an efficient shape of the R-tree index, which is constructed in the bulkload in a special way.
- Test data sets as scaled versions of *Trains* can be set up easily using SECONDO commands. A data set for long trajectories is created within the BerlinMOD benchmark.
- The algorithms for the filter step and for the refinement step can be defined as SECONDO operators $knearestfilter$ and $knearest$, respectively. They are offered within a *NearestNeighborAlgebra*. A query using these operations can be written as follows (see [11] for a detailed explanation):

```
query UnitTrains25_UTrip UnitTrains25 UnitTrains25Cover_RecId
  UnitTrains25Cover knearestfilter[UTrip, train742, 5]
  knearest[UTrip, train742, 5] count;
```

- Competing algorithms [9, 10] have been implemented in the same environment as operators $greeceknearest$ and $chinaknearest$, respectively. They can be run by queries:

```
query UTOrdered_RTreeBulk25 UTOrdered25 greeceknearest[UTrip, train742, 5] count;
query UnitTrains_UTrip_tbtree25 UnitTrains25 chinaknearest[UTrip,train742,5] count;
```

The plugin for [11] can be found at [3]. Scripts for repeating the experiments are available at [2].

# 3 BerlinMOD

**Purpose** Benchmarks have become the standard method to compare different DBMS. Each benchmark consists of a well defined data set and a workload, usually a set of queries. Though data structures, index structures and different operator implementations can be compared separately from other components, their impact on database performance becomes clearer when they are tested all together within in a real system. Benchmarks benefit researchers by simplifying the setup and description of experiments used to assess the efficiency of proposed inventions: The data properties are well defined and studied, the risk of introducing bias is minimized, and it becomes easier to repeat experiments. With BerlinMOD [7], we have proposed a benchmark to support research in the context of historic moving object database systems/ trajectory database systems.

**Data Generation** Though using moving object data (MOD) collected from the real world is widely considered preferrable, there exist severe problems with the amount and character of such data. Often, the trajectories are short, there are only few of them, or legal issues — such as data privacy or copyright — render their free use and distribution practically impossible. Therefore, BerlinMOD uses artificial MOD created by a data generator. The generator is implemented as a SECONDO script and allows for analysis and modification. With standard settings, data are obtained by long time observation (28 days) of 2,000 simulated vehicles' positions. The movements created are representative for employees' behaviour, who commute between their home and work location and do some additional trips (for visits, shopping, sports, etc.). The simulation uses geographic line data and tables with speed limits to create a traffic infrastructure network, and a combination of statistics on residential and

employers' locations and regions describing statistical districts to create representative destinations for trips. In the standard version, real world data on Berlin is used to this end.

The data generator itself is not a standalone application, but a SECONDO script. It uses SECONDO's available complex data types (like relations, vectors, R-trees, graphs) and operators from different algebras to process the data from only three simple input tables and some parameters into arbitrary amounts of representative moving object data. Mainly to increase the performance, some parts of the data generation have been implemented as a special algebra module (SimulationAlgebra). The position information is enriched with some standard data, like a unique licence plate number, a type and a brand for each vehicle, thus allowing for more rich semantics in queries. Also, some time instant, time period, point, region and standard data is created for the purpose of workload generation with varying range parameters. Data can be directly used within SECONDO or exported to shapefile or CSV format for import to any third party DBMS.

Whereas the observed area is determined by the dimensions of the imported map data, both the observation period and the number of observed vehicles is scalable. With standard settings, vehicle positions are mapped to positions on the road network, but it is also possible to create noisy data. Each trip is created with individual deceleration and stop events, considering the geometry and character of roads (inferred from their speed limit) used and crossed.

The amount of data generated with standard settings is considerable: 19.45 GB for 292,693 trips — in average, each trajectory consists of 26,963 positions. However, users may scale the data or even produce MOD from an alternative scenario by simply replacing the original map data or changing parameters within the data generator script.

**Queries** BerlinMOD defines two sets of queries. BerlinMOD/R provides 17 range and point queries formulated in common English and SQL. The queries apply combinations of simple to rather complex standard, temporal, spatial and spatio-temporal predicates, operations and aggregations to the dataset. Both, selection and join queries are covered. The selection allows for testing a wide range of index structures, access methods and implementations of spatio-temporal operators.

The second query set, BerlinMOD/NN, aims at nearest neighbour queries. Nine queries combine $k$-NN, reverse NN, and aggregated NN queries with both, static and moving query objects and candidate objects. While solutions to some of these combinations have already been proposed, others are still uncovered in the context of trajectory database systems. This part of BerlinMOD also gives a good prospect of how BerlinMOD can be extended to address new aspects of data processing while using standardized, existent data.

You can obtain BerlinMOD from a dedicated web page [1] on the SECONDO web site. As SECONDO itself, BerlinMOD is free and open software you are invited to use, examine, extend, publish and propagate free of all charges.

# 4 Plugins

To publish a plugin for a SECONDO extension, the implementor fills in an XML-file. Basically one needs to specify which SECONDO version is used, which dependencies to other algebra modules exist, and which extensions are provided in which files. Different XML tags describe the different kinds of extensions possible such as algebra modules, viewers, display classes for the Hoese viewer, and optimizer extensions. Afterwards, all files needed for the extension and the describing XML file are packed into a zip file which can be published at the author's web site. More details are given at the plugin web pages [3].

The reader of a paper referring to a plugin needs to have a working SECONDO installation which can be obtained from the SECONDO web site [4]. She gets the plugin *X.zip* from the author's web site. The SECONDO system contains an installer for plugins. The command `secinstall X.zip` integrates the components of

the plugin into the source code of the SECONDO system. Afterwards a call of `make` builds the system with the extension.

On the plugin website [3], we provide several plugins for download. These include implementations of a TB-Tree structure [14], an algebra containing X-Trees and M-Trees [5, 6], an algebra together with display classes for periodic moving objects, an algebra providing operators for finding the $k$ nearest neighbours to an object within a set of static or moving objects, and an algebra together with optimizer extensions for querying moving objects by their movement profile. The latter two are based on [11] and [15], respectively. The papers demonstrate how a publication can profit from the work invested into making the implementation available as a plugin.

# References


[1] BerlinMOD. http://dna.fernuni-hagen.de/secondo/BerlinMOD.html.

[2] Scripts to execute the experiments of the knn paper. http://dna.fernuni-hagen.de/papers/KNN/knn-experiment-script.zip.

[3] Secondo Plugins. http://dna.fernuni-hagen.de/Secondo.html/start_content_plugins.html.

[4] Secondo Web Site. http://dna.fernuni-hagen.de/Secondo.html/.

[5] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *VLDB*, pages 28–39, 1996.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435, 1997.

[7] C. Düntgen, T. Behr, and R.H. Güting. BerlinMOD: a benchmark for moving object databases. *VLDB J.*, 18(6):1335–1368, 2009.

[8] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *SIGMOD*, pages 319–330, 2000.

[9] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis. Algorithms for nearest neighbor search on moving object trajectories. *GeoInformatica*, 11(2):159–193, 2007.

[10] Y. Gao, C. Li, G. Chen, Q. Li, and C. Chen. Efficient algorithms for historical continuous $k$ nn query processing over moving object trajectories. In *APWeb/WAIM*, pages 188–199, 2007.

[11] R. H. Güting, T. Behr, and J. Xu. Efficient k-nearest neighbor search on moving object trajectories. Technical report, University of Hagen, Informatik-Report 352, 2009, *VLDB J.*, to appear.

[12] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and quering moving objects. *ACM TODS*, 25(1):1–42, 2000.

[13] R. H. Güting, V. Teixeira de Almeida, and Z. Ding. Modeling and querying moving objects in networks. *VLDB J.*, 15(2):165–190, 2006.

[14] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *VLDB*, pages 395–406, 2000.

[15] M.A. Sakr and R.H. Güting. Spatiotemporal pattern queries. Technical report, University of Hagen, Informatik-Report 352, 2009.