UPPSALA
UNIVERSITET

*Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 1352

# Main-Memory Query Processing Utilizing External Indexes

THANH TRUONG



ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2016

ISSN 1651-6214
ISBN 978-91-554-9509-1
urn:nbn:se:uu:diva-280374

Dissertation presented at Uppsala University to be publicly examined in 2446, ITC, Lägerhyddsvägen 2, Uppsala, Uppsala, Wednesday, 4 May 2016 at 13:15 for the degree of Doctor of Philosophy. The examination will be conducted in English. Faculty examiner: Professor Martin Kersten (National research institute for mathematics and computer science in the Netherlands (CWI)).

**Abstract**

Truong, T. 2016. Main-Memory Query Processing Utilizing External Indexes. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 1352. 45 pp. Uppsala: Acta Universitatis Upsaliensis. ISBN 978-91-554-9509-1.

Many applications require storage and indexing of new kinds of data in main-memory, e.g. color histograms, textures, shape features, gene sequences, sensor readings, or financial time series. Even though, many domain index structures were developed, very a few of them are implemented in any database management system (DBMS), usually only B-trees and hash indexes. A major reason is that the manual effort to include a new index implementation in a regular DBMS is very costly and time-consuming because it requires integration with all components of the DBMS kernel. To alleviate this, there are some extensible indexing frameworks. However, they all require re-engineering the index implementations, which is a problem when the index has third-party ownership, when only binary code is available, or simply when the index implementation is complex to re-engineer. Therefore, the DBMS should allow including new index implementations without code changes and performance degradation. Furthermore, for high performance the query processor needs knowledge of how to process queries to utilize plugged-in index. Moreover, it is important that all functionalities of a plugged-in index implementation are correct.

The extensible main memory database system (MMDB) *Mexima* (*Main-memory External Index Manager)* addresses these challenges. It enables transparent plugging in main-memory index implementations without code changes. Index specific rewrite rules transform complex queries to utilize the indexes. Automatic test procedures validate the correctness of them based on user provided index meta-data. Moreover, the same optimization framework can also optimize complex queries sent to a back-end DBMS by exposing hidden indexes for its query optimizer.

Altogether, Mexima is a complete and extensible platform for transparently index integration, utilization, and evaluation.

*Keywords:* Database indexing, query processing, index structures, main-memory, index validation

*Thanh Truong, Department of Information Technology, Division of Computing Science, Box 337, Uppsala University, SE-75105 Uppsala, Sweden.*



ISSN 1651-6214
ISBN 978-91-554-9509-1
urn:nbn:se:uu:diva-280374 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-280374)

# *Dedication*

*This humble thesis work is dedicated to:*

My parents ba Minh and má Thảo

My wife Diễm

My lovely daughters Anna and Lisa

# List of Papers

This Thesis is based on the following papers, which are referred to in the text by their Roman numerals.

I T. Truong and T. Risch: Transparent inclusion, utilization, and validation of main-memory domain indexes, *27th International Conference on Scientific and Statistical Database Management (SSDBM)*, San Diego, United States, June 29-Juli 1, 2015.
*I am the primary author of this paper.*

II T. Truong, T. Risch: Scalable Numerical Queries by Algebraic Inequality Transformations, *19th International Conference on Database Systems for Advanced Applications (DASFAA*), Bali, Indonesia, April 21-24, 2014.
*I am the primary author of this paper.*

III M.Zhu, S.Stefanova, T.Truong, and T.Risch: Scalable Numerical SPARQL Queries over Relational Databases, *4th international workshop on linked web data management (LWDM 2014)*, Athens, Greece, March 28, 2014.
*I am one of the co-authors of this paper.*

IV S.Badiozamany, L.Melander, T.Truong, C.Xu, and T.Risch: Grand Challenge: Implementation by Frequently Emitting Parallel Windows and User-Defined Aggregate Functions, *Proc. The 7th ACM International Conference on Distributed Event-Based Systems*, DEBS 2013, Arlington, Texas, USA, June 29 - July 3, 2013
*I am one of the co-authors of this paper.*

V K.Mahmood, T.Truong, and T.Risch: NoSQL Approach to Large Scale Analysis of Persisted Streams, *30th British International Conference on Databases, Edinburgh (BICOD)*, Scotland, July 6-8, 2015.
*I am one of the co-authors of this paper.*

# Contents

# Abbreviations

| | |
|---|---|
| DBMS | Database Management System |
| Mexima | Main-memory External Index Manager |
| BAO | Basic access operator |
| SSF | Special search function |
| ISF | Index sensitive function |
| MMDB | Main-memory database system |

# Acknowledgements

Many people helped to make this PhD possible. First and foremost, my supervisor Prof. Tore Risch. I appreciate your contributions of time, ideas, and funding to my Ph.D. Your enthusiasm for database research inspired and motivated me. It was countless time you caught my sloppy code, or pointed out that my arguments were either weak, or wrong. Honestly, several times I walked out of your office door and thought that I would never come back. However, I did bounce back thanks to your encouragements. It does not matter how much I was struggling along the way. It is important that I am finishing strong.

I would like to say many thanks to Anders Berglund for accepting me to MSc studies at Uppsala University, where I have met so many interesting people and friends. Your acceptance brought me on a very long journey that lasts several cold and dark winters and bright summers in Uppsala. There is no way I can pick up Rikssvenska but at least I think that surströmming is not that horrible.

To my fellow members in UDBL group, you guys made my PhD time here more memorable: Kjell Orsborn, thank you for assisting me in teaching assignments. Erik Zeilter, and Manivasakan Sabesan, thank you for literally alerting me to some obstacle I might encounter along the way. I am also thankful to Lars Melander, and Andrej Andrejev, whom I did not talk much with but I enjoyed our technical discussions during these years. Cheng Xu, I am grateful for your contribution to the accepted paper that we co-authored in 2013. Sobhan Badiozamany, you deserve my thanks for sharing your opinions in various things including research ideas, life in Sweden. Thanks you for giving me rides from Ultuna to Flogsta after playing futsal very late in 2014. Silvia Stefanova, I am thankful for your friendship and talks whenever we had break or lunch together. Minpengzhu, we have talked a lot regarding research, life, and other non-serious matters. I have learnt something from your determination, focus, and hard-works. As a side note, you have not ever said yes to a fika or beer somewhere in town. I will keep on asking you then.

To Matteo Magnani, thanks you for your friendship, your humor, and your Italian dishes that I gladly tasted every time. I also want to thank Yunyun Zhu for coordinating our language meetings. Yes, Lillemor Arronson, jag är tacksam för ditt hjälp varje gång vi träffas. I would like to thank Seif Alwan for your hospitality every time I come by and for listening to me complaining of

work and study. Ulrika Andersson, I want to thank you for your excellent support regarding working matters.

In regards to Vietnamese friends, I thank you for dinners, gatherings, in addition, soccer that we played together.

To all, whom I had the privilege of making friends with, we might not be able to talk or see each other often but you made my life more colorful and enjoyable. I am thankful for that.

I would like to thank my family for all their love and encouragement. For my parents ba Minh and má Thảo who raised me with curiosity for science and supported me in all decisions I made. You are my lifetime friends, sources of inspiration, and the ones whom I seek words of wisdoms from. To my sisters Thuỷ, Thư and my brother Thái, you might not interest in what I was doing these years but I am indebted to your caring and concern.

And Diễm. I have taken a long pause to think before I write this but I still do not know where I should begin. You are just amazing as my best friend, and the love of my life. We share not only dreams, goals, but also disappointments that life has thrown to us. Without your love, support, and encouragement, I would not have done my PhD. I am sure that our life will turn to a new chapter soon after this. Thank you for always being there for me.

Finally, Anna and Lisa, you girls are God's blessing to us and I thank him for it every day.

# 1 Introduction

Main-memory databases (MMDBs) [12] [30] [46] [80] [64][49] are common approaches for many applications such as financial analyses, real-time operating systems, industrial machine sensors, and scientific applications that require fast data access, storage, and manipulation. The emergence of such domain applications put new requirements on main-memory database systems to support new kinds of data, e.g.; color histograms, textures, shape descriptions in image databases, gene sequences in biology databases, sensor readings in machine-log databases, time series data in finance, etc. To efficiently access and manipulate such domain data, MMDBs need to include new kinds of domain indexes that provide scalable facilities to query and update domain-oriented data representations.

Even though many index structures were developed, very a few of them are implemented in database systems in practice; most database systems [29] [32] provide only B-trees and hash indexes. The reason is that it is very difficult to extend a DBMS with new index structures. The manual integration effort is very costly and time-consuming since the new index needs to interface with most subcomponent of the DBMS kernel. Therefore, for new emerging domain applications that need novel index structures, it may not be feasible to develop indexes from scratch and integrate them into the DBMS. The DBMS index manager should be extensible to enable including new index structures without changing the DBMS kernel.

Existing extensible indexing frameworks [36] [51] [53] [91] support adding new indexes but they require re-engineering the index code strictly following each particular framework's coding conventions and API primitives. This may still be a daunting task because the index may have third-party ownership, perhaps only binary code is available, or simply it is very complex to re-engineer the code. Therefore, one challenge is how to add new index implementations to a DBMS without code modifications.

When plugging-in a new index implementation to a DBMS, it is important to test that all index functionalities are correct. Correctness means that all index operations should return exactly the expected results and leave the database in a consistent state after updates or bulk loading. Therefore, another challenge is how to automate the test procedures for a new plugged-in index implementation.

Adding a new index to a DBMS requires teaching the query processor properties of the new index, e.g. its supported access methods and how to rewrite

queries to make the index utilized by the query optimizer. Furthermore, complex expression in queries, e.g. for advanced analytics, often hinder the query optimizer to utilize its indexes. Therefore, an extensible indexing system has a need for *extensible query processing* in which new index specific rewrite rules can be plugged in. Such index specific rewrites improve query performance. Extensible query rewrite mechanisms could also improve the performance of advanced queries sent to a regular DBMS by transforming queries involving complex expressions. Therefore, another challenge is how to plug-in index specific rewrite-rules in an extensible query processor for utilizing the new indexes in queries.

This Thesis addresses the above challenges of extensible indexing, index testing, and index-specific query transformations in a main-memory DBMS.

The following research questions are investigated:

1. The overall research question is: How should an extensible indexing system be designed to enable transparent inclusion of index implementations without code changes in neither the DBMS kernel nor the index?
2. How should the query processor of the extensible indexing system be provided with knowledge of the access methods of a new plugged-in index to utilize transparently the index in queries? In particular: How should the query processor be extensible with new index-specific rewrite rules so that the new plugged-in index is transparently utilized in queries?
3. How should the correctness of a plugged-in index be automatically validated? In particular, what are the functionalities to test and how can the testing be automated?
4. When data is stored in a back-end database, how should the query processor transform complex queries so indexes in the back-end DBMS are utilized?

To answer these questions, we developed an extensible MMDB system called *Mexima* (*Main-memory External Index Manager)*.

To answer Research Question one, Mexima enables plugging-in different kinds of main-memory index implementations without altering or re-engineering their original source code (Paper I). An *index extension developer* takes an existing *index implementation*, writes a simple *extension driver* (interface code), and then compiles the whole module as a dynamic library or shared object called an *index extension*. Mexima loads these index extensions at runtime. This inclusion requires little development efforts and no detailed knowledge about the DBMS kernel.

To answer Research Question two, for data operations on a plugged-in index, Mexima invokes corresponding index access and update operators (Paper I). The index operators that are common for all kinds of indexes are called *basic access operators (BAOs),* i.e. methods for creating, dropping, updating,

accessing, and mapping over indexed elements respectively. Moreover, each kind of index often has *special search functions (SSFs)* that utilize index specific properties for efficient search, e.g., interval search on B-trees [67], and K-nearest neighbor and proximity search on R-trees [1] and X-trees [5]. To utilize SSFs transparently in queries the system contains an *SSF translator* that transforms query conditions into calls to SSFs.

Furthermore, when queries involve complex inequality conditions, they may hinder the query optimizer from utilizing the presence of indexes. This causes expensive scans of entire tables rather than direct index access calls. The *Algebraic Inequality Query Transformations (AQIT)* (Paper II) transforms complex queries involving inequalities into equivalent ones, which are more efficient by exposing hidden indexes.

To answer Research Question three, for each index type, the *Mexima tester* generates a number of queries that automatically test the correctness of the index implementation. The test queries use *data generators*, which are queries specified by the index extension developer to generate relevant data for testing BAOs and SSFs (Paper I).

To answer Research Question four, Mexima allows optimizing complex numerical queries sent to a back-end relational database (Paper II). The same AQIT rules used to utilize plugged-in main-memory indexes can also be used for exposing indexes in back-end relational DBMSs. Furthermore, scalable execution of rewritten complex numerical queries sent to the back-end relational database requires translating the numerical expressions into SQL (Paper III). This avoids data transfer from the backend database and enables the back-end query optimizer to utilize indexes.

This Thesis overview is organized as follows. Chapter 2 presents technology background and related work. Chapter 3 describes Mexima's architecture, including its query processing based on rewrite rules. Chapter 4 concludes the Thesis and gives some future work discussions. Finally, Chapter 5 summarizes papers I-V and states my technical contributions to each of them.

# 2 Background and Related Work

A *database* is a collection of information that is organized by a general-purpose software system called a *database management system (DBMS)* [60]. The DBMS enables creation, construction, manipulation, and maintenance of databases. A *data model* is the paradigm used by a DBMS for representing the structure of its databases. For example, in the relational data model [18] databases are represented as a set of tables having columns and rows. The database is manipulated by *queries*. In a DBMS, the *query processor* is responsible for translating queries into a *query plan*, which is a sequence of database operations executed by the DBMS kernel. The *query optimizer*, a component of the query processor, determines for a given query an optimized query plan likely to be the most efficient way to execute it.

In this Thesis, we focus on two important aspects: database indexing and query processing in main-memory database systems.

## 2.1 Main-memory database system

Most major DBMSs are designed to store data on disk and bring disk pages into main-memory as needed for processing. This model assumes computers with main-memory smaller than the databases. Nowadays, a normal computer has enough main-memory to fit most databases entirely. Therefore, to take full advantage of modern hardware, a new class of database systems was introduced: *main-memory database systems (MMDBs)* [4] [12], [29], [30]. MMDBs are most commonly used in applications that demand fast data access, storage, and manipulation, For example: enterprise applications [89], sensor networks [76], industrial data [78], and telecom applications [56]. In addition, for applications running on traditional disk-based DBMSs, when data is skewed some frequently accessed “hot” data can be kept in a main-memory database. For this, major DBMS vendors have developed their own main-memory database processing integrated with their disk-based DBMSs, e.g., Oracle TimesTen [82], Heckaton in SQL Server [10], and MySQL in-memory tables [58]. Furthermore, in recent years parallel MMDBSs have been in focus [37][86][83][62], with commercial products such as SAP HANA [77] [89], MySQL Cluster [62], and VoltDB [48].

Colum-store systems such as MonetDB [80], VectorWise [64][49], C-Store [54] (or its commercial version Vertica [2] ), SybaseIQ [73], etc., are designed

to exploit the large main memories of modern computer systems efficiently when processing analytical and aggregate queries overs database in memory-mapped files. In particular, they partition a database into a collection of individual columns that are compressed and stored separately. It enables processing only the needed columns, rather than entire rows and discards other unneeded columns.

Even though main-memory database systems have been extensively studied in the past, the area of extensible indexing and query processing in a main-memory DBMS is little studied [29] and is the focus of this Thesis.

The next section discusses why indexing is needed in the context of main-memory database systems.

## 2.2 Indexing

A DBMS uses *indexes* to speed up the retrieval of records in response to certain search conditions [60]. An index associates a given key with a collection of addresses of matching records. To avoid physically scanning all records in a table for a given search condition on some search key, the DBMS can use the index to access only the relevant records. In a relational database, there can many indexes created for each table and an index can be associated with a single column or several columns. An index is *utilized* by a query when the query optimizer is able to use the index in the execution plan to speed up the execution of the query.

### 2.2.1 Indexing in MMDBs

*Figure 1* illustrates the memory hierarchy layers when accessing a regular disk-based database. The widths of the triangles indicate the storage capacity of the layers, while the thicknesses of the arrows indicate the data volume transferred between the layers.

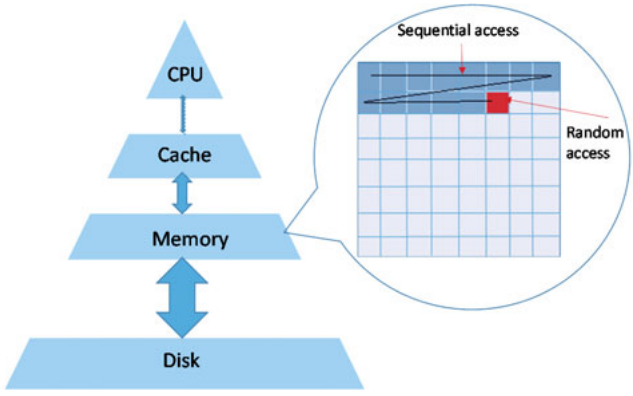


*Figure 1* Memory hierarchy

The main optimization objective of a disk-based DBMS is to reduce the number of I/O accesses, which is the primary performance bottleneck. In an MMDB, all data is assumed to fit in main-memory so therefore the goals of an MMDB are to reduce memory consumption, to optimize memory cache usage, and to minimize the number of CPU cycles for maintaining data in memory [29].

In an MMDB, before data can be processed by the CPU it needs to be transferred into the memory cache as a *data block*; this is called a *cache miss*. If the same data block is referenced a second time, the access time becomes substantially faster since no transfer is needed; this is called a *cache hit*. When a cache miss happens, the computer must wait for other CPU cycle(s) before transferring another data block from memory to the cache. Therefore, sequentially reading data blocks is much faster than randomly accessing main-memory. If there is much data to access, there will be many costly cache misses so indexing will improve performance substantially for large main-memory databases.

A cache miss is analogous to a buffer pool miss in a disk-based database. The difference is that the block size in a disk-based database is substantially larger than the memory cache in an MMDB and the performance difference between a cache hit and a cache miss is substantially higher for disk-based databases. Therefore, the improvement by indexing is higher for disk-based databases.

Even though column-store systems [80] [64] [49] [54] [2] [73] can achieve high cache utilization and CPU efficiency when scanning columns, indexes on top of column-stores improve the performance with orders of magnitude for queries that do not need to scan entire columns [81][14].

An index entry in an MMDB is a data structure containing pairs *(index key, list of handles),* where the handles refer the records having the same index key. A handle can be a physical memory address, or it can be an indirect reference to physical memory in order to allow flexible memory management

[72]. Different index data structures organize the index entries in different ways, e.g. as B-trees [67], hashing [63][93], R-trees [1], TV-trees [42], etc.

The next section addresses the necessity of extending DBMSs with new domain index structures.

### 2.2.2 Extensible indexing

*Figure 2* shows an application matrix proposed by Stonebraker in 1990s [55], which categories database applications into four quadrants. It motivates the need for DBMSs to support queries over complex data in the upper-right quadrant, which is required by many of today's applications such as multi-media, time series, and gene sequences. A key approach to support such new complex data types is the ability to include in a DBMS new data structures to represent domain data.

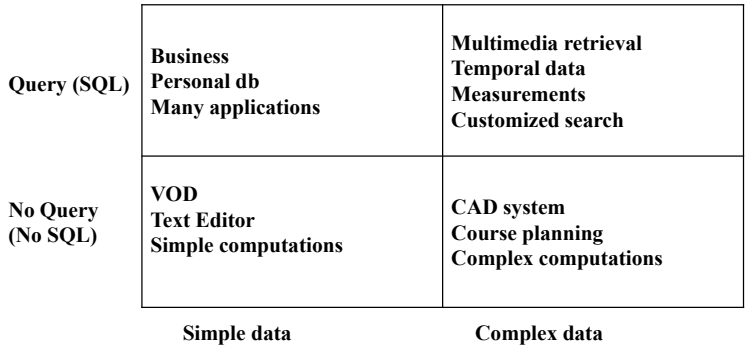


*Figure 2* Classifications of database applications

Adding a domain data type to a DBMS also requires supporting new domain operators in queries. For example, if an image data type is introduced, there is also need for query functions that compute image similarities. Importantly, inclusion of new data types requires new kinds of indexes to access the data efficiently.

*Figure 3* summaries most index structures invented during 1960-1996 [90]. However, very few of them were actually implemented in a DBMS [32].

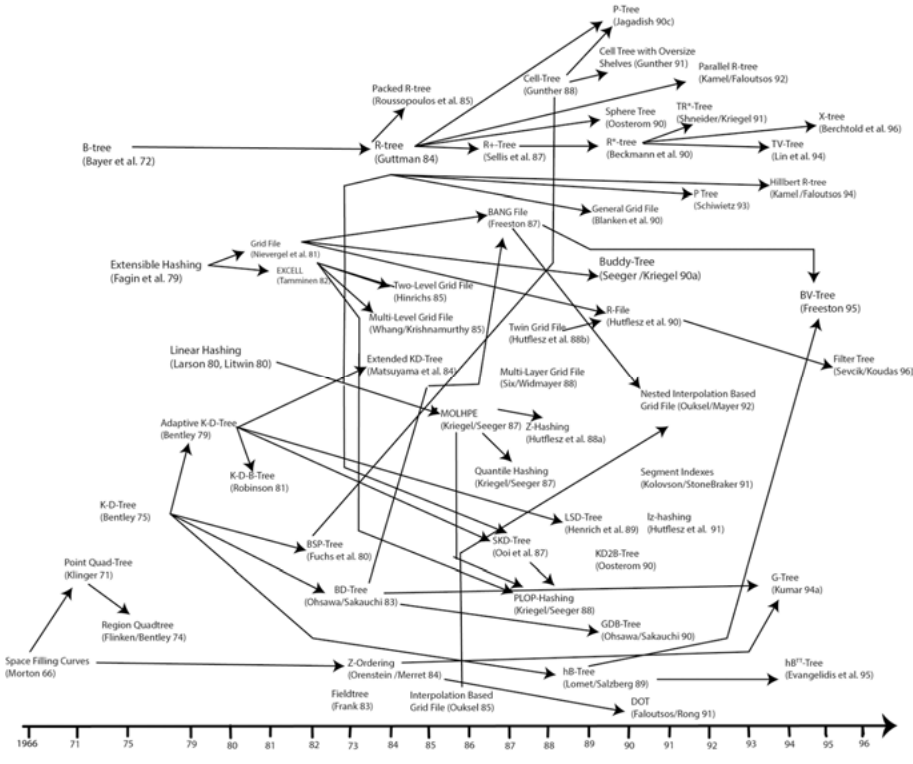


*Figure 3* Summary of indexes invented during 1966 - 1996

*Figure 4* shows what indexes are available today in well-known DBMSs. For example, both Oracle and MySQL employ R-trees up to 4D for geometry objects. In addition, Oracle Spatial engine supports also Linear Quad-trees [71], which uses a space-filling-curve technique [94] to decompose spatial data into linear-order data suitable for B-trees. Similarly, Microsoft SQL Server Spatial [94] and DB2 Spatial Extender [35] also use a space-filling curve to index spatial data. Alternatively, some DBMSs support function indexes [40][13], which are indexes on the result of a function. The DBMSs compute the result of the function for every update and materialize it in a B-tree or a hash table. Function indexes thus require computation for every single database update or insertion. That makes database updates more expensive and it is inapplicable for ad-hoc queries.

| Index | MySQL 5.6 | PostgreSQL 9.5.1 | MS SQL 2014 | Oracle 12c Release 1 |
|---|---|---|---|---|
| B-tree | Yes | Yes | Yes | Yes |
| Hash | Yes* | No | Yes* | No |
| Bit-map | No | Yes | No | Yes |
| Spatial index | Quadratic R-tree | R-tree (GiST) | No | Quadratic R-tree<br>Quad-tree |
| Function-based-index | No | Yes | Yes | Yes |

Notes
- Versions
  - Oracle 12c Release 1
  - SQL Server 2014
  - MySQL 5.6
- Hash index is only available for in-memory tables.

*Figure 4* Summary of indexes in some DBMSs

Another approach to improve performance of querying high-dimensional data is to use dimensionality reduction techniques such as DFT (Discrete Fourier Transform) [70], FFT (Fast Fourier Transform), or DWT (Discrete Wavelet Transform) that map from a high-dimensional space to lower dimensions. Then DBMSs can apply some low-dimensional index structures (R-trees or B-trees) to index the data. However, such reductions are lossy and thus they are applicable only on applications where loss of information is acceptable, e.g., financial databases [8] and images [7] [21].

MMDBs have even fewer index structures implemented than disk-based databases. For example, the following systems have only hash-based indexing: Memcached [4], Redis [74], RamCloud [38], Yahoo! S4 [43], and Piccolo [69]. SAP HANA has both CSB+ trees [77] [89] and hash indexes. H-store [68] and its commercial version VoltDB [48] have B-trees and hashing. For in-memory tables, MySQL 5.6 and MS SQL 2014 support hashing only.

The question is why most DBMSs implemented so few indexes, even though there is a demand for new kinds of indexing and there are many domain indexes. The answer is because it is very challenging to include new indexes into a DBMS kernel.

To simplify adding new indexes to DBMSs, several extensible indexing frameworks have been proposed. Generalized Search Trees [36] is a generalized template index structure, which provides a single code base for commonly invariant properties of B-tree-like search trees while leaving other characteristics to be specified by the user. GiST was realized in some prototype systems, e.g., Predator [66] and PostgreSQL [53]. Informix Dynamic Server with Universal Data Option (IDS/UDO) [51] simplified GiST's design while SP-GiST [91] extended GiST to include space-partitioning trees. A problem with GiST based approaches is that they require implementing the indexes completely following the coding conventions of the frameworks. This is still a challenging task because the index may have third-party ownership, or perhaps only binary code is available, or it is very complex to re-implement the code. It would be better if one could re-use an existing implementation of an

index without any code changes. This Thesis presents an extensible indexing framework to include new main-memory indexes without changing their implementations.

In order to utilize fully an index implementation it is very important that the query optimizer is aware of the presence of the index, its supported access methods, and how to process queries so that it is utilized in execution plans. The next section discusses this.

## 2.3 Query Processing

Query processing in general is first overviewed and it is then followed by a discussion of extensible query processing.

### 2.3.1 Overview

*Figure 5* illustrates the steps of a database query processor.

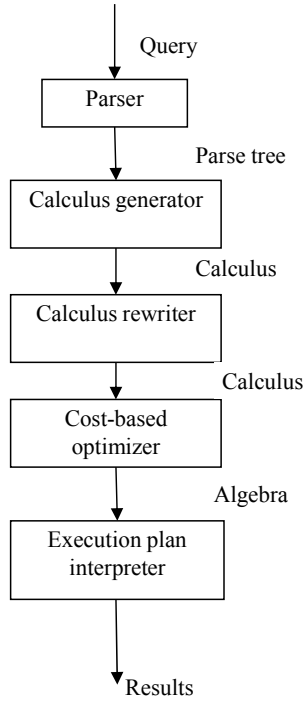


*Figure 5* Query processing in a DBMS

The query processing consists of the five following steps:

1. A *parser* constructs a *parse tree* for the input query doing type checking and some semantic checks for the validity of the objects being referred in the query.
2. A *calculus generator* converts the parse tree into a predicate *calculus* representation, e.g. relational tuple calculus [26] [9] where variables are bound to tuples (rows) in tables, or alternatively domain calculus such as Datalog [26] where variables are bound to atomic values. Mexima uses the domain calculus ObjectLog [92], which is a dialect of Datalog allowing user-defined objects, types, overloading, and *foreign functions* that allow accessing external algorithms and data structures.
3. A *calculus rewriter* transforms the calculus expression into an equivalent expression to improve performance and enable further optimization. One very important rewrite is to expand views to expose indexed columns hidden inside views. Mexima enables calculus rewrite rules for transparent utilization of new indexes by user-provided index specific rewrite rules and algebraic inequality transformations. This contrasts with other extensible indexing frameworks that recommend manually reformulating queries involving complex expressions [61] [11].
4. A *cost-based optimizer* [28] applies some optimization algorithm on the transformed calculus expression to produces an *execution plan,* which is a program in *physical algebra* accessing the database. The optimizer estimates the cost of executing a plan according to some *cost model* based on knowledge about database statistics, internal data representations, and search algorithms used in the plan. In Mexima, for given arguments, a function accessing a plugged-in index implementation can be associated with a *cost* and a *fanout*. The cost is an estimate of how expensive it is to retrieve the accessed tuples and the fanout estimates the expected number of emitted tuples in a results stream. If there is no specified cost, Mexima assumes a default cost and fanout based on the signature of the function, available index definitions, and some other heuristics.
5. Finally, the *execution plan interpreter* executes the plan and iteratively emits the result. In Mexima, special search functions supported by a plugged-in index are defined as foreign functions called from the execution plan.

### 2.3.2 Extensible query processing

In order to utilize a new index in queries, Oracle ODCIIndex framework [39] allows associating an index operator *op* with an index access path. Only conjunctive predicates where terms have the following forms are supported:

*op(...) relop <value expression>,* where *relop* is one of the relational operators: ≤, ≥, <,or >.

*op(...) LIKE <value expression>*

Oracle provides guidance [61] [11] on how to reformulate a query to utilize indexes when it is not exactly matching the above forms. In contrast, Mexima automatically transforms a wide range of query forms containing numerical expressions into queries that use index specific *special search functions (SSFs)* to utilize index properties.

In Starburst and DB2 [31], query transformations are driven by a rule engine. Internally, they represent queries by a *Query Graph Model (QGM)* in C++ structures. A rule table stores all rewrite rules and classifies them into different query classes. Each query class has different rewrite heuristics. The rule engine selects what rules to apply to transform the queries. Similarly, Volcano [24], Cascades [25], and Exodus [44] also use rules to transform relational algebra expression into physical operators.

In contrast, Mexima does not rely on procedural code since rewrite rules for SSFs are specified as declarative meta-data stored in *index property tables*. This is possible since the SSF rewriter is designed particularly for index utilization rather than for general relational algebra transformations as [31] [24] [25] [44]. Thus, a challenge is how to specify the rewrite rules as meta-data on a high-level. In Mexima each rewrite rule specified per index type describes a mapping from some terms of a *query fragment form* into an index-supported SSF function. Unlike Oracle, Mexima supports several query fragment forms. Altogether, the calculus rewriter in Mexima takes into account the existence of certain indexes, an extensible set of algebraic rules, and user-provided index rewrite rules to transform queries.

## 2.4 Database testing

It is critical that all functionalities of an index implementation are correct, meaning that query results and the database states after updates are the same, regardless of whether the index is used or not. Mexima includes a tester for automatic testing of the correctness of the functionalities provided by a plugged-in index implementation.

Testing of DBMS functionality in general has been studied in [23][45][59]. The database generator QAGen [23] provides general purpose testing of DBMS components. It generates test databases and test queries based on symbolic execution of queries. In [59] an inverse relational algebra generates query inputs for given query results. To implement unit testing for the query optimizer, the framework in [45] generates test queries based on user-defined transformation rules specified as trees of relational algebra operators. The JOB benchmark [88] tests the impact of a cost model and compares exhaustive dynamic programming with heuristic algorithms when enumerating sub-plans. It takes user-provided meta-data or DBMS statistics to generate dataset for given input queries.

QuEval [47] is a benchmark for testing spatial index implementations separate from a DBMS. QuEval produces test data sets based on user-provided parameters and built-in data generators. New index implementations can be developed and added to QuEval following its coding conventions.

Unlike QuEval, new complex indexes in C/C++ can be plugged into Mexima without any code changes. Furthermore, the Mexima tester generates and executes correctness tests of the plugged-in indexes, while the purpose of QuEval is to analyze performance of spatial index algorithms implementations in QuEval under different configurations.

## 2.5 Amos

Mexima extends the main-memory DBMS Amos [87]. Amos provides a functional and object oriented data model in which *objects, types,* and *functions* are the essential concepts. Functions are used to define properties, relationships, and computation over objects, which are classified by a type hierarchy stored in the database. The functional query language AmosQL supports queries in terms of functions over typed objects, where a *signature* and an *implementation* define a function. The signature declares the input and result parameter types and names, whereas the implementation defines how to compute outputs from inputs and vice versa. There are different kinds of functions. For example, a table is called a *stored* function and a *derived* function is a parameterized view defined by a single query. Furthermore, *foreign* functions can be defined in some external programming language, such as C/C++, Java, Python, or Lisp. Mexima uses the object-oriented data-model of Amos to represent index meta-data.

AmosQL queries are internally represented as ObjectLog [92], which is an extension of Datalog with objects, types, overloading, and foreign functions. A calculus rewriter transforms ObjectLog expressions to improve performance. After the rewrites, a cost-based optimizer produces an execution plan sent to the execution plan interpreter.

Mexima extends the query processor of Amos with calculus rewrite rules for transparent utilization of new indexes. It utilizes foreign functions to define SSFs to enable query transformation of queries into equivalent queries calling SSFs.

Amos represents all data objects internally as *physical objects* managed by a main-memory storage manager. Physical objects allocated inside a main-memory *database image* are persistent, which means that they can be saved on disk and later restored. A physical object, *po*, is accessed through an object *handle*, *hdl*, which is the offset to *po* from the start of the database image. Mexima provides a mechanism to access specialized external index storage managers for each index type so that index entries can be stored outside the database image.

# 3 Mexima

This chapter gives an overview of Mexima's architecture with references to the papers on which this Thesis is based. Followed by the general system architecture, the second section describes more in details the query processor, while the Mexima tester is described in the last sub-section.

## 3.1 Architecture

*Figure* 6 illustrates the overall architecture of Mexima. The system can used either as an extensible main-memory database or as a front-end query processor to a back-end relational DBMS, or a combination of the two.

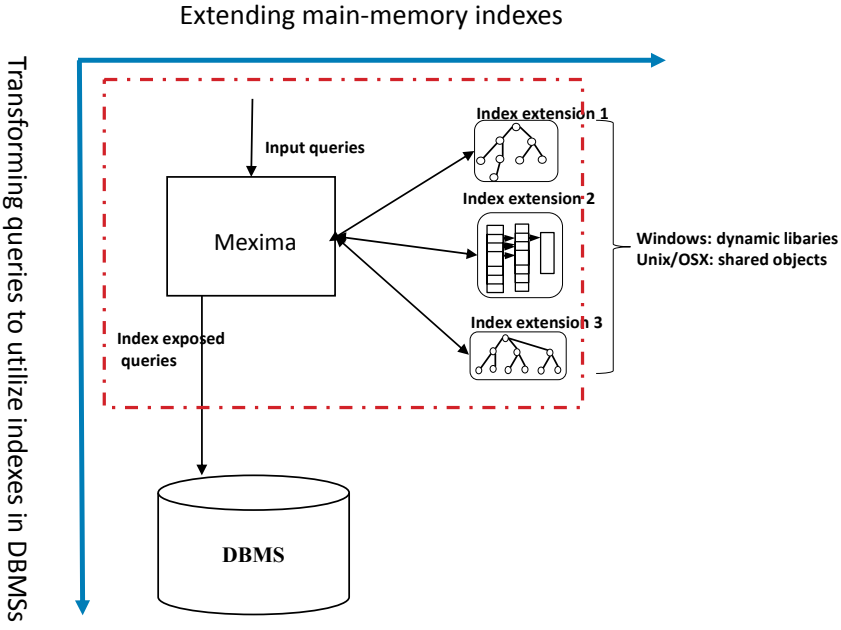


*Figure* 6 Mexima's architecture

Mexima enables plugging-in different kinds of main-memory index implementations in C or other programming languages without altering or re-engineering their original source code. An index extension developer can specify

meta-data about the plugged-in indexes. The meta-data can be index specific rewrite rules to hint the query processor on how to process queries to utilize indexes. The meta-data can also be data generators used to verify the correctness of the index. The details are in Paper I.

In addition, Mexima's query processor can transform complex queries over a back-end relational database so that indexes hidden inside complex expressions are exposed and utilized there, by applying algebraic transformation rules. In general, Mexima is a query processor with focus on index utilization. The details are mainly in Paper II and partly in Paper III.

*Figure 7* illustrates the software layers of main-memory query processing in Mexima.

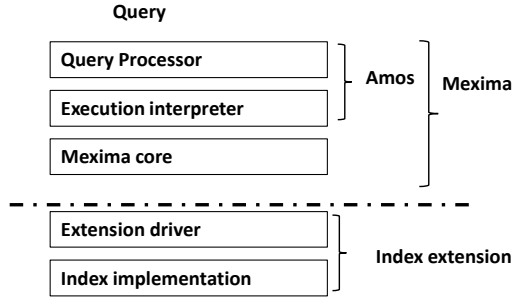


*Figure 7* Mexima software layers

Queries are compiled by the query processor and executed by the execution interpreter. The execution plans call the *Mexima core* to execute the basic index operations (BAOs) such as *create(), drop(), put(), get()*, and *map()*, as well as special search functions (SSFs) for each kind of plugged-in index. The *extension driver* is a plugged-in interface between Mexima and the unchanged *index implementation*.

*Figure 8* shows the details of the Mexima core component, with the Amos engine as the gray box. The *extension loader* loads at run-time the index extension as a dynamic library or a shared object. The extension loader registers the BAO index interfaces as C functions. The index name and its registered C functions are stored as meta-data in the *BAO table*.

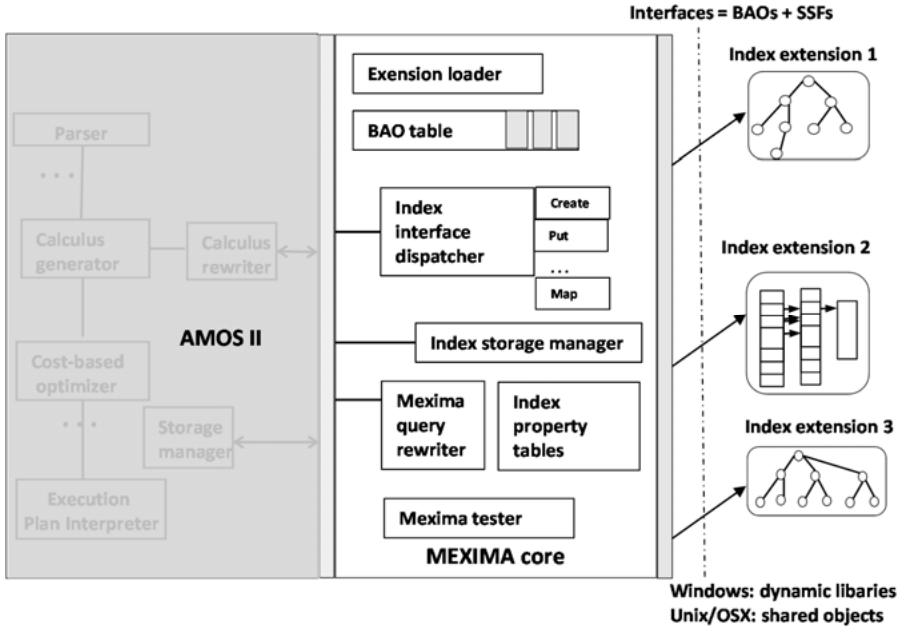


*Figure 8* Mexima core

When an instance of a plugged-in index is associated with an attribute of a main-memory table, for every data update the *index interface dispatcher* accesses the index by invoking the corresponding registered BAOs (*create(), put(), get()*,etc.) in the BAO table. Importantly, the index interface dispatcher maintains reference counters of index keys and values. This frees the extension developer from handling garbage collection issues manually.

Mexima includes an *index storage manager* for saving the index structures on disk and reloading them later when the system starts. This is important for main-memory index implementations that do not support persistency. In addition, if an index implementation has persistency implemented, Mexima provides *save()* and *restore()* hooks to call index persistency functions registered in the BAO table. The details of Mexima's core are presented in Paper I.

The next section discusses the query processor of Mexima followed by the Mexima tester.

## 3.2 Query processor

*Figure 9* illustrates Mexima's query processor. The calculus rewriter of Amos calls the *Mexima query rewriter* for transparent utilization of new indexes. The Mexima query rewriter applies rules to produce an *index exposed calculus*

expression, which is a calculus expression containing query fragments supported by an index. Without such rewrites, the query optimizer will not detect index access paths hidden inside complex expressions. The index extension developer populates the *index property tables* (*Figure 8*) containing *SSF translation rules*, which are index specific rewrite rules that describe how query fragments are translated to a new format that utilizes the index. Complex queries involving numerical expressions over indexed attributes are reformulated transparently so that the Mexima query rewriter can apply SSF translation rules to expose main-memory index implementation. The details are in Paper I and Paper II.

When the query plan is rewritten to expose a plugged-in index, the cost-based optimizer generates an execution plan that contains calls to the *Mexima core* to access the index. Paper I describes this in detail.

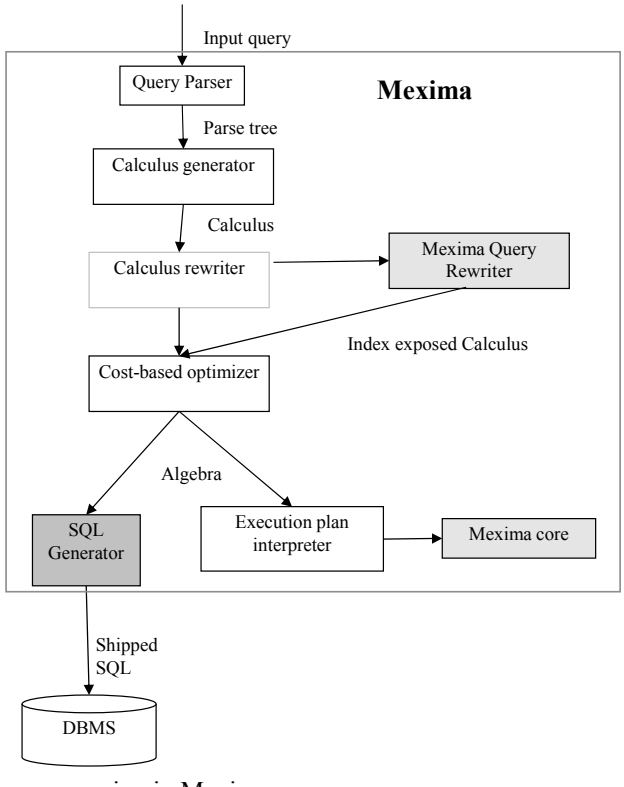


*Figure 9* Query processing in Mexima

When Mexima is used as a query processor in front of a back-end DBMS, it enables transparent query transformation to exploit the presence of indexes in the backend DBMS. In this case, the *SQL Generator* (Paper III) translates the index exposed calculus expression into an equivalent *shipped SQL query* sent to the back-end DBSM through JDBC for optimization and evaluation.

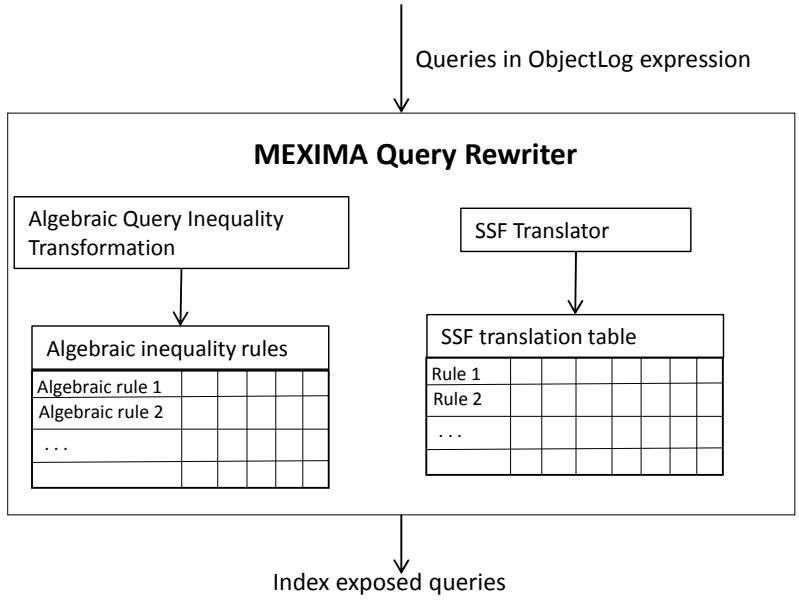


*Figure 10* Mexima Query Rewriter

*Figure 10* shows the Mexima query rewriter in details. For complex numerical expressions, the *Algebraic Inequality Query Transformations* (AQIT) component transforms a query into an equivalent one based on a set of *algebraic inequality rules*. AQIT transforms numerical complex inequality expressions so that query fragments supported by an index (B-trees in Paper II and high dimensional indexes in Paper I) are exposed.

The index extension developer populates the *SSF translation table* in which each row is an SSF translation rule for a particular index type. For plugged-in indexes, the *SSF translator* rewrites query fragments over indexed attributes into SSF calls (Paper I) based on these rules.

In summary, Mexima's query processor has the following features:

- SSF translation rules describe how query fragments are transformed to expose SSFs accessing plugged-in main-memory index implementations. Mexima currently supports six query fragment forms that can be transformed (Paper I).
- AQIT transformations enable transforming complex inequality expressions in queries to expose hidden indexes both for main-memory and back-end DBMS indexes (Paper I and Paper II).
- When data is stored in a back-end DBMS, queries having numerical expressions are translated to SQL queries sent to the back-end for execution (Paper III).

## 3.3 The Mexima tester

The *Mexima tester* illustrated by *Figure 11* automatically generates and runs test algorithms based on index meta-data provided by the index extension developer. The test algorithms require index specific random data generators. The system has some built-in random generators to generate keys as numbers and vectors of numbers respecting various distributions. User-defined data generators can easily be defined in terms of these built-in ones or as new kinds of data generators as queries. In addition, test keys stored in files can be declared as meta-data.

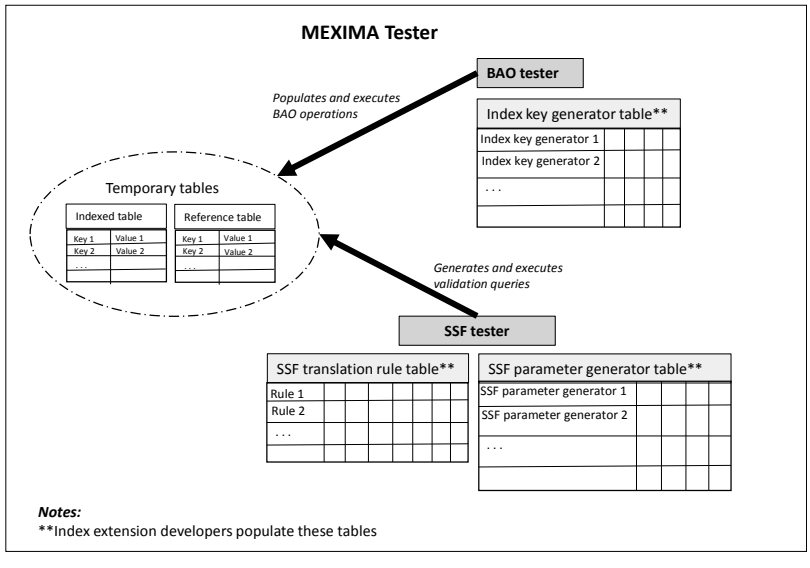


*Figure 11* Mexima tester

To test the correctness of BAOs, the *BAO tester* generates two temporary tables per tested index implementation, an *indexed table*, and a *reference table*, where the indexed table has an index of the tested kind while the reference table has a hash index. The BAO tester populates these two tables by executing *index key generator* queries stored in an index meta-data table producing random index keys. The idea is that having the index or not should not change query results or table contents. In particular, randomly loading, accessing keys, mapping over, and deleting keys should produce the same results with or without the index.

For testing correctness of SSFs, the *SSF tester* generates and executes *validation queries*, which test the following:

- When an SSF is used,a query result has to be the same as with non-indexed naive scans.
- The SSF rewrite rules are correct.

The validation queries are generated based on SSF translation rules and *SSF parameter generators* specified as index meta-data. The Mexima tester validates that executing the same validation queries on the indexed and reference tables should return the same result when SSF translation rules are enabled or disabled.

The details about the Mexima tester are presented in Paper I.

# 4 Conclusions and Future Work

Table 1 positions this Thesis in comparison to existing state-of-the-art extensible indexing framework [36] [39] [51] [53] [91] regarding the following aspects of extensible indexing:

- Code reuse: None of them provides solutions to reuse existing index implementations without any code changes.
- Index utilization query rewrites: Oracle [39] provides limited support for rewriting queries to utilize new indexes without changing the DBMS core optimizer, while Mexima transparently transforms a large class of queries involving complex expressions to utilize plugged-in index implementations or indexes in a back-end DBMS.
- Index validation: Unlike Mexima, none of them has automated validation of plugged-in indexes.

Table 1. *Summary of extensible indexing framework*

| | | Requirements | | |
|---|---|---|---|---|
| | | Code reuse | Index utilization query rewrites | Index validation |
| Extensible indexing framework | GiST<br>in PostgreSQL 9.3.5 [65] | No | Not in framework | No |
| | SP-GisT<br>version 0.0.1 [79] | No | Not in framework | No |
| | Oracle [39] | No | Limited | No |
| | DB2<br>in DB2 Universal Database 7.1, [34][27] | No | Not in framework | No |
| | This Thesis | Yes | Yes | Yes |

In summary, the Mexima framework allows transparent plugging-in of main-memory index implementations in a main-memory DBMS without code changes. The extension developer only writes a simple Mexima driver for the universal index operations (BAOs) and some index specific search functions (SSFs) that call the unchanged index implementation. Unmodified index implementations allow to easily utilizing highly optimized and complex index implementations such as Judy-tries [6]. For future work, more kinds of indexes should be plugged into Mexima than those evaluated so far B-trees [52], Linear-Hashing [52], Judy-Tries [6], X-trees [52], and R*-trees [20] (Paper I). This may add more requirements to the system.

To utilize plugged-in index implementations transparently in queries, the Mexima query processor uses SSF translation rules, and algebraic rewrite rules to rewrite queries. Future work should investigate how to extend the rewrite capabilities to support more algebraic rules and query fragment forms.

To validate the correctness of a new index, Mexima calls user-specified data generators, SSF translation rules, and query fragment forms that automate the correctness tests for a plugged-in index. As a future work, it should be investigated how to automate also performance tests of index operations.

Several experiments were made in Paper I: First, the penalty of calling an index implementation by plugging it into Mexima was compared with the stand-alone implementation showing that the overhead was less than one microsecond (μs) per index access. Furthermore, the experiments showed that rewrite rules provide substantial query performance improvements.

Moreover, when Mexima acts as a query processor in front of a DBMS, the experiments showed substantial query performance gains by Mexima's rewriting of queries to utilize indexes in the back-end DBMS.

For more future work, we shall investigate how extensible indexing and extensible query processing in Mexima can help to improve queries in NoSQL databases as discussed in Paper V. In addition, Mexima can be used to improve data access in distributed and parallel environments (Paper IV) to process massive stream in which each node is a Mexima node.

Altogether, Mexima is a complete and extensible platform for index integration, utilization, and evaluation.

# 5 Technical contributions

## 5.1 Paper I

T. Truong and T. Risch: Transparent inclusion, utilization, and validation of main-memory domain indexes, *27th International Conference on Scientific and Statistical Data-base Management (SSDBM)*, San Diego, United States, June 29-Juli 1, 2015.

**Summary**
In this paper, we presented the *Mexima (Main-memory External Index Manager)* system, which is an MMDB where new main-memory index structures can be plugged-in without modifying the index implementations or Mexima. To utilize plugged-in indexes in queries, the system transparently transforms query fragments into index operations based on user-provided *index property tables* containing index meta-data. The Mexima system includes a rule driven algebraic query transformation mechanism on complex numerical query expressions to expose potential utilization of a new index. To validate the correctness of an index implementation, Mexima generates and executes test queries based on general knowledge about indexing, the index meta-data, and the user-provided data generating queries. Several experiments were conducted to show that the index exposing rewrite mechanisms substantially improves performance and that the performance penalty of using an index plugged into Mexima is low compared to using the corresponding stand-alone C/C++ implementation. Finally, it is shown that the development effort of plugging in new indexes to Mexima is very small in comparison to other frameworks.

This paper partly answers Research Question one, two, and three.

**Contributions**
In 2011, Mexima allowed inclusion of new index structures and was used for research and education. Later in 2013, index utilization by transparently query rewrites was added whereas the index validation mechanism was designed and implemented during the autumn of 2014.

I am the primary author of the paper. The other authors contributed to discussion and paper writing.

## 5.2 Paper II

T. Truong, T. Risch: Scalable Numerical Queries by Algebraic Inequality Transformations, *19th International Conference on Database Systems for Advanced Applications (DASFAA)*, Bali, Indonesia, April 21-24, 2014.

**Summary**
This paper is based on a real industrial application scenario where data streams derived from sensor readings are bulk-loaded into a relational database system [78]. The application was prototyped as the *Stream Log Analysis System (SLAS),* which enables historical analyses of logged data streams by SQL queries. These historical queries often contains complex numerical query inequalities e.g. to find suspected deviations from normal behavior of measured sensor values during a time-period. However, such queries are often slow to execute, because the query optimizer is unable to utilize ordered indexes on some attributes hidden inside complex numerical inequalities. In order to speed up the queries, they should be reformulated so that the indexes become exposed. Therefore, we introduced the query transformation algorithm *AQIT (Algebraic Query Inequality Transformation)* that automatically transforms SQL queries involving a class of algebraic inequalities into more scalable SQL queries utilizing ordered indexes.

AQIT was originally implemented as part of query rewriter in Mexima. AQIT is used both when plugging in new main-memory indexes and when transforming queries having complex numerical expressions to be sent to a back-end DBMS. The experimental results show that the queries execute substantially faster by a commercial DBMS when AQIT has been applied to preprocess them.

This paper partly answers Research Questions two and four.

**Contributions**
In 2011, I prototyped the first algorithm of AQIT that was part of the query rewriter in Mexima. Later, I wanted to investigate Mexima, in particular AQIT, in a real industrial application described in the paper. Thus, in 2012, I developed the SLAS system where Mexima acts as a query processor (also known as AQIT query processor) with a relational database back-end. The paper was written in the autumn of 2012, later accepted in the beginning of 2013.

I am the primary author of the paper. The other authors contributed to discussion and paper writing.

## 5.3 Paper III

M.Zhu, S.Stefanova, T.Truong, and T.Risch: Scalable Nu-merical SPARQL Queries over Relational Databases, *4th international workshop on linked web data management (LWDM 2014)*, Athens, Greece, March 28, 2014.

**Summary**

In this paper, we investigated the problem of detecting past machine anomalies by querying historical sensor readings stored in a relational database. In this scenario, the main-memory database Mexima acts as query processor in front of the relational database. It takes anomaly detection queries containing numerical expressions, or inequality conditions, or string matching and produces equivalent SQL queries sent to the back-end database.

To enable scalable execution of such queries the numerical expressions should be translated into SQL rather than being post-processed in Mexima outside of the relational database. This is to avoid post-processing large data volumes, which must be transported back from the relational database server to Mexima. Furthermore, if the numerical expressions are post-processed, the indexes on the back-end database have no impact.

The paper presents the *NUMTranslator* algorithm, which translates numerical and other domain calculus operators into corresponding SQL expressions. The experiments showed that NUMTranslator substantially improves the query performance when the numerical expressions are highly selective.

This paper partly answers research question four.

**Contributions**

The NUMTranslator algorithm was first developed to enable scalable query execution of complex Mexima queries sent to a back-end relational database. Later, the NUMTranslator evolved to a part of a bigger system to harvest log databases [50]. The paper was written in autumn 2012, and then accepted in beginning of 2013.

I am one of the co-authors of the paper. In particular, I programmed the first limited version of the NUMTranslator, which later was fully developed as part of the FLOQ system [50]. I helped in paper writing and in data preparation for the experiments.

## 5.4 Paper IV – an application

*Paper IV is an example application of Mexima.*

S.Badiozamany, L.Melander, T.Truong, C.Xu, and T.Risch: Grand Challenge: Implementation by Frequently Emitting Parallel Windows and User-Defined Aggregate Functions, *Proc. The 7th ACM International Conference*

*on Distributed Event-Based Systems*, DEBS 2013, Arlington, Texas, USA, June 29 - July 3, 2013.

**Summary**
The paper describes an approach to monitor a soccer game that requires processing large volumes of data in real-time and delivers continuously physical summaries of the game as it is playing. The approach is based on an extensible DSMS in which high-volume data streams can be split and reduced into lower volume parallel streams by user-provided queries. Thus, expensive queries can be run in a parallel and distributed environment, in which each node is a main-memory Mexima database.

We experimented with plugging-in different indexes for indexing stream elements in a window. The application tested Mexima's performance and showed that Mexima can be used in a parallel and distributed environment.

**Contributions**
I am a co-author of the paper. I helped in prototyping the system and strategies.

## 5.5 Paper V – future development

*Paper V will put requirements to the future work.*

K.Mahmood, T.Truong, and T.Risch: NoSQL Approach to Large Scale Analysis of Persisted Streams, *30th British International Conference on Databases, Edinburgh (BICOD)*, Scotland, July 6-8, 2015.

**Summary**
In this paper, we first addressed some challenges in large scale persisting and analysis of numerical streaming logs. In order to investigate further these challenges, we propose to develop a benchmark that compares NoSQL stores with relational databases in storing and analyzing numerical logs. The benchmark is designed to serve as a base system for investigating query processing and indexing of large-scale numerical logs, in particular, how to reuse advanced indexing and query processing techniques in a scenario in which a main-memory is front-end while NoSQL stores data in the backend.

**Contributions**
I am a co-author of the paper.

# Summary in Swedish

Primärminnesdatabaser (PDBSer) [12] [30] [46] används för ett växande antal applikationer som kräver snabb dataåtkomst, lagring och manipulation, exempelvis applikationer för finansiella analyser, realtidsoperativsystem, sensorsystem i industriella maskiner och mer allmänt i många tekniska och vetenskapliga tillämpningar. Framväxten av denna typ av databastillämpningar ställer nya krav på databashanteringssystemen för att stödja nya sorters data, såsom färghistogram, texturer, bildmönster, gensekvenser, sensorsekvenser, eller tidsserier. För att effektivt komma åt och manipulera sådana domändata måste ett databashanteringssystem inkludera nya typer av domänorienterade indexstrukturer.

Trots att en hel del domänorienterade indexstrukturer utvecklats har mycket få av dem använts i praktiken, de flesta system [29] [32] använder bara B-träd och hash-index. Anledningen är att det är mycket svårt att utvidga ett databashanteringssystem med nya indexstrukturer vilket normalt kräver omfattande ändringar i dess kärna. Den manuella insatsen för att göra en sådan integration är mycket kostsam och tidskrävande eftersom det nya indexet måste samverka med de flesta andra delkomponenter i kärnan. Därför skulle det vara önskvärt att man kunde göra databashanterarens indexering utbyggbar så att man kan implementera och lägga till nya indexstrukturer utan att ändra i kärnan.

Befintliga ramverk för utbyggbar indexering [36] [51] [53] [91] har stöd för att lägga till nya index, men de kräver dock omkodning av index-algoritmerna där man strikt följer varje enskilt ramverks kodningskonventioner och programmeringsgränssnitt. Detta kan vara en svår uppgift då eventuellt bara binärkod är tillgänglig, det kan vara mycket komplicerat att förändra koden eller äganderätten till indexet kan vara beroende av tredje part. Det skulle således vara mycket önskvärt att kunna lägga till nya indeximplementationer utan kodändringar.

När man lägger till en ny indeximplementation i en databashanterare är det vidare viktigt att testa att alla indexfunktioner fungerar korrekt. Korrekthet innebär här att alla funktioner ska returnera exakt förväntade resultat och lämna databasen i ett konsistent tillstånd efter uppdateringar eller databasladdningar. Det är även önskvärt att kunna automatisera testförfarandet för varje tillagd indeximplementering.

Att lägga till ett nytt index i en databashanterare kräver vidare att dess frågehanterare har kunskap om egenskaperna för det nya indexet, såsom dess

olika sätt att söka data och hur databasfrågor kan transformeras för att frågeprocessorn skall kunna använda indexet effektivt. Komplexa uttryck i frågor, t.ex. för exempelvis avancerad analys, hindrar ofta frågeprocessorn från att utnyttja indexet. Det är alltså önskvärt med utbyggbar transformering av databasfrågor så att indexspecifika omskrivningsregler kan kopplas in för att förbättra frågeprestanda. Frågeprocessorn behöver dessutom hantera omskrivningsregler för att förbättra prestanda för avancerade frågor som skickas för exekvering till en extern vanlig databashanterare för exekvering.

Denna avhandling behandlar ovanstående utmaningar av utbyggbar indexering, index testning och indexspecifika frågetransformationer i ett PDBS.

Följande frågeställningar undersöks:

1. Den övergripande forskningsfrågan är: Hur kan ett utbyggbart indexeringssystem utformas för att möjliggöra transparent inkludering av olika indeximplementationer utan kodändringar i vare sig databassystemet kärna eller i indeximplementationen
2. Hur kan frågeprocessorn förses med kunskap om ett nytt inkopplat index för att transparent kunna använda indexet i databasfrågor? I synnerhet:
   a. Hur kan frågeprocessorn göras utbyggbar med nya indexspecifika omskrivningsregler så att ett nytt inkopplat index transparent kan tillämpas i frågor?
3. Hur ska korrektheten av ett inkopplat index automatiskt valideras? I synnerhet, vilka är funktionerna att testa och hur kan testerna automatiseras?
4. När data lagrats i en extern databas, hur kan frågeprocessorn omvandla komplexa frågor så att indexen i den externa databasen kan utnyttjas och tillämpas där?

För att besvara dessa frågor har vi utvecklat ett utbyggbart PDBS som kallas Mexima (Main-memory Extern Index Manager).

För att besvara frågeställning 1 möjliggör Mexima inkoppling av olika typer av primärminnesindex utan att ändra deras ursprungliga källkoder (Paper I). En utvecklare tar en existerande implementation av ett index, skriver gränssnittkod och kompilerar hela modulen som ett dynamiskt bibliotek. Denna inkludering kräver lite utvecklingsarbete och ingen kunskap om databashanteringssystemets kärna.

För att besvara frågeställning 2 kan Mexima anropa relevanta indexoperationer för en given databasfråga (Paper I). De indexoperationer som finns för alla typer av index kallas grundläggande accessoperationer (BAOs). De är metoder för att skapa, hämta, uppdatera och traversera indexstrukturens element. Dessutom finns det ofta index-specifika sökfunktioner (SSFs) som utnyttjar speciella egenskaper hos en indexstruktur för effektiv sökning, exempelvis intervallsökning för B-träd [62] och områdessökning för R-träd [1] och and X-träd [5]. För att frågeprocessorn skall kunna tillämpa index-specifika sökfunktioner innehåller Mexima ett system för omskrivning (transformation) av

databasfrågor för att identifiera mönster i frågorna där index-specifika sökfunktioner kan användas.

När databasfrågor innehåller komplexa uttryck kan de hindra frågeprocessorn från att utnyttja förekomsten av index. Detta orsakar dyra genomsökningar av hela tabeller snarare än direkta indexanrop. AQIT omvandlar komplexa numeriska frågor till motsvarande frågor som är mer effektiva genom att exponera dolda index (Paper II).

För att besvara frågeställning 3 genererar Mexima för varje indextyp ett antal testfrågor som automatiskt testar korrektheten av indexets implementering. Testmodulen drivs av datageneratorer och beskrivningar av indexets egenskaper (Paper I).

För att besvara forskningsfråga 4 tillåter Mexima att komplexa frågor till en relationsdatabas först transformeras så att dolda index exponeras (Paper II). Samma regler som används för att transformera databasfrågor mot ett primärminnesindex används också för att exponera index i externa databaser. En skalbar mekanism för att hantera omskrivna numeriska frågor som skickas till en extern relationsdatabas kräver vidare generering av SQL-frågor för att representera numeriska uttryck (Paper III).

# Bibliography


[1] A. Guttman: R-trees: A dynamic index structure for spatial searching, *Proc. SIGMOD Conf.*, pp 47–57, 1984.

[2] Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandier, Lyric Doshi, and Chuck Bear. The vertica analytic database: C-store 7 years later. In *Proceedings of the Very Large Data Bases Endowment* (PVLDB), 5(12):1790–1801, 2012.

[3] Aravind Menon. 2012. Big data @ facebook. *In Proceedings of the 2012 workshop on Management of big data systems (MBDS '12)*. ACM, New York, NY, USA, 31-32. DOI=http://dx.doi.org/10.1145/2378356.2378364

[4] B. Fitzpatrick and A. Vorobey. Memcached: a distributed memory object caching system. 2003. [Online]. Available: http://memcached.org/.

[5] B. Stefan, A.K. Daniel, H-P. Kriegel: The X-tree : An Index Structure for High-Dimensional Data, *Proc. 22nd of Very Large Databases Conference*., Bombay, India, pp. 28-39, 1996.

[6] Baskins, D. Judy home page. http://judy.sourceforge.net/ (Accessed at 2003).

[7] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. 1994. Efficient and effective querying by image content. *J. Intell. Inf. Syst. 3*, 3-4 (July 1994), 231-262. DOI=http://dx.doi.org/10.1007/BF00962238

[8] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. 1994. Fast subsequence matching in time-series databases. *Proc. of the 1994 ACM SIGMOD international conference on Management of data (SIGMOD '94),* Richard Thomas Snodgrass and Marianne Winslett (Eds.). ACM, New York, NY, USA, 419-429. DOI=http://dx.doi.org/10.1145/191839.191925

[9] Codd, E. F.: Relational completeness of database sublanguages. *J. Communications of the ACM. 13(6)*, pp 377-387, 1970

[10] Cristian Diaconu, Craig Freedman, Erik Ismert, Per-Ake Larson, Pravin Mittal, Ryan Stonecipher, Nitin Verma, and Mike Zwilling. 2013. Hekaton: SQL server's memory-optimized OLTP engine. *In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. ACM, New York, NY, USA, 1243-1254. DOI=http://dx.doi.org/10.1145/2463676.2463710

[11] D. Benoit, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin: Automatic SQL tuning in Oracle 10g, *Proc. of Thirtieth international conference on Very large data bases-Volume 30*, pp 1098-1109, 2004.

[12] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. Stonebraker, and D. A. Wood. 1984. Implementation techniques for main memory database systems. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of data*.

[13] D.J-H. Hwang. Function-Based Indexing for Object-Oriented Databases, *PhD Thesis, Massachusetts Institute of Technology*, 1994, 26-32.
[14] Daniel Abadi, Peter Boncz, and Stavros Harizopoulos. 2013. The Design and Implementation of Modern Column-Oriented Database Systems. *Now Publishers Inc*., Hanover, MA, USA
[15] David B. Lomet, Sudipta Sengupta, and Justin J. Levandoski. 2013. The Bw-Tree: A B-tree for new hardware platforms. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013*) (ICDE '13). IEEE Computer Society, Washington, DC, USA, 302-313. DOI=http://dx.doi.org/10.1109/ICDE.2013.6544834
[16] Donald E. Knuth. 1997. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.
[17] Douglas Comer. 1979. Ubiquitous B-Tree. *ACM Comput. Surv*. 11, 2 (June 1979), 121-137. DOI=http://dx.doi.org/10.1145/356770.356776.
[18] E. F. Codd. 1970. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (June 1970), 377-387. DOI=http://dx.doi.org/10.1145/362384.362685
[19] E. G. Coffman, Jr. and J. Eve. 1970. File structures using hashing functions. *Commun. ACM 13*, 7 (July 1970), 427-432 DOI=http://dx.doi.org/10.1145/362686.362693
[20] Efficient and Lightweight In-Memory Implementation of R*-Tree: *http://www.ics.uci.edu/~salsubai/rstartree.html*.
[21] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. *Proc. of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '01)*. ACM, New York, NY, USA, 245-250. DOI=http://dx.doi.org/10.1145/502512.502546
[22] F. Färber, N. May, W. Lehner, P. Große, I. Mller, H. Rauhe, and J. Dees, The sap hana database – an architecture overview. *IEEE Data Eng. Bull., 2012*.
[23] F. Haftmann, D. Kossmann and E. Lo: A framework for efficient regression tests on database applications, *The Very large data bases Journal, 16(1)*, pp. 145-164, 2007
[24] G. Goetz, W. J. McKenna: The Volcano optimizer generator: Extensibility and efficient search, *Proc. of IEEE Conference on Data Engineering*. pp. 209-218, 1993.
[25] G. Goetz: The cascades framework for query optimization, *IEEE Data Engineering Bulletin. 18(3)*, pp 19-29, 1995.
[26] Garcia-Molina, H., Ullman, J.D., Widom, J.: Database Systems: The Complete Book. Prentice Hall, Upper Saddle River, NJ, USA (2009).
[27] George Baklarz and Bill Wong. 2000. *The Db2 Universal Database V7.1 for Unix, Linux, Windows, and Os/2 with CD-ROM (4th ed.)*. Prentice Hall PTR, Upper Saddle River, NJ, USA.
[28] Graefe, G.: Query evaluation techniques for large databases. *J. ACM Computing Surveys (CSUR)*. 25(2):73-169, 1993.
[29] H Zhang, G Chen, BC Ooi, KL Tan, M Zhang: "In-memory big data management and processing: A survey", *IEEE*, 2015
[30] H. Garcia-Molina and K. Salem. 1992. Main Memory Database Systems: An Overview. *IEEE Trans. on Knowl. and Data Eng. 4*, 6 (December 1992), 509-516. DOI=http://dx.doi.org/10.1109/69.180602

[31] H. Pirahesh, T.C. Leung, & W. Hasan: A rule engine for query transformation in Starburst and IBM DB2 C/S DBMS. *Proc. of IEEE Conference on Data Engineering*, pp. 391-400, 1997.
[32] Hans-Peter Kriegel and Martin Pfeifle and Marco Pötke and Thomas Seidl, The Paradigm of Relational Indexing: A Survey, The Paradigm of Relational Indexing: A Survey, 2003, *In BTW, volume 26 of LNI. GI*, pp 285—304, Springer.
[33] Henrich A., Six H.-W., Widmayer P.: 'The LSD-Tree: Spatial Access to Multidimensional Point and Non-Point Objects', *Proc. 15th Conf. on Very Large Data Bases Conference*, Amsterdam, The Netherlands, pp. 45-53, 1989.
[34] http://www.ibm.com/developerworks/data/library/techarticle/dm-0312stolze/, Accessed January, 2016.
[35] IBM DB2 Spatial Extender User's Guide and Reference, Version 7 (2001).
[36] J Hellerstein. M., J. F. Naughton, and A. Pfeffer: Generalized search trees for database systems*, Proc. of The Very large data bases Conference.*, pp 562-573, 1995.
[37] J. Chhugani, A. D. Nguyen, V. W. Lee, W. Macy, M. Hagog, Y.-K. Chen, A. Baransi, S. Kumar et al., Efficient implementation of sorting on multi-core simd cpu architecture, in *PVLDB '08*, 2008.
[38] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazi` eres, S. Mitra, A. Narayanan et al., The case for ramclouds: Scalable high-performance storage entirely in dram, OSR, 2010.
[39] J. Srinivasan, R. Murthy, S. Sundara, N. Agarwal, and S. DeFazio: Extensible indexing: a framework for integrating domain-specific indexing schemes into oracle8i. *Proc. of IEEE Conference on Data Engineering.*, pp 91–100, 2000.
[40] J.Gray, A. Szalay, and G. Fekete. Using Table Valued Functions in SQL Server 2005 to Implement a Spatial Data Library, *Technical Report, Microsoft Research Advanced Technology Division 2005*.
[41] K.Mahmood, T.Truong, and T.Risch: NoSQL Approach to Large Scale Analysis of Persisted Streams, *30th British International Conference on Databases, Edinburgh (BICOD)*, Scotland, July 6-8, 2015.
[42] King Ip Lin, H. V. Jagadish, and Christos Faloutsos. 1994. The TV-tree: an index structure for high-dimensional data. *The VLDB Journal 3*, 4 (October 1994), pp 517-542.
[43] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *ICDMW '10*, 2010.
[44] M. Carey, et al: The architecture of the EXODUS extensible DBMS, *Proc. 1986 international workshop on Object-oriented database systems, IEEE Computer Society Press,* 1986.
[45] M. Elhamali and L. Giakoumakis: Unit-testing Query Transformation Rules, *Proc. of 1st International Workshop on Testing Database Systems*, 2008
[46] M. H. Eich, "Mars: The design of a main memory database machine," in *Database Machines and Knowledge Base Machines, ser. The Kluwer International Series in Engineering and Computer Science. Springer US*, 1988.
[47] M. Schäler, A. Grebhahn, R. Schröter, S. Schulze, V. Köppen, and G. Saake. 2013: QuEval: beyond high-dimensional indexing à la carte, *Proc. of the Very large data bases Endowment, 6(14)*, pp 1654-1665, 2013.
[48] M. Stonebraker and A. Weisberg, "The voltdb main memory dbms," *IEEE Data Eng. Bull.*, 2013.
[49] M. Zukowski, S. Héman, N. Nes, and P. Boncz. Cooperative scans: dynamic bandwidth sharing in a DBMS. In *Proceedings of the International Conference on Very Large Data Bases* (VLDB), pages 723–734, 2007.

[50] M.Zhu, S.Stefanova, T.Truong, and T.Risch: Scalable Numerical SPARQL Queries over Relational Databases, *4th international workshop on linked web data management (LWDM 2014)*, Athens, Greece, March 28, 2014.
[51] Marcel Kornacker. 1999. High-Performance Extensible Indexing. *Proc. of the 25th International Conference on Very Large Data Bases (VLDB '99)*, Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 699-708.
[52] Mexima homepage, http://www.it.uu.se/research/group/udbl/mexima
[53] Michael Stonebraker and Greg Kemnitz. 1991. The POSTGRES next generation database management system. *Commun. ACM 34*, 10 (October 1991), 78-92. DOI=http://dx.doi.org/10.1145/125223.125262
[54] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel R. Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alexander Rasin, Nga Tran, and Stan B. Zdonik. C-Store: A Column-Oriented DBMS. *In Proceedings of the International Conference on Very Large Data Bases* (VLDB), pages 553–564, 2005.
[55] Michael Stonebraker, Dorothy Moore, and Paul Brown. 1998. Object-Relational Dbmss: Tracking the Next Great Wave (2nd ed.). *Morgan Kaufmann Publishers Inc*., San Francisco, CA, USA.
[56] Mikael Ronström: Design and Modelling of a Parallel Data Server for Telecom Applications, PhD Thesis, Linköping University Dissertation No 520, 1998
[57] MongoDB Inc., "Mongodb," 2009. [Online]. Available:http://www.mongodb.org/
[58] MSQL, The MEMORY storage engine. 2016. [Online]. Available: http://www.mysql.com/
[59] N. Bruno, S. Chaudhuri and D. Thomas: Generating Queries with Cardinality Constraints for DBMS Testing, *IEEE Transactions on Knowledge and Data Engineering, 18(12)*, pp 1721-1725, 2006.
[60] Navathe, Ramez Elmasri, Shamkant B. (2010). Fundamentals of database systems (6th ed.). *Upper Saddle River, N.J.: Pearson Education*. pp. 652–660.
[61] Oracle Inc: Query Optimization in Oracle Database 10g Release 2. *http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-general-query-optimization-10gr-130948.pdf*, 2005.
[62] Oracle, "Mysql cluster," 2016. [Online]. Available: http://www.mysql.com/
[63] Per-Ake Larson. 1988. Linear hashing with separators—a dynamic hashing scheme achieving one-access. *ACM Trans. Database Syst. 13*, 3 (September 1988), 366-388. DOI=http://dx.doi.org/10.1145/44498.44500
[64] Peter Boncz, Marcin Zukowski, and Niels Nes. MonetDB/X100: Hyper- pipelining query execution. In *Proceedings of the biennial Conference on Innovative Data Systems Research* (CIDR), 2005.
[65] PostgreSQL version 9.3.5 http://www.postgresql.org/ftp/source/v9.3.5/
[66] Praveen Seshadri. 1998. PREDATOR: a resource for database research. *SIGMOD Rec. 27*, 1 (March 1998), 16-20. DOI=http://dx.doi.org/10.1145/273244.273251

[67] R. Bayer and E. McCreight. 1970. Organization and maintenance of large ordered indices. *In Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control (SIGFIDET '70)*. ACM, New York, NY, USA, 107-141. DOI=http://dx.doi.org/10.1145/1734663.1734671
[68] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden et al., "H-store: A high-performance, distributed main memory transaction processing system," in *PVLDB '08*, 2008
[69] R. Power and J. Li, "Piccolo: Building fast, distributed programs with partitioned tables," in *OSDI '10*, 2010
[70] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. 1993. Efficient Similarity Search In Sequence Databases. *Proc. of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO '93),* David B. Lomet (Ed.). Springer-Verlag, London, UK, 69-84, 1993.
[71] Ravi Kanth V Kothuri, Siva Ravada, and Daniel Abugov. 2002. Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD '02)*. ACM, New York, NY, USA, 546-557. DOI=http://dx.doi.org/10.1145/564691.564755
[72] Rifat Shahriyar, Stephen Michael Blackburn, Xi Yang, and Kathryn S. McKinley. 2013. Taking off the gloves with reference counting Immix. In *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications (OOPSLA '13*). ACM, New York, NY, USA, 93-110. DOI=http://dx.doi.org/10.1145/2509136.2509527
[73] Roger MacNicol and Blaine French. Sybase IQ multiplex - designed for analytics. In *Proceedings of the International Conference on Very Large Data Bases* (VLDB), pages 1227–1230, 2004.
[74] S. Sanfilippo and P. Noordhuis, "Redis," 2009. [Online]. Available: http://redis. io
[75] S.Badiozamany, L.Melander, T.Truong, C.Xu, and T.Risch: Grand Challenge: Implementation by Frequently Emitting Parallel Windows and User-Defined Aggregate Functions, *Proc. The 7th ACM International Conference on Distributed Event-Based Systems*, DEBS 2013, Arlington, Texas, USA, June 29 - July 3, 2013.
[76] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. 2005. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst. 30*, 1 (March 2005), 122-173. DOI=http://dx.doi.org/10.1145/1061318.1061322
[77] SAP, SAP HANA, 2010. [Online]. Available: http://www.saphana.com/
[78] Smart Vortex Project - http://www.smartvortex.eu/
[79] SP-GiST*: https://www.cs.purdue.edu/spgist/*
[80] Stratos Idreos, Fabian Gro!en, Niels Nes, Stefan Manegold, Sjoerd Mullender, and Martin L Kersten. MonetDB: Two Decades of Research in Column-oriented Database Architectures. *IEEE Data Eng. Bull.*, 35(1):40–45, 2012.
[81] StratosIdreos.DatabaseCracking:Towards Auto-tuning Database Kernels. CWI, PhD Thesis, 2010.
[82] T. Lahiri, M. A. Neimat, and S. Folkman. Oracle TimesTen: An In-Memory Database for Enterprise Applications. *IEEE Data Engineering Bulletin 36(2): 6--13 (2013)*.
[83] T. Muhlbauer, W. Rödiger, R. Seilbeck, A. Reiser, A. Kemper, and T. Neumann, Instant loading for main memory databases, in *PVLDB'13*, 2013.

[84] T. Truong and T. Risch: Transparent inclusion, utilization, and validation of main memory domain indexes, *27th International Conference on Scientific and Statistical Database Management (SSDBM)*, San Diego, United States, June 29-Juli 1, 2015.
[85] T. Truong, T. Risch: Scalable Numerical Queries by Algebraic Inequality Transformations, *19th International Conference on Database Systems for Advanced Applications (DASFAA)*, Bali, Indonesia, April 21-24, 2014.
[86] T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner, Simd-scan: Ultra fast in-memory table scan using on-chip vector processing units, in *PVLDB '09*, 2009.
[87] T.Risch, V.Josifovski, and T.Katchaounov: Functional Data Integration in a Distributed Mediator System, in P.Gray, L.Kerschberg, P.King, and A.Poulovassilis (eds.): Functional Approach to Data Management - Modeling, Analyzing and Integrating Heterogeneous Data, *Springer, ISBN 3-540-00375-4*,2004.
[88] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? In *Proc. VLDB Endow. 9, 3* (November 2015), 204-215. DOI=http://dx.doi.org/10.14778/2850583.2850594
[89] Vishal Sikka, Franz Färber, Anil Goel, and Wolfgang Lehner. 2013. SAP HANA: the evolution from a modern main-memory data platform to an enterprise application platform. *Proc. VLDB Endow. 6, 11 (August 2013)*, 1184-1185. DOI=http://dx.doi.org/10.14778/2536222.2536251
[90] Volker Gaede and Oliver Günther. 1998. Multidimensional access methods. *ACM Comput. Surv. 30*, 2 (June 1998), 170-231. DOI=10.1145/280277.280279
[91] W. G. Aref and I. F. Ilyas: An extensible index for spatial databases, *Proc. of Statistical and Scientific Database Management*, pp 49–58, 2001.
[92] W. Litwin and T.Risch: Main Memory Oriented Optimization of OO Queries Using Typed Datalog with Foreign Predicates, *IEEE Transactions on Knowledge and Data Engineering, 4(6)*, 1992.
[93] Witold Litwin. 1988. Linear Hashing: a new tool for file and table addressing.. *In Readings in database systems*, Michael Stonebraker (Ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA 570-581.
[94] Yi Fang, Marc Friedman, Giri Nair, Michael Rys, and Ana-Elisa Schmid. 2008. Spatial indexing in Microsoft SQL server 2008. *Proc. of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08)*. ACM, New York, NY, USA, 1207-1216. DOI=http://dx.doi.org/10.1145/1376616.1376737.

Acta Universitatis Upsaliensis

*Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 1352

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-280374

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2016