DAC: Database Application Context Analysis applied to Enterprise Applications

Johannes Wust **Carsten Meyer** **Hasso Plattner**

Hasso Plattner Institute for Software Systems Engineering,
University of Potsdam,
PO Box 14440, Potsdam, Germany,
Email: johannes.wust@hpi.uni-potsdam.de

Abstract

In today's fast-paced business environment, we see an ongoing trend towards the need for analytics on the latest operational data. The data management layer of enterprise applications needs to adapt to this requirement and In-Memory Column Stores have been proposed as a new architecture that can handle such mixed workload scenarios. A thorough understanding of the resulting query workload is required to validate and optimize data management concepts for this new challenge. Consequently, this paper introduces Database Application Context (DAC) analysis —an holistic framework to analyze database workloads, data characteristics as well as access patterns on specific domain types. We present results for a productive enterprise resource planning system, as well as widely accepted database benchmarks for transactional and mixed workloads. In contrast to existing work, we have analyzed correlations between issued queries and the domain types of accessed attributes. Our main findings are (i) that enterprise workloads are read heavy, (ii) that specific database operators predominantly operate on attributes with a specific domain type, and (iii) that data characteristics differ depending on the data type. Furthermore, based on our analysis of trends in modern enterprise applications, we expect workloads with an increased runtime share of complex queries in the future. These findings help in designing and optimizing the data management layer of modern enterprise applications.

Keywords: Database, In-Memory Database, Workload Analysis, Enterprise Applications

1 Introduction

A traditional enterprise IT landscape largely separates systems for operational data management and reporting. The main reasons for this fact are fundamental differences in functional and performance requirements of both domains (Chaudhuri & Dayal 1997). However, companies often demand more flexible, ad-hoc reporting on the latest data, also referred to as *Operational Business Intelligence* (Gillin 2007, Golfarelli et al. 2004, Kuno et al. 2010, White 2005). To avoid an additional load of analytical capabilities on the operational database, these applications typically rely on synchronized copies of the operational data, so-called *operational data stores* (White 2005). Maintaining such a redundant real-time copy is complex and expensive.

A radically different approach has been proposed by database architectures that are designed for a mixed workload of both, transactional and analytical queries (Plattner 2009, Kemper & Neumann 2010). The main reasons for the increased performance that allows processing both type of query workloads on a single database instance are massive intra-query parallelism on many-core CPUs and a primary data store in main memory instead of disks or SSDs. It is now possible to store and process data sets of enterprise applications, such as enterprise resource planning (ERP) systems, entirely in main memory (Plattner 2009). Holding the entire data set of an application in main memory, rather than on secondary storage such as hard disks and optimizing data access towards main memory and CPU-integrated memory, offers data access performance that is in orders of magnitudes faster than traditional disk-based systems. Applied to the domain of enterprise applications, the performance gain is so dramatic, that it becomes feasible to build analytical capabilities on top of transactional systems. Additionally, the performance increase allows a rethink of the design of database schema. As an example, complex materialized aggregates that require a predefinition of available aggregations can be replaced by dynamic views that aggregate on the fly, enabling a simpler and more flexible database schema.

Optimizing data structures of the data management layer for these new applications requires a thorough knowledge of the resulting workload in such a scenario with a mix of transactional and analytical applications on a single database instance. Therefore, the objective of this research is to improve the understanding of current and anticipate future database workloads of enterprise applications to optimize the data structures of in-memory databases for these scenarios. As we are right at the beginning of this revolutionary trend, we do not have access to a large productive database installation running in such a mixed scenario. Therefore, our approach presented in this paper is to start with an analysis of a large, productive ERP system as a representative for a transactional system. Based on this baseline, we look into two areas to understand how a mixed workload will look like: (i) we analyze an existing database benchmark, specifically designed for mixed workload, and (ii) we analyze recent developments in enterprise applications and formulate expected changes to the transactional baseline workload. It is important to note, that the obvious approach of combining the queries of existing transactional and analytical applications fails due to different data schema, as explained in more detail in Section 5. As a foundation of these

workload analyses, we have defined a framework to analyze database workloads by classifying queries and data access patterns. In contrast to existing work, we have also analyzed correlations between the domain types of attributes and data access patterns across workloads.

The contributions of this paper are as follows:

- Database Application Context (DAC) Analysis —A framework used to extract and parse queries, database operations, and data characteristics, as well as execution times from different SQL workloads
- A classification and quantification of queries, database operations and data schema definitions from a productive enterprise system, as well as a benchmark for mixed enterprise workloads
- An analysis of the expected changes of the database workload due to new enterprise applications, as well as changed database schema.

This paper is structured as follows: Section 2 gives a brief overview of the basic concepts of in-memory data management for enterprise applications and discuss trends we see in modern enterprise applications. Section 3 describes our framework DAC for classifying database workloads. In Section 4, we introduce the analyzed database workloads and present the results of our analysis in Section 5. We discuss the results of the workload analysis together with the trends we see in modern enterprise applications in Section 6. Section 7 discusses related work and the last Section closes with some concluding remarks.

2 Trends in Modern Enterprise Applications

This section gives a short overview of an in-memory database that allows processing of mixed database workloads of both, transactional, as well as analytical queries. Furthermore, we describe trends that we have identified by analyzing new applications that have been developed for these new database management systems. We will discuss these trends in the context of the results of the analyses of a productive enterprise application and of benchmarks in Section 6.

2.1 In-Memory Database Management

Here, we give a brief overview of the architecture of an in-memory database as introduced as SanssouciDB in (Plattner 2011). Other architectures for in-memory databases targeted towards mixed workloads have been proposed, for example HyPer (Kemper & Neumann 2010). In this paper we briefly introduce the architecture of SanssouciDB to demonstrate the change in database technology that triggers new usage patterns. However, the workload analyses presented in this paper are largely independent on a specific database architecture and can be generally applied as a starting point for optimization of data structures for enterprise application specific data management.

In SanssouciDB, all columns are stored dictionary-compressed to utilize main memory efficiently. Dictionary compression replaces all values by a small integer representative that references the original value that is uniquely stored in a dictionary. Databases that use a column-wise data store typically favor read-mostly analytical workloads, making updates and inserts into dictionary-compressed columns a challenge. To achieve high read and write performance, a common concept in column-oriented databases is to use an additional data store besides the read-optimized main partition (Krueger et al. 2011): a write optimized differential store.

To achieve durability in case of a system failure, the in-memory database writes log information to persistent memory. This log information is used to recover the latest consistent state in case of a failure and thus guarantees durability. We have proposed an efficient logging mechanism for dictionary-compressed columns in (Wust et al. 2012). We apply multi version concurrency control (MVCC) based on transaction IDs (TID) to determine which records are visible to each transaction when multiple transactions run in parallel. TIDs issued by a transaction manager for each arriving query define the start order of transactions. See (Plattner 2011) for more details.

2.2 Mixed Database Workloads

New enterprise applications that leverage the performance of new database architectures potentially lead to a changed database workload. Our objective is to describe what changes we have to expect in order to optimize database architectures for these use cases. However, observations from analyzing individual applications are hard to generalize. During our research we have looked at existing applications that have been redesigned for in-memory databases, cases of new feature-sets on existing data and also entirely new applications that rely on analytical queries running on transactional data. In this paper, we briefly describe two major trends we have observed, namely replacing materialized aggregates with on-the-fly aggregations, and applications that mix transactional queries and analytical queries. In Section 6, we discuss the implications of these trends on future database workloads, considering our findings from the analysis of existing enterprise workloads using our analysis framework *DAC*, as presented in the following Sections.

On-the-fly Aggregates Replace Materialized Views

The concept of materializing certain views in order to speed up the processing of queries is common practice in database systems (Yang & Larson 1987, P. Larson and H. Z. Yang 1985). Today, all major analytical database systems support the definition of de-normalized, pre-calculated views (Bello et al. 1998, Halevy 2001) for frequently executed queries. In SAP ERP (SAP 2013), the concept of materialized views is realized using additional, transactional tables that are maintained by the application logic. They hold subsets or even entire table projections of huge transactional relations, filtered and aggregated on pre-defined granularity. This redundancy has been necessary to get reasonable response times for complex aggregates and lookups on huge tables. However, there are three major limitations:

- Data redundancy,
- Reduced query flexibility and
- Maintenance cost (inserts and updates of materialized tables)

Consequently, an in-memory database replaces materialized aggregates using on-the-fly queries that run on the original tables (Plattner 2009). In order to estimate the impact of such a redesign, we have analyzed the workload on the most relevant materialized tables in SAP ERP Financial, based on the execution count:

- Sum table holding customer balances on a monthly basis

- Secondary index table that allows fast access to accounting documents
- Sum table holding balances of general ledger account on a monthly basis

Each query that calculates these aggregates on the fly based on the transactional schema contains a join on at least two transactional tables, calculating aggregates, grouped by a set of attributes.

Mixed Workload Applications
Providing a platform that can run transactional and analytical queries on a common dataset opens the way for new business applications. As an example, (Wust et al. 2011) proposes an application that gives sales representatives a tool to generate cross-selling recommendations on the fly adjusted to the actual customer. Product recommendations are calculated on-the-fly, in order to handle a range of different parameters (products, regions, branches or customers). Additionally, the abundance of pre-calculated, materialized result sets leads to a less complex data schema. Prior to xSellerate, product recommendations needed long-running, inflexible batch-job operations that were persisted in dedicated, materialized tables. xSellerate shows the need and feasibility of analytic queries on transactional data. As another example, (Tinnefeld et al. 2011) presents a real- time availability-to-promise service that calculates a stock projection in real-time for each request; this application includes transactional queries for ordering products that depend on analytical style queries calculating the current stock level, and therefore need to run on a consistent data set of a single database instance. Furthermore, (Krueger, Tinnefeld, Grund, Zeier & Plattner 2010) gives a detailed overview of the characteristics of these applications. Analyzing at the queries issued by these applications, we see large joins, as well as groupings and aggregations.

3 Database Application Context (DAC) Analysis

This Section introduces our framework to analyze database workloads. We start with a presentation of the overall process and then describe the individual steps in more detail in the subsequent section.

3.1 Overview

The motivation of this research is to get a thorough understanding of the database workload issued by today's enterprise application and derive characteristics of future mixed workload scenarios. In the context of columnar databases we intend to provide rationales for optimizations of data structures, compression and operators based on existing domain type schema definitions.

Typical questions that arise are: What kind of statements and operations are issued most by certain workloads and on what attributes do they operate? Answers to these questions can help to optimize the data management layer for specific workloads. To find answers to these questions, we designed a framework called Database Application Context (DAC) analysis, shown in Figure 1.

An application's workload, data schema and data characteristics are the elementary aspects of an application that determine the performance of a database. DAC describes and quantifies those aspects with the objective of understanding the predominantly used

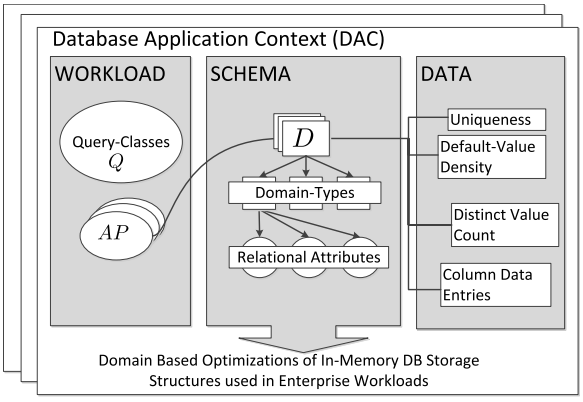


Figure 1: Database Application Context Analysis

queries, database operations, data types, and data characteristics for a given application.

First, we use the access pattern model AP, to describe database operations such as joins, groupings, selection, projections and aggregations parsed in each query statements. Based on that the query-class model Q, separates statements into distinct groups. Then all table attributes in AP are classified using a domain model D, that separates columns based on their domain information.

As a first step of the analysis, the application workload, its data schema definition and data statistics are captured from various sources. Then the workload is parsed for specific AP and the schema information is classified before it gets consolidated into a central data repository. Typically the workload is captured as an SQL trace, or as a snapshot of the SQL plan cache of a database, ideally annotated with performance data, such as execution count, runtime per query, and returned rows. Obviously, retrieving all performance data is only partially possible in a live system due to potential overhead. The data schema and information about the data can be extracted directly from the database.

The different models are applied to classify workloads into distinct query classes (Q), characteristic operations into access pattern (AP) and data segments, respectively relational attributes into domain groups (D), based on their domain-type information. Additionally, the data of each domain group in D is characterized, using the average data distribution, distinct value count and total number of the columns in each group. The extracted workload characteristics are then mapped with characteristics of the data schema and data statistics to find correlations for each workload.

3.2 Models of the DAC Analysis

This section describes the models used in DAC analysis in detail. We apply different models to analyze workload queries, access patterns as well as an application's data schema and data characteristics.

Classification of Queries
A first step towards characterizing a workload is to classify the queries of each workload.

The separation of statements into query classes is done by parsing the type of SQL statement as well as specific operations in the query. We currently consider the SQL statement types SELECT, INSERT, DELETE, and UPDATE. That allows a simple separation of all select- or read- (Q_R), delete- (Q_{DEL}),

	Description
AP_{Agg}	Aggregate operations (e.g. count(), max(), min(), sum())
AP_{Join}	Join operation on a field
AP_{Sel}	Data selection using where predicates
AP_{Grp}	Group by operation on a field
AP_{Sort}	Order by operation on a field
AP_{Proj}	Projection of a field

Table 1: Access Patterns

update- (Q_{UPD}), and insert- (Q_{INS}) queries. By analyzing the data characteristics stored in the database, Q_R-queries are further divided into Point or Single Select (Q_{Single}), Range Select (Q_{Range}), and Complex Select ($Q_{Complex}$) statements. We define Range Select (Q_{Range}) as statements that have a low selectivity and read a set of data entities. In contrast to Complex Selects ($Q_{Complex}$), they do not aggregate, join or group a set of data, but only read them. Point or Single Select (Q_{Single}) statements have a high selectivity, reading only single data sets. Finally, Complex Selects ($Q_{Complex}$) are defined as statements that operate and modify sets of data. Aggregate, grouping and join operations are considered complex, because they change and re-define the original data structure. Q_{Single}-queries must provide a equi-predicate on the (compound)-key attribute. Q_{Range}-queries are identified by a range-select operation or by a selection on non-key table attributes. $Q_{Complex}$-queries are characterized by complex database operations, such as joins, groupings, or aggregations.

Classification of Access Pattern

Data Access Patterns (AP) considered in this paper are all standard SQL operations such as a projection, selection, join, grouping, aggregation or sort on a particular table column. Table 1 gives an overview of the considered access patterns. Provided that we have performance indicators for the issued SQL statements of the analyzed workload, such as execution count, rows processed and time elapsed, we can rank the relevance of database operations (AP) that are extracted from the statements.

We believe that a detailed access pattern analysis helps to characterize the application workload and focus on the relevant optimizations strategies. For example, conducting an access pattern analysis, it is possible to tell on which data segments join operations are executed, how often they are executed and if there are multi-column or single-column joins.

Listing 1: Example SQL Statement

```
SELECT * FROM DISTRICT WHERE
    WID=:B2 AND DID>B1
```

As an example, consider the SQL query illustrated in Listing 1. Assuming that it was executed ten times, and that we retrieved the average runtime of 500 ms on the query on statement level. Then, we could capture access patterns and performance characteristics as shown in Table 2. Besides the AP we use an additional text field to capture AP relevant information, such as the aggregate function in AP_{Agg} or the predicate for AP_{Sel}.

It is important to note in that example, that the (run-)time performance for each AP is derived from the original query. Although, a more fine granular run-time information on access pattern level is supported by the model and would allow a better analysis

Type	Table	Attr	Exec	Time	Info
AP_{Proj}	DISTRICT	*	10	500	
AP_{Sel}	DISTRICT	WID	10	500	=
AP_{Sel}	DISTRICT	DID	10	500	>

Table 2: Extracted AP of Listing 1

of the importance of each access pattern, it is typically not a feasible option for analyzing productive workloads due to the overhead of generating the performance data. Therefore, we used the runtime information on statement level in our analysis presented in Section 5.

Analysis of the Data Schema

In a relational database, relations are the basic entities, where each element is defined by attributes. Each attribute has a defined data type, either a primitive type, predefined by the database system or a user defined type, also called domain-type. In enterprise applications, domain-types are used in a schema to enforce data integrity among attributes of different relations. A domain-type limits the range of possible values, of all columns of that type. It is an abstraction of a primitive data-type, having a set of additional, application-specific constraints. For example, a check constraint in a domain-type MONEY requires all price columns to be greater than zero or a foreign-key constraint in domain PRODUCT_ID limits all columns to the values defined in the primary table column.

Depending on the domain definition (constraints) the number and change frequency of legal values differs. Domains, such as NAME, STREET or DESCRIPTION, defined only by a data type do not have a well-defined, reasonably-sized list of values. They are called qualified domains. In contrast, the value range of enumerated domains (e.g. GENDER, CATEGORY_ID, PRODUCT_ID or CUSTOMER_ID) and all columns defined by them is predetermined and sorted by default. Either by a incremental primary table column or even fixed by the application itself.

Domain constraints provide valuable information for columns that are dictionary compressed. Especially enumerated domain constraints, as they provide a dictionary-like, well defined value range. Moreover, we believe that a column's domain type also has an impact on how it is accessed by a workload. For example, an enumerated domain column is likely to be used in a join operation in order to link the primary table relation with its own. In order to prove this assumption, the DAC analysis references a model to segment columns, based on their domain.

Our column classification first separates qualified (D_Q) and enumerated domain (D_E) columns. Within D_Q there are three subgroups: datetime type (D_{Q-DT}), numeric type (D_{Q-N}), and character type (D_{Q-C}) qualified domains, are separated based on the data type of the domain. We divided enumerated domains (D_E) by the type of their primary domain table and adopted the SAP specific definition of master data, transaction data and Organizational and customizing tables (SAP 2013).

While master data tables and their list of domain values ($D_{E-Master}$) are frequently read, new inserts are rare. Transaction data, respectively domain values ($D_{E-Trans}$) are frequently inserted. Organizational and customizing domain values (D_{E-Cust}) are unchanged during runtime, because they are defined before the system is run productively. Finally,

D_{E-Fix} domain columns have a value range that is defined at design time. We adopted this distinction, as it can provide valuable insights for designing specific compression techniques for different types.

Analysis of Data Characteristics
Data characteristics, such as number of distinct and total data items, value distribution and default-value density are essential for database compression and optimization. As an example, the effectiveness of dictionary compression depends on the number of distinct values of a column.

Therefore, being part of our *DAC* analysis, we extract three basic data indicators. In contrast to previously conducted data characterizations, we aggregate these indicators on domain level for enumerated domain types. We determine the following figures for each column:

- Default-value count,
- Distinct value count and
- Total data items (length).

The default-value count describes the number of data entries that are equal to the most frequent value in the column. This might be a NULL value as well as any other domain value. Distinct values count is the number of unique values in a column. Total data items of a column is equal to the number of rows in the table. Based on this information we derive the following characteristics for each domain group D.

- Distinct item count,
- Total data item count and
- Default value share

For D_Q domains the distinct data items are the sum of distinct values of all columns in that domain. In contrast, the number of distinct values in a D_E domain is only based on the distinct values of the primary domain column, because all other columns use a subset of the primary domain column. Total data items in a domain are the sum of all column data items. Default value share is calculated by the number default values across all domain columns and the total data item count. It gives us the share of default data items in a domain.

4 Analyzed Application Contexts

This Section introduces the different database application contexts we have analyzed.

Transactional applications are the backbone of any enterprise. Invoices, sales orders and accounting documents - all enterprise specific business entities are first captured and processed in a transactional systems. Analytical applications built upon a star schema, typically trade intense data compression, data redundancy as well as limited update performance for the efficiency to process and analyze increasing amounts of data in a fraction of time. However, transactional applications, built on a normalized data schema remain the necessary prerequisite and source for these systems. As proposed by Plattner in (Plattner 2009), our research builds on on the assumption that the central data source for mixed-workloads is based on a normalized transactional schema. Consequently, all application contexts analyzed here are based on a normalized data schema.

The first application context is a productive transactional workload, captured from a productive SAP ERP (SAP 2013) system of a company with roughly 20,000 employees. It provides the status-quo and the basis of future mixed-workload applications. The well established *TPC-C* benchmark (TPC 2010) is used to compare a transactional benchmark with the productive, transactional workload. Furthermore, we have analyzed the mixed-workload benchmark *CH-benCHmark* (Cole et al. 2011).

4.1 Transactional *Enterprise* Application

Our transactional context analysis is based on data from a large productive SAP ERP system, used for financial-, sales-, distribution- and production- processing. The workload of that application was traced over the period of one work-week. During that time, there were no untypical periodic loads, such as year-end-processing, etc. The analysis of a single system may not be representative for transactional enterprise applications in general. However, as the analyzed system had only few modifications to the standard SAP ERP system, which is widely used and a de-facto standard in most industries, it can be considered as highly representative. Besides, we consider the possibility to analyze a productive ERP system of a company of that size as a great opportunity and expect that these results are valuable for the community. We constantly try to find more companies that are willing to share data to increase the data base.

In the presented case, the workload information was extracted from the shared pool buffer of the underlying database of the running ERP system. Along with the executed SQL statements (without variable binding), we tracked relevant usage statistics of each statement, such as the number of executions, the time elapsed (runtime) and the number of rows processed.

In total, we extracted the following quantities:

- Total number of SQL queries: 144.000
- Analyzed SQL queries: 23.000 (92% load)
- Users: $\approx$ 1.000 active SAP Users (24h)
- Number of SQL executions: 3.300.000.000
- Rows processed: 20.200.000.000
- Total query runtime: 1.900 h

4.2 Transactional *TPC-C* Benchmark

In order to allow references to existing research and compare the transactional, enterprise application with a benchmark context, we analyzed the workload and schema of a TPC-C benchmark (TPC 2010). In the following, this workload is referred to W_{TPC-C}. W_{TPC-C} consists of 33 transactional queries, each having a defined share of the entire runtime and execution count.

4.3 Mixed-Workload *CH-benCHmark* Benchmark

With the motivation of comparing the performance of mixed workload databases, a group of researchers designed a benchmark, called CH-benCHmark (Cole et al. 2011). CH-benCHmark is a composite benchmark, combining the well known, transactional TPC-C (TPC 2010) and the analytical TPC-H (TPC 2013) on the normalized TPC-C data schema.

(Cole et al. 2011) examine three scenarios, weighting OLTP and decision support (DS) workload streams with different factors. Then they compare the execution results to see how those workloads effect each other in terms of *tpmC* and *QphH*, using

a reference number where each workload was run in isolation. The number of OLTP and DS streams does not emulate the number of business users, but can be interpreted as a ratio between those two workload types. For our workload analysis we chose two workload mixtures. One, where OLTP and DS queries have the same share - W_{CH11} and, as a reference for a analytical workload, one that consists of TPC-H queries only - W_{CH01}.

As we do not intend testing performance scalability of a system under load or how the different workloads effect each other, we are only interested in relative execution statistics (execution count, runtime) of each query in the same time frame. The benchmark conducted in (Cole et al. 2011), running on a PostgreSQL system provided the reference numbers. It is based on the assumption, that there is no contention between workloads, that they run on the same database size and that there is no think-time between requests. After a 5 minutes warm-up, one OLAP client running in isolation performs in average 5200 tpmC, whereas one OLAP stream runs 902 QphH. These reference numbers were used to calculate the execution statistics of each query presented in Section 5.

5 Analysis of Workload Characteristics

In this Section, we present the results of our *DAC* analysis of the workloads described in Section 4.

The results of our analysis are presented for each of the models introduced in Section 3 (i) analysis of query classes, (ii) analysis of access patterns, (iii) analysis of the accessed domain types, and (iv) data characteristics. A comprehensive discussion of the results will follow in Section 6.

5.1 Classification of Queries

Figure 2 shows the query classes of the workloads introduced in Section 4, by displaying for each of the six query classes (Q_{Range}, Q_{Single}, $Q_{Complex}$, Q_{INS}, Q_{UPD}, Q_{DEL}) of the Q model the percentage of total executions and total runtime for each query class. This way, we can compare the relative importance of each query class between workloads, even as they differ in absolute numbers.

Looking at the results of $Q_{Complex}$ in Figure 2, it is notable that the share of the total runtime is higher than the share of executions, which is in line with our expectations when defining the query classes. Comparing the workloads, it is striking that W_{TPC-C} and W_{CH11} have a much higher number of update and insert statements as the productive workload W_{Trans}. Comparing W_{TPC-C}, W_{CH01} and W_{CH11} we see an increasing percentage of complex query runtime, which is as expected given the increasing amount of analytical queries.

The total number of executed queries of W_{CH01} is almost negligibly low compared to W_{TPC-C}, as the analytical queries run orders of magnitude longer. The W_{CH11} workload is a mix of W_{TPC-C} and W_{CH01}. Consequently, the total number of executed queries in W_{CH11} is almost equal to W_{TPC-C}. However, comparing the runtime of the statements we see an increase of $Q_{Complex}$ queries for W_{CH11} and obviously for W_{CH01} as well, compared to the transactional workloads W_{TPC-C} and W_{Trans}.

The "total execution count" provides a runtime-independent figure of the share and the impact of each query class Q. Unfortunately, this is not meaningful for the W_{CH11} benchmark. Because the runtime for the TPC-C part is the same as for TPC-H part, it causes very few TPC-H queries to be executed. A difference in execution count between W_{TPC-C} and W_{CH11} is almost not perceivable. However, comparing the runtime of the statements, we see an increase of $Q_{Complex}$ queries for W_{CH11}, and obviously for W_{CH01} as well, compared to the transactional workload W_{TPC-C}. Hence, we consider runtime as the performance indicator to analyze the relevance of access patterns in the next section.

5.2 Classification of Access Patterns

In order to measure the occurrence and importance of individual database operations, we use the extracted *AP* and the performance figures of their SQL statement. For each *AP* we know to which SQL statement it belongs. Based on the performance figures of those statements, we measure the share and impact of *AP*. This way, we determine the relevance of an *AP* in a workload. It is important to mention that we could only extract runtime information on statement level. Thus, the indicated percentage of total runtime for an *AP* indicates that the execution time of all queries that contained this specific *AP* accounts for this percentage, but not necessarily the *AP* itself. Extracting more fine granular performance data on database operation level would have had a too high performance impact on the live system and was not a viable option.

Figure 3 illustrates and compares the share of statements having certain *AP*, in W_{Trans}, W_{TPC-C}, W_{CH11} and W_{CH01}. It shows that joins do not account for a significant percentage of the total runtime in W_{TPC-C}, whereas W_{Trans} shows a relevant share. As expected the impact of complex database operations such as aggregations, sorting, joins or grouping plays a much greater role in W_{CH11} and W_{CH01}. Although this analysis does not reveal the individual runtime contribution of each of these more complex *AP*, we can confirm that more complex *AP* play a significant role in analytical style workloads executed on a normalized data schema.

In the next section the qualitative characteristics of access patterns in workloads will be analyzed, illustrating detailed figures on the data segments that are accessed by AP.

5.3 Classification of Data Schema

This section shows the qualitative analysis of the access patterns *AP* and their correlations with accessing certain domain types. We restrict the presentation here to read queries Q_R, as they account for most executions and runtime in our productive application context W_{Trans}.

Each *AP* references a data segment (table and attribute), has two performance indicators (execution count and runtime) and an additional text-field, used to qualify the *AP* with additional information, such as select-predicate or aggregate function (Section 3.2).

In this paper, we show in detail the access patterns of selections and joins, as selections are the most used access pattern and join is the access pattern that typically has the longest execution time. We have further analyzed the accessed data types of aggregations and groupings and briefly mention the results.

For aggregations, SUM, AVG as well as MAX and MIN are primarily used on D_{Q-N} columns. Especially SUM as the basic analytic operation is only conducted on numeric domain columns. The COUNT function is typically applied to the anonymous star-column and returns the number of rows in a query.

Group by operations are the requirement for most aggregate functions. As shown in Figure 3 they are

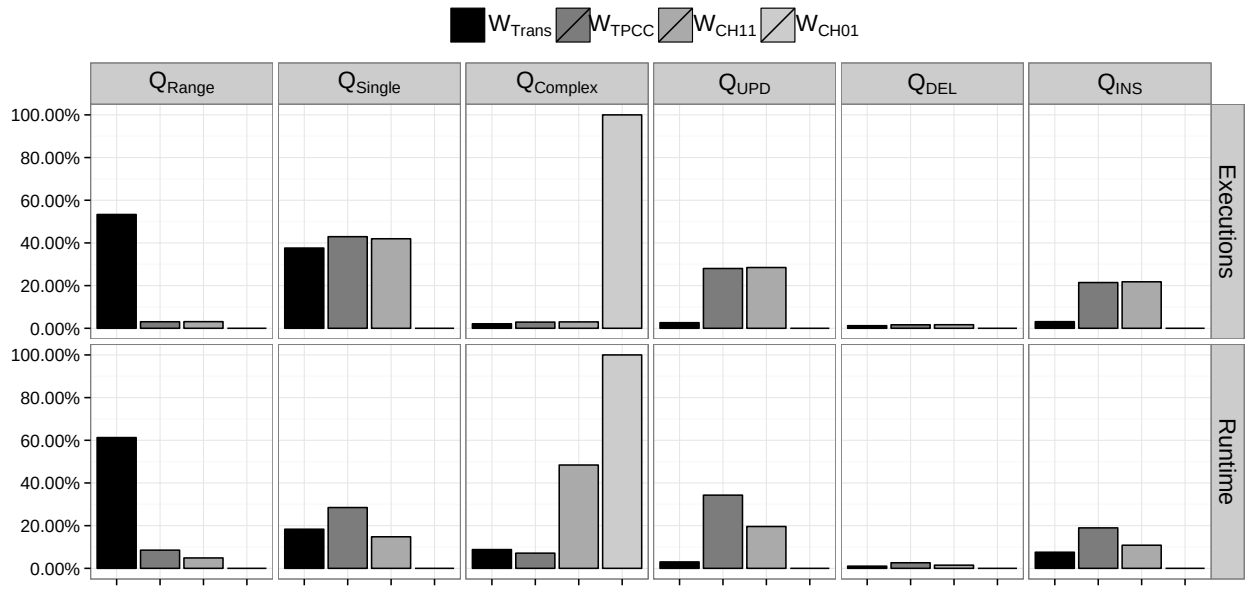


Figure 2: Query-Class Workload Characteristics

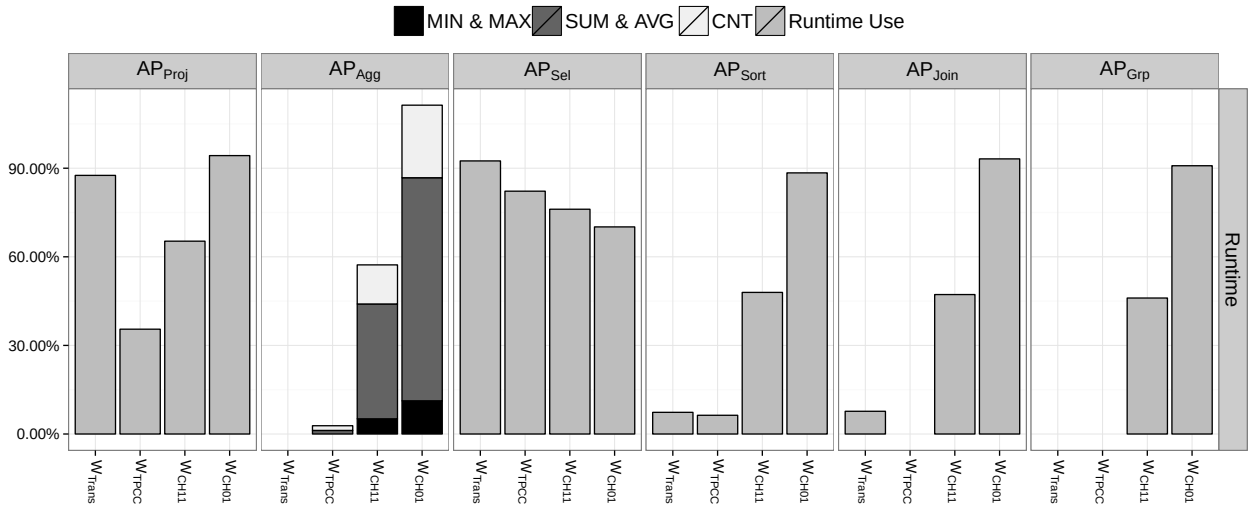


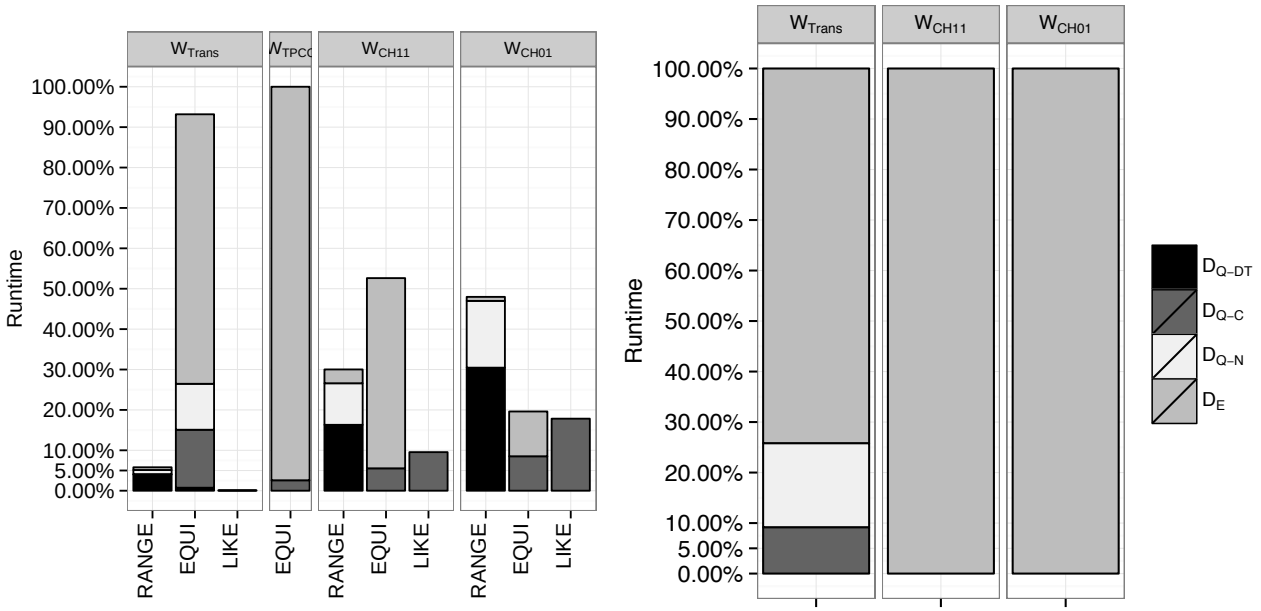
Figure 3: Access Pattern Workload Characteristics, based on Total Runtime

essential and intensively used in analytic queries. Our analysis showed that groupings happen largely on enumerated domain types. Although groupings happen on name or other qualified domains in W_{CH11} and W_{CH01}, we would expect them to be implemented as enumerated domains in productive applications, as names are typically not enforced to be unique. E.g. instead of using the customer.name attribute the customer.id attribute would be used for grouping. The analysis of accessed domain types revealed that in W_{Trans}, 82% of all AP_{Grp} are conducted on D_E columns. Besides, there are 17% D_{Q-DT} columns used to group selected data sets, based on their datetime attribute.

Data Selection on Domain Columns

Data selection is the most used AP in all analyzed workloads. There is almost no query without data selection operation(s) as shown in Figure 3. The AP_{Sel} has an additional variable, called "predicate-type". In Figure 4a, we use three predicate-groups: RANGE (<,<=,>,>=,between), LIKE and EQUI (=,!=,IN), to concentrate similar predicates in groups. The bars in the chart represent the overall fraction of AP_{Sel} predicate-groups within each W based on the runtime of their SQL statements.

In the purely transactional workloads W_{Trans} and W_{TPC-C} the EQUI-predicate is predominant. RANGE and LIKE predicates play an important role in the workloads W_{CH11} and W_{CH01} that also consist of analytical queries. Looking at the domain-type distribution of EQUI predicated, most selections operate on D_E columns: From left, W_{Trans} to right, W_{CH11}, 77%, 98%, 89% and 57% operate on D_E and especially on D_{E-Cust} columns. Analogously, most RANGE predicates operate on qualified domains: 89% in W_{Trans}, 89% in W_{CH11}, and 99% in W_{CH01}. Only a small fraction of 11% of RANGE-AP_{Sel} operate on $D_{E-Trans}$ columns.



(a) Select operation

(b) Join operation

Figure 4: Operation / domain group dependencies based on runtime

Join Operations on Domain Columns

The access pattern AP_{Join} is mapped to the join conditions of SQL statements. One AP_{Join} always reflects one join attribute. Thus, a standard join-condition on one attribute, over two tables yields two AP_{Join}. This way, we can compare the domain-types of join-couples and see that they always relate to the same domain. It shows that while attribute names can be different, their legal set of possible values is always the same.

Figure 4b shows the domain-types used in AP_{Join} based on the SQL runtime. The presented analysis only considers equi-joins, as other join types have been negligible in runtime and execution time in the analyzed workloads. In W_{Trans} we see that 79% of all join attributes are D_E columns. In W_{CH11} and W_{CH01} all join attributes are D_E columns. Convinced that only D_E columns are appropriate to join relations we further investigated the 21% of D_Q columns used for join operations in W_{Trans}. It shows that these columns are also key attributes, having a primary table column. However, they are not defined as an D_E domain type in the schema definition. This might be due to historic reasons or simply suboptimal schema design.

Domain-Type Distribution and Characteristics in Data Schema

In this section we analyze the distribution of domain groups D in the schema of W_{Trans}, as well as the characteristics of the data that is stored in those groups. We only show W_{Trans}, as the data schema of the benchmarks W_{TPC-C}, and similarly W_{CH01} and W_{CH11}, are too simplified to be representative for productive enterprise systems. While D_E domains have a shared, well-defined set of distinct values for all columns of the same domain, D_Q columns do not have such a list of domain values. Every column defines its own distinct values.

The y-facet "Distinct Domain Items (%)" of Figure 5 shows the sum of all distinct values for each domain group. It points out that D_Q domains make up 90% of all distinct values stored in the analyzed W_{Trans}. Especially D_{Q-C} and D_{Q-N} have a considerably high number of distinct values. D_E columns contribute only 10% of the distinct values. However, the share of "Data Items (%)" is almost equally distributed between D_E and D_Q. While not shown in the figure, the same spreading between D_E and D_Q can be seen for the number of columns, respectively domains in the data schema of W_{Trans}. In detail there are $\approx$ 17% D_{E-Fix}, $\approx$ 28% D_{E-Cust}, $\approx$ 3% $D_{E-Trans}$ and $\approx$ 7% $D_{E-Master}$ columns in the productive data schema. Additionally, there are $\approx$ 7% D_{Q-DT}, $\approx$ 15% D_{Q-N} and $\approx$ 23% D_{Q-C} columns.

The low share of distinct values per domain, accompanied by the high share of data items in D_E columns results in a very low uniqueness (share of distinct values among all data items).

As generally shown in (Huebner et al. 2011), the distinct values of a domain are not equally distributed. Our analysis confirms this and reveals a significant share of default values among D_E columns. On average 84% of all D_{E-Fix} and 73% of all D_{E-Cust} data items contain the column's default value. As these are just average numbers across all domain group columns a default-value aware compression algorithm can be very effective for certain relational attributes.

6 Discussion of Results

This section summarizes the main findings from the results of our DAC analysis presented in Section 5.

Our major insights of the DAC Analysis are:

- Enterprise workloads are read heavy; interestingly, we see a much higher percentage of read queries compared to the industry benchmark TPC-C.
- Specific database operators predominantly operate on attributes with a specific domain type.
- Data characteristics differ depending on the domain type.

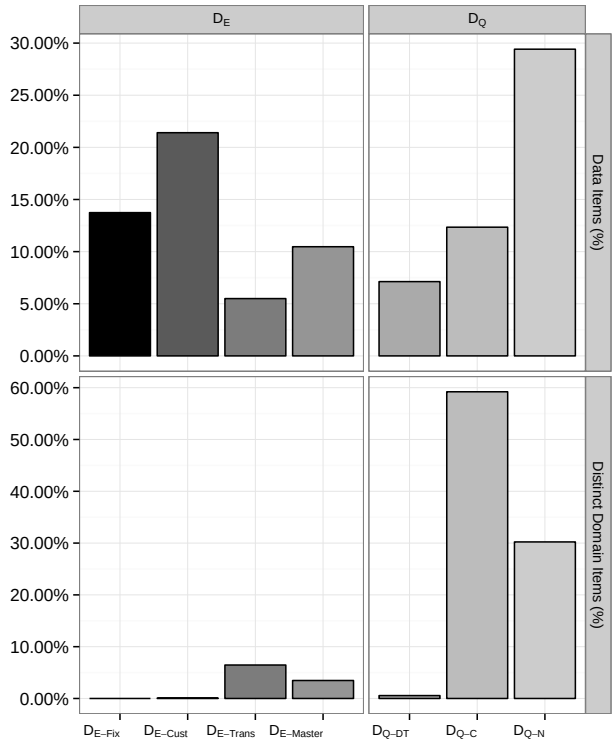


Figure 5: Data Characteristics of W_{Trans}

- We expect a trend towards more complex queries and operators with the introduction of modern enterprise applications.

We will briefly explain each of the insights and discuss implications.

Based on the analysis presented in Section 5.1, we see that more than 90% executed statements which account for around 90% execution time in the productive transactional workload are read-only statements. In contrast, the TPC-C workload shows an almost equal distribution of read and write statements. This is in line with the findings in a previous analysis of data access patterns (Krueger, Grund, Zeier & Plattner 2010), which derive the adequacy of read-optimized column stores based on this insight. This confirms that the system we have analyzed does not show an exceptional behavior and gives us confidence that we can generalize our findings.

When analyzing the domain types of attributes and the operators that access these attributes, as shown in Section 5.3, we see a clear correlation between domain types and operators across all workloads. Equi-selects, being the most important operation in all workloads, used in all query-classes (except Q_{INS}), strongly depend on enumerated domain columns, whereas D_{Q-N} and D_{Q-DT} columns make up more than 90% of all range-selects in all workloads. The importance of enumerated domains is also shown for join and grouping operations, that primarily depend on D_E domain-types. Character columns are only used in like-selection and data projections. These findings are valuable for optimizing operator implementations for domain types they mostly operate on. We think that a domain based optimization of data structures is a promising approach for highly normalized relational databases. Each group of domain types holds specific characteristics that can be leveraged by a different data structure. What we have seen is that while D_Q columns contain any possible, user-defined data, D_E columns are determined and controlled by the application, only. As a consequence D_E columns are essential to enforce data consistency as well as join, select and group data. As they are deterministic they are preferred, if not required for many database operations. On the other hand range-selections, aggregate functions and like-selection are irrelevant or even illegal on these D_E columns, because their domain values are always of nominal scale.

A closer look at the value distributions of the data set of the productive ERP system shows that the number of distinct values in all columns with a qualified domain is roughly ten times higher than the distinct values in all columns with enumerated domains which share values among columns (See Section 5.3).

Domain type information are available during design time. Compared to approaches that analyze workloads and data characteristics during runtime, it does not pose any overhead. We see that the effectiveness of different compression techniques such as sorted, unsorted as well as shared and attribute-wise dictionary compressions depends on the specific domain context. A shared or global dictionary for example seems reasonable for D_E attributes, in order to leverage a common dictionary encoding during join processing. Besides, our result show that it is questionable if D_{Q-C} columns profit from column-wise data structures as there are almost no operations on single D_{Q-C} columns. Therefore, we plan to leverage these insights by designing specific encoding and compression techniques depending on the domain type of attributes.

Based on our identified trends in Section 2.2, we see more complex data operations in future enterprise workloads. Ad hoc aggregations as well as more analytical style queries will lead to a mix of shorter running transactions, as well as potentially longer running queries that read, join and process large amounts of data. In line with the analysis of W_{CH11} shown in Figure 2, we expect a higher absolute number of transactional queries, but a larger share of the total execution time accounted by analytical queries. This will be further intensified by a trend to push logic into the database instead of only reading data from the database in bulk and process it in the application. As a potential implication, short running transactional queries, as well as the more complex queries, might compete for resources. In the worst case, complex queries can occupy all database resources and block short transactional queries from executing. Hence, a direction for future work is to design workload management systems for mixed workloads.

7 Related Work

We have identified related work in two fields: characterization of database workloads, as well as the underlying data. Our work extends prior work in workload characterization by analyzing the correlation of database queries of a workload as well as accessed data. Furthermore, we present results that have been obtained from analyzing the workload of a large scale, productive enterprise application system.

7.1 Workload Characterization

Workload analysis and characterization is the requirement for many performance studies and a foundation for various benchmarks, built as the synthesized, controlled workload, which can be used to measure and compare the performance of different environments. (Elnaffar & Martin 2002) summarize and compare

workload analysis conducted in different application fields. They basically classify and characterize analysis techniques into two categories: *static* and *dynamic*. Static techniques, such as descriptive statistics, histogram, component analysis and clustering are used to characterize static workloads components. Dynamic techniques are used to describe the probability of system transitions. Knowing a certain performance figure at a certain point in time is not sufficient. Instead the workload components must be captured periodically to show dependencies between different parameters. Based on our motivation, the analysis conducted in this paper is of static nature.

7.2 Data Characterization

In general, data characteristics of enterprise systems have been investigated in (Krueger et al. 2011), (Krueger, Grund, Zeier & Plattner 2010) and (Huebner et al. 2011). They show that many columns in standard enterprise systems have a low number of distinct values. A share of 35% of all analyzed table columns is even unused, due to the wide table schema needed in standard software to support various industries and customer requirements. Besides, Huebner et al. analyze the value distribution of FI table columns in an SAP ERP system. They found that half of all columns are best approximated by a uniform distribution, while the other part adheres to a zipf pdf distribution. Based on that information they built a merge strategy that is optimized for pdf zipf distributed columns.

8 Conclusion and Future Work

In this paper, we have presented Database Application Context (DAC) Analysis, a framework used to extract and classify queries, used database operations, and data access patterns, as well as execution times from SQL workloads. With *DAC*, we have analyzed a productive enterprise resource planning system, as well as established database benchmarks. Based on an analysis in trends in modern enterprise applications, we have derived expected changes to the workloads of enterprise applications in the future. We are confident that our results can be a valuable input for optimization data structures of the data management layer of enterprise applications. We plan to leverage these findings in two dimensions: (i) develop efficient database operations and compression techniques for enumerated domains that are shared among attributes, and (ii) implement effective query schedulers to handle a mixed workload of different query classes.

References

Bello, R. G., Dias, K., Downing, A., Feenan, Jr., J. J., Finnerty, J. L., Norcott, W. D., Sun, H., Witkowski, A. & Ziauddin, M. (1998), Materialized views in oracle, *in* 'VLDB'.

Chaudhuri, S. & Dayal, U. (1997), 'An overview of data warehousing and olap technology', *SIGMOD* .

Cole, R., Funke, F., Giakoumakis, L., Guy, W., Kemper, A., Krompass, S., Kuno, H., Nambiar, R., Neumann, T., Poess, M. & Others (2011), The mixed workload CH-benCHmark, *in* 'DBTest'.

Elnaffar, S. & Martin, P. (2002), Characterizing computer systems' workloads, Technical report, School of Computing, Queens University.

Gillin, P. (2007), 'Bi @ the speed of business', *Computer World Technology* .

Golfarelli, M., Rizzi, S. & Cella, I. (2004), 'Beyond data warehousing: what's next in business intelligence?', *DOLAP* .

Halevy, A. Y. (2001), 'Answering queries using views: A survey', *VLDB* .

Huebner, F., Boese, J.-H., Krger, J., Renkes, F., Tosun, C., Zeier, A. & Plattner, H. (2011), A cost-aware strategy for merging differential stores in column-oriented in-memory dbms, *in* 'BIRTE'.

Kemper, A. & Neumann, T. (2010), Hyper hybrid oltp&olap high performance database system, Technical Report May, Technische Universitaet Muenchen.

Krueger, J., Grund, M., Zeier, A. & Plattner, H. (2010), Enterprise application-specific data management, *in* 'EDOC 2010'.

Krueger, J., Kim, C., Grund, M., Satish, N., Schwalb, D., Chhugani, J., Dubey, P., Plattner, H. & Zeier, A. (2011), 'Fast updates on read-optimized databases using multi-core cpus', *VLDB* .

Krueger, J., Tinnefeld, C., Grund, M., Zeier, A. & Plattner, H. (2010), A case for online mixed workload processing, *in* 'DBTest'.

Kuno, H. A., Dayal, U., Wiener, J. L., Wilkinson, K., Ganapathi, A. & Krompass, S. (2010), Managing dynamic mixed workloads for operational business intelligence, DNIS.

P. Larson and H. Z. Yang (1985), 'Computing Queries from Derived Relations', *VLDB* .

Plattner, H. (2009), A common database approach for oltp and olap using an in-memory column database, SIGMOD.

Plattner, H. (2011), Sanssoucidb: An in-memory database for processing enterprise workloads, *in* 'BTW', pp. 2–21.

SAP (2013), 'Sap erp', http://www54.sap.com/solutions/bp/erp.html. [Online; accessed 22-August-2013].

Tinnefeld, C., Mueller, S., Zeier, A. & Plattner, H. (2011), Available-to-promise on an in-memory column store, *in* 'BTW'.

TPC (2010), Tpc benchmark c (standard specification) - revision 5.11, Technical report, Transaction Processing Performance Council.

TPC (2013), Tpc benchmark h (standard specification) - revision 2.16.0, Technical report, Transaction Processing Performance Council.

White, C. (2005), 'The next generation of business intelligence:operational bi', *DM Review Magazine* .

Wust, J., Boese, J.-H., Renkes, F., Blessing, S., Krueger, J. & Plattner, H. (2012), Efficient logging for enterprise workloads on column-oriented in-memory databases, *in* 'CIKM'.

Wust, J., Krueger, J., Blessing, S., Tosun, C., Zeier, A. & Plattner, H. (2011), xsellerate: Supporting sales representatives with real-time information in customer dialogs, *in* 'In-Memory Data Management'.

Yang, H. Z. & Larson, P.-A. (1987), Query transformation for psj-queries, *in* 'VLDB'.