# Analyzing Data Relevance and Access Patterns of Live Production Database Systems

Martin Boissier, Carsten Meyer, Timo Djürken, Jan Lindemann, Kathrin Mao,
Pascal Reinhardt, Tim Specht, Tim Zimmermann, Matthias Uflacker
Hasso Plattner Institute, University of Potsdam, Germany
{firstname.lastname}@hpi.de

## ABSTRACT

Access to real-world database systems and their workloads is an invaluable source of information for database researchers. However, usually such full access is not possible due to tracing overheads, data protection, or legal reasons. In this paper, we present a tool set to analyze and compare synthetic and real-world database workloads, their characteristics, and access patterns. This tool set processes SQL workload traces and collects fine-grained access information without requiring direct read access to the production system. To gain insights into large real-world systems, we traced a live production enterprise system of a Global 2000 company and compare it with the synthetic benchmarks TPC-C and TPC-E.

## Keywords

Mixed Workloads; OLxP; Data Smartist; Data Relevance; Data Tiering; Access Patterns; ERP; Production Systems

## 1. INTRODUCTION

Database researchers regularly face the problem of how to evaluate new ideas or algorithms. Synthetic benchmarks do not accurately reflect the properties of real-world systems [2]. To understand why some research projects provide only limited benefits in realistic scenarios, access to production systems is required. An example for the mismatch between real-world systems and benchmarks is the often employed partitioning along the warehouse identifier in TPC-C with an almost perfect distribution of both data and workload. A scenario barely found in real-world systems.

Insights from real-world applications help to make better and more profound assumptions. In some cases, it is possible to access the data of a system or the database plan cache (i.e., cached storage of execution plans without predicate bindings). But neither facilitates the ability to answer questions, such as "Which queries join tables $X$ and $Y$ and return data older than three years?" or evaluate different partitioning schemas. For those questions, access to both – data as well as the full SQL workload trace – is needed.

We present a tool set that answers such questions by parsing SQL workload traces and obtaining access information without the need of direct read access to the production database system and thus without access to business-critical information. With this tool set, we can analyze access patterns and quantify the relevance of data down to the level of accesses per tuple.



*CIKM'16 October 24–28, 2016, Indianapolis, IN, USA*



ACM ISBN 978-1-4503-4073-1/16/10.

DOI: http://dx.doi.org/10.1145/2983323.2983336

### 1.1 Motivation

Mixed workloads (OLxP) and in-memory databases are two of the major trends in enterprise systems computing. Both led to a dramatic shift in the workload modern database systems are expected to handle. This change has vast implications for research topics such as data aging/tiering or partitioning. For data aging, the classification of relevant/hot data is comparatively straightforward for classical OLTP systems with single tuple-focused accesses. But modern applications with OLxP characteristics – such as complex joins and aggregations – exhibit completely different access patterns [1]. Hence, data relevance questions cannot be studied using OLTP-oriented benchmarks as TPC-C/-E.

In this project, we want to provide profound insights into real-world enterprise resource planning (ERP) systems by analyzing both, single tuple accesses as well as complex analytical queries in order to quantify data relevance down to the finest level of granularity based on production database workloads.

### 1.2 Analyzed Workloads

Krueger et al. have analyzed ERP systems and shown that synthetic benchmarks as TPC-C do not reflect the properties found in real-world enterprise applications accurately [2]. This was our major motivation to focus on complete SQL workload traces of production systems instead of synthetic benchmarks or plan cache exports. Using SQL workload traces allows the analysis of topics as partitioning and query pruning, which is not possible using solely the database plan cache. Hence, we think it is crucial to look at 1) real-world data instead of synthetically created data and 2) real-world workloads to analyze and evaluate database systems.

Besides synthetic benchmarks, we have analyzed real-world systems, i.a., the following *live production SAP-based Enterprise Resource Planning (ERP) system of a Global 2000 company*:

- Over seven terabytes of uncompressed data
- Handles ~1.5 billion queries each day
- Sampled SQL workload traces of the financial and controlling modules over several days, resulting in ~50M queries
- One of the most recent versions of an SAP ERP system including operational reporting on transactional data (i.e., OLxP)
- Plan cache exports over a time span of several weeks

## 2. WORKFLOW

In short, the workflow of our analyses looks as follows: First, the logging functionality of the traced database is activated to log

all queries. Depending on the system load, the tracing can be iteratively activated for short intervals. The resulting query logs are copied to another system, which runs the analyses to keep the load on the production system as low as possible. On the analysis system, the query logs are parsed to extract relevant information (e.g., selections, predicates, projections, join partners, et cetera).

In case it is possible to access a snapshot of the database – directly or indirectly – a SQL log replay on a mirrored source system is performed to obtain tuple access information (Section 3). The results of the analyses are visualized in an HTML-based front end.

# 3. SQL LOG REPLAY

Having the complete SQL workload trace of a live production system is already a tremendous source of information. However, to discuss issues such as data aging/tiering, data relevance, and partitioning, further analyses are necessary. We replay SQL workload traces on a database snapshot with which we can determine for every tuple, which queries have accessed it, when, and how often.

*Indirect Snapshot Access.* Our research focuses on production ERP systems. Consequently, getting direct access to a snapshot of the enterprise system for the replay is hardly possible as it contains business-critical financial data. Therefore, we implemented the replay process in a way that the process can be run by the owner of the ERP system and will output only statistical data (i.e., *indirect access*). This data includes database meta data as table sizes, existing indices et cetera. But more importantly, we exported the internal tuple identifiers (i.e., the row identifier) of each replayed query. We use the tuple identifiers of the result set of each query to answer questions as "Which data is actually relevant and regularly accessed?". To obtain the tuple identifiers for returned tuples, we replace the projected columns by the internal system column (e.g., `$ROW_ID$` for SAP HANA) storing tuple identifiers.

```
-- original query
SELECT CUSTOMER_NAME, PRODUCT_ID, AMOUNT, CURRENCY
FROM CUSTOMER_ORDERS_VIEW
WHERE CUSTOMER_ID=17
GROUP BY CUSTOMER_ID;
-- rewriting to access base tables and obtain tuple
-- identifiers instead of the actual projection
SELECT REWR_VIEW.ORDERS_ROWID, REWR_VIEW.CUSTOMERS_ROWID
FROM (SELECT O.CUST_ID, O.$ROW_ID$ AS "ORDERS_ROWID",
        C.$ROW_ID$ AS "CUSTOMERS_ROWID"
   FROM ORDERS__TIMESLOT_004 AS O JOIN CUSTOMERS AS C
     ON O.CUSTOMER_ID=C.CUSTOMER_ID
) AS REWR_VIEW WHERE REWR_VIEW.CUSTOMER_ID=17;
```

**Listing 1: Rewriting of queries using views.**

## 3.1 Data Growth

Another issue is missing information about data growth when only a single database snapshot is available. Due to data protection, the obtained traces include full `SELECT` statements but only masked `INSERT/UPDATE` statements (i.e., no values) as they potentially include business data. If we now replay the SQL workload traces that have been traced over a long period, queries such as '`SELECT * FROM ORDERS WHERE CUSTOMER_ID = 3 ORDER BY DATE DESC LIMIT 0, 10`' would always return the same set of tuples even though the data might have changed in the meanwhile.

To deal with that issue we recorded the maximum and minimum tuple identifiers of each traced table each time the SQL workload trace was started or stopped (i.e., twice every ten minutes in the case of the ERP system). This information is used to create a logical view on top of the base table of the static snapshot, limiting the size of the table to an approximation its size at the traced point of time. For each traced table, a view accessing the original table is created that selects on the tuple identifier column to limit the table to the corresponding table size at that point in time.

| | `SELECT` (joining[1]) | | `UPDATE` | `INSERT` | `DELETE` |
|---|---|---|---|---|---|
| **SAP ERP** | 97.10% | (21.20%) | 0.25% | 2.65% | 0.00% |
| **TPC-C** | 69.48% | (2.52%) | 18.57% | 9.76% | 2.20% |
| **TPC-E** | 88.14% | (15.91%) | 6.66% | 4.79% | 0.4% |

**Table 1: Distribution of reading and modifying statements.**

## 3.2 Views

Plattner [4] discussed modern enterprise applications that make increasing use of logical database views as they ease the transition to new systems. Most interestingly, most complex OLxP queries including aggregations and joins are written using views. We found views in the financial module that join up to five tables and reference other views themselves (up to six layers of views). In order to obtain all accesses on a particular table for our analyses, we rewrite queries on views to directly access the actual base tables. There are several issues to consider when rewriting views as handing down tuple identifiers from the base tables up to the top level projection, including resolving potential attribute name collisions and more. An exemplary rewrite of a query on a view to directly access the base tables and project tuple identifiers is shown in Listing 1.

# 4. REPLAY PERFORMANCE

The performance of the query replay is crucial since there are hard time constraints whenever the ERP system owner has to provide a system copy (e.g., a standby system). Besides obvious measures as proper parallelization and efficient storing of results, our focus is on the reduction of queries (e.g., recognition of semantic duplicates), which are to be executed.

In short, the replay works as follows:

1. To group queries, the interval length is defined. As a lower bound, the interval length depends on the granularity of which the table size information is exported (e.g., every ten minutes for 90 seconds). Shorter intervals improve accuracy but in parallel impede performance (cf. Step 2).
2. Per time slot the following steps are taken:
   (a) For each query, we sort search arguments lexically by attribute name and replace projections by `$ROW_ID$`.
   (b) A mapping between distinct replay queries and their initial trace instances is created. For the replay, only distinct replay queries have to be executed, because the initial queries that map to that replay query yield the same tuple accesses anyways.
3. The queries of each time slot are executed in parallel. The eventually executed query projects its own query identifier to allow for later identification and the tuple identifiers.
4. After the replay, tuple accesses are again mapped to the original query trace instances and stored.

# 5. DEMONSTRATION

Our demonstration allows to browse through our HTML-based workload analyses and investigate the differences between synthetic workloads and a traced production SAP ERP system. We provide

[1]Percentage of `SELECT` queries that join tables.

guided access to both our visualization for easy browsing and to the database storing the trace information to answer more detailed questions from the audience.

The following two sections present the two main analyses of the front end that the audience, amongst others, can walk through.

## 5.1 Partitioning Evaluation

Effective partitioning is crucial for analytical applications, which usually access data without indices and hence profit from effective pruning. We provide a tool that allows evaluating different partitioning schemas interactively. With the help of that tool, we were also able to evaluate several published automated partitioning approaches, e.g., *aggressive data skipping* [6] or *PREF* [7].

## 5.2 Workload Characteristics

We revisited the numbers of Krueger et al. [2] who compared TPC-C with a production ERP system. For the ERP system, we calculated workload fractions for tables corresponding to the TPC-C benchmark. As listed in Table 1, the examined part of the ERP system is even more read-dominated than expected and executes joins significantly more often than TPC-C. Comparing TPC-C and TPC-E solely by these numbers, TPC-E does a better job reflecting production ERP systems. To further ease the comparison of workloads, we visualize the distribution of queries over time, query selections and their particular value distributions (Fig. 1), and the ratios of query types per table. Such analyses of read/write ratios and the composition of expensive queries are of particular interest for OLxP-optimized databases, which deploy read-optimized (columnar) data structures [1, 3, 5].

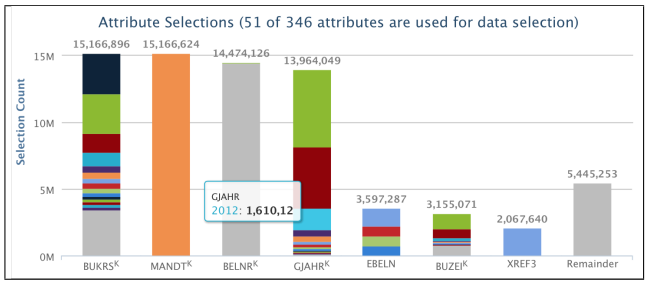


**Figure 1: Demo screen showing which columns of SAP ERP's main accounting table *BSEG* have been selected with which values.**

*Time-Correlation of Data Accesses.* Apart from calculating access frequencies, access skew, et cetera, we tried to visualize data access patterns, which we have obtained from the workload replay. We decided to combine time series and heat maps to visualize decreasing relevance of transactional data over time.

Fig. 2 shows a heat map visualizing accesses to a table storing customer master data. For this particular example, it can be seen that recent data is not accessed at a significantly higher frequency than less recent data. In contrast, transactional data (e.g., TPC-C's `ORDER` or SAP EPR's `BSEG` table) exhibit more time-correlated patterns where recent data is accessed at a much higher frequency while accesses decrease over time as tuples age.

However, comparing the heat maps of the traced ERP system and synthetic workloads shows a significant difference in the number of accesses to non-recent data. One reason is the increasing analytical load of such OLxP systems. More interestingly, another reason is the removal of materialized aggregates in favor of on-the-fly aggregations in modern enterprise systems [4]. Without materialized aggregates, the fraction of frequently accessed tuples is much larger, because aggregations are not longer lookups to recently modified tuples (thus probably cached) but complex scans and joins of large sets of data. This development can be nicely observed and compared using the heat maps and is particularly interesting for research areas as buffer management or data tiering.

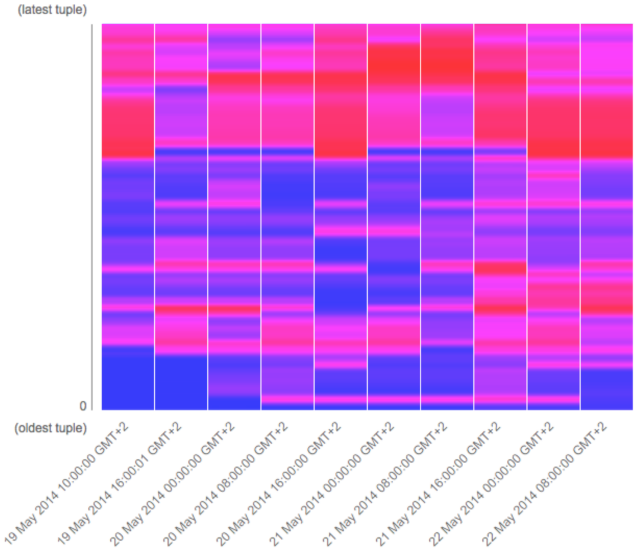


**Figure 2: *Heat map visualization* of tuple accesses for an SAP ERP master data table. Each column shows the accesses to the table for an 8h long time slot.**

# 6. CONCLUSION

The presented tool set analyzes large SQL workload traces and gathers access information down to the finest level of granularity without the need of direct access to the production system. It visualizes various aspects that allow exploring and comparing workload characteristics and thus enable thorough evaluations in a realistic OLxP scenario.

Our analyses show that both TPC-C and TPC-E fall short of representing production systems accurately. The demo provides valuable insights for the audience to spur research towards *real-world-optimized* database systems, especially for upcoming enterprise systems without materialized aggregates and OLxP workloads. We will include the audience's feedback into a publication, which analyses and quantifies the differences between synthetic and realistic systems in detail.

# 7. REFERENCES


[1] F. Funke, A. Kemper, and T. Neumann. Compacting transactional data in hybrid OLTP & OLAP databases. *PVLDB*, 5(11):1424–1435, 2012.

[2] J. Krueger et al. Fast updates on read-optimized databases using multi-core CPUs. *PVLDB*, 5(1):61–72, 2011.

[3] J. Lee et al. High-performance transaction processing in SAP HANA. *IEEE Data Eng. Bull.*, 36(2):28–33, 2013.

[4] H. Plattner. The impact of columnar in-memory databases on enterprise systems. *PVLDB*, 7(13):1722–1729, 2014.

[5] V. Raman et al. DB2 with BLU acceleration: So much more than just a column store. *PVLDB*, 6(11):1080–1091, 2013.

[6] L. Sun et al. Fine-grained partitioning for aggressive data skipping. In *Proc. ACM SIGMOD*, pages 1115–1126, 2014.

[7] E. Zamanian et al. Locality-aware partitioning in parallel database systems. In *Proc. ACM SIGMOD*, pages 17–30, 2015.