Statisticum: Data Statistics Management in SAP HANA

Anisoara Nica, Reza Sherkat, Mihnea Andrei, Xun Cheng Martin Heidel, Christian Bensberg, Heiko Gerwens SAP SE

firstname.lastname@sap.com

ABSTRACT

We introduce a new concept of leveraging traditional data statistics as dynamic data integrity constraints. These data statistics produce transient database constraints, which are valid as long as they can be proven to be consistent with the current data. We denote this type of data statistics by constraint data statistics, their properties needed for consistency checking by consistency metadata, and their implied integrity constraints by implied data statistics constraints (implied constraints for short). Implied constraints are valid integrity constraints which are powerful query optimization tools employed, just as traditional database constraints, in semantic query transformation (aka query reformulation), partition pruning, runtime optimization, and semi-join reduction, to name a few. To our knowledge, this is the first work introducing this novel and powerful concept of deriving implied integrity constraints from data statistics. We discuss theoretical aspects of the constraint data statistics concept and their integration into query processing. We present the current architecture of data statistics management in SAP HANA and detail how constraint data statistics are designed and integrated into this architecture. As an instantiation of this framework, we consider dynamic partition pruning for data aging scenarios. We discuss our current implementation for constraint data statistics objects in SAP HANA which can be used for dynamic partition pruning. We enumerate their properties and show how consistency checking for implied integrity constraints is supported in the data statistics architecture. Our experimental evaluations on the TPC-H benchmark and a real customer application confirm the effectiveness of the implied integrity constraints: (1) for 59% of TPC-H queries, constraint data statistics utilization results in pruning cold partitions and reducing memory consumption, and (2) we observe up to 3 orders of magnitude speed-up in query processing time, for a real customer running an S/4HANA application.

1. INTRODUCTION

Data statistics management in database systems has been carefully designed and integrated in many commercial sys-

Copyright 2017 VLDB Endowment 2150-8097/17/08.

tems as the absence of data statistics results in poor performance [6] while, on the other side, data statistics can be very expensive to create and maintain [11]. Selecting the most suitable data statistics depends on the current physical design, the current data, the current database workload, and, most importantly, on the current capabilities of the query optimizer and query execution engines to efficiently use data statistics. As many commercial DBMSs, SAP HANA - an in-memory columnar RDBMS - has extensive support for data statistics, which are used for statistical needs such as cardinality estimation. The system features a special DataStatistics component which manages data statistics objects, including the maintenance of these objects when data changes [22]. The DataStatistics component is designed to isolate the consumers of data statistics from the management and the storage of these objects. This design provides a common API, for answering statistical questions without the consumers being aware of which data statistics were used.

With increasing use of data aging¹ configurations and applications which are agnostic to how the data is stored, managed, and aged, there is a need to internally create and use special integrity constraints which capture the movement of data from hot to cold partitions, from column store to extended store in SAP HANA. These transient integrity constraints describe a snapshot of the database state such as data location and system topology. Such constraints can be extremely useful for efficient query processing by helping in restricting the access to cold data irrelevant to a query [10]. However, these dynamic integrity constraints can be extremely expensive to discover and maintain. Comparing the building and the maintenance of traditional data statistics to dynamic discovery of transient integrity constraints, we make the observation that the two are very similar in how the base data is processed for these objects, how they have to be built, maintained and stored in a compact way, and how they can be used in query processing.

With these observations in mind, we introduce in this paper traditional data statistics which also can be used as transient integrity constraints. In our implementation, we leverage the existing DataStatistics component's features, such as (1) the isolation of maintenance details from the consumers of DataStatistics services, (2) the persistence of the data statistics in a compact format in the catalog, (3) the support for distributed landscapes, and (4) the dynamical maintenance of these objects when data changes. We intro-

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.

Proceedings of the VLDB Endowment, Vol. 10, No. 12

¹Data aging in SAP HANA physically partitions data into one hot partition with recent data, and cold partition(s) with historical data (see "**Data aging**" in Section 2).

duce the theory behind the new concept of using traditional data statistics as dynamic data integrity constraints. These data statistics produce transient database constraints, which are valid as long as they can be proven to be consistent with the current data. We denote this type of data statistics by constraint data statistics, their properties needed for consistency checking by *consistency metadata*, and their implied integrity constraints by implied data statistics constraints (*implied constraints* for short). Implied constraints are valid integrity constraints which are powerful query optimization tools employed, just as traditional database constraints, in semantic query transformation (aka query reformulation), partition pruning, runtime optimizations, and semi-join reduction, to name a few. To our knowledge, this is the first work introducing this novel and powerful concept of implied integrity constraints from data statistics.

We demonstrate the usefulness of our proposed framework, considering practical use-cases in data aging scenarios. We design effective data statistics objects that: (1) integrate well into current SAP HANA's architecture; they rely on existing information for efficient runtime consistency verification, with no maintenance overhead. In this regard, an implicit passive mechanism is in place to maintain the consistency of our constraint data statistics objects, and (2) are compact objects and can be kept in memory with very small overhead to verify their consistency. This makes them desirable for efficient dynamic pruning.

Dynamic partition pruning based on constraint data statistics objects is complementary to static partition pruning. Static pruning, which uses the fixed partitioning scheme definition, can be applied when there is explicit reference(s) to the partitioning key columns. However, dynamic partition pruning can help to re-formulate query predicates which do not reference the partitioning columns. Dynamic pruning is particularly useful when there is a correlation between the partitioning key columns and the columns with constraint data statistics in query predicate. We show that constraint data statistics can be used beyond partition pruning, e.g. for semi-join reduction and runtime optimization using cached plans. Our experiments confirm that our constraint data statistics are powerful objects which can help to effectively prune cold partitions for 59% of queries in TPC-H benchmark, and can reduce query processing time up to 3 orders of magnitude for a real customer application.

1.1 Contribution

Our main contributions are presented as following:

- In Section 3, we present an architectural overview of SAP HANA's DataStatistics component; we describe the grand vision behind its design, as well as the services it provides.
- In Section 4, we introduce our novel type of data statistics objects, namely constraint data statistics, and present the theoretical framework of query optimization with constraint data statistics.
- In Section 5, we provide a use case for constraint data statistics, in a data aging scenario. We enumerate the requirements for an implied integrity constraint, and design one, seamlessly integrated into SAP HANA.
- In Section 6, we document an end-to-end experimental evaluation of our proposal, using both the standard TPC-H benchmak and a real customer's productive system in data aging configuration.

2. AN OVERVIEW OF SAP HANA

In this section, we present a brief overview of SAP HANA's in-memory column store, query processing, and data aging. The section provides the necessary background for the topics covered in this paper. The architecture of SAP HANA is described in full details in [14].

In-memory column store - In SAP HANA's column store, data is stored in a read-optimized main fragment and a write-optimized delta fragment. Inserting new rows or updating existing rows are both regarded as changes. Changes do not physically modify existing rows (i.e. no in-place update) but append new rows into the delta fragments. During the delta merge operation, all committed row changes from the delta fragment are moved into a newly constructed main fragment. Each query on a column is evaluated independently on the main and on the delta fragment of the column. The two result sets are united and returned, with some rows removed after applying proper row visibility rules. In the main fragment, a data vector is used per column to store which row position contains what value identifier from the column's dictionary. When searching a column for a value, an inverted index may help to speed-up the search of the corresponding rows in the data vector.

Query processing - Semantic query transformations (aka query reformulation) are applied in different stages of query processing. During query optimization, the rule based optimizer applies normalization steps to the relational algebra tree which is later used for cost-based optimization. The relational algebra tree contains relational operators such as projection, selection, join, and aggregations. Normalization transformations which can safely be applied for plans which will be cached - such as predicate transformation from disjunctive normal form to conjunctive normal form, tautology elimination, static partition elimination - are applied in this step of query optimization. Traditional database constraints, which are always consistent with any database state, are applied in query reformulation, similar to work described in [10], in this phase. Static partition elimination process for partitioned tables uses the normalized predicates to prescribe complete elimination of partitions which have no relevant rows for the query. After cost-based query optimization, which in general will cache the best plan, runtime optimizations are applied by different execution engines processing parts of the query plan. At runtime, new predicate normalization process is applied as values of the query parameters are now known; implied constraints, introduced in this paper, can be used at this stage for query reformulation. We denote the process of using some transient runtime information, such as implied integrity constraints, for partition elimination by dynamic partition pruning.

Data aging - The most-prominent way to avoid full data scan of a table is to create physically independent table partitions. This way, evaluating predicates on table partitions can benefit from server parallelism. Furthermore, it is possible to skip partitions irrelevant to a query (described shortly). With table partitioning in place, the WHERE clause of a query, after query reformulation using database integrity constraint, and predicate normalization, can be used to determine whether data from a partition can satisfy a query predicate condition. This can be achieved by query reformulation using the partition definition (i.e., the partitioning scheme). Partitions that are guaranteed – by their scheme definition – not to have any rows qualified

by the query predicate can be pruned, i.e. not loaded during query execution. This type of pruning is always correct as it is based on the static partitioning scheme definition. Often the partitioning scheme is chosen in a way that old data, which is rarely accessed, is stored in other partitions separate from frequently accessed data. This is often described with the metaphor of hot (for current) and cold (for old) data. Segmenting data into hot and cold partitions has the key advantage that smaller amount of data needs to be scanned and processed, when the majority of queries touch the most recent data. This is beneficial, as the cold data is often placed in containers with slower access time, which can get (partially) loaded into main memory on demand [25]. Unfortunately, general table layout does not foresee the direct implementation of data aging by range partitioning. As an example, a business object may be made up of two tables; Header and Lineitem. If only the Header table carries an age information and/or a status that indicates whether the business process is completed and the object is no longer required for normal processing, there is no option to properly employ partitioning techniques on the Lineitem table. This is why in S/4HANA, an artificial column has been added to the data model. Once a business object is identified to be no longer relevant for most OLTP processes, it gets closed by setting a business date into the artificial column of all related tables. Partitioning rules then ensure that the corresponding rows are being moved to the proper cold partitions.

Query processing on aged tables - The standard usage scenario of partition pruning in data aging configuration is limited to the predicates referring to the partitioning criterion. Without such predicates, partitions cannot be pruned by classic schema based partition pruning. Some applications may be able to explicitly provide the age in the query predicate, or at least derive relevant range for the age based on available business data. However, this information is not always evident from the query predicate. In this case, there are three possible ways to process the query. First, scan all table partitions and probe them against the query predicate; this is prohibitively expensive. Second, create an index on columns that are referenced in query predicates, and utilize this index to efficiently access data in each partition. While attractive, we do not opt for this solution; in SAP HANA each column index is stored with the corresponding column partition, meaning that this solution would also require loading of all column partitions². The third possible solution is to use constraint data statistics, defined per column partition, and perform consistency checking as described in Section 5.2, to avoid accessing irrelevant partition(s).

3. ARCHITECTURE OF DATA STATISTICS COMPONENT IN SAP HANA

Data statistics are handled by a dedicated DataStatistics component, that is responsible for building and maintaining all data statistics objects. This component is in charge of answering statistical questions posed by every database system component. For instance, the query optimizer uses DataStatistics API during query optimization while query execution engine uses it for runtime optimization such as dynamic partition pruning. DataStatistics answers to each statistical question by using the data statistics object(s) most relevant to that particular request. In SAP HANA, there are



Figure 1: DataStatistics component in SAP HANA

currently two categories of data statistics objects. The first category consists of the basic runtime data statistics objects. These objects are stored and maintained close to the data. The second category consists of objects which are created by DDLs; they are built and dynamically maintained by the DataStatistics component. These objects are treated as firstclass citizen database objects³. The main architecture of the DataStatistics component is depicted in Fig. 1. Each node in the distributed landscape has its own DataStatistics component which receives all statistical requests from local consumers. To fulfil each request, DataStatistics has direct access to the distributed metadata which stores information on the defined data statistics objects. This architecture has the primary goal of *isolating* the consumers of data statistics from the management aspects of these objects, which include dynamical maintenance of content when data changes, as well as life-cycle management⁴.

By design, all requests for statistical questions are sent to local DataStatistics component. Because of this, the architecture achieves another very important goal. The component provides seamless integration of the DataStatistics Adviser into SAP HANA's architecture [9]. The adviser, in an advisory mode, collects information on any type of statistical requests, and advises of missing data statistics, as well as unused or not useful ones. The DataStatistics Adviser is designed from a unique perspective that data statistics needs can be understood only based on the *current* implementation of database components consuming data statistics in their normal query processing. This implies that:

 Data statistics usefulness depends on the current physical database design (e.g., partitioning scheme, data location⁵).
 Data statistics usefulness depends on the current work-

load. A change in the workload properties may result in a dramatic change in what data statistics are useful.

3. Data statistics usefulness depends on the characteristics of the current data accessed by the workload.

4. Data statistics usefulness depends on the current implementation of the consumers of data statistics, i.e., the query optimizer and the execution engines.

 $^{3}\mathrm{They}$ have dedicated schema, unique object identifier, and are stored in database catalog.

⁴E.g. create, drop, alter, and refresh.

⁵SAP HANA supports multi-store tables which have some partitions stored in SAP IQ, and remote tables which are tables stored in a remote system.

 $^{^2\}mathrm{The}$ inverted index size is linear to the size of the column.

The last implication, immediately points to the extensibility of the DataStatistics Adviser architecture. For instance, when new statistical requests are triggered by new features added to the query optimizer (such as applying integrity constraints for query reformulation, or using constraint data statistics for partition pruning) the DataStatistics Adviser, without any changes, considers these requests for statistics recommendations. The constraint data statistics, introduced in this paper, were integrated into DataStatistics's architecture by adding support for:

1. A new type of request, to find implied integrity constraints from data statistics which are relevant to certain predicates. In the current implementation, these requests come from *runtime* optimization for dynamic partition pruning (Section 5.3). The DataStatistics component finds relevant constraint data statistics, and decides if their implied integrity constraints are consistent with the database state visible to the requesting transaction (Section 5.2).

2. New methods for building and maintaining constraint data statistics: when such an object is built, its implied integrity constraint must be consistent with the database state⁶. A consistency metadata is attached to the object which is used by DataStatistics component to decide the consistency of its implied integrity constraint when used for a request as described in (1). When the constraint data statistics become inconsistent with the current database state, the DataStatistics component dynamically, asynchronously, rebuilds them⁷.

3. New method for DataStatistics Adviser to recommend the creation of missing constraint data statistics based on this new type of statistical questions. This feature is essential to improve workload performance using constraint data statistics, as even for a simple workload and partition topology, defining useful constraint data statistics for dynamic partition pruning is inherently difficult for a human.

In the next section, we provide theoretical framework for constraint data statistics and their implied integrity constraints, and present how these are used by the query optimizer and execution engines during query processing.

4. QUERY OPTIMIZATION WITH CONSTRAINT DATA STATISTICS

In this section, we first introduce the novel concept of constraint data statistics objects (Section 4.1). We then show how these objects can be used during runtime query optimization (Section 4.2). Constraint data statistics objects can be integrated into our DataStatistics component. In Section 5, we provide a concrete instantiation of constraint data statistics objects, and its integration into DataStatistics component for dynamic partition pruning in data aging scenario.

4.1 Constraint Data Statistics

We define *constraint data statistics* by leveraging traditional data statistics which can be used to generate valid integrity constraints. These are data statistics objects which are *dynamically* maintained by the system when data changes. In particular, data statistics objects are augmented with additional consistency properties. These properties allow the system to *decide* if the data statistics object implies a valid integrity constraint consitent with the data visible to a current transaction. If it can be decided that an instance of a data statistics is *consistent* to the snapshot of the transaction, then its implied integrity constraint can be used, just like normal database constraints, in semantic query transformation (aka query reformulation [10]) and runtime optimization. This utilization can result in dynamic partition pruning, inference of new, more restrictive predicates, and semi-join reduction, to name a few. Many types of traditional data statistics can be seen as implying new, valid, integrity constraints. For example, a two bucket histogram built for the column A of table T as [10, 100, *payload*1], [200, 300, *payload*2] naturally implies the integrity constraint:

(T.A between 10 and 100) or (T.A between 200 and 300).

If such implied integrity constraint can be decided to be consistent with data visible to the current transaction, then it can be used, just as any other database constraint, for query reformulation [10]. Another example is the partitioned table with simple data statistics depicted in Table 1 in Section 5: their implied integrity constraints are as follows⁸:

iic_0 :	(l_shipdate > '1996-12-12' and Linitem(current)) or Lineitem $(1,, 5)$				
iic_1 :	(l_shipdate BETWEEN '1995-11-08' and '1996-12-12' and Lineitem(1)) or Linitem(current,2,,5)				

```
iic_2: (l_shipdate BETWEEN '1994-12-08'
and '1995-29-12' and
Linitem(2)) or Linitem(current,1,3, ...,5)
```

and likewise for iic_4 and iic_5 . Let a relevant predicate in a query be l_shipdate >= '1996-01-01'. Assume that it can be decided that all implied integrity constraints iic_0, \ldots, iic_5 are consistent with the database state visible to the query transaction. Under these conditions, we can apply query reformulation using the above implied integrity constraints resulting in the following predicate:

 $\label{eq:l_shipdate} = `1996-01-01' \mbox{ and Lineitem}(current,1,\ldots,5) \mbox{ and } ((l_shipdate>'1996-12-12' \mbox{ and Lineitem}(1,\ldots,5)) \mbox{ and } ((l_shipdate BETWEEN `1995-11-08' \mbox{ and } '1996-12-12' \mbox{ and Lineitem}(1)) \mbox{ or Linitem}(current,2,\ldots,5)) \mbox{ and } ((l_shipdate BETWEEN `1994-12-08') \mbox{ and } `1995-29-12' \mbox{ and Linitem}(2) \mbox{ or Linitem}(current,1,3,\ldots,5)) \mbox{ and } \ldots$

which, after predicate normalization, is equivalent to: (l_shipdate > '1996-12-12' and Linitem(current)) or (l_shipdate BETWEEN '1996-01-01' and '1996-12-12' and Lineitem(1)) . Hence, in this particular case, query reformulation using implied integrity constraints results in dynamic partition pruning of partitions 2, 3, 4, 5; moreover, the predicate for the unpruned partition 1 is a more restrictive predicate applicable only for partition 1.

The main challenges are related to building and maintaining constraint data statistics, and to make possible to decide if the implied integrity constraint is consistent with the database state for a given transaction consuming them. Our current architecture for supporting such data statistics is fully integrated in our DataStatistics component (Section 3). This provides a complete isolation between consumer of data

 $^{^6{\}rm For}$ example, data statistics built using sampling do not have this property.

⁷Note that these data statistics can be used as traditional data statistics objects regardless of their consistency.

⁸We use the notation $T(i_1,...)$ to denote the predicate 'the row belongs to either of the partitions $(i_1,...)$ '.

statistics and their maintenance, hence it is fully responsible for solving all the main challenges described above. To make possible this type of decisions, we identify the following properties we need to provide for a data statistics object: (1) when a data statistics object is built, the build process must see all the data, visible or not to the current transaction; (2) the implied integrity constraint is invariant to data deletes; (3) the implied integrity constraint can be made inconsistent only by data updates and data inserts; (4) the data statistics stores enough information on the data state when it was built: we denote this information as the consistency metadata of the data statistics object; (5) the consumer transaction must be able to provide information on its visible database state which, together with the consistency metadata of a data statistics, is used in the decision if the implied integrity constraint is consistent or not to the database state of the transaction: if the decision is yes, the implied integrity constraint can be used, just as any integrity constraint, for semantic query transformation.

As we explain in Section 5.2.2 in detail, for *consistency metadata* of a data statistics object, our current design uses a high-water mark for each physical partition, namely the maximum record identifier of the last row inserted in that particular partition. A logical data statistics object, defined on a single table, has a physical data statistics object for each partition of its data source table. The consistency metadata of a data statistics object is kept for each of its physical objects. It consists of the maximum record identifier (MaxRowID) of the partition the physical object is built on, at the time the physical object was last refreshed.

Hence, *constraint data statistics* is a traditional data statistics which have an implied integrity constraint that can be decided if it is consistent with a database state.

4.2 **Runtime Query Optimization**

One of the main goal of this work is to show the power of constraint data statistics for a specific runtime optimization, namely dynamic partition pruning. In theory, both static and dynamic partition pruning are methods of applying query reformulation [10] using conditions consistent with the data in some or all of the partitions. The goal is to use the rewritten predicates to prove that some of the partitions have no useful rows for the given query, hence achieving partition pruning. Practically, this type of query reformulation is applied if the query already contains predicates on relevant columns for which such conditions exist. For example, if the query has local predicates (e.g., equality with a constant, between predicates) on a column which is used in the range partitioning scheme, predicate reformulation using the partitioning scheme is used to achieve static partition pruning.

Example: The table T has the range partitioning scheme PARTITION BY RANGE (A) (PARTITION 1 <= VALUES < 5, PARTITION VALUE = 44, PARTITION OTHERS

resulting in three partitions for which the following constraints hold^9 :

 $c_1: (1 \le T.A \le 5 \text{ and } T(1)) \text{ or } T(2,3)$

 c_2 : (T.A = 44 and T(2)) or T(1,3)

 c_3 : (T.A <> 44 and (T.A < 1 or T.A >= 5) and T(3)) or T(1,2).

A query predicate p(T.A) on the column T.A can be rewritten as follows:

 $p(T.A) == p(T.A) \text{ and } T(1,2,3) \stackrel{using c_1, c_2, c_3}{=}$

p(T.A) and T(1,2,3) and

 $((1 \le T.A \le 5 \text{ and } T(1)) \text{ or } T(2,3))$ and

((T.A = 44 and T(2)) or T(1,3)) and

((T.A <> 44 and (T.A < 1 or T.A >= 5) and T(3)) or T(1,2))

For p(T.A) = T.A = 4', the above predicate is rewritten as (T.A = 4 and T(1)), hence achieving partition pruning for partitions (2,3); while for $p(T.A) = T.A \ge 44'$, the above predicate is rewritten as

(T.A = 44 and T(2)) or (T.A > 44 and T(3)), with partition (1) being pruned.

The constraint data statistics objects can be consumed for query reformulation by using their implied integrity constraints. If an implied integrity constraint can be proven to be consistent with the database state visible to the query transaction, then it can be used, just as any integrity constraint for query reformulation. For each partition T(i), for each column T.A, the implied integrity constraint is theoretically defined as: iic_{A_i} : $(p_{A_i}(T,A) \text{ and } T(i))$ or $T(1,\ldots,i-1,i+1,\ldots)$: the predicate $p_{A_i}(T,A)$ is defined using the current physical data statistics object which can be, for example, a range predicate if data statistics object is a simple min/max statistics, or can be a disjunction of range predicates for a data statistics object of type histogram. As an example, if the min/max statistics for l_shipdate, for partition 1 of table Lineitem is ('1995-11-08', '1996-12-12'), its implied integrity constraint is

 $iic_{\texttt{l.shipdate}_1}$: ('1995-11-08' <= <code>l.shipdate</code> < '1996-12-12'

and Lineitem(1)) or Lineitem(current, 2,3,4,5).

For a query execution process, the implied integrity constraints, once proven to be consistent to the data visible to the query transaction, can be used in these phases of the query processing:

- Query optimization, if the plan being optimized will not be cached: cached plans cannot use transient integrity constraints.
- Query optimization, if the plan being optimized will be cached as a parameterized plan: if the implied integrity constraints are used for a cached plan, the data statistics and its current consistency metadata can be a parameter to the cached plan: new transactions can use that parameterized plan, if an only if the stored consistency metadata can be use to prove the consistency of the implied integrity constraints.
- Query execution: as any optimization done during query executions are done only for the current plan, implied constraints can be used for any such optimization: these may include dynamic partition pruning, semi-join reductions using runtime values after a scan of a table to achieve dynamic partition pruning on the other join table.

5. CONSTRAINT DATA STATISTICS FOR DYNAMIC PARTITION PRUNING

In this section, we present a use case of constraint data statistics for dynamic partition pruning. We first motivate our design by presenting the requirements for constraint

⁹We use $T(i_1, i_2, ...)$ to denote rows which belong to either of the partitions $(i_1, i_2, ...)$ of table T

Table 1: Ranges of L_shipdate column for cold partitions of aged Lineitem table (details in Sec. 6.1)

	cold1	cold2	cold3	cold4	cold5
min	11Aug95	12Aug94	09Aug93	11Aug92	03Jan92
max	12Dec96	29Dec95	29Dec94	29Dec93	28Dec92

data statistics. For simplicity, we formulate this as improvement opportunities for critical data aging operations (Sec. 5.1). Then, we present our technical implementation of a constraint data statistics, and we describe how it is integrated into SAP HANA's architecture (Sec. 5.2). Finally, in Sec. 5.3, we overview how our proposed data statistics objects can be used during runtime query optimization.

5.1 Motivation: Data Aging in SAP HANA

We present two practical uses cases in this section. In data aging, each table is segmented into a hot partition and one or many cold partitions. The hot partition contains activity pertaining to recent (i.e. current) data. Each cold partition contains activities that are not recent, i.e. *closed* in data aging terminology. We present two practical operations and motivate that they both benefit from constraint data statistics. We carefully identify the requirements for such data statistics objects in Sec. 5.1.3, and show how we support them in SAP HANA (Sec. 5.2).

5.1.1 Query processing on cold partitions

Queries on aged tables may explicitly (or implicitly) provide hint(s) to query execution engine, to consider and/or skip selected partition(s) of an aged table. For instance, a hint could specify that only recent data, or data from designated cold partition are relevant for desired historical analvsis. Otherwise, every partition of the aged table needs to be loaded and examined; this has very large memory footprint and prohibitive CPU cost. One possibility to mitigate this cost is to build (inverted) index for any column anticipated to be queried. However, such indexes are stored with the column partition, meaning that probing the index would again trigger a load. Keeping the index in memory brings an extra overhead, as the size of an index is linear to the size of the corresponding column. It is very desirable to prune partitions by comparing query predicate against the constraint data statistics of each column partition. We use an example to illustrate this using the aged Lineitem table¹⁰ from the TPC-H benchmark and the query Q06 as

SELECT sum(l_extendedprice * l_discount) as revenue FROM Lineitem

```
WHERE l_shipdate >= to_date('1994-01-01')
AND l_discount between round((0.6-0,01),2)
and round((0.6+0,01),2) AND l_quantity < 24.</pre>
```

Table 1 contains an instance of the minimum/maximum statistics for l_shipdate column, for each cold partition. When evaluating Q06, these ranges can be leveraged to skip cold partitions cold4 and cold5. This is because of the predicate which references l_shipdate in the query is guaranteed not to have any candidate results in partitions cold4 and cold5. These two partitions have l_shipdate earlier than 29Dec93 and 28Dec92. In our current implementation, this pruning is achieved by creating constraint data statistics of type simple

for the column Lshipdate which contains the minimum and maximum values for each partition. The implied integrity constraint is used for dynamic partition elimination which, indeed, results in pruning partition cold4 and cold5.

5.1.2 Enforcing uniqueness constraint

Enforcing uniqueness of values in a column (or a set of columns) is an important requirement of many business applications. Enforcing uniqueness is challenging particularly in a data aging configuration. This is because for any OLTP operation that changes the data, one has to rigorously examine all partitions and verify that uniqueness would remain valid. One possible way to simplify uniqueness check is to augment the table's partitioning columns with the set of columns that are present in a uniqueness constraint. Using this technique, enforcing uniqueness becomes equivalent to verifying whether each update violates the uniqueness constraint in exactly one partition only, as opposed to several partitions. This approach, while significantly simplifying the problem, has the risk of violating business uniqueness semantics. This is because augmenting the primary key constraint with physical row placement columns opens the doors to having rows with identical primary key (uniqueness constraint) but different physical partitioning columns. As an alternative, we propose to verify uniqueness efficiently using constraint data statistics to the uniqueness checking for some partitions based on their implied integrity constraints. Instead of examining all partitions for each update, we propose to use first the constraint data statistics by checking the updated value against the implied integrity constraint. These objects are compact data structure and probing them must be quick. For instance, consider an insert operation. Let the constraint data statistics have the minimum m and the maximum M of a column, for a certain partition. If the inserted value is not inside the range [m, M], there is no need to examine that partition for uniqueness. Otherwise, the partition has to be examined for uniqueness violation(s).

5.1.3 Remarks and requirements

Although dynamic partition pruning may seem similar to classic schema-based partition pruning [24], it is quite different for two reasons. First, a table can only be physically partitioned by one set of primary $column(s)^{11}$. Because of this restriction, schema-based pruning can be applied when either (1) there is a predicate with conditions on the primany partitioning column, or (2) there is an opportunity for query re-formulation to learn the primary partitioning column(s). Second, dynamic partition pruning extends the possibility of pruning partitions; dynamic partition pruning can be applied as a complementary pruning step after schema-based partition pruning, to further reduce the number of partitions that are candidate for pruning, but not pruned by the schema based step. Such pruning becomes more fruitful, particularly when there is a correlation between column values and the partitioning key. However, dynamic partition pruning is useful only if the data statistics object is consistent with the database snapshot of the consumer transaction. Otherwise, using such data statistics for pruning may produce incorrect query results if a partition is pruned incorrectly. One approach to guarantee the consistency is to rigorously maintain data statistics objects values

¹⁰We explain the aging of the Lineitem table in Sec. 6.1.

¹¹Often an implied TEMPERATURE column consistent with business logic is selected in data aging applications.

per database snapshot. This solves the consistency problem, but is very expensive to manage. Instead, we advocate for a much simpler approach which fits well in the context of data aging applications. We create data statistics objects with an implied consistency metadata. In our design, we carefully consider the following important requirements for ideal consistency metadata:

- 1. Runtime verification of metadata consistency state for each data statistics object must be efficient,
- 2. Each snapshot can use data statistics objects having a metadata state consistent with the snapshot, and
- 3. No extra cost is associated with maintaining the consistency metadata of each data statistics object.

We present our implementation in the next section.

5.2 Implementation

We describe an instantiation of a constraint data statistics object. We use the term *synopsis* to refer to our instantiation. Each synopsis is constructed for a single column of a partitioned table¹². Each synopsis has a content (Sec. 5.2.1) and a consistency metadata (Sec. 5.2.2), which together ensure it satisfies the requirements we mentioned in Sec. 5.1.3, when integrated into the DataStatistics component (Sec. 5.2.3).

5.2.1 (Min,Max) as constraint data statistics

Let C be a column of the partitioned table T. A min/max data statistics is defined, for each partition T(p), as the pair (min(T(p).C), max(T(p).C)) which are the smallest and the largest values, respectively, that appear in the the main and the delta fragments of column C, of the partition T(p) assuming that every row in the partition T(p) is visible when minimum and maximum are computed. This content can be constructed in constant time by consulting the main dictionary and the delta dictionary of the column C, of the partition T(p). Note that we extract the content from both the main fragment and the delta fragment of each column partition. This, corresponds to the physical content of the column and does not filter out any row. Not including visibility rules in the construction of the synopsis content ensures that the synopsis can be consumed by any snapshot that applies filtering rule(s) over the same physical content of the column. We next formalize this property. Let $min(T(p),C,S_i)$ and $max(T(p),C,S_i)$ be, respectively, the minimum and the maximum values of rows in T(p). C that are visible under the snapshot S_i . Our construction of min/max synopsis guarantees that the range $[min(T(p).C, S_i), max(T(p).C, S_i)]$ is included in the range [min(T(p).C), max(T(p).C)] defined above. Therefore, if a transaction with snapshot S_i wants to perform dynamic partition pruning, e.g. based on the predicate T.C < value', it can safely use the range [min(T(p).C), max(T(p).C)] instead of the range $[min(T(p).C, S_i), max(T(p).C, S_i)]$, and skip partition T(p) if value < min(T(p).C). The advantage of using synopsis content over the current range visible to the snapshot S_i , i.e. $[min(T(p).C, S_i), max(T(p).C, S_i)]$, is that we do not need to maintain synopsis per snapshot. This is part of the second requirement mentioned in Sec. 5.1.3.

5.2.2 MaxRowID as consistency metadata

Key requirements for a good consistency metadata are that (1) it brings no extra cost to operations that modify column content, and (2) the verification of data statistics consistency with the current state of a column of a partition must have low overhead. For these purposes, we use MaxRowID for consistency metadata for each table partition. For each table partition T(p), MaxRowID (T(p), t) denotes the maximum of the largest row identifier in the main fragment and the delta fragment of T(p) at time t. When a row is inserted into a table partition, the MaxRowID of that partition is increased automatically in the current SAP HANA implementation. In this regard, MaxRowID represents table partition's state at no extra cost: it is an existing piece of information already managed by the system for each partition. The consistency check for a constraint data statistics is done as following. Assume that the synopsis is built, for the table partition T(p), at time t_i , with $r_i = \text{MaxRowID} (T(p), t_i)$. r_i is stored in the consistency metadata. Consider that a transaction \mathcal{T} requests to use the synopsis at time $t_j \geq t_i$. Assume that the *current* maximum rowid for the partition T(p)is $r_j = \text{MaxRowID}(T(p), t_j)$. If $r_i = r_j$, then the DataStatistics component decides that the synopsis content is consistent with the requesting transaction database state. Therefore, the implied integrity constraint can be used. However, if $r_i > r_i$ (note that r_i cannot be less than r_i as the maximum row id is monotonically increasing), it cannot be safely decided that the implied integrity constraint is consistent with the current database state. Therefore, the constraint data statistics cannot be used by the transaction \mathcal{T} . In this scenario, the DataStatistics component marks the data statistics as inconsistent and it will rebuild it (discussed in Sec. 5.2.3).

Using the MaxRowID for consistency metadata provides a very efficient consistency checking mechanism. Furthermore, MaxRowID has a low storage cost. Because MaxRowID is an information that is already maintained by the system, no extra cost is incurred in retrieving and using it. In each snapshot, the DataStatistics component can read snapshotindependent MaxRowID, and use it to determine the consistency based on the stored consistency metadata of a data statistics. Thus, our design addresses all the requirements we listed in Sec. 5.1.3, for the ideal consistency checking.

5.2.3 Integration into DataStatistics Component

We briefly describe how constraint data statistics are integrated into data statistics API, our simple data structures that use them, and the life-cycle of such statistics. In order to efficiently support requests for implied integrity constraints, the DataStatistics component locally caches a data statistics in the node which stores the column partition. Data statistics objects are persisted in the catalog which is accessible by each node in the SAP HANA landscape. When the content of a synopsis is modified (described shortly), we immediately update the local cache version and fire an asynchronous job to update the persistent image of the synopsis in the catalog. As described in Sec. 3, DataStatistics provides a new type of requests for the consumers of constraint data statistics, e.g., for dynamic partition pruning. Once DataStatistics receives such request, a look-up in the local cache is performed, with the key being a combination of the table identifier, the partition identifier, and the column identifier. If a synopsis is found with its MaxRowID equal to that of the table partition's *current* MaxRowID, the cached

¹²Synopsis can either be requested by a DDL statement, or be recommended by statistics adviser.

synopsis is used and declare success. Else, the DataStatistics component reads directly from the catalog¹³ and updates the local cache. Again, when the MaxRowID of the synopsis read from catalog matches with that of the table partition, this constraint data statistics can be used and declare success. Otherwise, this data statistics object is marked as inconsistent, and it has to be refreshed asynchronously. The content of each synopsis and its MaxRowID is modified in one of three possible ways. First, during the DDL statement, CREATE or REFRESH. Such statements can be also triggered by SAP HANA's data statistics adviser. Second, the during delta merge of a table partition, data statistics are re-built for every column of the partition that has data statistics defined in the catalog. Third, a daemon job can periodically look for synopsis that are marked as inconsistent by the DataStatistics component and re-build them using a background job. We deem this solution sufficiently good, as updates to cold partitions are rare by design in data aging configurations.

5.3 Integration with partition pruning

Traditionally, partition pruning used by query optimization can be categorized based on its usage of different query execution components [19]. Using local filter predicates, the pruning optimization can eliminate unnecessary partition scan of a single table. Using transitive closure of the local filter for a join table and the join predicates, the partition pruning can be extended to the other join table. Lastly, the partition pruning can be applied partitionwise based on matching partitioning schemes of the two join tables: it avoids unnecessary joins between different partitions. SAP HANA supports all three forms of partition pruning. Partition pruning can also be categorized based on the source of the constraint used for query reformulation. The traditional partitioning scheme allows for static partition pruning. The constraint data statistics objects (introduced in Sec. 5.2) can be consumed by dynamic partition pruning. The underlying optimization for dynamic partition pruning is similar to that for static partition pruning. Practically, they both compare query range predicates on a column against the constraint describing the data in a partition to find out whether they are disjoint. However, dynamic partition pruning differs from static partition pruning as it uses a transient constraint which describes the data at the query execution time. Its column, content of the constraint and the consistency of the content all come from the constraint data statistics object and its maintenance. It is reflected in its more dynamic nature of the integration with the execution plan. Dynamic partition pruning in SAP HANA is nevertheless integrated seamlessly with the existing static partition pruning, i.e., in query optimization and semi-join reduction. We next briefly review such integrations.

5.3.1 Query optimization

During query optimization, static partition pruning using literal filter predicates can eliminate partitions for any plan. All surviving partitions are further prepared with the static partition pruning information based on parameterized predicates. Dynamic pruning adds to this preparation with the extra pruning information based on both literal and parameterized predicates, and on the existing relevant constraint data statistics objects. Normally, the dynamic partition pruning is only prepared during the plan generation. The second phase – when the actual pruning optimization happens – is done during query plan execution. This allows the generated plan to be cached for future repeated executions. During each execution, dynamic pruning checks the consistency of the synopsis to decide whether it can still use the constraint. If the synopsis is still valid, its current content then participates in the actual pruning optimization, together with any runtime static partition pruning. If the plan is not cached, dynamic pruning can also happen during the query optimization. This provides an extra optimization opportunity as more types of query reformulation are applied during query optimization.

5.3.2 Semi-join reduction optimization

Constraint data statistics are also used in the semi-join reduction optimization which potentially can improve the performance of SAP HANA partition-wise join strategies. The semi-join optimization is a runtime optimization applied to the join operators. There are two slightly different types. The first type happens at the beginning of the plan execution. It uses the plan data such as the surviving operator based on the filter predicates and the existing consistent data statistics object, to further restrict the partition-wise joinable pairs. This can further avoid unnecessary join between partitions¹⁴. The second optimization waits for the first join table to finish its scan. Then, it uses the actual qualifying values for the columns used in the join predicates as the filter predicate (an interval) to check against the second join table for partition pruning using relevant constraint data statistics objects. The first type can in particular help equi-joins through primary key to foreign key equality predicates, when the foreign key can not be part of the partition columns. As long as primary keys are generated using an ever increasing counter across the partitions, the surviving primary key partitions can fully eliminate the non-joinable foreign key partitions.

6. EXPERIMENTAL EVALUATIONS

We present an experimental study of dynamic partition pruning in a data aging configuration. We evaluate the usefulness of dynamic partition pruning for two different workloads: (1) the TPC-H benchmark ¹⁵ on aged tables described in Section 6.1, and (2) a running SAP ERP system.

6.1 Datasets and Metrics

Dataset I (aged TPC-H) - We used the data and the queries from the standard TPC-H benchmark, configured in a data aging scenario. We simultaneously aged the Orders and the Lineitem tables, using the configuration similar to data aging performed in a typical SAP S/4HANA ERP system. For each entity, the activities that pertain to each calendar year are stored in one physical partition. This was achieved by performing aging runs on tables. To age the Orders and the Lineitem tables, we first added a new column named TEMPERATURE to both tables. To populate the TEMPERATURE column in Lineitem, we first grouped rows in Lineitem by the Lorderkey column. For each order key

¹³E.g. Another thread updated the data statistics after we read it from the local cache, or we cannot find the relevant synopsis in the cache after a database restart.

 $^{^{14}\}mathrm{E.g.}$ see the analysis of Q04 of TPC-H in Sec. 6.2

¹⁵http://www.tpc.org/tpch/

group, we extracted the maximum of three date columns {l_shipdate, l_receiptdate, l_orderdate} for every row in the same order key group. This maximum date represents the activity date of the latest line item corresponding to an order. For all line items in each order key group, we set the TEMPERATURE column equal to the maximum date from the corresponding order key group. Next, we set the TEMPERA-TURE attribute of each row in the Orders table equal to the TEMPERATURE of each line item row that refers to the order by its primary key. Because the line items having the same order key get the same TEMPERATURE by our construction, each row in the Orders table received one date value for the TEMPERATURE column. By our construction, both Lineitem and Orders tables have common agreement on the activity of each transaction: all orders and line items in each transaction have the same TEMPERATURE. This resembles the business logic of a typical SAP S/4HANA ERP system. We aged the Orders table, and similarly the Lineitem table, by range partitioning each table on the TEMPERATURE column. This produced 7 physical table partitions for each aged table. The current (hot) partition stored orders from the last two years (1997 and 1998). The 6 cold partitions, each contained orders for one calendar year (from 1996 to 1992 and beyond). We stored every column of each cold partition using SAP HANA's page loadable column class [25]. We report experiment results for SF 0.1. Furthermore, we report experiments with larger dataset on real production system.

Dataset II (real production system) - This is a system setup for a SAP S/4HANA financial application. There are three aging tables in this system, with 598M, 38M, and 151M rows. Queries are from a typical financial application. The cold partitions of aged tables were stored using page loadable column class [25]. Total memory size consumed by this workload was 10 Gigabytes, and all experiments were conducted on a dedicated system, running no workload but our tests, when the timing measurements were collected.

Constraint Data Statistics - We created constraint data statistics designated to be used for dynamic partition pruning purpose, for each column that appeared in at least one predicate of the queries in each workload. The construction of each of these data statistics objects takes constant time per column in each partition; it only requires to access the main and the delta dictionaries of each column in order to get the minimum and the maximum values that appear per column partition. As the TPC-H benchmark does not have updates, the statistics we constructed remain valid in our experiment. For Dataset I., we constructed 15 data statistics objects for the aged Lineitem and Orders tables.

Measures - Two measures are of our interest. First, the number of partitions pruned per query by a valid constraint data statistics. Larger number of pruned partitions is more desirable and it translates directly into better memory utilization. The largest number of this measure for Dataset I is 6 per aged table (i.e. the total number of cold partitions). Second, we report the query runtime when constraint data statistics are used for partition pruning, over the runtime of the same query *without* dynamic pruning. We expect no regression when there is no pruning , i.e. the ratio is equal to one. A ratio smaller than one is observed when partition(s) are pruned. For each query, we report the average runtime ratio for 200 runs. In each run we first cleared the plan cache and unloaded the tables referenced in the query. As we do not re-write the queries to use TEMPERATURE columns, thus



Figure 2: Number of pruned partitions in Lineitem & Orders tables, for 22 TPC-H queries

relative query execution time



Figure 3: Avg. runtime ratio (query processing time with pruning over processing time without pruning) for 22 TPC-H queries

no static partition pruning is performed in our experiments.

6.2 TPC-H Results

Fig. 2 demonstrates that for 13 queries (out of total 22 queries), dynamic pruning could prevent loading of at least one partition from either the Lineitem or the Orders tables. After examining the constraint data statistics usage (we provide examples for two queries later in this section), we noticed that only the data statistics defined on Lshipdate, 1_commitdate, 1_receiptdate, and o_orderdate columns were the ones that were successfully used for dynamic partition pruning. Interestingly, these four columns are all correlated with the TEMPERATURE column, however, TEMPERATURE does not appear explicitly in any of the 22 queries. This demonstrates that, while the aged tables are physically partitioned, the latent semantic correlation between TEMPERATURE and the other four date columns has provided an effective mechanism that offer pruning without explicitly mentioning the primary partitioning column, which otherwise would benefit from static schema-based pruning. Despite pruning cold partitions for 59% of TPC-H queries, we did not observe endto-end performance improvements (Fig. 3). In most cases, the running time with partition pruning was identical to that without runtime pruning. To figure out the reason behind this, we profiled all queries that could benefit from partition pruning. We present the summary of our analysis of query execution for two representative queries: Q04 and Q06 in Fig. 4. Our optimizer assigns each of these two queries to be executed by two different SAP HANA execution engines; Q04 by Join Engine and Q06 by OLAP Engine. Without dynamic partition pruning, the SAP HANA's semijoin reduction plan executes as illustrated in Fig. 4a. The predicate conditioned on the order date is pushed down to JEEvalPrecond layer, and is evaluated in parallel on every 7 partition of the aged Orders table. With the consistent data statistics objects defined on the Orders table and considering the joining columns, the actual execution of the semi-join is further improved (Fig. 4c). On the leaf scan level, the filter condition pruning is able to eliminate the loading of all but 3 partitions. Furthermore, the dynamic pruning achieves the optimal partition-



Figure 4: Execution plans for two TPC-H queries (Q04 and Q06) with and without dynamic partition pruning

wise join between JEStep1 and JEStep2 by eliminating all but one pair from any source partition¹⁶. However, because of SAP HANA's inter-partition parallelism, the end-to-end execution time of JEEvalPrecond step is dominated by the execution time of the most expensive access to aged Orders table. The most expensive access, however, is common to both Fig. 4a and Fig. 4c and corresponds to accessing the cold partition which returned 4,215 rows. Because of this parallelism, no improvement were observed in end-to-end runtime as a result of partition pruning for Q04, even though it skipped 4 cold partitions (marked with * in Fig. 4a) and reduced overall memory consumption by 25%. For Q06, as demonstrated in Fig. 4b,d, we observe that OLAP Engine's preliminary step can benefit from partition pruning to skip 4 partitions of aged Lineitem. It reduced memory consumption by 21%. However, there are two factors that prevent runtime reduction here. First, similar to the case for Q04, SAP HANA's inter-partition parallelism masks runtime saving by pruned partitions (Fig. 4b) because the runtime for reading two cold partitions dominate the BwPopSearch step. Second, for this query, the parallel aggregation step for cold

partitions dominate the end-to-end running time of Q06. Thus, improvement in BwPopSearch step was not observed in query execution time, even though both BwPopSearch and BwPopAggregateParallel steps benefited from dynamic partition pruning.

6.3 Production System Results

Fig. 5 shows the runtime of a set of representative queries on a large SAP S/4HANA Central Finance system, with and without constraint data statistics. The queries access the same data from the perspective of different applications e.g., general ledger, asset management or controlling, and for different purposes, e.g., point and analytical queries. Each of the queries includes a time-based restriction (e.g. year or posting date) which is correlated to the partitioning field (TEMPERATURE). The content of the partitioning field is populated by the data aging process for financial accounting documents¹⁷. For the fields used as a time based restriction, a corresponding constraint data statistics object was created. We observe that for a large number of queries, dynamic partition pruning significantly improved the query execution time, from 2 to 3 orders of magnitude, especially in

¹⁶The 1 row connection between JEStep1 and JEStep2 here indicates that no actual join happened between the pair.

¹⁷http://help-legacy.sap.com/sfin



Figure 5: Impact of using dynamic partition pruning on the query execution time of a real system. Horizontal axis: queries, vertical axis: runtime (ms).

situations where none of the cold partitions were loaded into memory. For the queries with no performance improvement, however, we do not observe any regression. After careful investigation, we learned that the savings come directly from pruning access to cold partitions. On such a large system, this pruning contributes to significant performance runtime improvement and memory reduction.

6.4 Remarks and Discussions

We anticipate that the end-to-end running time could be significantly improved in three cases, at least. First, if the query explicitly asks to access only the current hot partition. This scenario happens frequently, when analysis is performed on hot data. In this case, cold partitions get pruned *implicitly*, i.e. without a predicate in the query which refers to the TEMPERATURE column. Second, when the database server is heavily loaded and SAP HANA decides to serialize execution, rather than using inter-partition parallelism. In this case, the impact of pruning cold partitions becomes immediately visible in the end-to-end running time. Finally, there are valid configurations in which the cold partitions reside on remote systems or on extended storage. In such configurations, given the fact that data statistics objects are available locally and in transient data structures, partition pruning can achieve significant performance improvement by reducing communication to remote system, to push result sets for join or to fetch results.

7. RELATED WORK

Partitioning is a well established database performance optimization technique [1, 4, 7]. Skipping data not relevant to a query can be done using Small Materialized Aggregates [21], zone map filters [15], B+Tree index structures [2], Quadtree-like index structures [20], and features extracted from data and workload [26, 27]. Data pruning has been implemented at different granularity levels (e.g. column block and table partition) and it has been deployed in many commercial systems and large scale big data scenarios [8, 17, 23]. Aside from horizontal and vertical table partitioning, hybrid segmentation has been proposed in the literature to improve the performance of in-memory databases depending on workload [16, 18]. For instance, Grund et al. [16] propose to horizontally partition a selective set of columns accessed mostly by OLAP-style queries, and vertically segment a set of columns that are mostly involved in OLTP workload.

Our work on using data statistics objects for dynamic partition pruning is related to the Small Materialized Aggregates (SMA) [21]. Each SMA contains aggregate values extracted from a bucket of tuples. SMAs can be used to evaluate a predicate or to compute aggregate values. SMAs can be constructed during column load [15]. We study two aspects in dynamic partition pruning, which were not studied before. First, the management of our data statistics objects can be performed efficiently with no overhead on the OLTP performance. This is because we rely on automatic maintenance of MaxRowId, which is readily available at query processing time, for verifying consistency. Second, even though a data statistics object is not completely consistent with the database state at a given time, the corresponding content can be used whenever small error in precision can be tolerated, e.g. during query optimization. We also enumerate other possible use cases of constraint data statistics objects, apart from dynamic pruning.

The recent work related to defining new types of integrity constraints [12, 13], such as matching dependencies and conditional functional dependencies, is very important to our work: our data statistics structures are used in defining implied integrity constraints, which we envision, in the future work, will be these complex new types of database constraints. Hence, work related to how these new types are used in query reformulation [10] is of most importance for this work. Moreover, adding support for matching dependencies and conditional functional dependencies in the query optimizer for query reformulation, will immediately extend the usefulness of our constraint data statistics for a larger class of queries. Beside the traditional data dependencies defined by user as part of the database schema, research work such as [3] addresses the problem of dynamically discovering interesting data dependencies on an instance of the database, and use those in the query optimization. Such constraints are built by first identifying interesting columns which may be related, and then use the values to generate integrity constraints. In contrast, our work relies on traditional data statistics objects already managed by the database system, and extends their usage, when possible, as sources of dynamically defined data dependencies. Our approach is based on the fact that data statistics objects are defined for interesting columns which are used in the WHERE clause. Hence, such columns are already the prime candidates to benefit from the existence of integrity constraints.

The authors of [5] define the concept of data integrity constraints which are applicable to a single sandbox and are visible and relevant to the applications running in the same sandbox. The custom integrity constraints, defined by the user, are allowed to be inconsistent with the data, however the applications cannot access the data but in the states where these integrity constraints are valid. For this purpose, the system must be able to decide if a data state is consistent to a set of custom integrity constraints defined for a certain sandbox. In contrast, in this paper, we introduce implied integrity constraints which are dynamically defined from physical data statistics objects, for certain types of traditional data statistics. Implied integrity constraints are transient database constraints, redefined any time the implicant data statistics are rebuild; their consistency with a database state of a current transaction can be decided by the definition of the data statistics object and the consistency metadata attached to each physical data statistics object.

8. CONCLUSION

We presented the architecture of the unified statistics management component in SAP HANA. We emphasized the grand vision of this architecture; an extensible component that can answer any data statistics requests. Our data statistics adviser can monitor all data statistics requests, and can suggest creating missing relevant statistics, as well as refreshing existing but stale ones, and dropping those less needed. We proposed a new theoretical concept, which we name it *implied data statistics constraint*. We formalized these constraints as a class of integrity constraints which are valid as long as they can be proven to be consistent with the current database state. This offers powerful query optimization tools, which can be utilized - like any traditional database constraint - for semantic query transformation, partition pruning, runtime optimization, and semi-join reduction, among others. We presented the implementation of our constraint data statistics objects, and demonstrated its seamless integration for dynamic partition pruning. Our implementation brings zero maintenance overhead; constraint consistency can be verified at *runtime* by piggybacking on existing piece of metadata maintained at table partition level. Our experiments showcase the impact of (1) our new theoretical concept, and (2) our technical implementation integrated in our unified and overarching architechture; 59% of TPC-H queries skip loading cold partition(s), and queries on a large production system run up-to 3 orders of magnitude faster. We plan to investigate alternatives for consistency metadata in scale-out setting.

9. ACKNOWLEDGMENTS

We thank the anonymous reviewers for the comments and suggestions, which contributed to improving significantly the presentation of this paper. We give special thanks to all colleagues and contributors to the Data Statistics project.

10. REFERENCES

- S. Agrawal, V. Narasayya, and B. Yang. Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. In ACM SIGMOD, 2004.
- [2] K. Alexiou, D. Kossmann, and P. Larson. Adaptive Range Filters for Cold Data: Avoiding Trips to Siberia. *PVLDB*, 6(14), 2013.
- [3] P. G. Brown and P. J. Haas. BHUNT: Automatic Discovery of Fuzzy Algebraic Constraints in Relational Data. In *VLDB*, 2003.
- [4] S. N. S. Ceri, G. Wiederhold, and J. Dou. Vertical Partitioning Algorithms for Database Design. *TODS*, 9(4), 1984.
- [5] B. Chandramouli, J. Gehrke, J. Goldstein,
 D. Hoffmann, Moritz Kossmann, J. Levandoski,
 R. Marroquin, and W. Xie. READY: Completeness is in the Eye of the Beholder. In *CIDR*, 2017.
- [6] S. Chaudhuri and V. Narasayya. Automating Statistics Management for Query Optimizers. In *IEEE ICDE*, 2000.
- [7] C. Curino, E. Jones, Y. Zhang, and S. Madden. Schism: A Workload-driven Approach to Database Replication and Partitioning. *PVLDB*, 2010.
- [8] B. Dageville et al. The Snowflake Elastic Data Warehouse. In ACM SIGMOD, 2016.

- [9] A. D'Costa, W. Gordon, L. Linta, and S. Wang. Data Statistics Adviser in Database Management Systems. In ACM SIGMOD, 2017.
- [10] A. Deutsch, L. Popa, and V. Tannen. Query Reformulation with Constraints. ACM SIGMOD Record, 35(1), 2006.
- [11] A. El-Helw, I. Ilyas, W. Lau, V. Markl, and C. Zuzarte. Collecting and Maintaining Just-in-time statistics. In *IEEE ICDE*, 2007.
- [12] W. Fan. Dependencies Revisited for Improving Data Quality. In ACM PODS, 2008.
- [13] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering Conditional Functional Dependencies. *IEEE Trans. Knowl. Data Eng.*, 23(5), 2011.
- [14] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA Database – An Architecture Overview. *IEEE Data Engineering Bulletin*, 35(1):28–33, 2012.
- [15] G. Graefe. Fast Loads and Fast Queries. In DaWaK, 2009.
- [16] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. HYRISE: A Main Memory Hybrid Storage Engine. *PVLDB*, 4(1), 2010.
- [17] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, and V. Srinivasan. Amazon Redshift and the Case for Simpler Data Warehouses. In ACM SIGMOD, 2015.
- [18] R. Hankins and J. M. Patel. Data Morphing: An Adaptive, Cache-conscious Storage Technique. In *VLDB*, 2003.
- [19] H. Herodotou, N. Borisov, and S. Babu. Query Optimization Techniques for Partitioned Tables. In ACM SIGMOD, 2011.
- [20] L. Lins, J. Klosowski, T. J. T., and C. Scheidegger. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12), 2013.
- [21] G. Moerkotte. Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing. In VLDB, 1998.
- [22] G. Moerkotte, D. DeHaan, N. May, A. Nica, and A. Boehm. Exploiting Ordered Dictionaries to Efficiently Construct Histograms with q-error Guarantees. In ACM SIGMOD, 2014.
- [23] P. Pedreira, C. Croswhite, and L. Bona. Cubrick: Indexing Millions of Records Per Second for Interactive Analytics. *PVLDB*, 9(13), 2016.
- [24] E. Rahm. Towards Large-Scale Schema and Ontology Matching. In Schema Matching and Mapping. Springer, 2011.
- [25] R. Sherkat, C. Florendo, M. Andrei, A. Goel, A. Nica, P. Bumbulis, I. Schreter, G. Radestock, C. Bensberg, D. Booss, and H. Gerwens. Page As You Go: Piecewise Columnar Access In SAP HANA. In ACM SIGMOD, 2016.
- [26] L. Sun, M. J. Franklin, S. Krishnan, and R. S. Xin. Fine-grained Partitioning for Aggressive Data Skipping. In ACM SIGMOD, 2014.
- [27] L. Sun, M. J. Franklin, J. Wang, and E. Wu. Skipping-oriented Partitioning for Columnar Layouts. *PVLDB*, 10(4), 2016.