IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Synopsis-alloyed Index for Exact and Approximate Query Processing

MIKA TAKATA[1], HIROKI YUASA [1,2], and KAZUO GODA[1] (Member, IEEE).
[1]The University of Tokyo, Tokyo, Japan (e-mail: mtakata,yuasa,kgoda@tkl.iis.u-tokyo.ac.jp)
[2]Currently works for Koei Tecmo Games.

Corresponding author: Mika Takata (e-mail: mtakata@tkl.iis.u-tokyo.ac.jp).

This work was in part supported by JSPS Grant-in-Aid for Scientific Research (B) JP20H04191 and Cross-ministerial Strategic Innovation Promotion Program (SIP) on Integrated Health Care System.

ABSTRACT Structured database systems, which organize business data, widely adopt data structures like a B^+-tree to speed up selective queries. However, users in many analytical scenarios increasingly value query speed and are satisfied with high-quality approximate answers. To bridge this gap, this paper introduces the Synopsis-alloyed Index (SAI), a novel data structure that incorporates synopsis (i.e., summary or abstraction information of concerned table records) directly into existing index structures. We design two query methods, SAS and SAS+, which exploit the embedded synopsis to return exact or approximate results more efficiently than traditional approaches. To demonstrate practical viability, we realize SAI within PostgreSQL through a Foreign Data Wrapper (FDW) prototype. Experiments show that our approach achieves up to 97.9% faster performance on approximate queries compared with baseline PostgreSQL when the synopsis satisfies the query constraints and required derivative attributes.

INDEX TERMS Synopsis, Data structure, Approximate Query Processing

I. INTRODUCTION

Structured database systems organize business data to efficiently process business queries. The B^+-tree [1], [2] has long been a standard index structure, reducing search space and improving query performance under selective constraints. A B^+-tree consists of two different types of nodes: leaf nodes, which store table records or index key entries in the designated key order, and internal nodes, which hold navigation information to the leaf nodes. The navigation information reduces the search space based on a query constraint, improving query processing efficiency, particularly when the query constraints are substantially selective, along with the designed key attributes. However, in many analytical scenarios, such as interactive data exploration or real-time dashboard visualization, obtaining a fast, approximate answer is often more valuable than waiting for a precise, exact answer. This demand has led to the integration of Approximate Query Processing (AQP) techniques into traditional database systems.

This paper proposes the *Synopsis-alloyed Index* (SAI), incorporating synopsis, i.e., summary or abstraction information of concerned table records, into an existing data structure such as B^+-tree. By embedding the synopsis in internal nodes, SAI enables efficient retrieval of exact or approximate values without scanning all records.

Our preliminary work [3] introduced a design of a synopsis-alloyed B^+-tree data structure, along with synopsis-alloyed search methods (SAS) and synopsis-alloyed search plus considering inter-attribute correlation (SAS+). These search methods leverage the synopsis of the synopsis-alloyed tree index when it satisfies the query constraints and selective attributes, thereby preventing access to all the concerned table records. This paper extends the preliminary work and presents an integration of the search methods into an existing relational database system, PostgreSQL [4] using PostgreSQL's foreign data wrapper functions. Compared with native PostgreSQL, our prototype achieves up to a 97.9% speedup when the synopsis satisfies the query constraints and derivative attributes. Furthermore, the prototype-based experiments on a real dataset demonstrates that the proposed synopsis-alloyed search methods achieve a 65.2% speedup over native PostgreSQL.

The rest of the paper is structured as follows. Section II reviews the related work. Section III presents the synopsis-alloyed index, and Section IV presents query processing techniques using the synopsis-alloyed index. Section V describes their prototype implementation on an existing database engine, and Section VI reports the performance experiments.

Finally, Section VIII concludes the paper.

II. RELATED WORK

Analytical query processing has long been studied to improve performance in database systems. Traditional approaches include index structures such as B^+-tree [1], [2], which reduce search space for selective queries, and filtering data structures such as Bloom filter [5], Cuckoo filter [6], XOR filter [7] and other extended filters [8], [9]. Materialized views [10] also accelerate query processing by storing precomputed results, though they incur storage and maintenance overhead as data grows. In addition, many database systems collect statistical profiles such as histograms on single or multiple attributes and exploit them in query optimizers to generate effective access paths [11], [12]. These statistics-driven techniques improve query planning efficiency but still return exact results, which may lead to overhead as data size increases.

To address scenarios where fast, approximate answers are sufficient, AQP has been widely explored. AQP methods are typically classified into two categories: online processing and offline approaches. Online processing methods compute and estimate aggregate answers at query time using sampled data [13], [14]. Typical approaches refine accuracy by iteratively improving estimates across interactive queries [15]. The main advantage of online processing is that it avoids preprocessing and storage overhead for precomputed values; however, query latency may increase due to sampling operations performed during execution.

Offline approaches precompute and store summary information, such as statistical profiles, to provide approximate answers at query time [16], [17]. For example, wavelet-based methods compress data into synopses to reduce search space [18], [19], while histogram-based techniques capture dataset distributions for efficient approximation [20]. Offline approaches often achieve lower computation costs and higher accuracy compared with online processing, but they introduce additional overhead for maintaining precomputed information.

By considering both advantages and disadvantages of online and offline approaches, BlinkDB [21] has proved the massively parallel engines on large volumes of data by comparison with other existing AQP approaches by adopting an online sample selection strategy and multi-dimensional stratified sampling based on the workload history to obtain highly accurate approximate answers. Similarly, [22] proposed AQP++, which combines the synopsis and sampling to return an approximate answer of the aggregate values with an interactive response time. To improve the accuracy of approximate value estimation, many sampling methods have been presented [23]–[26]. A proposed method by [27] achieved high accuracy by determining whether to use online processing or sampled data for AQP depending on the query. This research extends operational efficiency by optimizing the maintenance cost [28]. By applying the AQP for more complex datasets, PairwiseHist [23], ShadowAQP [24] have considered the application for the larger datasets. SAQE [29] considering the federation of the distributed data. Although these approaches show high accuracy, they have not considered enough integration into existing database systems.

Recent studies have also explored AQP in distributed and cloud-native environments, such as Spark SQL [30] and Presto/Trino [31], where synopsis and sampling techniques are applied at scale to interactive analytics. Commercial systems like Google BigQuery [32], Amazon Redshift Spectrum [33], and Snowflake [34] incorporate approximate query functions to handle massive datasets efficiently. Middleware approaches such as Aqua [35], IDEA [36], and VerdictDB [37] extend these distributed engines by rewriting queries to exploit precomputed synopses or stratified samples. While these systems highlight responsiveness and accuracy at scale, systematic research on AQP in distributed environments remains relatively limited, and most solutions assume either a distributed execution framework or an additional middleware layer.

In contrast, our work focuses on direct, non-invasive integration into widely adopted relational database systems such as PostgreSQL, embedding the synopsis into index structures to achieve both responsiveness and accuracy with low storage overhead without requiring specialized infrastructure.

III. SYNOPSIS-ALLOYED INDEX

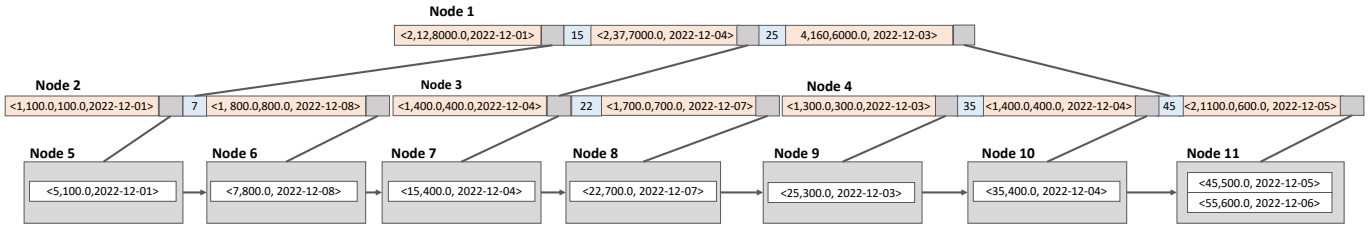
This section introduces a *synopsis-alloyed* index (SAI), incorporating the synopsis, i.e., summary or abstraction information of concerned table records, into an existing data structure. We take B^+-tree, for instance, but the same idea can be extended to other data structures.

B^+-tree [1], [2] index is a well-known data structure used in relational database systems to speed up the query processing. According to the standard design, a B^+-tree structure is composed of multiple nodes (i.e., fixed-length pages) that are structured in a balanced tree form. Table records are stored only in the leaf nodes in the sorted order of the designated index key attribute. Internal nodes hold one or more designated index keys and pointers to navigate to their descendant nodes. Starting at the root (internal top) node, the search query processing recursively visits only those nodes that satisfy the query constraints, following the pointers until fetching all the table records that satisfy the query constraints at the leaf nodes.

The SAI extends this structural design to allow each internal node to incorporate the synopsis regarding its descendant nodes. Figure 1(a) shows an example of the relational table that illustrates basic information about customer orders in a product line with attributes such as ORDERID for order ID, VALUE for product price per order, and SHIPDATE for shipping date. Figure 1(b) illustrates a synopsis-alloyed B^+-tree index where internal nodes (Node 1 to 4) hold a designated index key of ORDERID, pointers referencing its descendant nodes, and four types of synopsis information: (1) count of ORDERID, (2) sum of VALUE, (3) maximum of VALUE, and (4) minimum of SHIPDATE of its descendant

ORDERID	VALUE	SHIPDATE
5	100.0	2022-12-01
7	800.0	2022-12-08
15	400.0	2022-12-02
22	700.0	2022-12-07
25	300.0	2022-12-03
35	400.0	2022-12-04
45	500.0	2022-12-05
55	600.0	2022-12-06

(a) Relational table



(b) Synopsis-alloyed B$^+$-tree

FIGURE 1: Example of synopsis-alloyed B$^+$-tree. The B$^+$-tree data structure is composed of Nodes 1 to 11. The internal nodes, Node 1 to 4, contain the synopsis (colored in orange) as well as separation values of ORDERID for the given relational table (colored in blue) and references to its descendant nodes.

nodes, whereas the leaf nodes (Node 5 to 11) store the table records.

IV. EXACT AND APPROXIMATE QUERY PROCESSING USING SYNOPSIS-ALLOYED INDEX

This paper proposes efficient search algorithms using the SAI data structure to return an exact or approximate answer to a given query. Subsection A presents a search algorithm using an SAI to return an exact answer for a query constraint. Subsection B illustrates an inter-attribute result composition technique to return an approximate answer for composite query constraints under the assumption of inter-attribute independence. Subsection C extends the search algorithm to return an approximate answer to composite query constraints in cases where independence between attributes is invalid by considering inter-attribute correlation.

A. EXACT SEARCH METHOD EXPLOITING SYNOPSIS-ALLOYED INDEX

First, we present a search algorithm for exact query processing that utilizes the synopsis-alloyed B$^+$-tree index to improve the processing efficiency, called synopsis-alloyed search (SAS). SAS uses the synopsis embedded in internal nodes to reduce unnecessary data access to leaf nodes in traditional depth-first index searches, thereby improving data retrieval efficiency.

Algorithm 1 illustrates the pseudo-code of synopsis-alloyed search (SAS) algorithm for exact query processing, which utilizes synopsis-alloyed B$^+$-tree to improve the efficiency of query processing by extending the standard depth-first search on B$^+$-tree. Starting at the root node, the standard depth-first search recursively examines the descendant nodes if the designated index key satisfies the query constraints until fetching the matched table records stored in leaf nodes. In contrast, SAS keeps statistical values on child nodes that

Algorithm 1 synopsis-alloyed search (SAS)

Require: n_r: a reference to the root node
Ensure: R: result buffer

```
1:  R = ∅
2:  SAS(n_r)
3:  function SAS(n)
4:      N ←Fetch n
5:      if N is an internal node then
6:          for each e ∈ N do
7:              if e satisfies query constraint then
8:                  if e contains synopsis sufficient for
9:                      a disjoint part of query then
10:                     R ← R ∪ e
11:                 else SAS (e's child reference)
12:                 end if
13:             end if
14:         end for
15:     else
16:         for each e ∈ N do
17:             if e satisfies query constraint then
18:                 R ← R ∪ e
19:             end if
20:         end for
21:     end if
22: end function
```

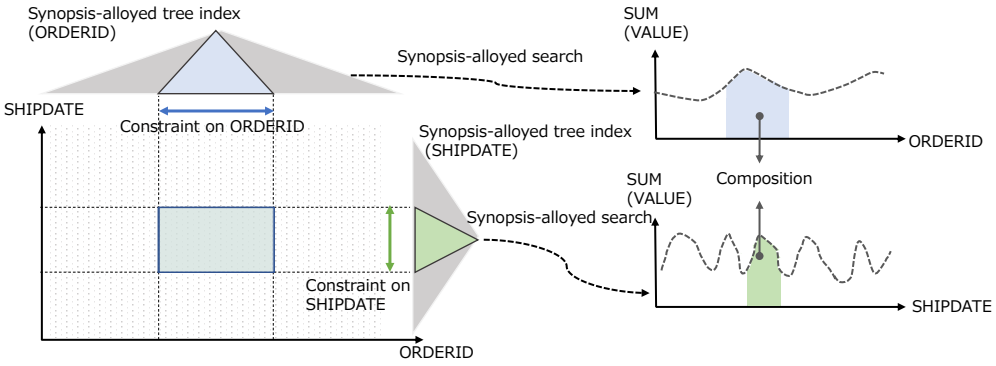


FIGURE 2: Inter-attribute result composition. For a query having a composed constraint, separate synopsis-alloyed search offers a partial result and the inter-attribute result composition technique composes the results to obtain an approximate answer.

satisfy the query search range, i.e., partial exact values for the query, in the result buffer (line 7) and stops the navigation below the internal node when the synopsis of the reached internal node is sufficient for the selective attributes (lines 8-10). When reaching out to a leaf node, SAS fetches matched records (lines 17-18). By reducing the cost of navigating the tree when the synopsis satisfies the selective attributes, SAS improves the efficiency of query processing to output the exact answer.

For example, consider Figure 1(b) and the query MAX(VALUE) with ORDERID < 46. A standard B^+-tree traverses the internal nodes of Node 1, 2, and leaf nodes of Node 5-11. In contrast, SAS examines only three nodes: Node 1, 4, and 11. Node 1 satisfies the partial query constraint, i.e., ORDERID < 25, in the designated index key and selective attributes in the synopsis. For the rest of the query constraint, i.e., ORDERID >=25 and ORDERID < 46, SAS examines the internal nodes of Node 4 and the leaf node of Node 11 for satisfying the query constraints and selective attributes.

As this example shows, utilizing the synopsis-alloyed B^+-tree, SAS described in Algorithm 1 potentially improves the efficiency of query processing while keeping the answer exactness, if the synopsis is mathematically sufficient to form a part of the answer to a given query. This technical benefit will be experimentally demonstrated in Section VI.

B. APPROXIMATE QUERY PROCESSING FOR COMPOSITE QUERY CONSTRAINTS

Next, this paper presents an approximate query processing for composite query constraints.

Algorithm 1 efficiently obtains exact values using an SAI only if the synopsis in the index is mathematically sufficient for a query constraint of a given query. In the case that synopsis may not be mathematically sufficient for a given query constraint, this section presents a search method to answer an approximate answer by extending Algorithm 1. This extension can potentially be deployed for more queries that require immediate responses rather than exact values.

This paper introduces the *inter-attribute result composition* technique that combines exact values derived from each SAI for a query constraint to answer an approximate value. Figure 2 illustrates that the two different SAIs holding two different attributes as a designated index key output exact values for each different query constraint of a given query, which are combined to produce an approximate value. Suppose a query to obtain the sum of VALUE under the condition of ORDERID < 46 and SHIPDATE >= 2022-12-05 on Figure 1(b). Preparing two SAIs holding ORDERID and SHIPDATE as a designated index key, the SAI (ORDERID) outputs a sum of VALUE (3200) with a query constraint of ORDERID (ORDERID < 46) whereas the SAI (SHIPDATE) outputs a sum of VALUE (2600) with a query constraint of SHIPDATE (SHIPDATE >= 2022-12-05). The required value can be approximately calculated from the first query constraint multiplied by another value obtained from another query constraint and divided by the total value from the all search space to compose both two conditions, i.e., $3200 \times 2600/3800 = 2189.5$, where 3800 is the total value from all search space.

This mathematical composition assumes that the concerned attributes are statistically independent. In the case the assumption is effective, Algorithm 1 and the inter-attribute result composition technique efficiently performs query processing by reducing the search cost of the search space using the synopsis. Also, the composition technique potentially outputs an approximate answer with a negligible error when attribute distributions are sufficiently independent and unbiased. The query performance and accuracy will be demonstrated in section VI.

C. CONSIDERATION OF INTER-ATTRIBUTE CORRELATION

Furthermore, this paper extends SAS to consider inter-attribute correlations. The inter-attribute result composition assumes the independence between the concerned attributes as described in subsection B. If the concerned attributes are correlated, the composite result may include non-negligible errors. To consider the inter-attribute correlation, this pa-

per presents an extended synopsis-alloyed search, called synopsis-alloyed search plus considering inter-attribute correlation; SAS+.

Assume that inter-attribute correlations are added in an SAI. With Algorithm 1, the search query processing examines each index node, utilizing the precomputed synopsis if the synopsis is sufficient for the selective attributes. The extended version of the search method utilizes the synopsis only if the inter-attribute correlation is lower than specific criteria because the inter-attribute compositions technique assumes that the concerned attribute is independent. If the inter-attribute correlation is higher, the search method recursively examines internal nodes until leaf nodes are fetched, obtaining exact table records. SAS+ potentially improves the error rate if the concerned attributes are correlated, although SAS+ may reduce the query processing performance than SAS.

V. IMPLEMENTATION ON AN EXISTING DATABASE SYSTEM

Section V introduces an implementation to integrate synopsis-alloyed search methods (SAS, SAS+) using an SAI in an existing database system for improving query processing beyond existing B^+-tree in query processing performance of the query result.

PostgreSQL supports foreign data wrapper (FDW) that enables the PostgreSQL server to access data stored aside from the database domain through its APIs by defining the external data as foreign tables. When a query is given, PostgreSQL engine initiates the query parsing, rewriting, and building of an access plan, subsequently executing the query to retrieve the desired value. PostgreSQL provides APIs that implement the process of building and executing access plans to access external data sources. There are two main types of APIs required for query processing of external data: one for planning the query process and another one for execution. Those APIs are called when a query is input in the PostgreSQL interface. PostgreSQL provides postgres_fdw [38] as a type of foreign data wrapper to access data stored in an external PostgreSQL server using those APIs. This paper implements *GetForeignRelSize, GetForeignPaths, GetForeignPlan* for planning of query processing and *BeginForeignScan, IterateForeignScan, ReScanForeignScan, EndForeignScan* for the query execution to execute the synopsis-alloyed search methods (SAS, SAS+) by referencing the APIs of postgres_fdw.

Figure 3 shows a query processing process of PostgreSQL server to access the SAI using the FDW (Approximate_fdw). When a query is received through the standard SQL interface, PostgreSQL engine takes responsibility for parsing and rewriting. For access to a foreign table, PostgreSQL calls planning hooks (*GetForeignRelSize*, *GetForeignPaths*, *GetForeignPlan*), and execution hooks (*BeginForeignScan*, *IterateForeignScan, ReScanForeignScan*, *EndForeignScan*). Our Approximate_FDW invokes SAS/SAS+ in IterateForeignScan, reading the SAI files from storage and returning results to the SQL interface. When the synopsis-alloyed search methods return an exact or an approximate value to the given query, the value is returned to the SQL interface.

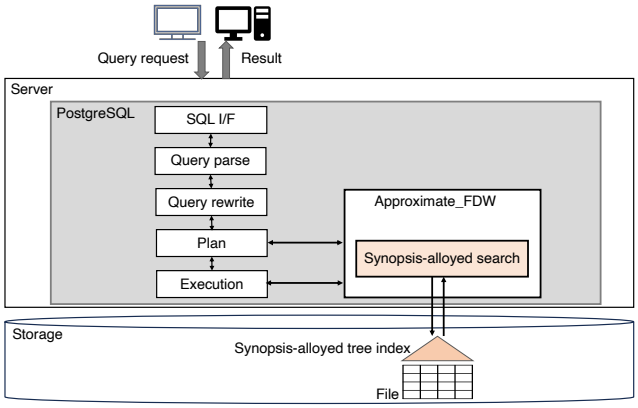


FIGURE 3: A system design of PostgreSQL foreign data wrapper. Given a query accessing a foreign table, a specific foreign data wrapper (e.g., Approximate_FDW) is called, where the synopsis-alloyed search is integrated.

This paper presents an implementation approach to apply synopsis-alloyed search methods to PostgreSQL using FDW to return exact or approximate answers to various queries. While the detailed design may differ, a similar design may be applicable to other existing database systems because they have the ability to access external data sources similar to FDW.

VI. EXPERIMENTS

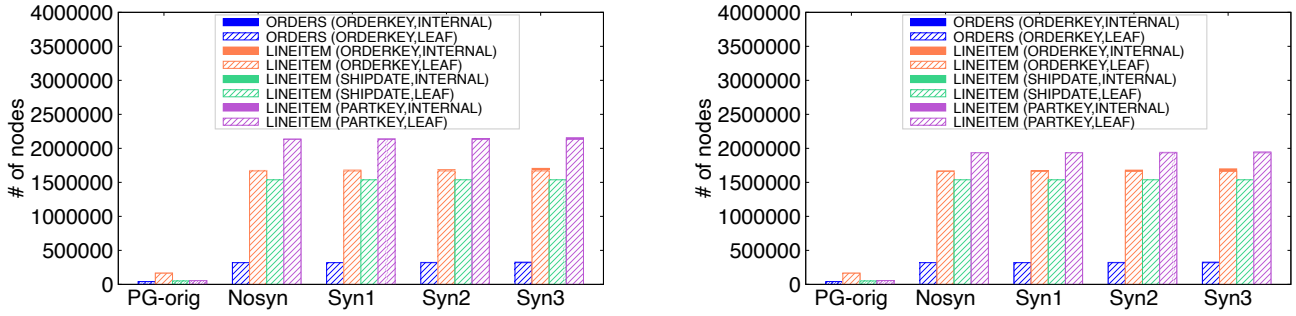
In this section, we present our intensive prototype-based experiments to clarify the overhead of synopsis-alloyed tree index with different types of synopsis information and the benefit of SAS on different queries having one query constraint using uniform [39] and skewed TPC-H datasets [40]. In addition, we demonstrate the comparison of SAS and SAS+ to show the benefits of consideration of inter-attribute correlation. Finally, we present the benefits of SAS on different queries for real Intel wireless dataset.

A. EXPERIMENTAL SETUP

This evaluation compares two types of datasets: a uniform dataset where a TPC-H data generator (dbgen) provides [39] and a skewed dataset where revised dbgen generates [40], both with the scale factor 1,10,100. Unless otherwise noted, the scale factor is 10. The zipf value was set to 1.0 for the skewed dataset.

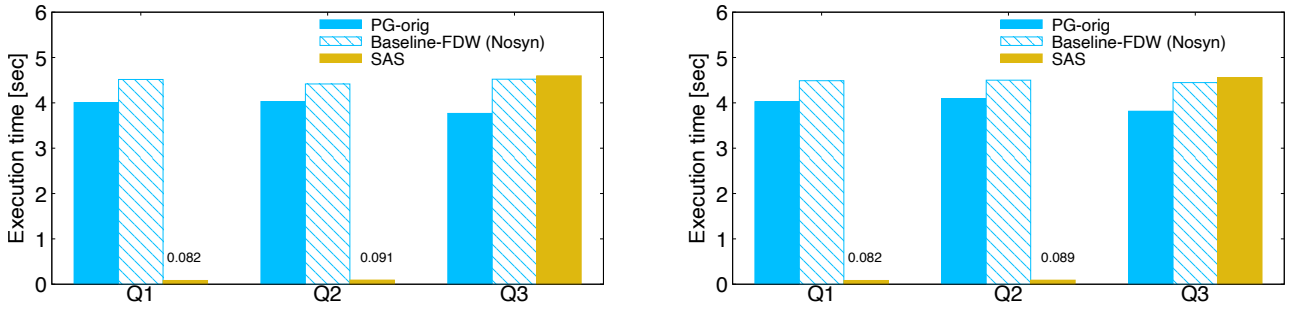
We experimentally used the following queries to evaluate execution time and error rate in approximate query processing when accessing the datasets. Q1 to Q3 are test queries with a single constraint, whereas Q4 to Q7 are test queries with two constraints. The selective ratio for all the queries is around 10% in the concerned tables.

For evaluating the queries, we prepared four B^+-tree data structures for both uniform and skewed TPC-H datasets:



(a) Uniform dataset (b) Skewed dataset

FIGURE 4: Synopsis-alloyed tree requires small storage space overheads. Even the intensive case **Syn3** incurs only 1.82% of additional nodes. LEAF and INTERNAL denote leaf and internal nodes respectively.



(a) Uniform dataset (b) Skewed dataset

FIGURE 5: SAS using SAI performs significantly faster for Q1 and Q2, where the SAI satisfies the query constraint and derivative attribute. By contrast, SAS does not support Q3, where the synopsis lacks the necessary information for derivative attribute.

TABLE 1: Test queries with a single constraint

Q1	SELECT SUM(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_ORDERKEY BETWEEN 54000001 and 60000001;
Q2	SELECT MAX(L_EXTENDEDPRICE) FROM LINEITEM WHERE L_ORDERKEY BETWEEN 54000001 and 60000001;
Q3	SELECT MAX(L_SHIPDATE) FROM LINEITEM WHERE L_ORDERKEY BETWEEN 54000001 and 60000001;

TABLE 2: Test queries with a composite constraint

Q4	SELECT SUM(L_EXTENDEDPRICE) from LINEITEM where L_ORDERKEY BETWEEN 36000000 and 42000000 AND L_SHIPDATE BETWEEN 1992-01-01 and 1992-12-31;
Q5	SELECT MAX(L_EXTENDEDPRICE) from LINEITEM where L_ORDERKEY BETWEEN 36000000 and 42000000 AND L_SHIPDATE BETWEEN 1992-01-01 and 1992-12-31;
Q6	SELECT SUM(L_EXTENDEDPRICE) from LINEITEM where L_ORDERKEY BETWEEN 36000000 and 42000000 AND L_PARTKEY BETWEEN 1800000 and 2000000;
Q7	SELECT MAX(L_EXTENDEDPRICE) from LINEITEM where L_ORDERKEY BETWEEN 36000000 and 42000000 AND L_PARTKEY BETWEEN 1800000 and 2000000;

- ORDERS (ORDERKEY) stores the ORDERS records by the key of O_ORDERKEY;
- LINEITEM (ORDERKEY, LINENUMBER) stores the LINEITEM records by the key of L_ORDERKEY and L_LINENUMBER ;
- LINEITEM (SHIPDATE) stores the LINEITEM records by the key of L_SHIPDATE.
- LINEITEM (PARTKEY) stores the LINEITEM records by the key of L_PARTKEY.

For each data structure, we comparatively alloy the following types of synopsis.

- **PG-orig** indicates a native B^+-tree design of PostgreSQL as a baseline.
- **Nosyn** holds no synopsis information. This case is equivalent to the native B^+-tree design integrated in foreign data wrapper.
- **Syn1** holds the following synopsis information; the count, sum, maximum, and minimum values of the key attribute, i.e., O_ORDERKEY for ORDERS (ORDERKEY), L_ORDERKEY for LINEITEM

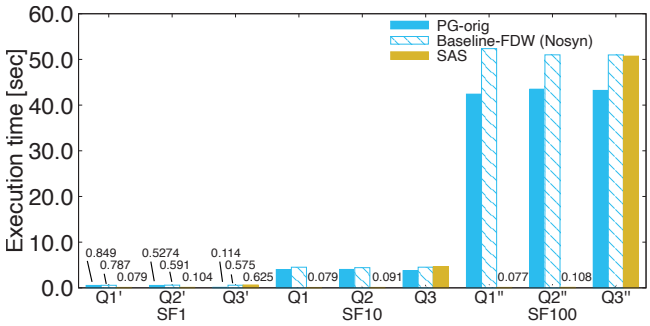


FIGURE 6: Execution time using various scaled uniform datasets. SAS performs significantly faster for series of Q1 and Q2 where the SAI satisfies the query constraint and derivative attributes at any scale factor, while SAS does not support series of Q3.

(ORDERKEY, LINENUMBER), L_SHIPDATE for LINEITEM (SHIPDATE), L_PARTKEY for LINEITEM (PARTKEY);

- **Syn2** further holds the count, sum, maximum, and minimum values of the additional attributes, i.e., O_TOTALPRICE for ORDERS (ORDERKEY), L_EXTENDEDPRICE for LINEITEM (ORDERKEY), L_EXTENDEDPRICE for LINEITEM (SHIPDATE), and L_EXTENDEDPRICE for LINEITEM (PARTKEY); and
- **Syn3** further holds the correlation value between the key attribute and another attributes, i.e., O_ORDERKEY and O_ORDERDATE for ORDERS (ORDERKEY), L_ORDERKEY and L_SHIPDATE for LINEITEM (ORDERKEY, LINENUMBER), L_SHIPDATE and L_ORDERKEY for LINEITEM (SHIPDATE), and L_PARTKEY and L_ORDERKEY for LINEITEM for LINEITEM (PARTKEY).

We implemented our prototype of the synopsis-alloyed search method on the well-known B^+-tree design [2] with PostgreSQL version 14.12 foreign data wrapper (FDW). All the experiments were performed on 2 CPU cores, 14 GB Memory, 1.5 TB storage, running on OS Amazon Linux release 2 (Karoo) on AWS cloud service. All the data structures were stored on xfs file system constructed on the storage. The node size of B^+-tree was set to 8192 bytes, the same as the common page size of PostgreSQL. We performed five trials for each measurement, and the average values of the trials were used for the report.

B. EXACT QUERY PROCESSING WITH A SINGLE CONSTRAINT

First, we present the overhead of synopsis-alloyed tree index with different data structures and synopsis information. Figure 4 demonstrates the storage overhead incurred by the synopsis information alloyed in B^+-tree using both normal (a) and skewed datasets (b). By comparison with the baseline (**Nosyn**), the synopsis-alloyed tree of **Syn1** to **Syn3** requires quite small overhead for both uniform and skewed datasets. Even the most intensive case of **Syn3** requires 1.82% additional overhead in addition to the baseline of **Nosyn**. This is because the leaf nodes require more overhead than internal nodes do. Internal nodes occupy less than 1% of the baseline (**Nosyn**) and at most 2.2%, even in the most intensive case (**Syn3**). Thus, the total overhead required is pretty small, even with the additional overhead incurred by synopsis in internal nodes.

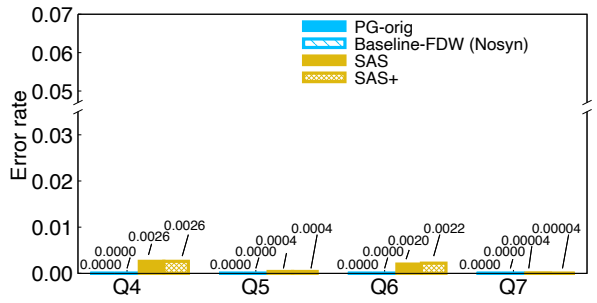
Second, we present the benefit of SAS on different queries (Q1 to Q3) denoted in Table 1 having a single constraint using uniform (Figure 5(a)) and skewed datasets (Figure 5(b)). Baseline of FDW (**Baseline-FDW (Nosyn)**) adopting the same access as PostgreSQL (**PG-orig**) shows slight overhead compared with PG-orig due to FDW use, but SAS using synopsis-alloyed B^+-tree significantly performs faster for Q1 and Q2, where the synopsis satisfies the query constraint and selective attribute. In contrast, SAS does not support Q3, where the synopsis lacks the necessary information for derivative attributes. This demonstrates that the SAI efficiently improves query processing if the designated index key satisfies the query constraint and the synopsis satisfies the selective attribute.

Moreover, Figure 6 illustrates the benefits of SAS using different dataset sizes with scale factors of 1,10, 100. To keep 10% selectivity for all sizes of datasets, we adjusted query ranges: narrower ranges for SF=1 (Q1′ to Q3′) and wider ranges for SF=100 (Q1″ to Q3″). SAS using synopsis-alloyed B^+-tree demonstrates significantly effective performance in each different size of datasets for series of Q1 and Q2, where the synopsis satisfies the query constraint and selective attribute. In contrast, SAS does not support series of Q3, where the synopsis lacks the necessary information for derivative attributes on any scale factor dataset. Thus, SAI presents benefits regardless of the dataset size if the designated index key satisfies the query constraint and the synopsis satisfies the selective attribute.

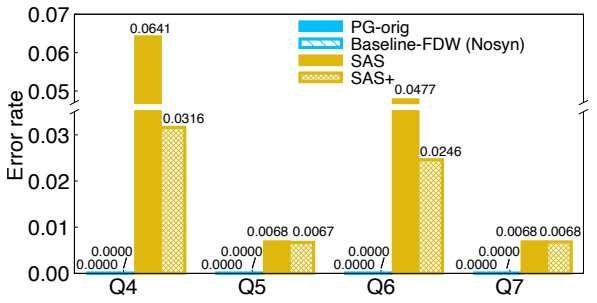
C. APPROXIMATE QUERY PROCESSING WITH COMPOSITE CONSTRAINTS

Furthermore, this paper presents that synopsis-alloyed search plus considering inter-attribute correlation (SAS+) performs faster than baseline with moderate error rates.

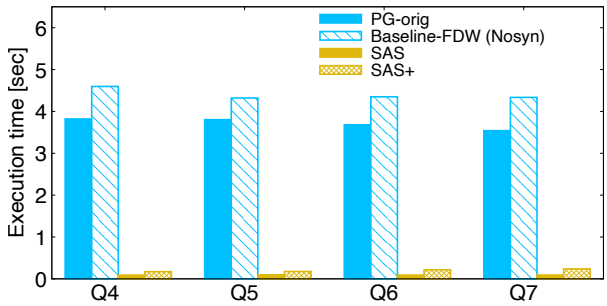
We executed the Q4 to Q7 queries denoted in Table 2 on LINEITEM (ORDERKEY), LINEITEM (SHIPDATE), and LINEITEM (PARTKEY) data structures with the **Syn3** synopsis configuration and measured the execution time and error rate for both uniform and skewed datasets. The synopsis-alloyed data structure does not offer an exact answer for Q4 to Q7 queries, but each of the data structure support each query to narrow down the search space, and the synopsis information of **Syn3** offers the derivative attributes. This experiment reports the execution time and error rate using SAS and the inter-attribute result composition techniques (presented in Section IV).



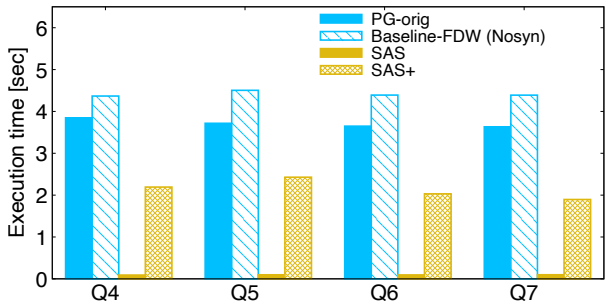
(a) Error rate: Uniform dataset



(b) Error rate: Skewed dataset



(c) Execution time: Uniform dataset



(d) Execution time: Skewed dataset

FIGURE 7: SAS with inter-attribute result composition performs significantly faster than the baseline case (PG-orig) with a negligible error. SAS+ offers a balanced property, performing faster than the baseline case with a smaller error than SAS.

Figure 7 shows the execution time and error rates involved in performing synopsis-alloyed search (SAS) and synopsis-alloyed search plus considering inter-attribute correlation (SAS+) for both uniform and skewed datasets. Compared to the baseline of PostgreSQL (**PG-orig**), SAS significantly outperforms the execution time because the synopsis of **Syn3** offers the derivative attributes of all the queries for both uniform (Figure 7(c)) and skewed datasets (Figure 7(d)). The uniform dataset does not have a high correlation between the key attribute (L_ORDERKEY) and additional attributes (L_SHIPDATE, L_PARTKEY) of the data structure, bringing no significant error. However, the skewed dataset offers negligible error for the skewed dataset because SAS assumes uniform distribution for concerned attributes.

By contrast, SAS+ utilizes the inter-attribute correlation information to decide if the query processing uses synopsis at each internal node. SAS+, like SAS, performed faster than the baseline with only a small error (0.0026) on the uniform dataset. Although SAS reduces the execution time rather than SAS+ on the skewed dataset, SAS+ offered a balanced performance on the skewed dataset, faster than the baseline by up to 45.6% and a smaller error rate of up to 0.03. In summary, this experiment clarifies that synopsis-alloyed search plus considering the inter-attribute result composition (SAS+) significantly speeds up a test query with a quite small error on the uniform dataset, offering balanced performance for the skewed dataset.

TABLE 3: Queries for Intel Wireless Dataset

Q11	select sum(light) from sensors where date between '2004-03-01' and '2004-03-04' and time between '00:00:00.000000' and '23:59:59.999999';
Q12	select max(light) from sensors where date between '2004-03-01' and '2004-03-04' and time between '00:00:00.000000' and '23:59:59.999999';
Q13	select max(humidity) from sensors where date between '2004-03-01' and '2004-03-04' and time between '00:00:00.000000' and '23:59:59.999999';

D. EVALUATION WITH REAL DATASET

We evaluate on a real dataset. This paper utilized an Intel Wireless Dataset [41] that has about 2.3 million records with 8 columns including temperature, humidity, light, and voltage per date and time. The dataset was stored as a table SENSORS, and the following queries are utilized.

Similarly to evaluation using TPC-H data, this paper prepared four B^+-tree data structures that store Intel Lab data records by the key of date and time. The data structure allows the date, time, and light as synopsis.

We present the benefit of SAS on different queries (Q11 to Q13) denoted in Table 3 for Intel Lab dataset. **Baseline-**

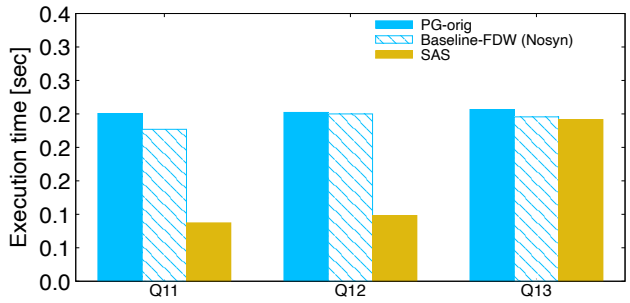


FIGURE 8: Execution time using queries for Intel Wireless Dataset. SAS using SAI performs significantly faster for Q11 and Q12, where the synopsis-alloyed data structure satisfies the query constraint and derivative attributes. By contrast, SAS does not support Q13, where the synopsis lacks the necessary information for derivative attribute.

FDW (Nosyn)) performs similar to (**PG-orig**), SAS using synopsis-alloyed B^+-tree achieves up to 65.2 times speedup for Q11 and Q12, where the synopsis satisfies the query constraint and selective attribute. In contrast, SAS does not support Q13, where the synopsis lacks the necessary information for derivative attributes, humidity in this case. This demonstrates that the SAI efficiently improves query processing if the designated index key satisfies the query constraint and the synopsis satisfies the selective attribute for the real dataset.

VII. DISCUSSION

This paper presents exact and approximate query processing using SAI in an existing relational database system and non-distributed data. Our plans consider the following challenges.

First, this paper focuses on simple queries that target a single table. Although this paper concentrated on simple queries, the notion of the SAI and the search methods employing the SAI should be applicable and valid for more intricate queries, including JOIN, GROUP BY, or HAVING by incorporating summary or abstraction information into internal nodes. Specifically, the joined values can be incorporated into the internal nodes of SAI as the synopsis. If needed, queries can be reformulated to leverage the SAI. As with the search over a single table, while the search processing recursively visits the nodes, if the synopsis satisfies the query constraints and selective values, the search processing retains the partial exact values that satisfy the query in the result buffer. Thus, SAI could be theoretically applicable for complex queries involving JOIN, GROUP BY, or HAVING, but issues remain. One reason is that complex queries may require a broader variety of synopses, increasing the storage overhead of SAI. Under such constraints, optimizing synopsis selection becomes critical. Additionally, if the database becomes larger, errors may increase in skewed data distributions because the current SAS with inter-attribute result composition is based on a uniform distribution. Evaluation targeting multiple tables and extensions to distributed data is demonstrated in detail in the following paper by addressing the issues.

Next, this study assumes a bulk-loading scenario, which is common in analytical workloads and offline data preparation pipelines. Under this assumption, the cost of creating the synopsis embedded in the index can be very low. As our experimental results demonstrate, the overhead incurred during index creation is negligible compared to the total preparation required for data ingestion. However, in dynamic environments with frequent insertions, updates, or deletions, maintaining the accuracy of the embedded synopsis can produce significant overhead. Updating the synopsis directly with each record-level modification could be computationally expensive and impractical. To mitigate this issue, we could consider deferred maintenance strategies, such as periodic synopsis reconstruction or amortized updates, to balance performance and accuracy. Designing efficient and adaptive maintenance mechanisms for SAIs in mutable datasets remains an open research challenge. We regard this as a promising direction for future work and acknowledge the current limitations in this area.

Thus, we plan to investigate these remaining challenges in future studies, with the aim of developing practical and scalable maintenance strategies.

VIII. CONCLUSION

Structured database systems organize business data to efficiently process business queries. This paper proposes a synopsis-alloyed index (SAI), incorporating the synopsis that is summary or abstraction information of concerned table records. Using the synopsis, query processing quickly retrieves exact or approximate values without accessing all the concerned table records. This paper extends our previous work and presents the implementation design of the efficient search methods using the synopsis-alloyed B^+-tree integrated into an open-source database system, PostgreSQL. The prototype-based experiments demonstrate that the search methods speed up to 97.9%, where synopsis satisfies the query constraint and derivative attributes compared with PostgreSQL's native B^+-tree. This paper presents significant evaluation results for SAI regarding simple queries on a single table. However, the SAI application to more complex queries involving multiple tables and distributed data remains challenging and will be examined in the following article.

REFERENCES

[1] D. J. Abel, "A b+-tree structure for large quadtrees," ICVGIP, vol. 27, no. 1, pp. 19–31, Jul. 1984.
[2] R. Elmasri, "Fundamentals of database systems seventh edition," 2021.
[3] H. Yuasa, K. Goda, and M. Kitsuregawa, "Exploiting embedded synopsis for exact and approximate query processing," in Proc. DEXA, 2022, pp. 235–240.
[4] "PostgreSQL," https://www.postgresql.org/, accessed on July. 25, 2024.
[5] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Commun. ACM, vol. 13, no. 7, pp. 422–426, 1970.
[6] B. Fan, D. G. Andersen, M. Kaminsky, and M. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in Proc. CoNEXT, 2014, pp. 75–88.

[7] T. M. Graf and D. Lemire, "Xor filters," J. Exp. Algorithmics, vol. 25, pp. 1–16, 2020.
[8] ——, "Binary fuse filters: Fast and smaller than xor filters," J. Exp. Algorithmics, vol. 27, pp. 1.5:1–1.5:15, 2022.
[9] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in Proc. Algorithms - ESA, ser. Lecture Notes in Computer Science, vol. 4168. Springer, 2006, pp. 684–695.
[10] I. Mami and Z. Bellahsene, "A survey of view selection methods," SIGMOD Rec., vol. 41, no. 1, pp. 20–29, 2012.
[11] M. V. Mannino, P. Chu, and T. Sager, "Statistical profile estimation in database systems," ACM Comput. Surv., vol. 20, no. 3, pp. 191–221, 1988.
[12] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," in Proc. SIGMOD, P. A. Bernstein, Ed., 1979, pp. 23–34.
[13] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra, "Scalable approximate query processing with the dbo engine," TODS, vol. 33, no. 4, pp. 1–54, 2008.
[14] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica, "G-ola: Generalized on-line aggregation for interactive analysis on big data," in Proc. SIGMOD, 2015, pp. 913–918.
[15] "Interactive data analysis: The control project," Computer, vol. 32, no. 8, pp. 51–59, 1999.
[16] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi, "Distributed and interactive cube exploration," in Proc. ICDE, 2014, pp. 472–483.
[17] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, "The aqua approximate query answering system," in Proc. SIGMOD, 1999, pp. 574–576.
[18] I. Mytilinis, D. Tsoumakos, and N. Koziris, "Distributed wavelet thresholding for maximum error metrics," in Proc. SIGMOD. Association for Computing Machinery, 2016, pp. 663–677.
[19] A. N. Sazish and A. Amira, "An efficient architecture for HWT using sparse matrix factorisation and DA principles," in Proc. APCCAS, 2008, pp. 1308–1311.
[20] G. Cormode, A. Deligiannakis, M. Garofalakis, and A. McGregor, "Probabilistic histograms for probabilistic data," Proc. VLDB Endow., vol. 2, no. 1, pp. 526–537, Aug. 2009.
[21] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "BlinkDB: queries with bounded errors and bounded response times on very large data," in Proc. EuroSys, 2013, pp. 29–42.
[22] J. Peng, D. Zhang, J. Wang, and J. Pei, "AQP++: Connecting approximate query processing with aggregate precomputation for interactive analytics," in Proc. SIGMOD, 2018, pp. 1477–1492.
[23] A. Hurst, D. E. Lucani, and Q. Zhang, "Pairwisehist: Fast, accurate, and space-efficient approximate query processing with data compression," Proc. VLDB Endow., vol. 17, no. 6, pp. 1432–1445, 2024.
[24] R. Gu, H. Li, H. Dai, W. Huang, J. Xue, M. Li, J. Zheng, H. Cai, Y. Huang, and G. Chen, "Shadowaqp: Efficient approximate group-by and join query via attribute-oriented sample size allocation and data generation," Proc. VLDB Endow., vol. 16, no. 13, pp. 4216–4229, 2023.
[25] Q. Ma and P. Triantafillou, "Dbest: Revisiting approximate query processing engines with machine learning models," in Proc. SIGMOD, 2019, pp. 1553–1570.
[26] R. Li, P. Wang, J. Zhu, J. Zhao, J. Di, X. Yang, and K. Ye, "Building fast and compact sketches for approximately multi-set multi-membership querying," in Proc. SIGMOD, 2021, pp. 1077–1089.
[27] X. Liang, S. Sintos, Z. Shang, and S. Krishnan, "Combining aggregation and sampling (nearly) optimally for approximate query processing," in Proc. SIGMOD, 2021, pp. 1129–1141.
[28] X. Liang, S. Sintos, and S. Krishnan, "Janusaqp: Efficient partition tree maintenance for dynamic approximate query processing," in Proc. ICDE. IEEE, 2023, pp. 572–584.
[29] J. Bater, Y. Park, X. He, X. Wang, and J. Rogers, "SAQE: practical privacy-preserving approximate query processing for data federations," Proc. VLDB Endow., vol. 13, no. 11, pp. 2691–2705, 2020.
[30] "Apache Spark," https://spark.apache.org/docs/latest/api/python/index.html, accessed on Nov. 19, 2025.
[31] "Trino," https://trino.io/, accessed on Nov. 19, 2025.
[32] "Google BigQuery," https://docs.cloud.google.com/bigquery/docs, accessed on Nov. 19, 2025.
[33] "Amazon Redshift Spectrum," https://docs.aws.amazon.com/redshift/, accessed on Nov. 19, 2025.
[34] "Snowflake," https://docs.snowflake.com/, accessed on Nov. 19, 2025.
[35] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, "Join synopses for approximate query answering," in Proc. SIGMOD, 1999, pp. 275–286.
[36] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska, "Revisiting reuse for approximate query processing," Proc. VLDB Endow., vol. 10, no. 10, pp. 1142–1153, 2017.
[37] Y. Park, B. Mozafari, J. Sorenson, and J. Wang, "Verdictdb: Universalizing approximate query processing," in Proc. SIGMOD, 2018, pp. 1461–1476.
[38] "postgres_fdw," https://www.postgresql.jp/document/14/html/postgres-fdw.html, accessed on Feb. 26, 2024.
[39] "Tpc-h," https://www.tpc.org/tpch/, accessed on Jan. 27, 2023.
[40] "Ysu-data-lab/tpc-h-skew," https://github.com/YSU-Data-Lab/TPC-H-Skew, accessed on Jan. 31, 2023.
[41] P. Bodik, W. Hong, C. Guestrin, S. Madden, M. Paskin, and R. Thibaux, "Intel lab data," https://db.csail.mit.edu/labdata/labdata.html, accessed on Aug. 13, 2025.

MIKA TAKATA received her B.E. and the M.E. in computer science from Waseda University in 2007 and 2009 respectively, and M.S. in bioinformatics from the University of Georgia in 2012. She joined R&D group of Hitachi, Ltd in 2012 and also worked for Hitachi America, Ltd from 2016 to 2019. She is a member of IPSJ and JSAI.

HIROKI YUASA received the B.E. and the M.S. in information and communication engineering from The University of Tokyo in 2020 and 2022 respectively. He currently works for Koei Tecmo Games.

KAZUO GODA is a professor at Institute of Industrial Science, The University of Tokyo. He received his B.E. in electrical engineering, his M.E. in information and communication engineering, and his Ph.D. in information science and technology from The University of Tokyo in 2000, 2002, and 2005 respectively. His research interests include database systems and storage systems. He is a member of ACM, IEEE, USENIX, IEICE (The Institute of Electronics, Information and Communication Engineers), IPSJ (Information Processing Society of Japan), and DBSJ (The Database Society of Japan.)