# Efficient Execution Methods of Pivoting for Bulk Extraction of Entity-Attribute-Value-Modeled Data

Gang Luo, Lewis J. Frey

***Abstract*—Entity-Attribute-Value (EAV) tables are widely used to store data in electronic medical records and clinical study data management systems. Before they can be used by various analytical (e.g., data mining and machine learning) programs, EAV-modeled data usually must be transformed into conventional relational table format through pivot operations. This time-consuming and resource-intensive process is often performed repeatedly on a regular basis, e.g., to provide a daily refresh of the content in a clinical data warehouse. Thus, it would be beneficial to make pivot operations as efficient as possible. In this paper, we present three techniques for improving the efficiency of pivot operations: (1) filtering out EAV tuples related to unneeded clinical parameters early on, (2) supporting pivoting across multiple EAV tables, and (3) conducting multi-query optimization. We demonstrate the effectiveness of our techniques through implementation. We show that our optimized execution method of pivoting using these techniques significantly outperforms the current basic execution method of pivoting. Our techniques can be used to build a data extraction tool to simplify the specification of and improve the efficiency of extracting data from the EAV tables in electronic medical records and clinical study data management systems.**

***Index Terms*—Clinical study data management system, database, electronic medical record, Entity-Attribute-Value, pivot**

## I. INTRODUCTION

The Entity-Attribute-Value (EAV) data model is widely used for data storage in electronic medical records (EMRs) and clinical study data management systems (CSDMSs). It is particularly suitable for supporting fast transaction processing when data are sparse and have many applicable attributes, but only a small fraction of them applies to a specific entity [1]. Among all sparse data storage models [1-3] that have been proposed in the relational database literature, the EAV data model is the most widely used in clinical systems. Example EMR systems using the EAV data model include the Regenstrief EMR [4], the Columbia-Presbyterian EMR [5], the TMR EMR [6], Intermountain Healthcare's HELP EMR [7], and the Cerner Powerchart EMR [8]. Example CSDMSs using the EAV data model include Oracle Clinical [9], Clintrial [10], TrialDB [11], i2b2, LabKey, OpenClinica, Opal, and REDCap [12, 13].

The EAV data model uses tables with at least three columns: the entity, the attribute, and the value. Typically, the entity column includes the ID of a clinical event, which can be regarded as the combination of a patient ID and a date/time stamp [14, page 58]. The attribute column includes the ID of a clinical parameter. The value column includes the clinical parameter's value. An example EAV table is shown in Fig. 1.

EAV table: clinical_data

| event_ID (entity) | parameter_ID (attribute) | value |
|---|---|---|
| E1 | test 1 | 100 |
| E2 | test 2 | 200 |
| E1 | test 2 | 300 |
| E2 | test 3 | 400 |
| E3 | test 4 | 500 |

pivot →

conventional relational table: test_data

| event_ID (entity) | test_1 | test_2 | test_3 |
|---|---|---|---|
| E1 | 100 | 300 | null |
| E2 | null | 200 | 400 |
| E3 | null | null | null |

Fig. 1. Pivot to obtain the columns for the three clinical parameters '*test 1*,' '*test 2*,' and '*test 3*.'

EAV tables can support fast transaction processing for individual patients, but are inefficient for population-based online analytical processing because data of the same event are spread across multiple rows. Also, EAV tables cannot be directly used by most of the existing analytical (e.g., data mining and machine learning) programs, which require inputs in the conventional relational table format [20]. That is, each clinical parameter of interest should have its own column. To address these problems, row-modeled data in EAV tables in a production clinical system are transformed into column-modeled, conventional relational table format through pivot operations [15-17]. The transformed data are then bulk loaded into an enterprise clinical data warehouse, which typically runs on separate hardware from the production system [14, 18] and supports various management, strategic decision making, and clinical decision support applications [19]. Fig. 1 shows an example of the pivot operation.

A production clinical system typically contains a huge amount of data. Frequently, hundreds to thousands of conventional relational tables, e.g., one per form in an EMR or CSDMS [14, page 344, 20], need to be generated in a clinical data warehouse through a bulk extraction process using pivot

Gang Luo is with the Department of Biomedical Informatics, University of Utah, Suite 140, 421 Wakara Way, Salt Lake City, UT 84108, USA (email: gang.luo@utah.edu).

Lewis J. Frey is with the Department of Public Health Sciences, Medical University of South Carolina, 135 Cannon Street Suite 303, MSC 835, Charleston, SC 29425-8350, USA (email: frey@musc.edu).

operations. For instance, 9,000 tables and 10 terabytes of data are included in Intermountain Healthcare's enterprise data warehouse [19]. Thus, the bulk extraction process often consumes much time and significant computing resources. As new data continually arrive at the production system, this process is performed repeatedly on a regular basis, such as every night [19]. It would be beneficial to make this process, particularly the pivot operations, as efficient as possible.

Several execution methods of pivoting have been proposed in the research literature [16, 21]. Among them, the one sketched in Cunningham *et al*. [16] is the most efficient for data sets larger than memory. In this paper, we focus on the common case that the data set is larger than memory. We describe the execution method sketched in Cunningham *et al*. [16] in detail. Then we present three techniques for improving the efficiency of pivot operations. The first technique uses a special property of clinical applications to filter out EAV tuples related to unneeded clinical parameters early on. The second technique supports pivoting across multiple EAV tables. The third technique conducts multi-query optimization. We demonstrate the effectiveness of our techniques through an initial implementation.

A large healthcare provider typically has an information technology (IT) team dedicated to building and maintaining its clinical data warehouse. Frequently, for research, reporting, quality improvement, and other operational purposes, medical researchers and hospital operational people who do not necessarily have much computing background need to construct conventional relational tables from the EAV tables in the EMR. At present, such data extraction is conducted by submitting requests to the IT team. These data extraction requests are often fulfilled after a long delay or not at all, because the IT team tends to be constantly busy and pivot operations usually consume much time and significant computing resources.

Our techniques can be used to build a data extraction tool to simplify the specification of and improve the efficiency of extracting data from the EAV tables in the EMR. This tool can also facilitate data extraction from CSDMSs. After the user inputs the clinical parameters of interest and any other needed information, the data extraction tool automatically forms the corresponding pivot operations and then executes them. In this way, by eliminating dependency on the IT team, medical researchers and hospital operational people can use the data extraction tool to independently construct needed relational tables on demand and in a short amount of time. This would facilitate research, quality improvement, and hospital operations.

## II. METHODS

This section contains three parts. Section II-A reviews the definition of the pivot operation. Section II-B discusses existing execution methods of pivoting. Section II-C describes our techniques for improving the efficiency of pivot operations. A list of symbols used in this paper is provided in the appendix.

### *A. Definition of the pivot operation*

In this section, we review the definition of the pivot operation.

#### *1) SQL syntax*

An example syntax of the pivot operation in Structured Query Language (SQL) is as follows [22]:

```
SELECT <non-pivoted column(s)>, <alias 1>, <alias 2>,
    …, <alias n>
FROM <table-expression>
PIVOT (<value column(s)>
        FOR <column containing the values that will
            become column headers>
        IN (<value 1> AS <alias 1>, <value 2> AS <alias
            2>, …, <value n> AS <alias n>))
[WHERE clause]
[ORDER BY clause];
```

Here, <value 1>, <value 2>, …, and <value *n*> correspond to the *n* clinical parameters needed in the pivot result.

For the example shown in Fig. 1, the corresponding SQL query is as follows:

```
SELECT event_ID, test_1, test_2, test_3
FROM clinical_data
PIVOT (value FOR parameter_ID IN ('test 1' AS test_1,
        'test 2' AS test_2, 'test 3' AS test_3));
```

Here, '*test 1*,' '*test 2*,' and '*test 3*' are three clinical parameters needed in the pivot result. '*test 4*' is a clinical parameter that appears in the EAV table *clinical_data*, but is unneeded for the pivot result.

In the database research literature [15, 23], the pivot operation has been defined in two closely related, but different ways. Here, we call them *inner pivot* [23] and *outer pivot* [15], respectively. To differentiate them, we can introduce two reserved words INNER_PIVOT and OUTER_PIVOT into SQL. In the rest of this paper, pivot refers to outer pivot by default, unless inner pivot is specifically mentioned. We assume that in each EAV table, the entity column and the attribute column form a unique key, and hence any two values in the value column will never map to the same location in the pivot result. If this is not the case, we can use the methods described in Cunningham *et al*. [16] to handle data collisions, if any.

#### *2) Outer pivot*

Let $R$ denote an EAV table with an entity column $E$, an attribute column $A$, and a value column $V$. $A_1$, $A_2$, …, and $A_n$ are $n$ values of column $A$. Using relational algebra, the outer pivot operator is defined by [15]:

$$OUTER\_PIVOT_{V\ for\ A}^{[A_1, A_2, \ldots, A_n]}(R) = [\pi_E(R)] \ltimes [\bowtie_{i=1}^{n} \pi_{E,V}(\sigma_{A=A_i}(R))] \quad (1)$$

Here, $\ltimes$ is the left outer join operator. $\Theta_{i=1}^{n}\Psi_i$ is a short hand for $\Psi_1 \Theta \Psi_2 \Theta \cdots \Theta \Psi_n$. For a join operation including its outer species, the join predicate is assumed to be the equality of $E$ unless otherwise stated.

In general, the attribute column of the EAV table $R$ can include more values beyond $A_1$, $A_2$, …, and $A_n$. The pivot result will include an all-null relational tuple for an entity, if

the entity appears in $R$, but none of the entity's EAV tuples in $R$ relates to any of $A_1$, $A_2$, …, and $A_n$ [15]. An *all-null tuple* has a null value in each non-entity attribute. For instance, in the example shown in Fig. 1, the EAV table *clinical_data* includes a tuple of the entity *E3* related to the clinical parameter '*test 4*.' Consequently, the conventional relational table *test_data* formed through pivoting *clinical_data* includes an all-null tuple for *E3*.

*3) Inner pivot*

The inner pivot operation is almost the same as the outer pivot operation, except that it eliminates all-null tuples introduced by the outer pivot operation. The inner pivot operator is defined by [23]:

$$INNER\_PIVOT_{V\ for\ A}^{[A_1, A_2, \ldots, A_n]}(R) = \bowtie_{i=1}^{n} \pi_{E,V}(\sigma_{A=A_i}(R)) \quad (2)$$

Here, $\bowtie$ is the full outer join operator. For the example shown in Fig. 1, the corresponding inner pivot result is shown in Fig. 2.

| event_ID (entity) | test_1 | test_2 | test_3 |
|---|---|---|---|
| E1 | 100 | 300 | null |
| E2 | null | 200 | 400 |

Fig. 2. The inner pivot result corresponding to the example shown in Fig. 1.

*B. Existing execution methods of pivoting*

As shown in Equation (1), the simplest way to implement pivot is to conduct a series of (self-)joins, one per needed clinical parameter [23]. This method is rather inefficient due to the large number of joins typically needed [16]. In the past, researchers have proposed several more efficient execution methods of pivoting avoiding join [16, 21]. Among them, the one sketched in Cunningham *et al*. [16] is the most efficient for data sets larger than memory. In this section, we describe this execution method in detail, much of which has not been included in Cunningham *et al*. [16].

This execution method of pivoting implements pivot as GROUP BY (grouping) using either a sort-based approach [24, Chapter 15.4] or a hash-based approach [24, Chapter 15.5]. Either approach can support parallel query execution [24, Chapter 20.1], e.g., on multiple computers, by distributing EAV tuples in such a way that those of the same entity are processed in the same thread on the same computer [16]. In the following, we describe these two approaches one by one. Our description focuses on two-pass approaches, in which data from an EAV table is read into memory, processed, written out to disk, and then reread from disk to complete the pivot operation. As mentioned in Garcia-Molina *et al*. [24, Chapter 15.4], two passes are usually enough in practice, even for a very large EAV table. Also, it is not difficult to generalize to more than two passes.

*1) The sort-based approach*

The high-level idea of the sort-based approach is to use an efficient external sorting algorithm [25] to sort all tuples in the EAV table based on their entity values. Then all EAV tuples of the same entity become next to each other and are integrated into a conventional relational tuple for the entity. Several external sorting algorithms have been developed in the computer science literature [25]. In the rest of this sub-section, we present the details of the sort-based approach in a way similar to that in Garcia-Molina *et al*. [24, Chapter 15.4].

Let $M$ denote the number of pages available in the buffer pool in memory. We perform the following four steps:

Step 1: Read all tuples of the EAV table $R$ into memory, $M$ pages at time. Use a fast main-memory sorting algorithm, such as Quicksort, to sort each $M$ pages in ascending order of the entity value. Write each sorted sub-list out to disk. This produces $N=\lceil |R|/M \rceil$ sorted sub-lists in total, with $N \leq M$ for a two-pass approach. Here, $|R|$ denotes the size of $R$ in pages. $\lceil x \rceil$ is the ceiling function (e.g., $\lceil 1.3 \rceil=2$).

Step 2: Allocate $N$ buffer pages, one for each sorted sub-list. For each sub-list, load its first block into its corresponding buffer page.

Step 3: Find the smallest entity value $v$ among the first EAV tuples in the $N$ buffer pages. Then:

(a) Create a conventional relational tuple $t_v$ for $v$ with $n$ attributes, one for each clinical parameter needed in the pivot result. Here, $n$ denotes the number of clinical parameters needed in the pivot result. Initialize the value of each attribute of $t_v$ as null.

(b) In each of the $N$ buffer pages, the EAV tuples with entity value $v$, if any, must be the first several ones there because each sub-list is sorted in ascending order of the entity value. Examine each EAV tuple $t$ with entity value $v$ in the $N$ buffer pages. For $t_v$'s attribute corresponding to the clinical parameter ID $t$.attribute, set its value to be $t$.value. Then remove $t$ from its corresponding buffer page. If a buffer page becomes empty, load the next block, if any, from the corresponding sub-list into it.

(c) When no more EAV tuple with entity value $v$ remains in any of the $N$ buffer pages, all EAV tuples with entity value $v$ in the EAV table $R$ have been integrated into the relational tuple $t_v$. Output $t_v$ as a pivot result tuple.

Step 4: Repeat Step 3 until all EAV tuples in the $N$ sorted sub-lists have been consumed.

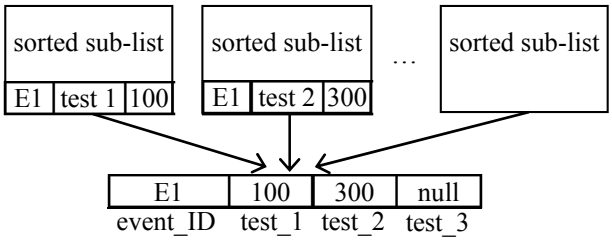



Fig. 3. In the second pass of the sort-based approach, integrate EAV tuples of the same entity from multiple sorted sub-lists into a conventional relational tuple for the entity.

In the above four steps, Step 1 constitutes the first pass. Steps 2-4 form the second pass that merges the $N$ sorted sub-lists to generate the conventional relational tuples in the pivot result. For the example shown in Fig. 1, Fig. 3 illustrates the second pass of the sort-based approach.

For an EAV table $R$, this two-pass, sort-based approach will work as long as $|R| \leq M^2$ [24, Chapter 15.4]. For database operations, disk I/O cost typically dominates CPU cost [2, 24, page 756]. In modeling a database operation's cost, we adopt the method used in Garcia-Molina *et al*. [24, page 756]: only disk I/O cost is considered and no charge is added for handling the output. This two-pass, sort-based approach takes $3|R|$ disk I/Os [24, Chapter 15.4]:

(1) $|R|$ to read the EAV table into memory to create the sorted sub-lists in the first pass.
(2) $|R|$ to write the sorted sub-lists out to disk in the first pass.
(3) $|R|$ to read the sorted sub-lists into memory in the second pass.

*2) The hash-based approach*

The high-level idea of the hash-based approach is to partition all tuples in the EAV table into buckets via hashing the entity value. For each entity, all EAV tuples of it fall into the same bucket. We ensure that within each bucket, all EAV tuples of the same entity are next to each other to facilitate integrating them into a conventional relational tuple for the entity. In the rest of this sub-section, we present the details of the hash-based approach in a way similar to that in Garcia-Molina *et al*. [24, Chapter 15.5].

Let $h_1$ and $h_2$ denote two hash functions taking an entity value as input. $h_1$ maps an entity value to an integer between 1 and $M$-1. We perform the following four steps:

Step 1: Initialize $M$-1 buckets, each of which uses an empty buffer page. The last buffer page holds blocks of the EAV table, one at a time.

Step 2: Read all tuples of the EAV table into memory, one block at a time. For each EAV tuple $t$, use the first hash function $h_1$ to hash $t$ to the bucket $h_1(t.\text{entity})$ and copy $t$ to the corresponding buffer page. If a bucket's buffer page becomes full, write it out to disk and then re-initialize it to be empty.

Step 3: After all tuples of the EAV table are consumed, write each bucket's buffer page out to disk if it is not empty. This produces $M$-1 buckets of EAV tuples. If the hash function $h_1$ works well, each bucket will have about $|R|/(M\text{-}1)$ blocks. For a two-pass approach, we should have $|R|/(M\text{-}1) \leq M$, or $|R| \leq M(M\text{-}1)$, so that each bucket can be held in memory in Step 4.

Step 4: Process the $M$-1 buckets one by one in $M$-1 phases, one bucket per phase. In each phase, process a bucket $B$ as follows:

(a) Initialize a hash table for the second hash function $h_2$.
(b) Read all EAV tuples in $B$ into memory. For each EAV tuple $t$, use $h_2$ to hash $t$ to the bucket $h_2(t.\text{entity})$ in the hash table. Add $t$ into this bucket while keeping all EAV tuples in this bucket sorted in ascending order of the entity value. This ensures that within this bucket, all EAV tuples of the same entity are next to each other.
(c) When all EAV tuples in $B$ have been added into the hash table, process all buckets in the hash table one by one. For each bucket in the hash table, sequentially scan all EAV tuples in the bucket, integrate all EAV tuples of the same entity into a conventional relational tuple $t_v$ for the entity, and output $t_v$ as a pivot result tuple. This is done in a way similar to that in Step 3 of the sort-based approach.

In the above four steps, Steps 1-3 form the first pass. Step 4 constitutes the second pass. This two-pass, hash-based approach will work when $|R| \leq M(M\text{-}1) \approx M^2$ and takes $3|R|$ disk I/Os [24, Chapter 15.5].

*C. Techniques for improving the efficiency of pivot operations*

In this section, we present three techniques for improving the efficiency of pivot operations. These techniques can be used in combination and implemented either inside or outside a relational database management system (RDBMS). If they are implemented inside an RDBMS, we will have more opportunities for query optimization [16]. Moreover, we can use them to speed up the maintenance of materialized views defined using pivot operations [22, 23].

*1) Filtering out EAV tuples related to the unneeded clinical parameters early on*

Consider a pivot operation transforming an EAV table to a conventional relational table. Usually, the clinical parameters needed in the pivot result are only a small subset of those appearing in the EAV table. Each entity appearing in the EAV table has a corresponding tuple in the outer pivot result table, but most of these tuples are all-null. For example, hundreds of thousands of clinical parameters exist in an EMR [14, page 56]. A relational table in a clinical data warehouse typically includes no more than a few dozen clinical parameters. For most clinical events, none of these clinical parameters was recorded [26].

All-null tuples occupy storage space, slow down query processing, and are useless for almost all clinical applications. It would be desirable to avoid all-null tuples introduced by outer pivot operations, and use inner pivot operations to generate conventional relational tables in a clinical data warehouse. If we ever need to obtain the all-null tuples that would be introduced by an outer pivot operation, we can always outer join an inner pivot result table with an entity table listing all entities.

One basic method for executing inner pivot is to conduct outer pivot and then filter out the all-null tuples introduced by the outer pivot operation [16]. This method is inefficient, as it repeatedly processes the EAV tuples related to the clinical parameters that appear in the EAV table, but are unneeded for the inner pivot result. A more efficient method for executing inner pivot is to filter out EAV tuples related to the unneeded clinical parameters early on.

This execution method can be implemented through pushing a filter below the inner pivot operator in the query execution plan. In both the sort-based approach and the hash-based approach, only EAV tuples related to the clinical parameters needed in the inner pivot result are retrieved and processed.

Let $p$ denote the percentage, based on total size, of the tuples in the EAV table $R$ that are related to the clinical parameters needed in the inner pivot result. Let $p'$ denote the percentage of pages in $R$ that contain at least one tuple related to the needed clinical parameters. The technique mentioned above reduces the disk I/O cost of the two-pass, sort-based or hash-based approach from $3|R|$ I/Os to $(q+2p)\cdot|R|$ I/Os:

(1) $q \cdot |R|$ to read the EAV tuples related to the needed clinical parameters into memory in the first pass. If the query optimizer chooses to perform table-scan on $R$, $q$=1. If an index exists on the attribute column of $R$ and the query optimizer chooses to perform index-scan on $R$, $q=p \leq 1$.
(2) $p \cdot |R|$ to write the EAV tuples related to the needed clinical parameters out to disk in the first pass.
(3) $p \cdot |R|$ to read the EAV tuples related to the needed clinical parameters into memory in the second pass.

Frequently, $p << 1$, $3/(q+2p) \geq 3/(1+2p) \approx 3$, and hence the disk I/O cost is reduced by almost three or more times.

For executing outer pivot, we can use a technique similar to the one mentioned above to speed up the sort-based approach. In the first pass of the sort-based approach, we can remove many EAV tuples related to unneeded clinical parameters early on as part of the in-memory sorting process. More specifically, during any stage of this sorting process, if we run into an EAV tuple $t$ with entity value $v$ related to an unneeded clinical parameter, and the EAV tuple $t'$ next to $t$ has the same entity value $v$, we throw $t$ away without further processing. The existence of $t'$ is sufficient for signaling that a conventional relational tuple needs to be generated for $v$ in the outer pivot result.

*2) Pivoting across multiple EAV tables*

An EMR or CSDMS often includes multiple EAV tables. To allow more compact storage as well as indexing by value, a separate EAV table is used for each data type [14, page 64, 26]. Also, special-purpose EAV tables can be used to store restricted kinds of homogeneous data, such as lab values, pharmacy orders and medications, surgical and medical interventions, and general clinical observations [27].

Consider a conventional relational table $S$ in a clinical data warehouse. The data of the clinical parameters included in $S$ are scattered across $m>1$ EAV tables $R_1$, $R_2$, …, and $R_m$ in an EMR or CSDMS. One basic method for generating $S$ from $R_1$, $R_2$, …, and $R_m$ is to compute and write to disk an inner pivot result table for each $R_i$ ($1 \leq i \leq m$). Then based on equality of the entity column, all inner pivot result tables are full outer joined together to form $S$. This requires performing $m$-1 joins, which often takes an excessive amount of time [21]. This is particularly the case if $m$ is not very small, as RDBMSs are inefficient at handling many joins in a single query.

A more efficient method for generating the conventional relational table $S$ from the EAV tables $R_1$, $R_2$, …, and $R_m$ is to support pivoting across multiple EAV tables [21]. In computing an inner pivot result table for each $R_i$ ($1 \leq i \leq m$), we skip the last step of producing the inner pivot result table and move on to generate $S$ directly. In this way, we save both the overhead of writing the inner pivot result table for each $R_i$ ($1 \leq i \leq m$) to disk and the overhead of conducting $m$-1 full outer joins among the $m$ inner pivot result tables. The larger the number $m$ of EAV tables, the greater performance advantage this method gains.

This method of supporting pivoting across multiple EAV tables can be implemented using either a sort-based approach or a hash-based approach. Either approach consists of two stages. The first stage processes each EAV table $R_i$ ($1 \leq i \leq m$) one by one to produce some intermediate computation results. The second stage handles the intermediate results computed from $R_1$, $R_2$, …, and $R_m$ simultaneously. The two stages of the sort-based approach match the two passes described in Section II-B-1, respectively. The two stages of the hash-based approach match the two passes described in Section II-B-2, respectively.

We first describe the sort-based approach. In the first stage, we perform the first pass of computing an inner pivot result table for each EAV table $R_i$ ($1 \leq i \leq m$) one by one. This produces and writes out to disk multiple sorted sub-lists of $R_i$. In the second stage, we allocate one buffer page for every sorted sub-list of each $R_i$ ($1 \leq i \leq m$). Then we simultaneously merge all sorted sub-lists of $R_1$, $R_2$, …, and $R_m$ to generate the conventional relational tuples in $S$. In comparison, the second pass described in Section II-B-1 merges the sorted sub-lists of only a single EAV table.

Next, we describe the hash-based approach. We use the same two hash functions $h_1$ and $h_2$ for each EAV table $R_i$ ($1 \leq i \leq m$). This ensures that in the first stage, those EAV tuples that have the same entity value, but come from different $R_i$, are put into buckets with the same bucket number. Also, in the second stage, those EAV tuples are put into the same bucket of the hash table. In the first stage, we perform the first pass of computing an inner pivot result table for each EAV table $R_i$ ($1 \leq i \leq m$) one by one. This produces and writes out to disk $M$-1 buckets of EAV tuples of $R_i$. In the second stage, we proceed in $M$-1 phases. In the $j$-th ($1 \leq j \leq M$-1) phase, we process the $j$-th bucket of each $R_i$ ($1 \leq i \leq m$) simultaneously to generate the conventional relational tuples in $S$ corresponding to this bucket. In comparison, in the $j$-th ($1 \leq j \leq M$-1) phase of the second pass described in Section II-B-2, we process the $j$-th bucket of only a single EAV table.

*3) Multi-query optimization*

Frequently, we need to perform multiple pivot operations on the same EAV table or across the same set of EAV tables [14, page 344]. Each pivot operation produces a separate, conventional relational table. For example, an EMR or CSDMS includes many forms. The data entered into these forms are stored in the same EAV table or the same set of EAV tables. For each form, a conventional relational table needs to be generated in a clinical data warehouse [14, page 344, 20].

Usually, those pivot operations on the same EAV table or across the same set of EAV tables share much work in common. This provides an opportunity for us to conduct multi-query optimization [28, 29] to avoid repeatedly performing this part of the work. In the rest of this section, we focus on the case that multiple pivot operations are performed on the same EAV table. The discussion with the case that multiple pivot operations are performed across the same set of EAV tables is similar and omitted.

In the first pass of the sort-based or hash-based approach of executing a pivot operation on an EAV table, the EAV table is often read into memory. For a large EAV table, this read will incur many disk I/Os. In a typical case, the pivot operation is an inner pivot one, the clinical parameters needed in the inner pivot result are only a small subset of those appearing in the

EAV table, and we can use the technique described in Section II-C-1 to filter out the EAV tuples related to the unneeded clinical parameters early on. Then most of the inner pivot operation's disk I/O cost will be spent on reading the EAV table into memory.

When multiple inner pivot operations are performed on the same EAV table, the EAV table will often be read into memory multiple times. This will repeatedly incur the same set of many disk I/Os and is inefficient. As shown in Fig. 4, a more efficient method of executing these inner pivot operations is to conduct multi-query optimization [28, 29] on them to let they share the work of reading the EAV table into memory. Then this work is performed only once rather than multiple times, and subsequently a large portion of the disk I/O cost is spared for all but one inner pivot operation. In the database area, the idea of letting multiple queries share the work of reading the same table into memory has been used before both in synchronized scan [30-32] and in multi-query optimization on other types of queries [28, 29]. The larger the number of inner pivot operations, the greater performance advantage the multi-query optimization method gains.

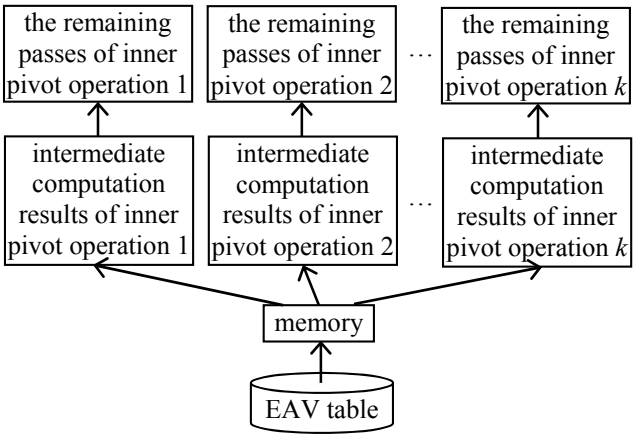



Fig. 4. Multiple inner pivot operations on the same EAV table share the work of reading the EAV table into memory.

To conduct multi-query optimization on multiple inner pivot operations on the same EAV table, we proceed as follows. For all of these inner pivot operations, the first pass of the sort-based or hash-based execution approach is run concurrently. A separate portion of the pages available in the buffer pool, rather than all pages available in the buffer pool, is allocated to each inner pivot operation to produce the intermediate computation results. For instance, the pages available in the buffer pool can be divided evenly among all of these inner pivot operations. For the sort-based approach, the intermediate computation results are sorted sub-lists of EAV tuples. For the hash-based approach, the intermediate computation results are buckets of EAV tuples. After the first pass is completed, the second pass of the sort-based or hash-based execution approach is run for each inner pivot operation one by one, using all pages available in the buffer pool.

## III. RESULTS

In this section, we present results from a prototype implementation of our techniques for improving the efficiency of pivot operations. As is the case with executing GROUP BY, the sort-based approach and the hash-based approach have similar performance in executing pivot operations [24, Chapters 15.4 and 15.5]. Our implementation used the sort-based approach.

### *A. Setup*

We conducted three experiments. In each experiment, we evaluated one of our three techniques for improving the efficiency of pivot operations, by comparing our optimized execution method of pivoting using this technique with the basic execution method of pivoting without using this technique. When measuring the execution time of an inner pivot query or the total execution time of a set of inner pivot queries, we excluded the time spent on outputting inner pivot result tuples, which was a constant independent of the method used to execute the query or the set of queries. In addition, we started executing the query or the set of queries on an unloaded system. This avoided any variable buffer pool caching effect that was left over from the execution of previous queries.

Each experiment included two sub-experiments. The first sub-experiment used one or more EAV tables containing synthetic data. Each EAV table had the same set of three integer attributes: (*event_ID*, *parameter_ID*, *value*), an index created on the *parameter_ID* attribute, and 512M tuples with a total size of 6GB.

The second sub-experiment used one or two EAV tables containing real clinical data from lab tests. The source of the clinical data was the Multiparameter Intelligent Monitoring in Intensive Care II (MIMIC II) database [33], which is publicly available at PhysioNet.org [34]. Each EAV table had the same set of three attributes: (*hospital_admission_ID*, *lab_test_ID*, *test_result_value*), an index created on the *lab_test_ID* attribute, and a total size of 6GB. Both *hospital_admission_ID* (entity) and *lab_test_ID* (attribute) were integer attributes. *test_result_value* (value) was always a float attribute except in one case. In one of the two EAV tables used in Sub-experiment 2.2, *test_result_value* was a variable-length character string attribute. The lab test results in the MIMIC II database came from around 36 thousand hospital stays and had a limited total size. To let each EAV table reach a total size of 6GB and still have the same data distribution as that in the MIMIC II database, we made multiple copies of the lab test result data in the MIMIC II database, performed one-by-one replacement of the hospital admission IDs in each copy so that each copy had a distinct set of hospital admission IDs, and combined the copies together to form the EAV tables.

Our measurements were performed on a Dell Inspiron 1525 PC with one 2-core 2GHz processor, 4GB main memory, one 109GB IDE disk, and running the Microsoft Windows Vista operating system.

#### *1) Setup of Experiment 1*

Using an EAV table, Experiment 1 tested our first technique of filtering out EAV tuples related to the unneeded clinical parameters early on. In the first sub-experiment, an inner pivot query was run on the EAV table to generate a conventional relational table including three clinical parameters. In the

second sub-experiment, an inner pivot query was run on the EAV table to generate a conventional relational table including multiple lab tests.

*2) Setup of Experiment 2*

Experiment 2 tested our second technique of supporting pivoting across multiple EAV tables. Both the basic and optimized execution methods of pivoting used our first technique of filtering out EAV tuples related to the unneeded clinical parameters early on.

In the first sub-experiment, we used $m$ EAV tables with each containing the same set of event IDs and unique parameter IDs. An inner pivot query was run on the $m$ EAV tables to generate a single conventional relational table, which included three clinical parameters from each of the $m$ EAV tables. In each of the $m$ EAV tables, every event ID was associated with the same set of three parameter IDs. Thus, all tuples in the EAV table were related to the clinical parameters needed in the inner pivot result.

In the second sub-experiment, we used two EAV tables. The first EAV table covered the lab tests with numeric results. Its *test_result_value* attribute was of float type. The second EAV table covered the lab tests with character string results. Its *test_result_value* attribute was of variable-length character string type. A first inner pivot query was run on the first EAV table to generate a conventional relational table, which included the 50 most frequently occurring lab tests with numeric results in the MIMIC II database. A second inner pivot query was run on both EAV tables to generate a single conventional relational table, which included the 50 most frequently occurring lab tests with numeric results as well as the 50 most frequently occurring lab tests with character string results in the MIMIC II database.

*3) Setup of Experiment 3*

Using an EAV table, Experiment 3 tested our third technique of conducting multi-query optimization. A set of $k$ inner pivot queries on the EAV table was submitted to the query execution system simultaneously. In the basic execution method of pivoting, the $k$ inner pivot queries were executed one by one. In our optimized execution method of pivoting, the $k$ inner pivot queries were executed concurrently using multi-query optimization. Both the basic and optimized execution methods of pivoting used our first technique of filtering out EAV tuples related to the unneeded clinical parameters early on.

In the first sub-experiment, every event ID was associated with the same set of parameter IDs in the EAV table. No two inner pivot queries shared any clinical parameter in common. Each of the $k$ inner pivot queries generated a conventional relational table including three clinical parameters. 10% of the tuples in the EAV table were related to the three clinical parameters.

In the second sub-experiment, each of the $k$ inner pivot queries generated a conventional relational table including three lab tests. The first inner pivot query used the three most frequently occurring lab tests with numeric results in the MIMIC II database. The second inner pivot query used the next three most frequently occurring lab tests with numeric results in the MIMIC II database, and so on. According to the data distribution in the MIMIC II database, 3.6% of the tuples in the EAV table are related to the most frequently occurring lab test with numeric result in the MIMIC II database. 1% of the tuples in the EAV table are related to the 30th most frequently occurring lab test with numeric result in the MIMIC II database.

*B. Test results for Experiment 1*

*1) Test results for Sub-experiment 1.1*

In the first sub-experiment, we varied $p$, the percentage of the tuples in the EAV table that are related to the clinical parameters needed in the inner pivot result, from 0 to 100%. Since each EAV tuple is of the same size, $p$ also represents the percentage, based on total size, of the tuples in the EAV table that are related to the clinical parameters needed in the inner pivot result.

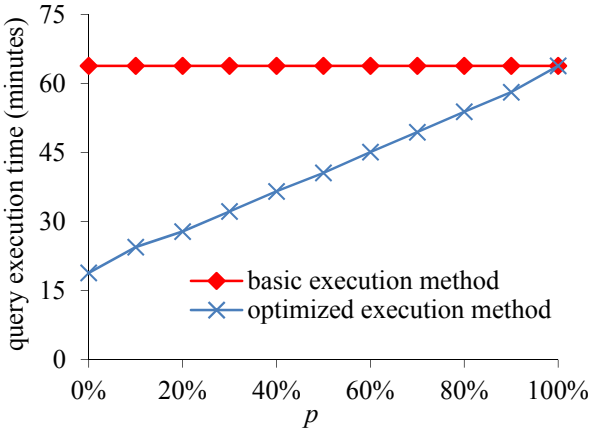



Fig. 5. Query execution time vs. $p$.

Fig. 5 shows the inner pivot query's execution time vs. $p$. The amount of time taken by the basic execution method of pivoting is basically a constant independent of $p$. In contrast, the amount of time taken by the optimized execution method of pivoting increases with $p$. When $p$<100% indicating that some tuples in the EAV table are related to the unneeded clinical parameters, the optimized execution method of pivoting always runs faster than the basic execution method of pivoting.

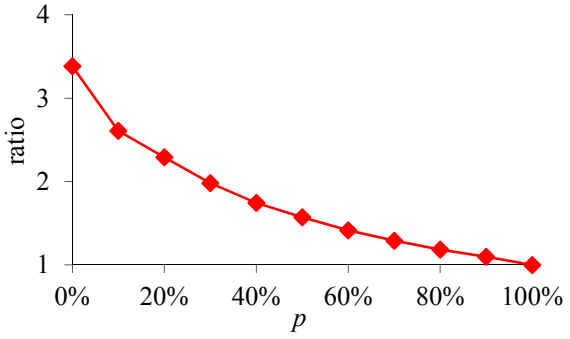



Fig. 6. The ratio of the amount of time taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. $p$.

Fig. 6 shows the ratio of the amount of time taken by the

basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. $p$. As explained by different approaches' disk I/O costs computed in Section II-C-1, the smaller the $p$, the greater performance advantage the optimized execution method of pivoting has over the basic execution method of pivoting.

*2) Test results for Sub-experiment 1.2*

In the second sub-experiment, we varied $n_t$, the number of lab tests needed in the inner pivot result, from two to ten. These $n_t$ lab tests are the $n_t$ most frequently occurring ones with numeric results in the MIMIC II database. The percentage of the tuples in the EAV table that are related to the lab tests needed in the inner pivot result increases with $n_t$.

Fig. 7 shows the inner pivot query's execution time vs. $n_t$. The amount of time taken by the basic execution method of pivoting is basically a constant independent of $n_t$. In contrast, the amount of time taken by the optimized execution method of pivoting increases with $n_t$. The optimized execution method of pivoting always runs faster than the basic execution method of pivoting, as most tuples in the EAV table are unrelated to the needed lab tests.

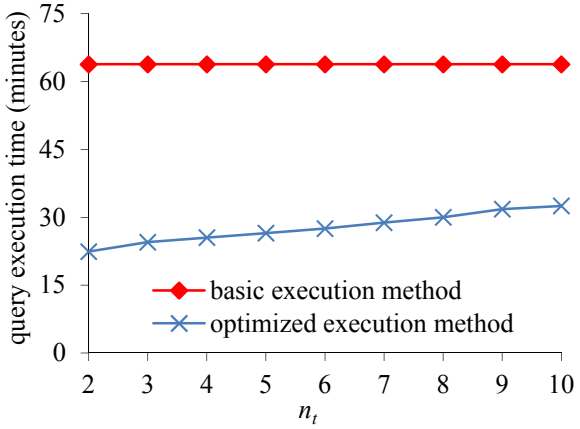



Fig. 7. Query execution time vs. $n_t$.

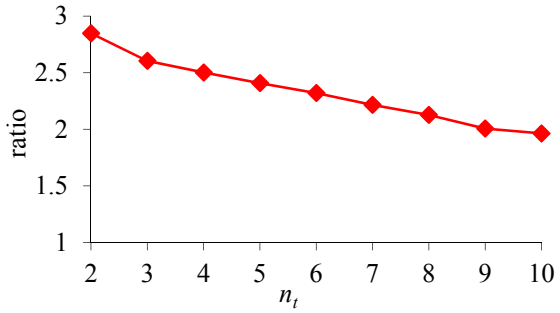



Fig. 8. The ratio of the amount of time taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. $n_t$.

Fig. 8 shows the ratio of the amount of time taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. $n_t$. The smaller the $n_t$, the greater performance advantage the optimized execution method of pivoting has over the basic execution method of pivoting.

*C. Test results for Experiment 2*

*1) Test results for Sub-experiment 2.1*

In the first sub-experiment, we varied $m$, the number of EAV tables, from 1 to 4. Fig. 9 shows the inner pivot query's execution time vs. $m$. Partly due to performing $m$-1 joins as described in Section II-C-2, the amount of time taken by the basic execution method of pivoting increases rapidly with $m$. In contrast, the amount of time taken by the optimized execution method of pivoting increases slowly with $m$. When $m$>1, the optimized execution method of pivoting always runs faster than the basic execution method of pivoting.

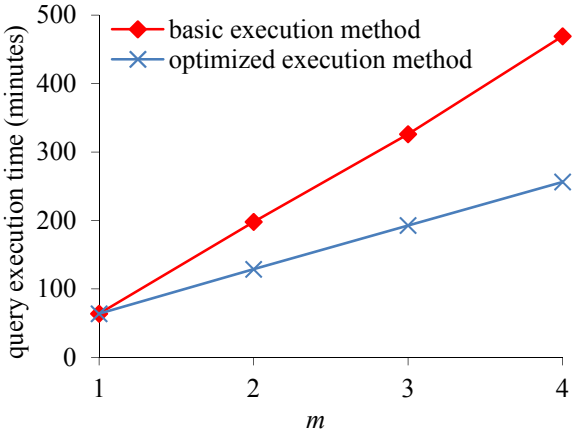



Fig. 9. Query execution time vs. $m$ in Sub-experiment 2.1.

Fig. 10 shows the ratio of the amount of time taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. $m$. As explained in Section II-C-2, the bigger the $m$, the greater performance advantage the optimized execution method of pivoting has over the basic execution method of pivoting.

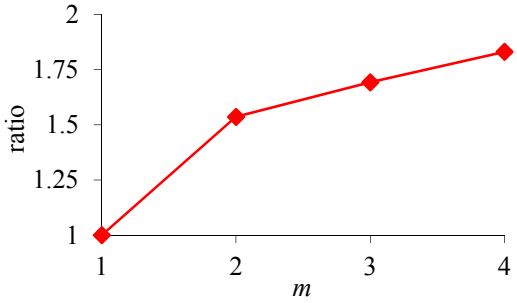



Fig. 10. The ratio of the amount of time taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. $m$ in Sub-experiment 2.1.

*2) Test results for Sub-experiment 2.2*

In the second sub-experiment, we varied $m$, the number of EAV tables on which the inner pivot query was run, from 1 to 2. Fig. 11 shows the inner pivot query's execution time vs. $m$. Fig. 12 shows the ratio of the amount of time taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. $m$. The trends shown in the second sub-experiment are the same as those shown in the first sub-experiment.

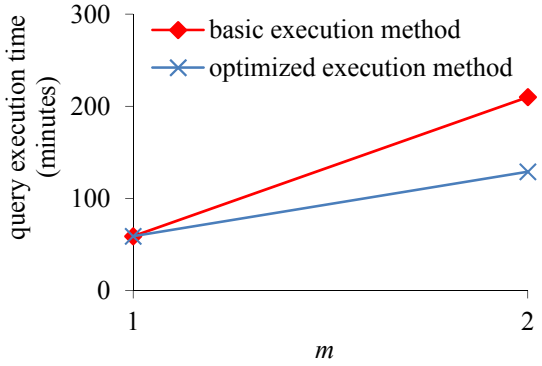



Fig. 11. Query execution time vs. *m* in Sub-experiment 2.2.

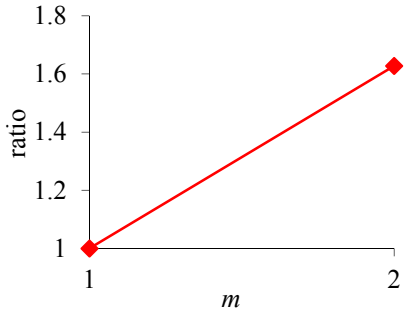



Fig. 12. The ratio of the amount of time taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. *m* in Sub-experiment 2.2.

*D. Test results for Experiment 3*

*1) Test results for Sub-experiment 3.1*

In the first sub-experiment, we varied *k*, the number of inner pivot queries on the same EAV table, from 1 to 10. Fig. 13 shows the total execution time of the set of inner pivot queries vs. *k*. Due to repeatedly reading the EAV table from disk into memory, the amount of time taken by the basic execution method of pivoting increases rapidly with *k*. In contrast, the amount of time taken by the optimized execution method of pivoting increases slowly with *k*. When *k*>1, the optimized execution method of pivoting always runs faster than the basic execution method of pivoting.

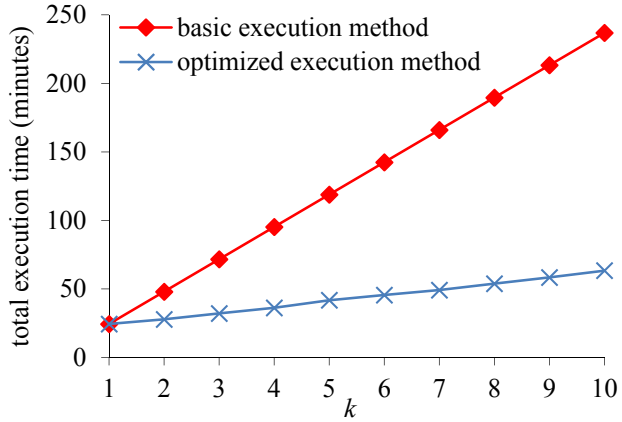



Fig. 13. Total execution time of the set of inner pivot queries vs. *k* in Sub-experiment 3.1.

Fig. 14 shows the ratio of the amount of time taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. *k*. As explained in Section II-C-3, the bigger the *k*, the greater performance advantage the optimized execution method of pivoting has over the basic execution method of pivoting.

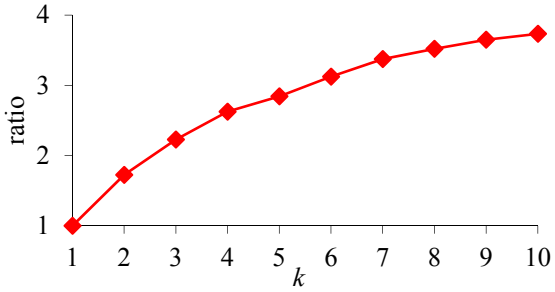



Fig. 14. The ratio of the amount of time taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. *k* in Sub-experiment 3.1.

*2) Test results for Sub-experiment 3.2*

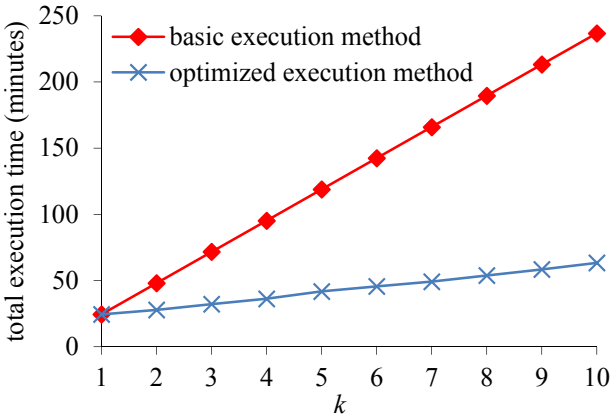



Fig. 15. Total execution time of the set of inner pivot queries vs. *k* in Sub-experiment 3.2.

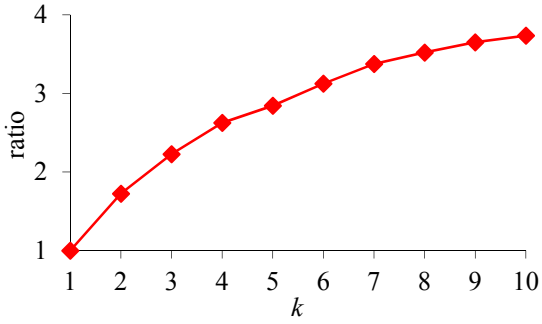



Fig. 16. The ratio of the amount of time taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. *k* in Sub-experiment 3.2.

In the second sub-experiment, we varied *k*, the number of inner pivot queries on the same EAV table, from 1 to 10. Fig. 15 shows the total execution time of the set of inner pivot queries vs. *k*. Fig. 16 shows the ratio of the amount of time

taken by the basic execution method of pivoting over the amount of time taken by the optimized execution method of pivoting vs. $k$. The trends shown in the second sub-experiment are the same as those shown in the first sub-experiment.

## IV. DISCUSSION AND CONCLUSIONS

We present three techniques for improving the efficiency of pivot operations. Through implementation, we show that our optimized execution method of pivoting using these techniques significantly outperforms the current basic execution method of pivoting. Our techniques can serve as the base of building a data extraction tool to simplify the specification of and improve the efficiency of extracting data from the EAV tables in EMRs and CSDMSs.

This paper focuses on the execution methods of pivoting. It has addressed neither query optimization in an RDBMS [16] nor maintenance of materialized views defined using pivot operations [22, 23]. Both query optimization and materialized view maintenance incorporating our three techniques are interesting areas for future work.

## Appendix

List of symbols

| | |
|---|---|
| $A$ | attribute column |
| $A_i$ $(1 \le i \le n)$ | a value of the attribute column $A$ |
| $B$ | bucket |
| $E$ | entity column |
| $h_1$, $h_2$ | hash function |
| $k$ | the number of inner pivot queries on the same EAV table |
| $M$ | the number of pages available in the buffer pool in memory |
| $m$ | for the clinical parameters included in a conventional relational table, the number of EAV tables across which they are scattered |
| $N$ | the number of sorted sub-lists |
| $n$ | the number of clinical parameters needed in the pivot result |
| $n_t$ | the number of lab tests needed in the inner pivot result |
| $p$ | the percentage, based on total size, of the tuples in the EAV table $R$ that are related to the clinical parameters needed in the inner pivot result |
| $p'$ | the percentage of pages in the EAV table $R$ that contain at least one tuple related to the clinical parameters needed in the inner pivot result |
| $R$, $R_i$ $(1 \le i \le m)$ | EAV table |
| $\|R\|$ | table $R$'s size in pages |
| $S$ | conventional relational table |
| $t$, $t'$ | EAV tuple |
| $t_v$ | conventional relational tuple for the entity value $v$ |
| $V$ | value column |
| $v$ | entity value |


## ACKNOWLEDGMENT

We thank Katherine Sward and Selena Thomas for helpful discussions.


## REFERENCES


[1] V. Dinu and P. M. Nadkarni, "Guidelines for the effective use of entity-attribute-value modeling for biomedical databases," *I. J. Medical Informatics*, vol. 76, no. 11-12, pp. 769-779, 2007.

[2] J. L. Beckmann, A. Halverson, R. Krishnamurthy, and J. F. Naughton, "Extending RDBMSs to support sparse datasets using an interpreted attribute storage format," in *Proc. ICDE'06*, 2006, 58.

[3] J. Corwin, A. Silberschatz, P. L. Miller, and L. Marenco, "Dynamic tables: an architecture for managing evolving, heterogeneous biomedical data in relational database management systems," *J. Am. Med. Inform. Assoc*., vol. 14, no. 1, pp. 86-93, 2007.

[4] C. J. McDonald, L. Blevins, W. M. Tierney, and D. K. Martin, "The Regenstrief medical records," *MD Computing*, vol. 5, no. 5, pp. 34–47, 1988.

[5] C. Friedman, G. Hripcsak, S. B. Johnson, J. J. Cimino, and P. D. Clayton, "A generalized relational schema for an integrated clinical patient database," in *Proc. Annu. Symp. Comput. Appl. Med. Care*, 1990, pp. 335–339.

[6] W. W. Stead, W. E. Hammond, and M. J. Straube, "A chartless record - Is it adequate?" in *Proc. Annu. Symp. Comput. Appl. Med. Care*, 1982, pp. 89–94.

[7] H. R. Warner, C. M. Olmsted, and B. D. Rutherford, "HELP - a program for medical decision-making," *Comput. Biomed. Res.*, vol. 5, no. 1, pp. 65–74, 1972.

[8] Cerner's electronic medical record homepage. https://www.cerner.com/solutions/Hospitals_and_Health_Systems/Electronic_Medical_Record/. Last accessed: May 4, 2014.

[9] Oracle Clinical homepage. http://www.oracle.com/us/products/applications/health-sciences/e-clinical/clinical/index.html. Last accessed: May 4, 2014.

[10] Oracle Health Sciences Clintrial homepage. http://www.oracle.com/us/industries/life-sciences/health-sciences-clintrial-363570.html. Last accessed: May 4, 2014.

[11] C. A. Brandt, P. Nadkarni, L. Marenco, B. T. Karras, C. Lu, L. Schacter, J. M. Fisk, and P. L. Miller, "Reengineering a database for clinical trials management: lessons for system architects," *Control Clin. Trials*., vol. 21, no. 5, pp. 440-61, 2000.

[12] https://metacpan.org/pod/CohortExplorer. Last accessed: May 4, 2014.

[13] P. A. Harris, R. Taylor, R. Thielke, J. Payne, N. Gonzalez, and J. G. Conde, "Research electronic data capture (REDCap) - a metadata-driven methodology and workflow process for providing translational research informatics support," *J. Biomed. Inform*., vol. 42, no. 2, pp. 377-381, 2009.

[14] P. M. Nadkarni, *Metadata-driven Software Systems in Biomedicine: Designing Systems that can Adapt to Changing Knowledge*. New York, NY, USA: Springer, 2011.

[15] R. Agrawal, A. Somani, and Y. Xu, "Storage and querying of E-commerce data," in *Proc. VLDB'01*, 2001, pp. 149-158.

[16] C. Cunningham, G. Graefe, and C. A. Galindo-Legaria, "PIVOT and UNPIVOT: optimization and execution strategies in an RDBMS," in *Proc. VLDB'04*, 2004, pp. 998-1009.

[17] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian, "On efficiently implementing SchemaSQL on an SQL database system," in *Proc. VLDB'99*, 1999, pp. 471-482.

[18] J. A. Lyman, K. Scully, and J. H. Jr. Harrison, "The development of health care data warehouses to support data mining," *Clin. Lab Med*., vol. 28, no. 1, pp. 55-71, 2008.

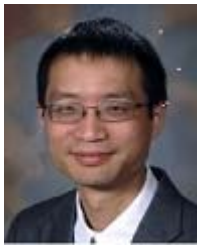
[19] R. S. Evans, J. F. Lloyd, and L. A. Pierce, "Clinical use of an enterprise data warehouse," in: *AMIA Annu. Symp. Proc*., 2012, pp. 189-98.

[20] C. Brandt, R. Morse, K. Matthews, K. Sun, A. M. Deshpande, R. Gadagkar, D. B. Cohen, P. L. Miller, and P. M. Nadkarni, "Metadata-driven creation of data marts from an EAV-modeled clinical research database," *I. J. Medical Informatics*, vol. 65, no. 3, pp. 225-241, 2002.

[21] V. Dinu, P. M. Nadkarni, and C. Brandt, "Pivoting approaches for bulk extraction of entity-attribute-value data," *Computer Methods and Programs in Biomedicine*, vol. 82, no. 1, pp. 38-43, 2006.

[22] B. Rashid and M. S. Islam, "Role of materialized view maintenance with PIVOT and UNPIVOT operators," in *Proc. IACC'09*, 2009, pp. 951-

955.
[23] S. Chen, and E. A. Rundensteiner, "GPIVOT: Efficient incremental maintenance of complex ROLAP views," in *Proc. ICDE'05*, 2005, pp. 552-563.
[24] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database Systems: the Complete Book*, 2nd Ed. Upper Saddle River, NJ, USA: Prentice Hall, 2008.
[25] J. S. Vitter, "Algorithms and data structures for external memory," *Foundations and Trends in Theoretical Computer Science*, vol. 2, no. 4, pp. 305-474, 2006.
[26] P. M. Nadkarni, L. N. Marenco, R. Chen, E. Skoufos, G. M. Shepherd, and P. L. Miller, "Organization of heterogeneous scientific data using the EAV/CR representation," *JAMIA*, vol. 6, no. 6, pp. 478-493, 1999.
[27] P. M. Nadkarni and C. Brandt, "Data extraction and ad hoc query of an entity-attribute-value database," *J. Am. Med. Inform. Assoc*., vol. 5, no. 6, pp. 511-27, 1998.
[28] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe, "Efficient and extensible algorithms for multi query optimization," in *Proc. SIGMOD'00*, 2000, pp. 249-260.
[29] T. K. Sellis, "Multiple-query optimization," *TODS*, vol. 13, no. 1, pp. 23-52, 1988.
[30] C. A. Lang, B. Bhattacharjee, T. Malkemus, S. Padmanabhan, and K. Wong, "Increasing buffer-locality for multiple relational table scans through grouping and throttling," in *Proc. ICDE'07*, 2007, pp. 1136-1145.
[31] C. A. Lang, B. Bhattacharjee, T. Malkemus, and K. Wong, "Increasing buffer-locality for multiple index based scans through intelligent placement and index scan speed control," in *Proc. VLDB'07*, 2007, pp. 1298-1309.
[32] G. Luo, J. F. Naughton, C. J. Ellmann, and M. W. Watzke, "Transaction reordering," *Data and Knowledge Engineering*, vol. 69, no. 1, pp. 29-49, 2010.
[33] M. Saeed, M. Villarroel, A. T. Reisner, G. Clifford, L. W. Lehman, G. Moody, T. Heldt, T. H. Kyaw, B. Moody, and R. G. Mark, "Multiparameter Intelligent Monitoring in Intensive Care II: a public-access intensive care unit database," *Crit. Care Med*., vol. 39, no. 5, pp. 952-60, 2011.
[34] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215-e220, 2000.



**Gang Luo** received the BSc degree in computer science from Shanghai Jiaotong University, Shanghai, P.R. China, in 1998, and the PhD degree in computer science from the University of Wisconsin-Madison, Madison, WI, USA, in 2004. He is an Assistant Professor in the Department of Biomedical Informatics at the University of Utah, Salt Lake City, UT, USA. From 2004 to 2012, he was a research staff member at IBM T.J. Watson Research Center. His research interests include health informatics (system building and data analysis), database, information retrieval, natural language processing, machine learning, data mining, and brain-computer interface.

**Lewis J. Frey** received the BS degree in mathematics from the University of Pittsburg, Pittsburg, PA, USA, in 1992, the MS degree in computer science from Vanderbilt University, Nashville, TN, USA, in 1994, and the PhD degree in computer science from Vanderbilt University in 2003. He is an Associate Professor in the Department of Public Health Sciences at the Medical University of South Carolina, Charleston, SC, USA. From 2003 to 2006, he was a Post Doctoral Fellow in the Department of Biomedical Informatics at Vanderbilt University. From June 2006 to April 2014, he was an Assistant Professor in the Department of Biomedical Informatics at the University of Utah, Salt Lake City, UT, USA. His research interests include biomedical informatics, machine learning, data mining, data integration, and big data analysis.