International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining Analysis

.R.Rajgowri[1], A.Archana[2],T.Srinithi[3],M.Hemapriya[4]

Muthayammal Engineering College, Rasipuram, Tamil Nadu, India

Department of Electronics and Communication Engineering, Muthayammal College of Engineering, Rasipuram, Tamil Nadu, India

Publication History: Received: 25.02.2026; Revised: 20.03.2026; Accepted: 25.03. 2026; Published: 28.03.2026.

ABSTRACT: Preparing a data set for analysis is generally the most time consuming task in a data mining project, requiring many complex SQL queries, joining tables, and aggregating columns. Existing SQL aggregations have limitations to prepare data sets because they return one column per aggregated group. In general, a significant manual effort is required to build data sets, where a horizontal layout is required. We propose simple, yet powerful, methods to generate SQL code to return aggregated columns in a horizontal tabular layout, returning a set of numbers instead of one number per row. This new class of functions is called horizontal aggregations. Horizontal aggregations build data sets with a horizontal denormalized layout (e.g.,point-dimension, observationvariable, instance-feature), which is the standard layout required by most data mining algorithms. We propose three fundamental methods to evaluate horizontal aggregations: CASE: Exploiting the programming CASE construct; SPJ: Based on standard relational algebra operators (SPJ queries); PIVOT: Using the PIVOT operator, which is offered by some DBMSs. Experiments with large tables compare the proposed query evaluation methods. Our CASE method has similar speed to the PIVOT operator and it is much faster than the SPJ method. In general, the CASE and PIVOT methods exhibit linear scalability, whereas the SPJ method does not.

KEYWORDS: Horizontal aggregations, SQL queries, data mining, dataset preparation, data transformation, relational databases, OLAP operations, feature engineering, big data analytics, query optimization

I. INTRODUCTION

IN a relational database, especially with normalized tables, a significant effort is required to prepare a summary data set [16] that can be used as input for a data mining orstatistical algorithm [17], [15]. Most algorithms require as input a data set with a horizontal layout, with several records and one variable or dimension per column. That is the case with models like clustering, classification, regression, and PCA; consult [10], [15]. Each research discipline uses different terminology to describe the data set. In data mining the common terms are point-dimension. Statistics literature generally uses observation-variable. Machine learning research uses instance-feature. This paper introduces a new class of aggregate functions that can be used to build data sets in a horizontal layout (denormalized with aggregations), automating SQL query writing and extending SQL capabilities. We show evaluating horizontal aggregations is a challenging and interesting problem and we introduce alternative methods and optimizations for their efficient evaluation.

1.1 Motivation

As mentioned above, building a suitable data set for data mining purposes is a time-consuming task. This task generally requires writing long SQL statements or customizing SQL code if it is automatically generated by some tool. There are two main ingredients in such SQL code: joins and aggregations [16]; we focus on the second one. The most widely known aggregation is the sum of a column over groups of rows. Some other aggregations return theaverage, maximum, minimum, or row count over groups of rows. There exist many aggregation functions and operators in SQL. Unfortunately, all these aggregations have limitations to build data sets for data mining purposes. The main reason is that, in general, data sets that are stored in a relational database (or a data warehouse) come from Online Transaction Processing (OLTP) systems where database schemas are highly normalized. But data mining, statistical, or machine learning algorithms generally require aggregated data in summarized form. Based on current available functions and clauses in SQL, a significant effort is required to compute aggregations when they are desired in a crosstabular (horizontal) form, suitable to be used by a data mining algorithm. Such effort is due to the amount and complexity of

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

SQL code that needs to be written, optimized, and tested. There are further practical reasons to return aggregation results in a horizontal (cross-tabular) layout. Standard aggregations are hard to interpret when there are many result rows, especially when grouping attributes have high cardinalities. To perform analysis of exported tables into spreadsheets it may be more convenient to have aggregations on the same group in one row (e.g., to produce graphs or to compare data sets with repetitive information). OLAP tools generate SQL code to transpose results (sometimes called PIVOT [5]). Transposition can be more efficient if there are mechanisms combining aggregation and transposition together. With such limitations in mind, we propose a new class of aggregate functions that aggregate numeric expressions and transpose results to produce a data set with a horizontal layout. Functions belonging to this class are called horizontal aggregations. Horizontal aggregations represent an extended form of traditional SQL aggregations, which return a set of values in a horizontal layout (somewhat similar to a multidimensional vector), instead of a single value per row. This paper explains how to evaluate and optimize horizontal aggregations generating standard SQL code. 678 IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 4, APRIL 2012. The authors are with the Department of Computer Science, University of Houston, Houston, TX 77204. E-mail: {ordonez, zchen6}@cs.uh.edu. Manuscript received 17 Nov. 2009; revised 2 June 2010; accepted 16 Aug. 2010; published online 23 Dec. 2010. Recommended for acceptance by T. Grust. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2009-11-0789. Digital Object Identifier no. 10.1109/TKDE.2011.16. 1041-4347/12/$31.00 _ 2012 IEEE Published by the IEEE Computer Society

1.2 Advantages

Our proposed horizontal aggregations provide several unique features and advantages. First, they represent a template to generate SQL code from a data mining tool. Such SQL code automates writing SQL queries, optimizing them, and testing them for correctness. This SQL code reduces manual work in the data preparation phase in a data mining project. Second, since SQL code is automatically generated it is likely to be more efficient than SQL code written by an end user. For instance, a person who does not know SQL well or someone who is not familiar with the database schema (e.g., a data mining practitioner). Therefore, data sets can be created in less time. Third, the data set can be created entirely inside the DBMS. In modern database environments, it is common to export denormalized data sets to be further cleaned and transformed outside a DBMS in external tools (e.g., statistical packages). Unfortunately, exporting large tables outside a DBMS is slow, creates inconsistent copies of the same data and compromises database security. Therefore, we provide a more efficient, better integrated and more secure solution compared to external data mining tools. Horizontal aggregations just require a small syntax extension to aggregate functions called in a SELECT statement. Alternatively, horizontal aggregations can be used to generate SQL code from a data mining tool to build data sets for data mining analysis.

1.3 Paper Organization

This paper is organized as follows: Section 2 introduces definitions and examples. Section 3 introduces horizontal aggregations. We propose three methods to evaluate horizontal aggregations using existing SQL constructs, we prove the three methods produce the same result and we analyze time complexity and I/O cost. Section 4 presents experiments comparing evaluation methods, evaluating the impact of optimizations, assessing scalability, and understanding I/O cost with large tables. Related work is discussed in Section 5, comparing our proposal with existing approaches and positioning our work within data preparation and OLAP query evaluation. Section 6 gives conclusions and directions for future work.

II. DEFINITIONS

This section defines the table that will be used to explain SQL queries throughout this work. In order to present definitions and concepts in an intuitive manner, we present our definitions in OLAP terms. Let F be a table having a simple primary key K represented by an integer, p discrete attributes, and one numeric attribute: FðK;D1; . . .;Dp;AÞ. Our definitions can be easily generalized to multiple numeric attributes. In OLAP terms, F is a fact table with one column used as primary key, p dimensions and one measure column passed to standard SQL aggregations. That is, table F will be manipulated as a cube with p dimensions [9]. Subsets of dimension columns are used to group rows to aggregate the measure column. F is assumed to have a star schema to simplify exposition. Column K will not be used to compute aggregations. Dimension lookup tables will be based on simple foreign keys. That is, one dimension column Dj will be a foreign key linked to a lookup table that has Dj as primary key. Input table F size is called N (not to be confused with n, the size of the answer set). That is, jFj ¼ N. Table F represents a temporary table or a view based on a "star join" query on several tables. We now explain tables FV (vertical) and FH (horizontal) that are used throughout

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

the paper. Consider a standard SQL aggregation (e.g., sum()) with the GROUP BY clause, which returns results in a vertical layout. Assume there are j þ k GROUP BY columns and the aggregated attribute is A. The results are stored on table FV having j þ k columns making up the primary key and A as a nonkey attribute. Table FV has a vertical layout. The goal of a horizontal aggregation is to transform FV into a table FH with a horizontal layout having n rows and j þ d columns, where each of the d columns represents a unique combination of the k grouping columns. Table FV may be more efficient than FH to handle sparse matrices (having many zeroes), but some DBMSs like SQL Server [2] can handle sparse columns in a horizontal layout. The n rows represent records for analysis and the d columns represent dimensions or features for analysis. Therefore, n is data set size and d is dimensionality. In other words, each aggregated column represents a numeric variable as defined in statistics research or a numeric feature as typically defined in machine learning research.

2.1 Examples

Fig. 1 gives an example showing the input table F, a traditional vertical sum() aggregation stored in FV , and a horizontal aggregation stored in FH. The basic SQL aggregation query is:

SELECT D1;D2,sum(A)
FROM F
GROUP BY D1;D2
ORDER BY D1;D2;

ORDONEZ AND CHEN: HORIZONTAL AGGREGATIONS IN SQL TO PREPARE DATA SETS FOR DATA MINING ANALYSIS 679

Fig. 1. Example of F, FV , and FH.

Notice table FV has only five rows because D1 ¼ 3 and D2 ¼ Y do not appear together. Also, the first row in FV has null in A following SQL evaluation semantics. On the other hand, table FH has three rows and two (d ¼ 2) nonkey columns, effectively storing six aggregated values. In FH it is necessary to populate the last row with null. Therefore, nulls may come from F or may be introduced by the horizontal layout. We now give other examples with a store (retail) database that requires data mining analysis. To give examples of F, we will use a table transactionLine that represents the transaction table from a store. Table transactionLine has dimensions grouped in three taxonomies (product hierarchy, location, time), used to group rows, and three measures represented by itemQty, costAmt, and salesAmt, to pass as arguments to aggregate functions. We want to compute queries like “summarize sales for each store by each day of the week”; “compute the total number of items sold by department for each store.” These queries can be answered with standard SQL, but additional code needs to be written or generated to return results in tabular (horizontal) form. Consider the following two queries:

```
SELECT storeId,dayofweekNo,sum(salesAmt)
FROM transactionLine
GROUP BY storeId,dayweekNo
ORDER BY storeId,dayweekNo;
SELECT storeId,deptId,sum(itemqty)
FROM transactionLine
GROUP BY storeId,deptId
ORDER BY storeId,deptId;
```

Assume there are 200 stores, 30 store departments, and stores are open 7 days a week. The first query returns 1,400 rows which may be time consuming to compare with each other each day of the week to get trends. The second query returns 6,000 rows, which in a similar manner, makes difficult to compare store performance across departments. Even further, if we want to build a data mining model by store (e.g., clustering, regression), most algorithms require store id as primary key and the remaining aggregated columns as nonkey columns. That is, data mining algorithms expect a horizontal layout. In addition, a horizontal layout is generally more I/O efficient than a vertical layout for analysis. Notice these queries have ORDER BY clauses to make output easier to understand, but such order is irrelevant for data mining algorithms. In general, we omit ORDER BY clauses.

2.2 Typical Data Mining Problems

Let us consider data mining problems that may be solved by typical data mining or statistical algorithms, which assume each nonkey column represents a dimension, variable (statistics), or feature (machine learning). Stores can be clustered based on sales for each day of the week. On the other hand, we can predict sales per store department based on the sales in other departments using decision trees or regression. PCA analysis on department sales can reveal which

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

departments tend to sell together. We can find out potential correlation of number of employees by gender within each department. Most data mining algorithms (e.g.,clustering, decision trees, regression, correlation analysis) require result tables from these queries to be transformed into a horizontal layout. We must mention there exist data mining algorithms that can directly analyze data sets
having a vertical layout (e.g., in transaction format) [14], but they require reprogramming the algorithm to have a better I/O pattern and they are efficient only when there many zero values (i.e., sparse matrices).

III HORIZONTAL AGGREGATIONS

We introduce a new class of aggregations that have similar behavior to SQL standard aggregations, but which produce tables with a horizontal layout. In contrast, we call standard SQL aggregations vertical aggregations since they produce tables with a vertical layout. Horizontal aggregations just require a small syntax extension to aggregate functions called in a SELECT statement. Alternatively, horizontal aggregations can be used to generate SQL code from a data mining tool to build data sets for data mining analysis. We start by explaining how to automatically generate SQL code.

3.1 SQL Code Generation

Our main goal is to define a template to generate SQL code combining aggregation and transposition (pivoting). A second goal is to extend the SELECT statement with a clause that combines transposition with aggregation. Consider the following GROUP BY query in standard SQL that
takes a subset L1; . . . ; Lm from D1; . . .;Dp:
SELECT L1; ::;Lm, sum(A)
FROM F
GROUP BY L1; . . . ; Lm;

This aggregation query will produce a wide table with m þ 1 columns (automatically determined), with one group for each unique combination of values L1; . . . ; Lm and one aggregated value per group (sum(A) in this case). In order to evaluate this query the query optimizer takes three input parameters: 1) the input table F, 2) the list of grouping columns L1; . . . ; Lm, 3) the column to aggregate (A). The basic goal of a horizontal aggregation is to transpose (pivot) the aggregated column A by a column subset of L1; . . . ; Lm; for simplicity assume such subset is R1; . . .;Rk where k < m. In other words, we partition the GROUP BY list into two sublists: one list to produce each group (j columns L1; . . . ; Lj) and another list (k columns R1; . . .;Rk) to transpose aggregated values, where fL1; . . . ; Ljg \ fR1; . . .;Rkg ¼ ;. Each distinct combination of fR1; . . .;Rkg will automatically produce an output column. In particular, if k ¼ 1 then there are j_R1 ðFÞj columns (i.e., each value in R1 becomes a column storing one aggregation). Therefore, in a horizontal aggregation there are four input parameters to generate SQL code:

1. the input table F,
2. the list of GROUP BY columns L1; . . . ; Lj,
3. the column to aggregate (A),
4. the list of transposing columns R1; . . .;Rk. Horizontal aggregations preserve evaluation semantics of standard (vertical) SQL aggregations. The main difference will be returning a table with a horizontal layout, possibly having extra nulls. The SQL code generation aspect is 680 IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 4, APRIL 2012 explained in technical detail in Section 3.4. Our definition allows a straightforward generalization to transpose multiple aggregated columns, each one with a different list of transposing columns.

3.2 Proposed Syntax in Extended SQL

We now turn our attention to a small syntax extension to the SELECT statement, which allows understanding our proposal in an intuitive manner. We must point out the proposed extension represents nonstandard SQL because the columns in the output table are not known when the query is parsed. We assume F does not change while a horizontal aggregation is evaluated because new values may create new result columns. Conceptually, we extend standard SQL aggregate functions with a “transposing” BY clause followed by a list of columns (i.e., R1; . . .;Rk), to produce a horizontal set of numbers instead of one number. Our proposed syntax is as follows:

SELECT L1; ::; Lj, HðABYR1; . . .;RkÞ
FROM F
GROUP BY L1; . . . ; Lj;

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

We believe the subgroup columns R1; . . .;Rk should be a parameter associated to the aggregation itself. That is why they appear inside the parenthesis as arguments, but alternative syntax definitions are feasible. In the context of our work, HðÞ represents some SQL aggregation (The function HðÞ must have at least one argument represented by A, followed by a list of columns. The result rows are determined by columns L1; . . . ; Lj in the GROUP BY clause if present. Result columns are determined by all potential combinations of columns R1; . . .;Rk, where k ¼ 1 is the default. Also, fL1; . . . ; Ljg \ fR1; . . .;Rkg ¼ ; We intend to preserve standard SQL evaluation semantics as much as possible. Our goal is to develop sound and efficient evaluation mechanisms. Thus, we propose the following rules.

1. The GROUP BY clause is optional, like a vertical aggregation. That is, the list L1; . . . ; Lj may be empty. When the GROUP BY clause is not present then there is only one result row. Equivalently, rows can be grouped by a constant value (e.g., L1 ¼ 0) to always include a GROUP BY clause in code generation.
2. When the clause GROUP BY is present there should not be a HAVING clause that may produce cross tabulation of the same group (i.e., multiple rows with aggregated values per group).
3. The transposing BY clause is optional. When BY is not present then a horizontal aggregation reduces to a vertical aggregation.
4. When the BY clause is present the list R1; . . .;Rk is required, where k ¼ 1 is the default.
5. Horizontal aggregations can be combined with vertical aggregations or other horizontal aggregations on the same query, provided all use the same GROUP BY columns fL1; . . . ; Ljg.
6. As long as F does not change during query processing horizontal aggregations can be freely combined. Such restriction requires locking [11], which we will explain later.
7. The argument to aggregate represented by A is required; A can be a column name or an arithmetic expression. In the particular case of countðÞ A can be the “DISTINCT” keyword followed by the list of columns.
8. When HðÞ is used more than once, in different terms, it should be used with different sets of BY columns.

3.2.1 Examples

In a data mining project, most of the effort is spent in preparing and cleaning a data set. A big part of this effort involves deriving metrics and coding categorical attributes from the data set in question and storing them in a tabular (observation, record) form for analysis so that they can be used by a data mining algorithm. Assume we want to summarize sales information with one store per row for one year of sales. In more detail, we need the sales amount broken down by day of the week, the number of transactions by store per month, the number of items sold by department and total sales. The following query in our extended SELECT syntax provides the desired data set, by calling three horizontal aggregations.

```
SELECT
storeId,
sum(salesAmt BY dayofweekName),
count(distinct transactionid BY salesMonth),
sum(1 BY deptName),
sum(salesAmt)
FROM transactionLine
,DimDayOfWeek,DimDepartment,DimMonth
WHERE salesYear¼2009
AND transactionLine.dayOfWeekNo
¼DimDayOfWeek.dayOfWeekNo
AND transactionLine.deptId
¼DimDepartment.deptId
AND transactionLine.MonthId
¼DimTime.MonthId
GROUP BY storeId;
```

This query produces a result table like the one shown inTable 1. Observe each horizontal aggregation effectively returns a set of columns as result and there is call to a standard vertical aggregation with no sub grouping columns. For the first horizontal aggregation, we show day names and for the second one we show the number of day of the week. These columns can be used for linear regression, clustering, or factor analysis. We can analyze correlation of sales based on daily sales. Total sales can be predicted based on volume of items sold each day of the week. Stores can be clustered based on similar sales for each day of the week or similar sales in the same department. Consider a more complex example where we want to know for each store subdepartment how sales compare for each region-month

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

showing total sales for each region/ month combination. Subdepartments can be clustered based on similar sales amounts for each region/month combination. We assume all stores in all regions have the same departments, but local preferences lead to different buying patterns. This query in our extended SELECT builds the required data set:

ORDONEZ AND CHEN: HORIZONTAL AGGREGATIONS IN SQL TO PREPARE DATA SETS FOR DATA MINING ANALYSIS 681

```
SELECT subdeptid,
sum(salesAmt BY regionNo,monthNo)
FROM transactionLine
GROUP BY subdeptId;
```

We turn our attention to another common data preparation task, transforming columns with categorical attributes into binary columns. The basic idea is to create a binary
dimension for each distinct value of a categorical attribute. This can be accomplished by simply calling maxð1BY::Þ, grouping by the appropriate columns. The next query produces a vector showing 1 for the departments where the customer made a purchase, and 0 otherwise.

```
SELECT
transactionId,
max(1 BY deptId DEFAULT 0)
FROM transactionLine
GROUP BY transactionId;
```

3.3 SQL Code Generation:
Locking and Table Definition
In this section, we discuss how to automatically generate efficient SQL code to evaluate horizontal aggregations. Modifying the internal data structures and mechanisms of the query optimizer is outside the scope of this paper, but we give some pointers. We start by discussing the structure of the result table and then query optimization methods to populate it. We will prove the three proposed evaluation methods produce the same result table FH.

3.3. Locking
In order to get a consistent query evaluation it is necessary to use locking [7], [11]. The main reasons are that any insertion into F during evaluation may cause inconsistencies:
1) it can create extra columns in FH, for a new combination of R1; . . .;Rk; 2) it may change the number of rows of FH, for a new combination of L1; . . . ; Lj; 3) it may change actual aggregation values in FH. In order to return consistent answers, we basically use table-level locks on F, FV , and FH acquired before the first statement starts and released after FH has been populated. In other words, the entire set of SQL statements becomes a long transaction. We use the highest SQL isolation level: SERIALIZABLE. Notice an alternative simpler solution would be to use a static (read-only) copy of F during query evaluation. That is, horizontal aggregations can operate on a read-only database
without consistency issues.

3.3.2 Result Table Definition
Let the result table be FH. Recall from Section 2 FH has d aggregation columns, plus its primary key. The horizontal aggregation function HðÞ returns not a single value, but a set of values for each group L1; . . . ; Lj. Therefore, the result table FH must have as primary key the set of grouping columns fL1; . . . ; Ljg and as nonkey columns all existing combinations of values R1; . . .;Rk. We get the distinct value combinations of R1; . . .;Rk using the following statement.

```
SELECT DISTINCT R1; :::;Rk
```

FROM F; Assume this statement returns a table with d distinct rows. Then each row is used to define one column to store an aggregation for one specific combination of dimension values. Table FH that has fL1; . . . ; Ljg as primary key and d columns corresponding to each distinct subgroup. Therefore, FH has d columns for data mining analysis and j þ d columns in total, where each Xj corresponds to one aggregated value based on a specific R1; . . .;Rk values combination.

```
CREATE TABLE FH(
L1 int
,. . .
,Lj int
```

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

,X1 real
,. . .
,Xd real
) PRIMARY KEY(L1; . . . ; Lj);
3.4 SQL Code Generation:
Query Evaluation Methods
We propose three methods to evaluate horizontal aggregations. The first method relies only on relational operations. That is, only doing select, project, join, and aggregation queries; we call it the SPJ method. The second form relies on the SQL "case" constructs; we call it the CASE method. Each table has an index on its primary key for efficient join processing. We do not consider additional indexing mechanisms to accelerate query evaluation. The third method uses the built-in PIVOT operator, which transforms rows to columns (e.g., transposing). Figs. 2 and 3 show an overview of the main steps to be explained below (for a sum() aggregation).

3.4. SPJ Method
The SPJ method is interesting from a theoretical point of view because it is based on relational operators only. The basic idea is to create one table with a vertical aggregation for each result column, and then join all those tables to produce FH. We aggregate from F into d projected tables with d Select-Project-Join-Aggregation queries (selection, projection, join, aggregation). Each table FI corresponds to one sub grouping combination and has fL1; . . . ; Ljg as primary key and an aggregation on A as the only nonkey 682 IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 4, APRIL 2012 TABLE 1 A Multidimensional Data Set in Horizontal Layout, Suitable for Data Mining column. It is necessary to introduce an additional table F0, that will be outer joined with projected tables to get a complete result set. We propose two basic sub strategies to compute FH. The first one directly aggregates from F. The second one computes the equivalent vertical aggregation in a temporary table FV grouping by L1; . . . ; Lj;R1; . . .;Rk. Then horizontal aggregations can be instead computed from FV , which is a compressed version of F, since standard aggregations are distributive [9]. We now introduce the indirect aggregation based on the intermediate table FV , that will be used for both the SPJ and
the CASE method. Let FV be a table containing the vertical aggregation, based on L1; . . . ; Lj;R1; . . .;Rk. Let V() representthe corresponding vertical aggregation for HðÞ. The statement to compute FV gets a cube:

INSERT INTO FV
SELECT L1; . . . ; Lj;R1; . . .;Rk, V(A)
FROM F
GROUP BY L1; . . . ; Lj;R1; . . .;Rk;
Table F0 defines the number of result rows, and builds the primary key. F0 is populated so that it contains every existing combination of L1; . . . ; Lj. Table F0 has fL1; . . . ; Ljg as primary key and it does not have any nonkey

column.
INSERT INTO F0
SELECT DISTINCT L1; . . . ; Lj
FROM fFjFV g;
In the following discussion I 2 f1; . . . ; dg: we use h to make writing clear, mainly to define Boolean expressions. We need to get all distinct combinations of subgrouping
Columns R1; . . .; Rk, to create the name of dimension columns, to get d, the number of dimensions, and to generate the Boolean expressions for WHERE clauses. Each
WHERE clause consists of a conjunction of k equalities
based on R1; . . .;Rk.
SELECT DISTINCT R1; . . .;Rk
FROM fFjFV g;
Tables F1; . . . ; Fd contain individual aggregations for
each combination of R1; . . .;Rk. The primary key of table FI
is fL1; . . . ; Ljg.
INSERT INTO FI
SELECT L1; . . . ; Lj;
FROM fFjFV g
WHERE R1 ¼ v1I AND .. AND Rk ¼ vkI
GROUP BY L1; . . . ; Lj;

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

Then each table FI aggregates only those rows that correspond to the Ith unique combination of R1; . . .;Rk, given by the WHERE clause. A possible optimization is synchronizing table scans to compute the d tables in one pass. Finally, to get FH we need d left outer joins with the d þ 1 tables so that all individual aggregations are properly assembled as a set of d dimensions for each group. Outer joins set result columns to null for missing combinations for the given group. In general, nulls should be the default Propose two basic sub strategies to compute FH. In a similar manner to SPJ, the first one directly aggregates from F and the second one computes the vertical aggregation in a temporary table FV and then horizontal aggregations are indirectly computed from FV . We now present the direct aggregation method. Horizontal aggregation queries can be evaluated by directly aggregating from F and transposing rows at the same time to produce FH. First, we need to get the unique combinations of R1; . . .;Rk that define the matching boolean expression for result columns. The SQL code to compute horizontal aggregations directly from F is as follows: observe V ðÞ is a standard (vertical) SQL aggregation that has a "case" statement as argument. Horizontal aggregations need to set the result to null when there are no qualifying rows for the specific horizontal group to be consistent with the SPJ method and also with the extended

relational model [4].

```
SELECT DISTINCT R1; . . .;Rk
FROM F;
INSERT INTO FH
SELECT L1; . . . ; Lj
,V(CASE WHEN R1 ¼ v11 and . . . and Rk ¼ vk1
THEN A ELSE null END)
..
,V(CASE WHEN R1 ¼ v1d and . . . and Rk ¼ vkd
THEN A ELSE null END)
FROM F
GROUP BY L1; L2; . . . ; Lj;
```

This statement computes aggregations in only one scan non F. The main difficulty is that there must be a feedback process to produce the "case" boolean expressions. We now consider an optimized version using FV . Based on FV , we need to transpose rows to get groups based on L1; . . . ; Lj. Query evaluation needs to combine the desired aggregation with "CASE" statements for each distinct combination of values of R1; . . .;Rk. As explained above, horizontal aggregations must set the result to null when there are no qualifying rows for the specific horizontal group. The boolean expression for each case statement has a conjunction of k equality comparisons. The following statements compute FH:

```
SELECT DISTINCT R1; . . .;Rk
FROM FV ;
INSERT INTO FH
SELECT L1,..,Lj
,sum(CASE WHEN R1 ¼ v11 and .. and Rk ¼ vk1
THEN A ELSE null END)
..
, sum (CASE WHEN R1 ¼ v1d and .. and Rk ¼ vkd
THEN A ELSE null END)
FROM FV
GROUP BY L1; L2; . . . ; Lj;
```

As can be seen, the code is similar to the code presented before, the main difference being that we have a call to sum ð Þ in each term, which preserves whatever values were previously computed by the vertical aggregation. It has the disadvantage of using two tables instead of one as required by the direct computation from F. For very large tables F computing FV first, may be more efficient than computing directly from F.

3.4.3 PIVOT Method

We consider the PIVOT operator which is a built-in operator in a commercial DBMS. Since this operator can perform transposition it can help evaluating horizontal aggregations. The PIVOT method internally needs to determine how many columns are needed to store the transposed table and it can be combined with the GROUP BY clause. The basic syntax to exploit the PIVOT operator to compute a horizontal aggregation assuming one BY column

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

REFERENCES

1. C. Cunningham, G. Graefe, and C.A. Galindo-Legaria, "PIVOT and UNPIVOT: Optimization and Execution Strategies in an
2. RDBMS," Proc. 13th Int'l Conf. Very Large Data Bases (VLDB '04),
3. pp. 998-1009, 2004.
4. C. Galindo-Legaria and A. Rosenthal, "Outer Join Simplification and Reordering for Query Optimization," ACM Trans. Database Systems, vol. 22, no. 1, pp. 43-73, 1997.
5. H. Garcia-Molina, J.D. Ullman, and J. Widom, Database Systems: The Complete Book, first ed. Prentice Hall, 2001.
6. C.Nagarajan and M.Madheswaran - 'Stability Analysis of Series Parallel Resonant Converter with Fuzzy Logic Controller Using State Space Techniques'- Taylor &Francis, Electric Power Components and Systems, Vol.39 (8), pp.780-793, May 2011. DOI: 10.1080/15325008.2010.541746
7. C.Nagarajan and M.Madheswaran - 'Experimental verification and stability state space analysis of CLL-T Series Parallel Resonant Converter' - Journal of Electrical Engineering, Vol.63 (6), pp.365-372, Dec.2012. DOI: 10.2478/v10187-012-0054-2
8. C.Nagarajan and M.Madheswaran - 'Performance Analysis of LCL-T Resonant Converter with Fuzzy/PID Using State Space Analysis'- Springer, Electrical Engineering, Vol.93 (3), pp.167-178, September 2011. DOI 10.1007/s00202-011-0203-9
9. S.Tamilselvi, R.Prakash, C.Nagarajan,"Solar System Integrated Smart Grid Utilizing Hybrid Coot-Genetic Algorithm Optimized ANN Controller" Iranian Journal Of Science And Technology-Transactions Of Electrical Engineering, DOI10.1007/s40998-025-00917-z,2025
10. [S.Tamilselvi, R.Prakash, C.Nagarajan," Adaptive sliding mode control of multilevel grid-connected inverters using reinforcement learning for enhanced LVRT performance" Electric Power Systems Research 253 (2026) 112428, doi.org/10.1016/j.epsr.2025.112428
11. S.Thirunavukkarasu, C. Nagarajan, 2024, "Performance Investigation on OCF and SCF study in BLDC machine using FTANN Controller," Journal of Electrical Engineering And Technology, Volume 20, pages 2675–2688, (2025), doi.org/10.1007/s42835-024-02126-w
12. C. Nagarajan, M.Madheswaran and D.Ramasubramanian- 'Development of DSP based Robust Control Method for General Resonant Converter Topologies using Transfer Function Model'- Acta Electrotechnica et Informatica Journal , Vol.13 (2), pp.18-31,April-June.2013, DOI: 10.2478/aeei-2013-0025.
13. C.Nagarajan and M.Madheswaran - 'DSP Based Fuzzy Controller for Series Parallel Resonant converter'- Springer, Frontiers of Electrical and Electronic Engineering, Vol. 7(4), pp. 438-446, Dec.12. DOI 10.1007/s11460-012-0212-0.
14. C.Nagarajan and M.Madheswaran - 'Experimental Study and steady state stability analysis of CLL-T Series Parallel Resonant Converter with Fuzzy controller using State Space Analysis'- Iranian Journal of Electrical & Electronic Engineering, Vol.8 (3), pp.259-267, September 2012.
15. C.Nagarajan and M.Madheswaran, "Analysis and Simulation of LCL Series Resonant Full Bridge Converter Using PWM Technique with Load Independent Operation" has been presented in ICTES'08, a IEEE / IET International Conference organized by M.G.R.University, Chennai.Vol.no.1, pp.190-195, Dec.2007
16. Suganthi Mullainathan, Ramesh Natarajan, "An SPSS and CNN modelling based quality assessment using ceramic materials and membrane filtration techniques", Revista Materia (Rio J.) Vol. 30, 2025, DOI: https://doi.org/10.1590/1517-7076-RMAT-2024-0721
17. M Suganthi, N Ramesh, "Treatment of water using natural zeolite as membrane filter", Journal of Environmental Protection and Ecology, Volume 23, Issue 2, pp: 520-530,2022
18. G. Graefe, U. Fayyad, and S. Chaudhuri, "On the Efficient Gathering of Sufficient Statistics for Classification from Large SQL Databases," Proc. ACM Conf. Knowledge Discovery and Data Mining (KDD '98), pp. 204-208, 1998.
19. J. Gray, A. Bosworth, A. Layman, and H. Pirahesh, "Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross- Tab and Sub-Total," Proc. Int'l Conf. Data Eng., pp. 152-159, 1996.
20. J. Han and M. Kamber, Data Mining: Concepts and Techniques, first ed. Morgan Kaufmann, 2001.
21. G. Luo, J.F. Naughton, C.J. Ellmann, and M. Watzke, "Locking Protocols for Materialized Aggregate Join Views," IEEE Trans. Knowledge and Data Eng., vol. 17, no. 6, pp. 796-807, June 2005.
22. C. Ordonez, "Horizontal Aggregations for Building Tabular Data Sets," Proc. Ninth ACM SIGMOD Workshop Data Mining and Knowledge Discovery (DMKD '04), pp. 35-42, 2004.
23. C. Ordonez, "Vertical and Horizontal Percentage Aggregations," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '04), pp. 866-871, 2004.
24. C. Ordonez, "Integrating K-Means Clustering with a Relational DBMS Using SQL," IEEE Trans. Knowledge and Data Eng., vol. 18, no. 2, pp. 188-201, Feb. 2006.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

25. C. Ordonez, "Statistical Model Computation with UDFs," IEEE Trans. Knowledge and Data Eng., vol. 22, no. 12, pp. 1752-1765, Dec. 2010.
26. C. Ordonez, "Data Set Preprocessing and Transformation in a Database System," Intelligent Data Analysis, vol. 15, n**o. 4, pp. 613-**
27. **Kiran, A., Rubini, P., & Kumar, S. S. (2025).** Comprehensive review of privacy, utility and fairness offered by synthetic data. *IEEE Access.*
28. **Gopinathan, V. R. (2024).** Real-Time Financial Risk Intelligence Using Secure-by-Design AI in SAP-Enabled Cloud Digital Banking. *International Journal of Computer Technology and Electronics Communication, 7(6), 9837-9845.*
29. **Udayakumar, R., Elankavi, R., Vimal, R., & Sugumar, R. (2023).** Improved Particle Swarm Optimization with Deep Learning-Based Municipal Solid Waste Management in Smart Cities. *Environmental & Social Management Journal, 17(4).*
30. **Anand, L. (2023).** An Intelligent AI and ML–Driven Cloud Security Framework for Financial Workflows and Wastewater Analytics. *International Journal of Humanities and Information Technology, 5(02), 87-94.*
31. **Soundappan, S. J. (2020).** Big Data Analytics in Healthcare: Applications for Pandemic Forecasting. *International Journal of Advanced Research in Computer Science & Technology, 3(1), 2248-2253.*
32. **Rajasekar, M. (2024).** Real-Time Predictive DevOps Intelligence for Risk-Aware Digital Business Processes in Cloud and SAP Ecosystems. *International Journal of Advanced Research in Computer Science & Technology, 7(4), 10713-10718.*
33. **Poornima, G., & Anand, L. (2024, May). Novel AI Multimodal Approach for Combating Against Pulmonary Carcinoma. In 2024 5th International Conference for Emerging Technology (INCET) (pp. 1-6). IEEE.**
34. **Prabha, P. S., & Rengarajan, A. (2025).** Adaptive Cloud Resource Allocation Using Attention-Driven Deep Reinforcement Learning. *Engineering, Technology & Applied Science Research, 15(6), 29334-29340.*
35. **Jagadeesh, S., & Sugumar, R. (2017).** A Comparative study on Artificial Bee Colony with modified ABC algorithm. *European Journal of Applied Sciences, 9(5), 243-248.*
36. **Varma, K. K., & Anand, L. (2025, March). Deep Learning Driven Proactive Auto Scaler for High-Quality Cloud Services. In International Conference on Computing and Communication Systems for Industrial Applications (pp. 329-338). Singapore: Springer Nature Singapore.**
37. **Kumar, S. A., & Anand, L. (2025). A Novel EEG-Based Deep Learning Framework for Enhancing Communication in Locked-In Syndrome Using P300 Speller and Attention Mechanisms. KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS, 19(11), 3841-3855.**
38. **Poornima, G., & Anand, L. (2025). Medical image fusion model using CT and MRI images based on dual scale weighted fusion based residual attention network with encoder-decoder architecture. Biomedical Signal Processing and Control, 108, 107932.**
39. **Archana, R., & Anand, L. (2025). Residual u-net with Self-Attention based deep convolutional adaptive capsule network for liver cancer segmentation and classification. Biomedical Signal Processing and Control, 105, 107665.Kumar, S. A., & Anand, L. (2025).** A Novel EEG-Based Deep Learning Framework for Enhancing Communication in Locked-In Syndrome Using P300 Speller and Attention Mechanisms. *KSII Transactions on Internet and Information Systems, 19(11), 3841-3855.*
40. **Rengarajan, A. (2025).** Cloud-Based AI-Driven Threat Detection Framework for Smart Grid Cybersecurity. *International Journal of Future Innovative Science and Technology, 8(6), 16065.*
41. **Murugeshwari, B., Sudharson, K., Panimalar, S. P., Shanmugapriya, M., & Abinaya, M. (2020).** SAFE–Secure Authentication in Federated Environment using CEG Key code.
42. **Raj A. A., & Sugumar, R. (2023).** Early Detection of COVID-19 with Impact on Cardiovascular Complications using CNN Utilising Pre-Processed Chest X-Ray Images. *2023 International Conference on Applied Intelligence and Sustainable Computing (ICAISC), IEEE.*
43. **Jagadeesh, S., & Sugumar, R. (2017). A Comparative study on Artificial Bee Colony with modified ABC algorithm. European Journal of Applied Sciences, 9(5), 243-248.**
44. **Selvi, G. V., Anbarasan, A. B., Murthy, B. A., & Prabavathy, S. (2023).** An Application Oriented Integrated Unequal Clustering Algorithm for Wireless Sensor Network. In *Underwater Vehicle Control and Communication Systems Based on Machine Learning Techniques* (pp. 140-154). CRC Press.
45. **Sruthi, R. S., Ananya, S., & Murugeshwari, B. (2010).** Web Based Virtual Control System Laboratory and On-Line Temperature Control of Electrophoresis Equipment using LabVIEW. *International Journal of Computer Applications, 975, 8887.*
46. **Vimal Raja, G. (2021).** Mining Customer Sentiments from Financial Feedback and Reviews using Data Mining Algorithms. *International Journal of Innovative Research in Computer and Communication Engineering, 9(12), 14705-14710.*

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802307

47. MATHEW, A. R. (2025). Neurosecurity and Brain-Computer Interfaces.
48. Soundappan, S. J. (2024). AI-Driven Customer Intelligence in Enterprise Lakehouse Systems Sentiment Mining Governance-Aware Analytics and Real-Time Data Synchronization. International Journal of Advanced Engineering Science and Information Technology (IJAESIT), 7(5), 14905.
49. Mathew, A. (2025). Human–AI Collaboration in Security Operations: Measuring Alert Trust, Automation Bias, and Analyst Upskilling in AI-Augmented SOC Environments. International Journal of Computer Technology and Electronics Communication, 8(5), 11375-11380.
50. Soundappan, S. J. (2022). AI-Based Fault Detection and Isolation for Reliability in Modern Power Systems. International Journal of Research Publications in Engineering, Technology and Management (IJRPETM), 5(4), 7106-7110.
51. **Poornima, G., & Anand, L. (2024, April). Effective Machine Learning Methods for the Detection of Pulmonary Carcinoma. In 2024 Ninth International Conference on Science Technology Engineering and Mathematics (ICONSTEM) (pp. 1-7). IEEE.Garg, V. K., Soundappan, S. J., & Kaur, E. M. (2020).** Enhancement in intrusion detection system for WLAN using genetic algorithms. *South Asian Research Journal of Engineering and Technology, 2(6), 62–64.*
52. Rengarajan, A., Jayakumar, C., & Sugumar, R. (2012). Optimization Of Recent Attacks Using Internet Protocol. National Journal of System and Information Technology, 5(1), 8.
53. Mathew, A. (2024). AI TRiSM: Trust, Risk, and Security Management in Cybersecurity. Cybersecurity, 4(3), 84-90.
54. Mathew, A. (2025). Deep seek vs. ChatGPT: A deep dive into AI Language mastery. Int J Multidisciplinary Res, 7(1), 1-5.