(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 March 2018 (22.03.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/052508 A1

(51) International Patent Classification:
*G06F 17/30* (2006.01)

(21) International Application Number:
PCT/US2017/040612

(22) International Filing Date:
03 July 2017 (03.07.2017)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
15/268,521 16 September 2016 (16.09.2016) US

(71) Applicant: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, Mail Stop 50P7, Redwood Shores, California 94065 (US).

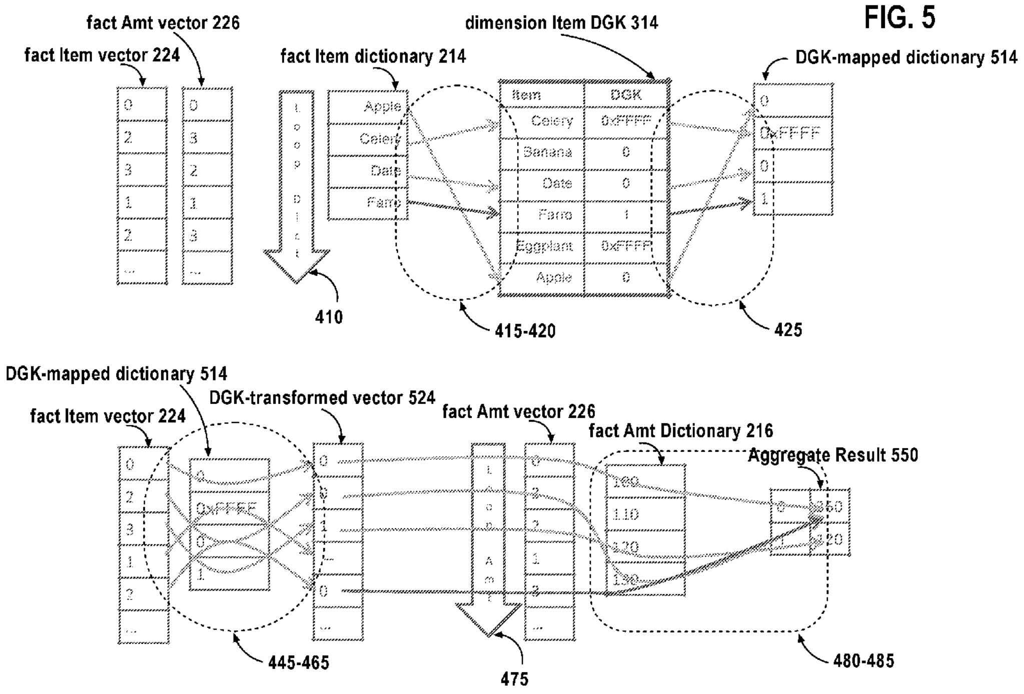
(72) Inventors: **HOPEMAN, Albert**; 500 Oracle Parkway, Mail Stop 50P7, Redwood Shores, California 94065 (US). **ROTH, Martin**; 500 Oracle Parkway, Mail Stop 50P7, Redwood Shores, California 94065 (US). **SOYLEMEZ, Ekrem**; 500 Oracle Parkway, Mail Stop 50P7, Redwood Shores, California 94065 (US). **KOCIUBES, Adam**; 500 Oracle Parkway, Mail Stop 50P7, Redwood Shores, California 94065 (US).

(74) Agent: **PAPANYAN, Khachatur V.** et al.; Hickman Palermo Becker Bingham LLP, 1 Almaden Boulevard, San Jose, California 95113 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA,

# (54) Title: TECHNIQUES FOR DICTIONARY BASED JOIN AND AGGREGATION

FIG. 5



(57) Abstract: Techniques are described herein for performing join and aggregation operations for a received query using column dictionaries. In an embodiment, a query is received that requests to aggregate a measure column of a fact table, stored in storage data units, based on an aggregate function and join the fact table with a dimension table on a join key column. For a storage unit having column dictionaries and corresponding column vectors for each column, the DBMS may generate a dictionary-grouping key mapping based on the fact join key dictionary and dense grouping keys from the dimension table. Based on the generated dictionary-grouping key mapping and the fact join key vector, the DBMS aggregates a column vector in that data storage unit that corresponds to the measure column in the received query.

WO 2018/052508 A1

[Continued on next page]

SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

**(84) Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

**Published:**

— *with international search report (Art. 21(3))*

INTERNATIONAL PATENT APPLICATION

FOR

# TECHNIQUES FOR DICTIONARY BASED JOIN AND AGGREGATION

## TECHNICAL FIELD

**[0001]** The technical field relates to database management systems particularly to techniques for dictionary based join and aggregation.

## BACKGROUND

**[0002]** Database management systems (DBMS's) are often designed to maintain large amounts of data. The amount of data stored is so large that the databases storing the information may colloquially be referred to as "data warehouses." Although the data stored in a data warehouse is large and describes a variety of entities, events, etc., the data items of the data may be very related.

**[0003]** To efficiently store large amounts of related data in a data warehouse, the DBMS may employ star/snowflake schema for the databases. The star/snowflake schema relates a set of factual data (sale transactions, quote requests) to analytic dimensions (customers, time, locations, merchandise types). Based on the schema the factual data is stored in "fact tables" that have one row per item, and the analytic dimensions are stored in "dimension tables," which describe multiple characteristics of items. Such a schema saves storage space in the data warehouse by eliminating repetition of the characteristics information common to multiple items for each of those items.

**[0004]** With time, as new facts are collected, factual data in a data warehouse tends to grow much faster than analytic dimension data, making fact tables generally much larger than dimension tables (having less rows than fact tables). The size and other data statistics as well as the database schema definition itself may be used by the DBMS to determine which tables are fact tables and which tables are dimension tables.

**[0005]** Dimension and fact tables are commonly joined by queries for business analysis. The combined data of the dimension and fact tables is aggregated to give the user a high level view of business performance. For example, sales might be aggregated by month and by the customer's home state.

**[0006]** Aggregation queries that are run against data that is organized in a star/snowflake schema typically specify a join key, an aggregate measure and a grouping key. The join key determines which row in a dimension table should be combined with a given row in the fact table, while the aggregate measure and grouping key determine which rows and columns are aggregated. For example, in
"select D.state, SUM (F.amount)
from F, D
where F.city_id = D.city_id
group by D.state" query, the "city_id" columns are join keys, the "amount" column is an aggregate measure and the "state" column is a grouping key.

**[0007]** The overall execution time is dominated by the processing of joins and aggregation. The query specifies that each row in fact table F should be joined with the row in dimension table D that has the same value for "city_id" as the fact table row. Further, once the rows from the fact table are joined to the appropriate rows in the dimension table, the rows should be divided into groups based on state, and the "amount" column values for each unique state are aggregated by summation.

**[0008]** One approach for executing such queries is to perform a hash join using hash tables. A "hash join" operation, generally speaking, comprises two phases. In the first phase, known as the "build phase," the DBMS generates a hash table by hashing each row of the first table according to a hash function on the join key column(s). In the second phase, known as the "probe phase," the DBMS then iteratively scans through each row of the second table. For each row in the second table, the DBMS uses the hash function and the hash table to identify rows in the first table with equivalent join key values. When matching rows are identified, the rows are merged and added to the result set for the join operation, assuming the rows also match any applicable join predicates.

**[0009]** For a large table, using hash tables to perform join and aggregation operations in this manner is inefficient as it requires a significant amount of memory for hashing numerous rows of large tables. Therefore, database systems that contain large amounts of information suffer significant performance degradation when performing join and aggregation operations using the hash join.

BRIEF DESCRIPTION OF THE DRAWINGS

**[0010]** In the drawings:

**[0011]** FIG. 1 is a block diagram that depicts a dense grouping key, a join-back table and a dense grouping key data structure, in an embodiment;

**[0012]** FIG. 2 is a block diagram that depicts a fact table and the corresponding data portion with column vectors and column dictionaries, in an embodiment;

**[0013]** FIG. 3 is a block diagram that depicts a dimension table, a dense grouping key and a dense grouping key data structure created based on data from the dimension table, in an embodiment;

**[0014]** FIG. 4A and B are flow diagrams that depict processes for performing join and aggregation operations using column vectors and dictionaries, in one or more embodiments;

**[0015]** FIG. 5 is a process diagram that depicts process for performing join and aggregation operations on example data structures of fact and dimension tables referenced in a query, in an embodiment.

**[0016]** FIG. 6 is a block diagram illustrating a computer system that may be used to implement the techniques described herein.

## DETAILED DESCRIPTION

**[0017]** In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

## GENERAL OVERVIEW

**[0018]** Techniques are described herein for efficiently processing of join and aggregation operations for fact and dimension tables leveraging column vectors and dictionaries for data. A DBMS performs in-depth pre-processing of dimension table rows to reduce processing of join and aggregation operations with fact table rows. Since dimension tables are smaller in size than fact tables, the DBMS can save computing resources by shifting the processing more to the dimension table side, for example by performing group/filtering operation first on the dimension table. “Dimension table,” as referred herein, is a pre-defined data container managed by the DBMS or a data container based on joining one or more pre-defined data containers. Similarly, “fact table,” as referred herein is a pre-defined data container managed by the DBMS or a data container based on joining one or more defined data containers, in which one or more attributes may be described in a corresponding dimension table. The DBMS may utilize the native column vectors and dictionaries for fact table rows to even further speed up the processing of the join and aggregation operations.

**[0019]** In an embodiment, for a dimension table of a received query, the DBMS generates a “dense grouping key (DGK)” data structure that represents a mapping between join key values and grouping key values of a dimension table. The term “dense grouping key (DGK)” refers to encoded representation of grouping key values, which are values of a group by column. In one embodiment, the DBMS generates DGKs as integers based on unique values in the group by columns. For example, “Massachusetts”, “California”, “Illinois”, “New York”, and “Kentucky” values in a group by column may be respectively assigned the DGKs of ‘1’, ‘2’, ‘3’, ‘4’ and ‘5’ (as depicted in the FIG. 1 example). The use of dense grouping key values to represent grouping key values, rather than the grouping key values themselves, typically makes processing more efficient.

**[0020]** For a fact table in the received query, the DBMS may maintain one or more dictionary data structures for columns of one or more “data portions” of the fact table. The term “data portion” refers herein to a table, or a partition or a segment of a table, or any other portion of a database object. A data portion may contain a portion or a complete data from join key column and/or aggregate measure column specified in a received query. Each data portion may be stored in a different storage data unit, a unit of storage in volatile or persistent storage that has a predetermined size. The data portion may be stored in a storage data unit in a row major format (cell elements of each row are stored in contiguous memory addresses) or a column major format (cell elements of each column are stored in contiguous memory addresses). The data in the data portion may also be compressed using a compression with a particular level of compression.

**[0021]** The term “column dictionary” is used to describe a dictionary data structure for a column vector in a data portion; the dictionary data structure maps dictionary encoded values in the data portion to the unique values of the column. In an embodiment, a column dictionary contains unique values of a join key column in the data portion, while a column vector of the data portion contains encoded join key values that are indices of the corresponding dictionary. Stated differently, each cell element of the join key vector of the data portion is a pointer to an entry in the corresponding column dictionary that contains the actual value of the corresponding cell of the join key column. For example, such pointer within a cell element may be an index into a vector of fixed width pointers or offsets that can be used to locate column dictionary values in the column dictionary.

**[0022]** When processing a query, the DBMS may access DGKs generated for a dimension table to map the DGKs with join key values of the fact table using fact table column dictionaries. In an embodiment, the DBMS iterates through a fact table column dictionary of the join key column (for example, 214 of FIG.’s 2 and 5) and matches the join key values with the corresponding join key values of the dimension DGK data structure (for example, 314 of FIG.'s 3 and 5). By doing so, regardless whether any column in the dimension table is referenced in the GROUP BY clause of the query, the DBMS effectively filters the values of fact table to be further processed. Only those rows of the fact table are to be processed which correspond to the fact dictionary join key values that have matched with the dimension join key values in the DGK data structure. Thus, the DBMS performs exact filtering of fact table rows based on the DGK data structure as opposed to other approximate filtering techniques used in query computation such as Bloom filters.

**[0023]** The exact nature of this DGK based filtering allows aggregation to be pushed before the traditional join since the exact set of rows that may need to be aggregated can be determined by the DBMS ahead of the aggregation. Using this filtering based on the matched join key values, the DBMS creates a DGK-mapped data structure (for example, 514 of FIG. 5) similar to the dimension join key column dictionary that maps encoded dimension join key values to the DGKs (for example, 314 of FIG.'s 3 and 5), and maps encoded join key values common to the dimension and fact join keys to DGKs (for example, 514 of FIG. 5). Since the index of the generated mapping data structure is similarly based on encoded join key values, the DBMS may easily access the corresponding DGK for each join key element of a fact table data portion. Based on such an access, aggregate measure values, of one or more aggregate key columns, corresponding to the join key cells are then aggregated for each unique DGK to yield the result for the one or more aggregate functions of the received query.

## DENSE GROUPING KEYS

**[0024]** FIG. 1 illustrates an example of dense grouping key values (DGKs) 101 and data structures 112, 122A-C, 123A-C based on dense grouping key values that may be created from data of a dimension table. For the purpose of illustrating a clear example, dimension table 102 is used in illustrating the example of creating dense grouping key data structure and "join back table" based on data of a dimension table.

**[0025]** Queries that specify aggregation operations identify one or more group-by column(s). The DBMS creates a dense grouping key value for each unique combination of values, from those group-by columns, that appears in any row of the table. A unique combination of values, from those group-by columns, that appears in any row of the table, is referred to herein as a "grouping key value".

**[0026]** In an embodiment, dense grouping key values are a sequence of consecutive numbers and all dense grouping key values have a fixed size. For example, if dense grouping key values are a series of integer values, then each dense grouping key value will be the size of an integer data type. Multiple rows of a dimension table may be represented or indicated by a single dense grouping key value.

## JOIN BACK TABLE

**[0027]** As used herein, the term "join back table" refers to a data structure that comprises one or more dense grouping key values and corresponding grouping key value or set of grouping key values. In an embodiment, the join back table comprises columns for dense grouping key values and corresponding grouping key values. In an embodiment, each entry of the join back table comprises a different dense grouping key value and the corresponding grouping key value or set of grouping key values.

**[0028]** In an embodiment, the join back table is a temporary table that is created only for a query and is discarded after the query is successfully processed.

## CREATION OF DENSE GROUPING KEY VALUES AND JOIN BACK TABLE

**[0029]** Dense grouping key values may be created based on the DBMS receiving a query such as:

```
SELECT g.state, sum (s.amount)
FROM sales s, geog g
WHERE g.CID = s.CID AND g.country = 'USA'
GROUP BY g.state
```

Query Q1.

**[0030]** The DBMS may identify the one or more columns indicated by the GROUP BY criteria specified in a query as the column or columns relevant for computing the aggregation operation. The DBMS, may then determine the dense grouping key values for the relevant column or columns and also determine the number of dense grouping key values and for each dense grouping key value, the corresponding value or values in the one or more columns indicated by the GROUP BY criteria.

**[0031]** In query Q1, table "sales" is a fact table and "geog" is a dimension table. For the purposes of illustrating a clear example, table "geog" is dimension table 102 represented in FIG. 1.

**[0032]** In an embodiment, the DBMS identifies one of the tables specified in a query to be a fact table based on certain metrics associated or corresponding to the tables. For example, if a DBMS determines that the size of one of the tables is substantially larger than other tables, then the DBMS identifies the large table as a fact table, and identifies the smaller tables as dimension tables.

**[0033]** The DBMS may determine and create dense grouping key values for the relevant column or columns, and the join back table used in grouping dimensional data and/or aggregating dimensional data, by executing the following example query:

```
CREATE TEMPORARY TABLE tt_geog as
SELECT g.state, keyvec_create(g.state ) dgk
FROM geog g
WHERE g.country = ‘USA’
GROUP BY g.state
```

Query Q2.

**[0034]** Query Q2 represents an example query that may be executed for each dimension table specified in the GROUP BY clause of query Q1 or in the WHERE clause of query Q1.

**[0035]** The DBMS by executing the keyvec_create( ) function of query Q2 creates a dense grouping key value for each unique value of the group key column specified by the GROUP BY clause. Thus a unique dense grouping key value is created for each unique grouping key value of the “state” column of dimension table 102.

**[0036]** Dense grouping key values are created for only the rows with values of a dimension table that are actually used in determining a result set. Therefore, the rows represented by dense grouping key values are rows with values that satisfy a filtering criteria in a predicate of the query. Thus, the number of rows represented by a dense grouping key value are reduced from all possible rows of the dimension table to only those rows of the dimension table that are actually used.

**[0037]** Therefore, prior to creating a dense grouping key value for a particular grouping key value, query Q2 also ensures that filtering criteria of query Q2 is satisfied by the row of the particular grouping key value. The filtering criteria of query Q2 is specified by the WHERE clause and it requires that the value in the “country” column of the dimension table 102 is ‘USA’. Consequently, dense grouping key values are created only for unique group-by values from rows that have a value of ‘USA’ in the country column of dimension table 102.

**[0038]** For example, in FIG. 1, dimension table 102 includes ‘Massachusetts’ in the “State” column of row one, and query Q2, prior to creating a dense grouping key value for ‘Massachusetts’, determines whether “Country” value associated with ‘Massachusetts’ is indeed ‘USA’, thus satisfying the filtering criteria. In dimension table 102, the “Country” value associated with ‘Massachusetts’ is indeed ‘USA’, therefore a dense grouping key value is created for ‘Massachusetts’. However, for ‘Queensland’, a dense grouping key value is not created because the “Country” value associated with ‘Queensland’ is ‘Australia’, thus the row of grouping key value ‘Queensland’ does not have a value of ‘USA’ in the “country” column of dimension table 102, thereby failing the filtering criteria.

**[0039]** In the example of query Q2 the temporary table tt_geog is the join back table. In FIG. 1 data structure 112 is the join back table. Join back table 112 comprises a column of dense grouping key values, indicated by column name “DGK”, and a column of grouping key values, indicated by “State”. Each entry of join back table 112 comprises a different dense grouping key (DGK) value and the corresponding grouping key value. In an embodiment, DGK values are assigned sequentially in the order of scanning the grouping key values by the DBMS from the table. For example, the “Massachusetts” grouping key value in Table 102 is the first value scanned for the “State” grouping key column and thus, is assigned to the DGK value of 1.

**[0040]** The DBMS by executing the keyvec_create( ) function of query Q2 also associates each dense grouping key value within the join back table 112 with a row in the dimension table 102 based on that dense grouping key value’s corresponding grouping key value. The DBMS may create a data structure such as array 101 to store the associations between each row of the dimension table and a dense grouping key value. Each row of the dimension table is an index into the array. Array 101 starts from zero index, therefore, the first row of dimension table corresponds to the first position of the array as depicted in FIG. 1.

**[0041]** In FIG. 1, since the corresponding dimension table 102 value of dense grouping key value ‘1’ is ‘Massachusetts’ in join back table 112, each row of dimension table 102 with ‘Massachusetts’ as the “State” column value will be associated with the dense grouping key value of ‘1’, as indicated by the first position of the array 101. The corresponding dimension table 102 value of dense grouping key value ‘2’ is ‘California’ in join back table 112, thus rows two and six of dimension table 102 are associated with dense grouping key value ‘2’, as indicated by positions two and six of array 101.

**[0042]** The corresponding dimension table 102 values of dense grouping key values ‘3’, ‘4’, ‘5’, are Illinois, New York, and Kentucky, respectively, in join back table 112. Therefore, rows seven and eight of dimension table 102 are associated with dense grouping key value ‘4’, rows three and four of dimension table 102 are associated with dense grouping key value ‘3’, and row nine of dimension table 102 is associated with dense grouping key value ‘5’. These associations are represented in array 101 by positions seven and eight for dense grouping key value ‘4’, positions three and four for dense grouping key value ‘3’, and position nine for dense grouping key value ‘5’.

**[0043]** In an embodiment, the DBMS may create a temporary dimension table with an additional column, where each row in the additional column comprises a dense grouping key value associated with that row of the dimension table.

**[0044]** In another embodiment, without generating array 101, the DBMS may directly proceed to generating a DGK data structure where different entries of the data structure correspond to unique join key values and a dense grouping key value corresponding to each of the unique join key values. While computing the query, the DBMS may use the resulting rows of the dimension table from one or more predicate evaluations to produce a list of unique join key values for those rows and the associated grouping key values of those rows.

**[0045]** The DBMS may generate a DGK data structure by executing the query referenced keyvec_create() function. The join key values used in creating the data structure are specified by the join key column of the WHERE clause of query Q1. The dimension table join key column specified in query Q1 is the "CID" column of "geog" dimension table 102 depicted in FIG. 1.

**[0046]** The DGK data structure created is depicted by data structures, 122a, 122b, 122c in FIG. 1. Data structures 122a, 122b, 122c are the same data structure depicted at different points in time. In an embodiment, if join key values are numeric or if a numeric value can be associated with a non-numeric join key value, then the DGK data structure created by the DBMS may be a vector. Use of a vector to store a dense grouping key value for the corresponding join key value may allow the DBMS to benefit from the fast read and write operations that can be performed on a vector data structure.

**[0047]** In query Q1, the values of the specified join key column are numeric. Therefore, in FIG. 1, the DGK data structure created is a vector depicted by DGK data structures 122a, 122b, 122c. DGK data structures 122a, 122b, 122c are the same vector depicted at different points in time. The size of the data structure is based on the number of join key values. If a filtering criteria is specified, then only rows of the dimension table that satisfy the filtering criteria will be considered for the DGK data structure. For example, query Q2 specifies a filtering criteria that requires the value in the "Country" column of a row in the dimension "geog" table to comprise 'USA'. Therefore, only join key values of rows in the dimension "geog" that satisfy the filtering criteria will be considered for the DGK data structure represented by DGK data structures 122a, 122b, 122c. Based on the data of dimension table 102, all rows except for the row where the "Country" value is 'Australia' will be considered for the DGK data structure represented by DGK data structures 122a, 122b, 122c.

**[0048]** Each join key value of “CID” column in dimension table 102 is an identifier for a unique value in the City column of dimension table 102. Therefore, each value in the City column of dimension table 102 has a corresponding value in the “CID” column in dimension table 102. For example, ‘Boston’ has corresponding “CID” value of ‘4’, and similarly ‘Springfield’ has a “CID” value of ‘6’.

**[0049]** In FIG. 1, the numeric values of join key column “CID” may be used as index into the vector, represented by DGK data structures 122a, 122b, 122c, for storing the corresponding dense grouping key value of the join key value. For example, the first row of join key column “CID” of dimension table 102 has a numerical value of ‘4’. Thus, the dense grouping key value associated with the first row of dimension table 102 will be stored in the fourth position of the vector, as depicted in DGK data structures 122a, 122b, 122c. In some embodiments, the size of the vector may be based on the maximum value among the values of the join key column. In FIG. 1, DGK data structures 122a, 122b, 122c are vectors of size 10. In an embodiment, the size of the vector may be based on the maximum value among the values of the join key column whose rows also satisfy a filtering criteria if specified.

**[0050]** Using the numerical values of the join key column, each row satisfying the filtering criteria specified in query Q2 is processed, and the dense grouping key value associated with the processed row is stored in DGK data structure 122a at the location indicated by the numerical value of the join key column. For example, in FIG. 1, as described above, at the fourth position of DGK data structure 122a, the dense grouping key value associated with first row of dimension table 102, dense grouping key value of ‘1’ will be stored. Similarly, at the second position of DGK data structure 122a, the dense grouping key value ‘4’ is stored.

**[0051]** Join key values of the join key column of a dimension table may not provide a unique constraint on the dimension table: one join key value may correspond to multiple grouping key values. For such a many-to-many join key-to-dense grouping key value relationship, multiple dense grouping key values in the vector corresponding to a same join key value occurring more than once within the join key column, are replaced with a flag value that indicates that a hash table associated with the vector should be used in performing join and aggregation operations as part of the DGK data structure. The vector flag value referenced row in the hash table contains the multiple dense grouping key values, in an embodiment.

**[0052]** For example, the “CID” value of ‘6’ in dimension table 102 is associated with more than one dense grouping key value. Therefore, the first time the “CID” value of ‘6’ is processed (the third row of dimension table 102), at the sixth position of DGK data structure 122b, the corresponding dense grouping key value for the third row of dimension table 102 is stored. The second time the "CID" value of '6' is processed, that is when the sixth row of the "CID" column of dimension table 102 is processed, the dense grouping key value at the sixth position of DGK data structure 122b is changed to a flag value as depicted in DGK data structure 122c. Additionally, a hash table is created that includes the join key value that is associated with more than one dense grouping key value and each of the corresponding dense grouping key values of the join key value as a DGK set.

**[0053]** In FIG 1, processes 130 and 131 depict the creation of hash table 123 where join key value of '6', depicted in first row of column 123a, is associated with a list of DGK sets, 123b, 123c comprising corresponding dense grouping key values 3 and 4 respectively. The third time the "CID" value of '6' is processed, that is when the ninth row of the "CID" column of the "geog" dimension table 102 is processed, the dense grouping key value at the sixth position of DGK data structure 122c still remains the flag value and an additional DGK set 123d with a value of 5, is added to the list of DGK sets in the hash table for join key value '6'.

**[0054]** DGK data structure 122c illustrates how the vector comprising dense grouping key values looks after all join key values that will actually be used in determining the result set of a query are processed. DGK data structure 122c also illustrates an example of a DGK data structure that can support processing many-to-many relation data.

**[0055]** In an embodiment, the DBMS receives a query in which one or more join key columns are string based rather than integer based. In such an embodiment, rather than using a vector in which an entry may be accessed by an integer, a DGK data structure may be used in which an entry may be accessed by a string, such as a hash table. In a related embodiment, for a grouping key of a short string or date data type, the DBMS uses the raw byte value of a grouping key value as an integer DGK value and/or applies a function to convert the grouping key value to an integer DGK value.

**[0056]** For example, the DBMS may receive a query such as recited below, in which the join key columns, "item", are a string value based columns. The DBMS may convert the string values of the "item" column to an integer for streamlining the processing of query Q3, however, FIG. 2 and FIG. 3 depict the "item" values as a string for ease of readability:

```
SELECT f.class, sum(s.amt)
FROM sales s, food f
WHERE s.item = f.item AND f.class != 'Vegetable'
```

GROUP BY f.class

Query Q3.

**[0057]** FIG. 2 depicts “Sales” table 202, which is the fact table of query Q3 comprising the “item” 204 and “amt” 206 columns. The “amt” column is the aggregate measure column referenced in the “SUM” aggregation function of query Q3. The “item” column is the join key column is joined in query Q3 with the “item” column of “Food” table 302 depicted in FIG. 3. The “Food” table 302 also contains the “class” column that is the grouping key column of query Q3.

**[0058]** The “item” join key columns have string based values. Accordingly, the generated dimension “Item” DGK data structure 314 has a string based index corresponding to the unique string values of the “item” column in “Food” dimension table 302. Dimension “Item” DGK data structure 314 maps the unique string join key values of the “item” column to dense grouping key values generated from the grouping key values of the “Class” column. To perform the join operation of query Q3, Item DGK data structure may be compared to dictionaries of join key values of fact “Sales” table 202 data portions stored in storage data units.

COLUMN DICTIONARIES

**[0059]** In an embodiment, one or more data portions store dictionary encoded values of actual values of a database object in data portion’s cell elements. Each cell element corresponds to a particular row and a particular column of the database object, such as a fact table. The cell elements may be arranged as a column vector that mimics the particular column arrangement within the database object. When multiple column vectors are stored in a data storage unit, the column vectors can be stored in a column major or a row major format.

**[0060]** Furthermore, the DBMS may maintain a column dictionary for each column vector that column vector’s dictionary encoded values to the corresponding values in the database object. Such column dictionaries for a data portion contain at least all unique database object values for data represented in the data portion for the corresponding columns of the database object.

**[0061]** Similar to DGK data structure, column dictionaries may arrange the mapping in many different ways using various hash based and non-hash based dictionary encodings. In an embodiment, dictionary encoded value in a cell element is an index to an entry in a column dictionary that contains the corresponding database object value. Thus, the corresponding database object value can be retrieved from the column dictionary by specifying the dictionary encoded value as an index to the column dictionary.

**[0062]** FIG. 2 depicts examples of a data portion that includes a dictionary structure for a database object, in an embodiment. Database object “Sales” table 202 contains two columns: “Item” column 204 and “Amount” column 206. Since “Item” column 204 contains four different values: “Apple”, “Celery”, “Date” and “Farro”, fact column dictionary 214 contains four values in a zero based index vector. Instead of data portion 222 storing the actual values from “Item” column 204, fact “Item” column vector 224 of data portion 222 stores in its cell elements the dictionary encoded indices of fact column dictionary 214 that correspond to the actual values at “Item” column 204 for a particular row. For example, the first row of “Item” column 204 contains the value of “Apple.” Based on fact column dictionary 214, the “Apple” value is at the zero index. Accordingly, fact “Item” column vector 224 stores zero at the first row. Similarly, the “Date” value at the second row of “Item” column in “Sales” table 202, corresponds to index two in fact column dictionary 214. Thus, the index value of two is stored at the second row of fact “Item” column vector 224. The rest of “Item” column 204 and “Sales” column 206 are stored similarly in fact column vectors 224 and 226 using fact dictionaries 214 and 216, respectively.

**[0063]** In an embodiment, dictionary encoded values in a column vector are based on memory addresses of entries in a corresponding column dictionary. The column dictionary may store unique actual values of a column in a contiguous memory space, each entry of a unique actual value being at a particular offset from the first entry’s memory address. These offsets are equivalent to indices of the column dictionary, and cell elements of the column vector may store the offsets of the corresponding unique actual value as dictionary encoded values. According to such an arrangement, a look up of an actual value is performed by request for a value at a particular offset from the first entry memory address.

**[0064]** In an embodiment, a data portion containing column vectors and dictionaries, is generated as part of Ozip compression of a database object. According to such techniques, a packed sequential plurality tokens are generated from the input database object data and a static dictionary data structure, such as a column dictionary, is used to decode the tokens into actual values of the input database object data.

**[0065]** In an embodiment, data portions are generated in response to a receipt and processing of a query. Continuing with the example of the DBMS processing query Q3, the DBMS determines that “food” table of Q3 is a dimension table for the “sales” fact table. Accordingly, to execute query Q3, the DBMS generates dimension DGK data structure 314 from "Food" dimension table 302 as depicted in FIG. 3 using the techniques described above. The DBMS may additionally generate column vectors with dictionary encoded values for "sales" and "food" tables and column dictionaries to decode the encoded values into actual values, as depicted in FIG. 2 and 3, respectively. Once generated, data portions may be stored in a volatile and/or persistent memory managed by the DBMS. Thus, next time a query referencing the same table is received, the data portions may not need to be regenerated but rather the already existing data portions are used.

**[0066]** In another embodiment, data portions are generated independent of a query receipt by the DBMS. Data portions may be generated when database objects that are stored persistently in the persistent data format, are mirrored into volatile memory in a mirror format.

**[0067]** In an embodiment, different data formats have different benefits. Therefore, techniques are described herein for maintaining data persistently in one format, but making that data available to a database server in more than one format. In one embodiment, one of the formats in which the data is made available for query processing is based on the on-disk format, while another of the formats in which the data is made available for query processing is independent of the on-disk format.

**[0068]** The format that corresponds to the on-disk format is referred to herein as the "persistent format" or "PF". Data that is in the persistent format is referred to herein as PF data. An in-memory format that is independent of the on-disk format is referred to as a "mirror format" or "MF". Data that is in the mirror format is referred to herein as MF data. For example, in one embodiment, the persistent format is row-major disk blocks, and the mirror format is a column-major format.

**[0069]** According to one embodiment, the mirror format is completely independent of the persistent format. However, the MF data is initially constructed in memory based on the persistently stored PF data, not based on any persistent MF structures. Since persistent MF structures are not required, users of existing databases need not migrate the data or structures in their existing databases to another format. Thus, a conventional database system that uses row-major disk blocks may continue to use those disk blocks to persistently store its data without performing any data migration, while still obtaining the performance benefit that results from having a column-major representation of the data available in volatile memory.

**[0070]** In-memory MF data is maintained transactionally consistent with the PF data. The MF data is transactionally consistent in that any data items provided to a transaction from the MF data will be the same version that would have been provided if the data items were provided from the PF data. Further, that version reflects all changes that were committed before the snapshot time of the transaction, and no changes that were committed after the snapshot time of the transaction. Thus, when a transaction, that made a change to a data item that is mirrored in the MF data, is committed, the change is made visible relative to both the PF data and the MF data. On the other hand, if a transaction that made a change is aborted or rolled back, then the change is rolled back relative to both the PF data and the MF data.

**[0071]** In one embodiment, the same transaction manager that ensures consistency among the reads and writes of the PF data is also used to ensure consistency among the reads and writes of the MF data. Because the MF data is kept current in a transactionally consistent manner, if the in-memory MF data includes the data required by a database operation, then the database operation may be satisfied either from the in-memory MF data, or from the PF data.

**[0072]** The MF data mirrors data that already exists in the PF data. However, while all items in the MF data are mirror versions of corresponding items in the PF data (albeit organized in a different format), not all items in the PF data need be mirrored in the MF data. Thus, the MF data may be a subset of the PF data.

**[0073]** Because not all of the PF data is necessarily mirrored in the MF data, in some situations queries may require data that can only be satisfied by the PF data. For example, if a table has columns A, B and C, and only column A is mirrored in the MF data, then a query that requires values from column B must obtain those values from the PF data.

**[0074]** However, even in those circumstances, the MF data may still be used to (a) satisfy a portion of the query, and/or (b) speed up the retrieval of required data from the PF data. For example, the MF data may be used to identify the specific rows that must be retrieved from the PF data.

**[0075]** According to one embodiment, to reduce overhead, no on-disk copy of the MF data is maintained. In an alternative embodiment, a copy of the MF may be stored, but no attempt is made to keep the on-disk copy of the MF data in sync with updates that are being performed on the PF data. Consequently, after a failure, the in-memory MF data must be reconstructed based on the persistent copy of the PF data.

**[0076]** In some embodiments, the MF data is compressed. The compression can be performed at various compression levels, either specified by the user or based on access patterns.

**[0077]** While examples shall be given hereafter in which the mirror format is columnar, the mirror format may be any format, different from the persistent format, that is useful for running in-memory queries. For example, in an alternative embodiment, the PF format is column-major, while the MF format is row-major. Regardless of the particular mirror format used, the mirror format data is created in memory based on existing PF structures (e.g. tables and indexes) without causing a change to the format of those structures.

**[0078]** MF data may mirror all of the PF data, or a subset thereof. In one embodiment, a user may specify what portion of the PF data is “in-memory enabled”. The specification may be made at any level of granularity. For example, the specification of what is in-memory enabled may be made at least at the following levels of granularity:

- the entire database
- specified tables
- specified columns
- specified row ranges
- specified partitions
- specified segments
- specified extents
- any combination thereof (e.g. specified columns and partitions)

**[0079]** As shall be described hereafter, in-memory enabled data is converted to the mirror format and stored as MF data in volatile memory. Thus, when in-memory enabled data is required by a query, the database server has the option of providing the data from either the PF data or the MF data. The conversion and loading may occur at the time the database is started, or in a lazy or on-demand fashion.

**[0080]** In an embodiment in which the MF data is compressed, the MF data may be organized, within volatile memory of a DBMS, into “in-memory compression units” (IMCUs). Each IMCU stores a different set of MF data. For example, one IMCU may store half of column vectors, and another IMCU may stores the other half of column vectors, which together would make up a table or column-based partition thereof. Specifically, one IMCU may include a vector portion that stores half of the values from a first column of a table, and another vector portion that stores half of the values from a second column of the table. Similarly, another IMCU may include a vector portion that stores the remaining half of the values from the first column of the table, and yet another vector portion that stores the remaining half of the values from the second column.

**[0081]** In such an example, the IMCUs divide the MF data based on the rows to which the data belongs, where one IMCU corresponds to a portion of the rows in the table, and another IMCU corresponds to another portion of rows in the table. However, this is only one of many different ways that the MF data may be spread among IMCUs. For example, different IMCUs may store MF data for different tables, different partitions of a table, different columns of a table, different segments, different extents, etc.

**[0082]** To determine whether the MF data has the data required to process a query, and if so, to find the MF data required to process the query, the database server needs to know which PF data is mirrored in the MF data, and specifically which specific PF data is mirrored by each IMCU. Therefore, according to one embodiment, metadata for the MF data is maintained in volatile memory.

**[0083]** In one embodiment, metadata includes a data-to-IMCU mapping. The data-to-IMCU mapping indicates which data is contained in each IMCU. This indication may be made in a variety of ways, including storing data that indicates, for each IMCU, one or more of the following:

- the table(s) whose data is stored in the IMCU
- the column(s) whose data is stored in the IMCU
- the range of rows stored in the IMCU
- the range of the disk blocks whose data is stored in the IMCU
- the segments whose data is stored in the IMCU
- the table partitions whose data is stored in the IMCU
- the extents whose data is stored in the IMCU
- the manner in which the data, within the IMCU, has been compressed
- the dictionary for decompressing the data stored in the IMCU (when a dictionary-type encoding has been used to compress the PF data)

**[0084]** In a related embodiment, a data portion may be maintained as an IMCU and one or more column dictionaries for the data portions may be stored as the metadata of the IMCU. Accordingly, once IMCUs are generated as the MF data of a database object, the DBMS may use the IMCUs that include column dictionaries to perform join and/or aggregation operations.

## JOINING AND AGGREGATING FACT AND DIMENSION TABLES

**[0085]** The DBMS may perform join and aggregation operation of a received query using DGK data structures and data portions that include column vectors and column dictionaries of the fact and dimension tables of the target data set of the query. In an embodiment, rather than performing the join and aggregation operations of a query after scanning the target data set, the DBMS performs the join and aggregation operations while scanning the target data set. The joining and aggregating during the scan significantly reduces the resource utilization of the DBMS, especially when column vectors and column dictionaries, have been pre-generated before the scan as is the case for IMCUs.

**[0086]** FIG. 4A is a flow diagram that depicts a process for performing join and aggregation operations, in an embodiment. Using the techniques described for generating a DGK data structure, at block 400, the DBMS generates a DGK data structure for a dimension table of the target set of a received query. For example, FIG. 3 depicts dimension "Item" DGK data structure 314 for the "Item" column of "Food" dimension table 302. Dimension "Item" DGK data structure 314 may have been generated in response to the DBMS receiving query Q3, which references "Food" dimension table 302.

**[0087]** To perform a join operation between a dimension table and a fact table, data structures of the fact table and the dimension table may need to be on a same compute node of the DBMS. In an embodiment in which a DGK data structure is stored on a different compute node than data portions of the fact table, a copy of the DGK data structure may be transferred to the compute nodes that store data portions for evaluation of the query.

**[0088]** In an embodiment, the join operation is performed by comparing the DGK data structure of a dimension table with column dictionaries of a fact table data portion. To do so, a new dictionary-grouping key mapping is generated that maps each join key value in the fact table portion to the dense grouping key for which the associated dimension join key value matches the join key value from the fact table data portion. A DGK-mapped dictionary data structure may be initialized at block 405 to store the dictionary-grouping key mapping. The new DGK-mapped dictionary has the same index as the join key column dictionary of the fact table data portion and maps join key values of the fact table rather than the dimension table with DGKs of the dimension table. Such a mapping effectively performs a join operation between join key columns of the dimension table and the fact table while preserving the association with the DGKs. Alternatively or additionally, the new DGK-mapped dictionary data structure may be dynamically allocated as new entries for the mapping dictionary data structure are determined and populated in the new data structure by the process.

**[0089]** To generate a dictionary-grouping key mapping for the new DGK-mapped dictionary, at block 410, a join key column dictionary of a fact table data portion is scanned.

Each retrieved join key value is used for an index to look up into the DGK data structure at block 415. If an entry in the DGK data structure at the index exists, at block 420, the DGK for the matched dictionary entry is added to the new DGK-mapped dictionary at the same index as the retrieved join key value at block 425. If an entry does not exist, no dictionary-grouping key mapping exists for the fact join key value and no entry may be added to the DGK-mapped dictionary. In another embodiment, a special null value is added to the DGK-mapped dictionary to indicate that the join key value for the fact table does not exist in the dimension table. The process repeats blocks 415 through 425 until the join key column dictionary is fully scanned at block 430.

**[0090]** For example, FIG. 5 depicts dimension “Item” DGK data structure 314 and fact “Item” dictionary 214 that are used for generating dictionary-grouping key mapping in DGK-mapped dictionary 514 as part of the execution of query Q3. Fact “Item” dictionary 214 as well as fact “Item” column vector 224 and “Amt” column vector 226 are also depicted in FIG. 2 and are part of data portion 222 of “Sales” fact table referenced in query Q3.

**[0091]** DGK-mapped dictionary 514 is initialized with the same indexes as fact “Item” column dictionary 214: for example, the first entry has the index of ‘0’ which, like fact “Item” column dictionary 214’s index ‘0’, corresponds to the encoded value of ‘0’ in fact “Item” column vector 224. Each entry in fact “Item” dictionary 214 is retrieved and compared with dimension “Item” DGK data structure 314 as described at blocks 415-420 of FIG. 4. For example, the first entry value of “Item” column dictionary, ‘Apple’, is looked up in dimension “Item” DGK data structure 314. The lookup matches the ‘Apple’ value with the last index of dimension DGK data structure 314 that contains the value of ‘0’ for the DGK of that entry and retrieves this DGK value. As described at block 425 of FIG. 4, the process adds the retrieved DGK value at the same index in DGK-mapped dictionary 514 as the original look up value in fact “Item” dictionary 214. For the join key value of ‘Apple’, that index is 0 (the first entry), and thus, the DGK value of ‘0’ is added to the first entry at index 0 of DGK-mapped dictionary 514. The DBMS similarly generates the rest of the mapping and populates the rest of the entries in DGK-mapped dictionary 514 as depicted in FIG. 5.

**[0092]** In an embodiment, at block 440, the process may initialize a DGK-transformed vector of the same size as the join key vector of the data portion. Alternatively or additionally, the new DGK-transformed vector may be dynamically allocated as new mappings are added as a result of join key value scan. The DGK-transformed vector expands the mapping information between a fact table join key value and a DGK by storing a DGK for each row of the data portion. At block 445, the encoded join key values of the fact table column vector are scanned. Each retrieved encoded join key value is used as an index to look up a corresponding DGK in the dictionary-grouping key mapping at block 450. At block 460, the corresponding DGK is added to the DGK-transformed vector at the same row as the retrieved encoded join key value that was used to look up the DGK. Once the scan is complete at block 465, the DGK-transformed vector contains DGK values for each row of the fact table data portion.

**[0093]** In an embodiment, to evaluate an aggregate function on an aggregate measure column of the fact table, the aggregate measure values are aggregated based on DGKs of same rows. In particular, at block 470, an aggregate result data structure may be initialized with indexes corresponding to the unique values of DGKs. In another embodiment, the aggregate result data structure is dynamically expanded when a new unique DGK value is encountered. At block 475, the process scans rows of the aggregate measure vector and, at block 480, looks up the aggregate measure value in the aggregate measure dictionary based on the row value of the aggregate measure vector. Based on the DGK value at the same row, the process retrieves the previous aggregate result from the aggregate result data structure and aggregates the previous aggregate result with the looked up aggregate measure value at block 485 to yield the new aggregate result for the DGK. At the end of the scan at block 490, the aggregate result data structure contains aggregate results for each DGK for the data portion.

**[0094]** In an embodiment, in which multiple data portions contain the data of the fact table, multiple aggregate result dictionaries are generated for the query execution. The results from the multiple aggregate result dictionaries are further aggregated per each common DGK, to yield the final aggregate result for the query at block 495.

**[0095]** In an embodiment, final aggregate results per DGKs are converted through join back table of the DGKs to grouping key values. Thus, the DBMS may return the final aggregate result with the corresponding grouping key value.

**[0096]** Continuing with the example of query Q3 execution referencing “Sales” table 202 and “Food” table 302, the DBMS uses generated DGK-mapped dictionary 514 to aggregate measure values based on DGKs. In particular, DGK-mapped dictionary 514 is expanded to have a corresponding DGK for each encoded row value of fact “Item” column vector 224. To do so, each encoded join key value of fact “Item” column vector 224 is used as an index into DGK-mapped dictionary 514 to look up the corresponding DGK. For example, the last row of fact “Item” column vector 224 has the value of ‘2’. DGK-mapped dictionary at the index value of ‘2’, contains DGK value ‘0’. Accordingly, DGK value of ‘0’ is added as the last row of DGK-transformed vector 524.

**[0097]** Based on generated DGK-transformed vector 524, encoded aggregate measure values of fact "Amt" column vector 226 are mapped to DGKs of same rows. Based on this mapping, aggregate measure values in fact "Amt" column dictionary 216 can be aggregated per each unique DGK. In particular, for each row in "Amt" column vector, the encoded row value is used to look up the actual "Amt" column value in fact "Amt" column dictionary 216. The looked up value corresponds to the same row DGK value in DGK-transformed vector 524, and can be aggregated for the DGK value in aggregate results 550. For example, the scan of the first row of "Amt" column vector yields value of '0', which corresponds to the value of '100' at the index of '0' in fact "Amt" dictionary 216. The first row in DGK-transformed vector 524 has the DGK of 0, and since query Q3 specifies SUM aggregate function, the "Amt" value of '100' is summed with an initial value for the DGK value of '0'. The scan of the second row in fact "Amt" column vector 226 yields the value of '3', which corresponds to the value of '130' at the index of '3' in fact "Amt" dictionary 216. The second row of DGK-transformed vector 524 has the same DGK value of '0' and thus, the retrieved "Amt" value of '130' is summed with the previous aggregate result of '100' at DGK index value of '0' in aggregate result data structure 550. After all values in fact "Amt" column vector 226 are scanned and processed according to blocks 480-485 of FIG. 4, aggregate result data structure 550 contains the aggregate results for query Q3 for data portion 222. Since DGK values are common across data portions of fact table, the final result for each DGK may be obtained by aggregating the results of the data portion per each DGK.

**[0098]** In an alternative embodiment, rather than generating the DGK-transformed vector, the DGK may be dynamically retrieved as part of the aggregation process. FIG. 4B is a flow diagram that depicts a process for performing join and aggregation operations for such an embodiment. The DBMS executes blocks 400B-430B similar to blocks 400-430 in FIG. 4A to similarly generate a DGK-mapped dictionary at block 430B. At block 435B, the DBMS initializes an aggregate result data structure that stores aggregate results for each unique DGK value. For each aggregate encoded value scanned from an aggregate measure column at block 440B, the DBMS uses the row index of the scan to look up the corresponding DGK value in the DGK-mapped dictionary at block 450B. The DBMS looks up the aggregate measure value based on the aggregate encoded value in the aggregate column dictionary at block 480B. At block 485B, for the corresponding DGK value in the aggregate result data structure, the DBMS aggregates the looked up aggregate measure value with the existing value in the aggregate result data structure for the corresponding DGK value. At block 490B, the blocks 440B-490B are repeated for each row of the aggregate column of the fact table. In an embodiment, in which multiple data portions contain the data of the fact table, multiple aggregate result dictionaries are generated for the query execution. The results from the multiple aggregate result dictionaries are further aggregated per each common DGK value, to yield the final aggregate result for the query at block 495B.

## DATABASE MANAGEMENT SYSTEMS

**[0099]** A database management system (DBMS) manages a database. A DBMS may comprise one or more database servers. A database comprises database data and a database dictionary that are stored on a persistent memory mechanism, such as a set of hard disks. Database data may be stored in one or more data containers. Each container contains records. The data within each record is organized into one or more fields. In relational DBMSs, the data containers are referred to as tables, the records are referred to as rows, and the fields are referred to as columns. In object-oriented databases, the data containers are referred to as object classes, the records are referred to as objects, and the fields are referred to as attributes. Other database architectures may use other terminology.

**[0100]** In an embodiment, a DBMS may be connected to or include a cluster of compute nodes that may store one or more tables. The DBMS may manage tables stored on the cluster of compute nodes similar to managing tables stored in persistent storage.

**[0101]** Users interact with a database server of a DBMS by submitting to the database server commands that cause the database server to perform operations on data stored in a database. A user may be one or more applications running on a client computer that interact with a database server. Multiple users may also be referred to herein collectively as a user.

**[0102]** As used herein, "query" refers to a database command and may be in the form of a database statement that conforms to a database language. In one embodiment, a database language for expressing the query is the Structured Query Language (SQL). There are many different versions of SQL, some versions are standard and some proprietary, and there are a variety of extensions. Data definition language ("DDL") commands are issued to a database server to create or configure database objects, such as tables, views, or complex data types. SQL/XML is a common extension of SQL used when manipulating XML data in an object-relational database. Although the embodiments of the invention are described herein using the term "SQL", the invention is not limited to just this particular database query language, and may be used in conjunction with other database query languages and constructs.

**[0103]** A client may issue a series of requests, such as requests for execution of queries, to a database server by establishing a database session, referred herein as "session." A session comprises a particular connection established for a client to a database server, such as a database instance, through which the client may issue the series of requests. The database server may maintain session state data about the session. The session state data reflects the current state of the session and may contain the identity of the user for which the session is established, services used by the user, instances of object types, language and character set data, statistics about resource usage for the session, temporary variable values generated by processes executing software within the session, and storage for cursors and variables and other information. The session state data may also contain execution plan parameters configured for the session.

**[0104]** A multi-node database management system is made up of interconnected compute nodes that share access to the same database. Typically, the compute nodes are interconnected via a network and share access, in varying degrees, to shared storage, e.g. shared access to a set of disk drives and data blocks stored thereon. The compute nodes in a multi-node DBMS may be in the form of a group of computers (e.g. work stations, personal computers) that are interconnected via a network. Alternately, the compute nodes may be the compute nodes of a grid, which is composed of compute nodes in the form of server blades interconnected with other server blades on a rack.

**[0105]** Each compute node in a multi-node DBMS hosts a database server. A server, such as a database server, is a combination of integrated software components and an allocation of computational resources, such as memory, a compute node, and processes on the compute node for executing the integrated software components on a processor, the combination of the software and computational resources being dedicated to performing a particular function on behalf of one or more clients.

**[0106]** Resources from multiple compute nodes in a multi-node DBMS can be allocated to running a particular database server's software. Each combination of the software and allocation of resources from a compute node is a server that is referred to herein as a "server instance" or "instance". A database server may comprise multiple database instances, some or all of which are running on separate computers, including separate server blades.

## HARDWARE OVERVIEW

**[0107]** According to one embodiment, the techniques described herein are implemented by one or more special-purpose computing devices. The special-purpose computing devices may be hard-wired to perform the techniques, or may include digital electronic devices such as one or more application-specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs) that are persistently programmed to perform the techniques, or may include one or more general purpose hardware processors programmed to perform the techniques pursuant to program instructions in firmware, memory, other storage, or a combination. Such special-purpose computing devices may also combine custom hard-wired logic, ASICs, or FPGAs with custom programming to accomplish the techniques. The special-purpose computing devices may be desktop computer systems, portable computer systems, handheld devices, networking devices or any other device that incorporates hard-wired and/or program logic to implement the techniques.

**[0108]** For example, FIG. 6 is a block diagram that illustrates a computer system 600 upon which an embodiment of the invention may be implemented. Computer system 600 includes a bus 602 or other communication mechanism for communicating information, and a hardware processor 604 coupled with bus 602 for processing information. Hardware processor 604 may be, for example, a general purpose microprocessor.

**[0109]** Computer system 600 also includes a main memory 606, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 602 for storing information and instructions to be executed by processor 604. Main memory 606 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 604. Such instructions, when stored in non-transitory storage media accessible to processor 604, render computer system 600 into a special-purpose machine that is customized to perform the operations specified in the instructions.

**[0110]** Computer system 600 further includes a read only memory (ROM) 608 or other static storage device coupled to bus 602 for storing static information and instructions for processor 604. A storage device 610, such as a magnetic disk, optical disk, or solid-state drive is provided and coupled to bus 602 for storing information and instructions.

**[0111]** Computer system 600 may be coupled via bus 602 to a display 612, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 614, including alphanumeric and other keys, is coupled to bus 602 for communicating information and command selections to processor 604. Another type of user input device is cursor control 616, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 604 and for controlling cursor movement on display 612. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

**[0112]** Computer system 600 may implement the techniques described herein using customized hard-wired logic, one or more ASICs or FPGAs, firmware and/or program logic which in combination with the computer system causes or programs computer system 600 to be a special-purpose machine. According to one embodiment, the techniques herein are performed by computer system 600 in response to processor 604 executing one or more sequences of one or more instructions contained in main memory 606. Such instructions may be read into main memory 606 from another storage medium, such as storage device 610. Execution of the sequences of instructions contained in main memory 606 causes processor 604 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions.

**[0113]** The term “storage media” as used herein refers to any non-transitory media that store data and/or instructions that cause a machine to operate in a specific fashion. Such storage media may comprise non-volatile media and/or volatile media. Non-volatile media includes, for example, optical disks, magnetic disks, or solid-state drives, such as storage device 610. Volatile media includes dynamic memory, such as main memory 606. Common forms of storage media include, for example, a floppy disk, a flexible disk, hard disk, solid-state drive, magnetic tape, or any other magnetic data storage medium, a CD-ROM, any other optical data storage medium, any physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, NVRAM, any other memory chip or cartridge.

**[0114]** Storage media is distinct from but may be used in conjunction with transmission media. Transmission media participates in transferring information between storage media. For example, transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 602. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

**[0115]** Various forms of media may be involved in carrying one or more sequences of one or more instructions to processor 604 for execution. For example, the instructions may initially be carried on a magnetic disk or solid-state drive of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 600 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 602. Bus 602 carries the data to main memory 606, from which processor 604 retrieves and executes the instructions. The instructions received by main memory 606 may optionally be stored on storage device 610 either before or after execution by processor 604.

**[0116]** Computer system 600 also includes a communication interface 618 coupled to bus 602. Communication interface 618 provides a two-way data communication coupling to a network link 620 that is connected to a local network 622. For example, communication interface 618 may be an integrated services digital network (ISDN) card, cable modem, satellite modem, or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 618 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 618 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

**[0117]** Network link 620 typically provides data communication through one or more networks to other data devices. For example, network link 620 may provide a connection through local network 622 to a host computer 624 or to data equipment operated by an Internet Service Provider (ISP) 626. ISP 626 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the “Internet” 628. Local network 622 and Internet 628 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 620 and through communication interface 618, which carry the digital data to and from computer system 600, are example forms of transmission media.

**[0118]** Computer system 600 can send messages and receive data, including program code, through the network(s), network link 620 and communication interface 618. In the Internet example, a server 630 might transmit a requested code for an application program through Internet 628, ISP 626, local network 622 and communication interface 618.

**[0119]** The received code may be executed by processor 604 as it is received, and/or stored in storage device 610, or other non-volatile storage for later execution.

**[0120]** In the foregoing specification, embodiments of the invention have been described with reference to numerous specific details that may vary from implementation to implementation. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. The sole and exclusive indicator of the scope of the invention, and what is intended by the applicants to be the scope of the invention, is the literal and equivalent scope of the set of claims that issue from this application, in the specific form in which such claims issue, including any subsequent correction.

## EXTENSIONS AND ALTERNATIVES

**[0121]** In the foregoing specification, embodiments of the invention have been described with reference to numerous specific details that may vary from implementation to implementation. Thus, the sole and exclusive indicator of what is the invention, and is intended by the applicants to be the invention, is the set of claims that issue from this application, in the specific form in which such claims issue, including any subsequent correction. Any definitions expressly set forth herein for terms contained in such claims shall govern the meaning of such terms as used in the claims. Hence, no limitation, element, property, feature, advantage or attribute that is not expressly recited in a claim should limit the scope of such claim in any way. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

CLAIMS

What is claimed is:

1. A method comprising:

receiving a query that aggregates a measure column of a fact table based on an aggregate function and joins the fact table with a dimension table on a join key column, wherein data of the fact table is stored in one or more storage data units:

each storage data unit storing a respective data portion of the fact table,

the respective data portion comprising one or more column vectors corresponding to one or more columns of the fact table,

each cell element of the one or more column vectors of said respective data portion corresponding to a corresponding row and a corresponding column, of the one or more columns, of the fact table, said each cell element, of the one or more column vectors, comprising a respective dictionary encoded value of a corresponding value at the corresponding column and at the corresponding row, wherein the respective dictionary encoded value is mapped to the corresponding value of said each element by a respective dictionary data structure of the corresponding column;

in response to receiving the query, for a particular data storage unit of the one or more data storage units:

based on the query, identifying a fact join key vector of the one or more column vectors of the particular data storage unit and a fact join key dictionary data structure corresponding to the fact join key vector;

generating a dictionary-grouping key mapping based on the fact join key dictionary data structure and a dense grouping key data structure, the dense grouping key data structure representing a mapping of a plurality of unique dimension join key values of the join key column of the dimension table to grouping key values of a grouping key of the dimension table;

for each measure cell element in a measure column vector that corresponds to said measure column:

selecting a dense grouping key value corresponding to a respective cell element of the fact join key vector, selecting a respective measure value, and

associating the respective measure value with said dense grouping key value;

aggregating the measure column vector in said particular data storage unit by aggregating the respective measure value with other measure values that are associated with the same said dense grouping key value; and

wherein the method is performed by one or more computing devices.

2. The method of Claim 1, wherein said each storage data unit is an in-memory compression unit (IMCU).

3. The method of Claim 1, further comprising:

prior to receiving the query, generating the respective data portion that comprises the one or more column vectors, of the one or more columns of the fact table as part of mirroring the one or more columns of the fact table from persistent storage to volatile memory.

4. The method of Claim 1, wherein the fact join key vector is stored in a column-major format.

5. The method of Claim 1, wherein the plurality of unique dimension join key values includes only those unique dimension join key values which rows of the dimension table satisfy criteria of one or more predicates in the query.

6. The method of Claim 1, wherein the respective dictionary encoded value is encoded based on a location of an entry in the respective dictionary data structure for the corresponding value at the corresponding column and at the corresponding row.

7. The method of Claim 1, wherein generating the dictionary-grouping key mapping further comprises:

comparing a plurality of unique fact join key values of the fact join key dictionary data structure with the plurality of unique dimension join key values of the dense grouping key data structure; and

based on comparing the plurality of unique fact join key values of the fact join key dictionary data structure with the plurality of unique dimension join key values, generating the dictionary-grouping key mapping that maps the plurality of unique fact join key values with a sub-set the grouping key values of the dense grouping key data structure.

8. The method of Claim 1, wherein the dictionary-grouping key mapping is stored in a mapped dictionary data structure that stores a mapping of unique dictionary encoded fact join

key values of the fact join key dictionary data structure with corresponding dense grouping keys of a plurality of dense grouping key representing the grouping key values of the dense grouping key data structure.

9. The method of Claim 1, further comprising:

based on the query, identifying a dimension join key vector corresponding to the join key column of the dimension table and a dimension join key dictionary data structure corresponding to the dimension join key vector;

based on the dimension join key dictionary data structure and the dimension join key vector, generating the dense grouping key data structure that comprises a plurality of dense grouping key values, each entry of the dense grouping key data structure storing a mapping of one or more dense grouping key values, of the plurality of dense grouping key values to a unique dimension join key value from the plurality of unique dimension join key values.

10. The method of Claim 9, wherein the dense grouping key data structure is stored on storage of a first compute node and said respective data portion is stored on storage of a second compute node different from the first compute node, and the method further comprising:

transferring a copy of the dense grouping key data structure to the storage of the second compute node from the storage of the first compute node.

11. The method of Claim 9, further comprising:

based on the query, identifying a grouping key vector, of the one or more column vectors, corresponding to the grouping key of the dimension table and a grouping key data structure corresponding to the grouping key vector;

generating the dense grouping key data structure comprises:

for an entry of the dimension join key dictionary data structure, generating a corresponding entry of the dense grouping key data structure,

selecting one or more encoded grouping key values that are retrieved based on a same one or more locations in the grouping key vector as one or more locations in the dimension join key vector that corresponds to the entry of the dimension join key dictionary data structure, and

storing in the corresponding entry of the dense grouping key data structure a particular dense grouping key value that represents the selected one or more encoded grouping key values.

12. The method of Claim 11, wherein, when more than one encoded grouping key values are selected as the one or more encoded grouping key values, the particular dense grouping key value represents a pointer to locations in which the more than one encoded grouping key values are stored.

13. A method comprising:

receiving a query that aggregates a measure column of a fact table based on an aggregate function and joins the fact table with a dimension table on a join key column, wherein data of the fact table is stored in one or more storage data units:

each storage data unit storing a respective data portion of the fact table,

the respective data portion comprising one or more column vectors corresponding to one or more columns of the fact table,

each cell element of the one or more column vectors of said respective data portion corresponding to a corresponding row and a corresponding column, of the one or more columns, of the fact table, said each cell element, of the one or more column vectors, comprising a respective dictionary encoded value of a corresponding value at the corresponding column and at the corresponding row, wherein the respective dictionary encoded value is mapped to the corresponding value of said each element by a respective dictionary data structure of the corresponding column;

in response to receiving the query, for a particular data storage unit of the one or more data storage units:

based on the query, identifying a fact join key vector of the one or more column vectors of the particular data storage unit and a fact join key dictionary data structure corresponding to the fact join key vector;

generating a dictionary-grouping key mapping based on the fact join key dictionary data structure and a dense grouping key data structure, the dense grouping key data structure representing a mapping of a plurality of unique dimension join key values of the join key column of the dimension table to grouping key values of a grouping key of the dimension table;

generating a transformed vector based on the dictionary-grouping key mapping and the fact join key vector, the transformed vector representing the grouping key values of the dictionary-grouping key mapping arranged according to the fact join key vector;

based on the transformed vector, aggregating a column vector in said particular data storage unit that corresponds to said measure column; and

wherein the method is performed by one or more computing devices.

14. The method of Claim 13, wherein the dictionary-grouping key mapping is stored in a mapped dictionary data structure that stores a mapping of unique dictionary encoded fact join key values of the fact join key dictionary data structure with corresponding dense grouping keys of a plurality of dense grouping key representing the grouping key values of the dense grouping key data structure.

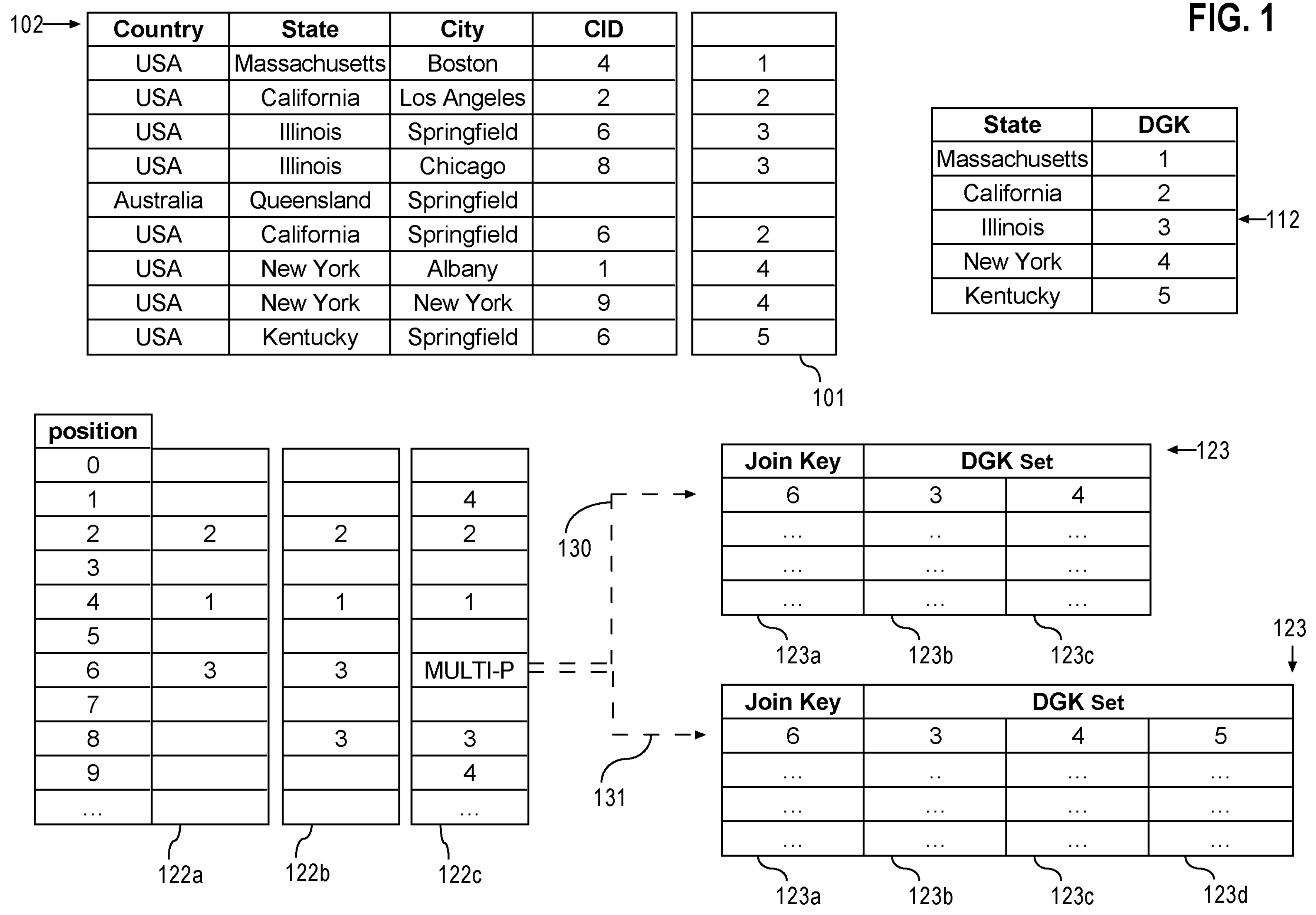
15. The method of Claim 14, wherein generating the transformed vector further comprises:

generating a cell element of the transformed vector for each cell element of the fact join key vector by selecting a dense grouping key value from an entry of the mapped dictionary data structure, that corresponds to said each cell element of the fact join key vector, and

storing the selected dense grouping key value in the cell element of the transformed vector.

16. One or more non-transitory computer-readable media storing instructions which, when executed by one or more hardware processors, cause the performance of the method recited in any of Claims 1-15.

FIG. 1



| Country | State | City | CID |
|---|---|---|---|
| USA | Massachusetts | Boston | 4 |
| USA | California | Los Angeles | 2 |
| USA | Illinois | Springfield | 6 |
| USA | Illinois | Chicago | 8 |
| Australia | Queensland | Springfield | |
| USA | California | Springfield | 6 |
| USA | New York | Albany | 1 |
| USA | New York | New York | 9 |
| USA | Kentucky | Springfield | 6 |

| |
|---|
| 1 |
| 2 |
| 3 |
| 3 |
| |
| 2 |
| 4 |
| 4 |
| 5 |

| State | DGK |
|---|---|
| Massachusetts | 1 |
| California | 2 |
| Illinois | 3 |
| New York | 4 |
| Kentucky | 5 |

| position | | | |
|---|---|---|---|
| 0 | | | |
| 1 | | | 4 |
| 2 | 2 | 2 | 2 |
| 3 | | | |
| 4 | 1 | 1 | 1 |
| 5 | | | |
| 6 | 3 | 3 | MULTI-P |
| 7 | | | |
| 8 | | 3 | 3 |
| 9 | | | 4 |
| … | | | … |

| Join Key | DGK Set | |
|---|---|---|
| 6 | 3 | 4 |
| … | .. | … |
| … | … | … |
| … | … | … |

| Join Key | DGK Set | | |
|---|---|---|---|
| 6 | 3 | 4 | 5 |
| … | .. | … | … |
| … | … | … | … |
| … | … | … | … |

FIG. 2

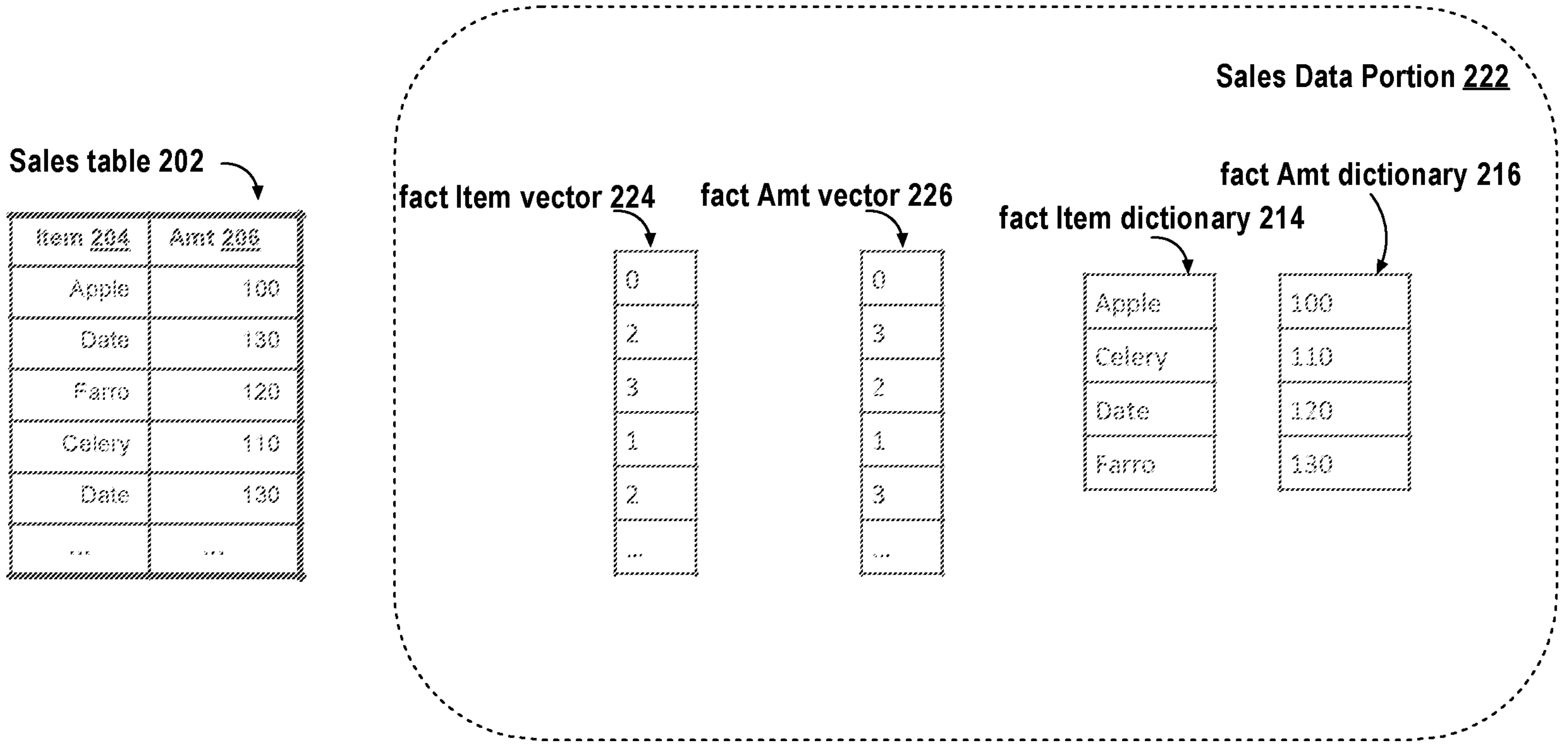

Sales table 202
Item 204
Amt 206
Apple
100
Date
130
Farro
120
Celery
110
Date
130
...
...
Sales Data Portion 222
fact Item vector 224
0
2
3
1
2
...
fact Amt vector 226
0
3
2
1
3
...
fact Item dictionary 214
Apple
Celery
Date
Farro
fact Amt dictionary 216
100
110
120
130

FIG. 3

Food table 302

| Category | Class | Item |
|---|---|---|
| Produce | Fruit | Apple |
| Produce | Fruit | Banana |
| Produce | Vegetable | Celery |
| Produce | Fruit | Date |
| Produce | Vegetable | Eggplant |
| Produce | Grain | Farro |

dimension Item DGK 314

| Item | DGK |
|---|---|
| Celery | 0xFFFF |
| Banana | 0 |
| Date | 0 |
| Farro | 1 |
| Eggplant | 0xFFFF |
| Apple | 0 |

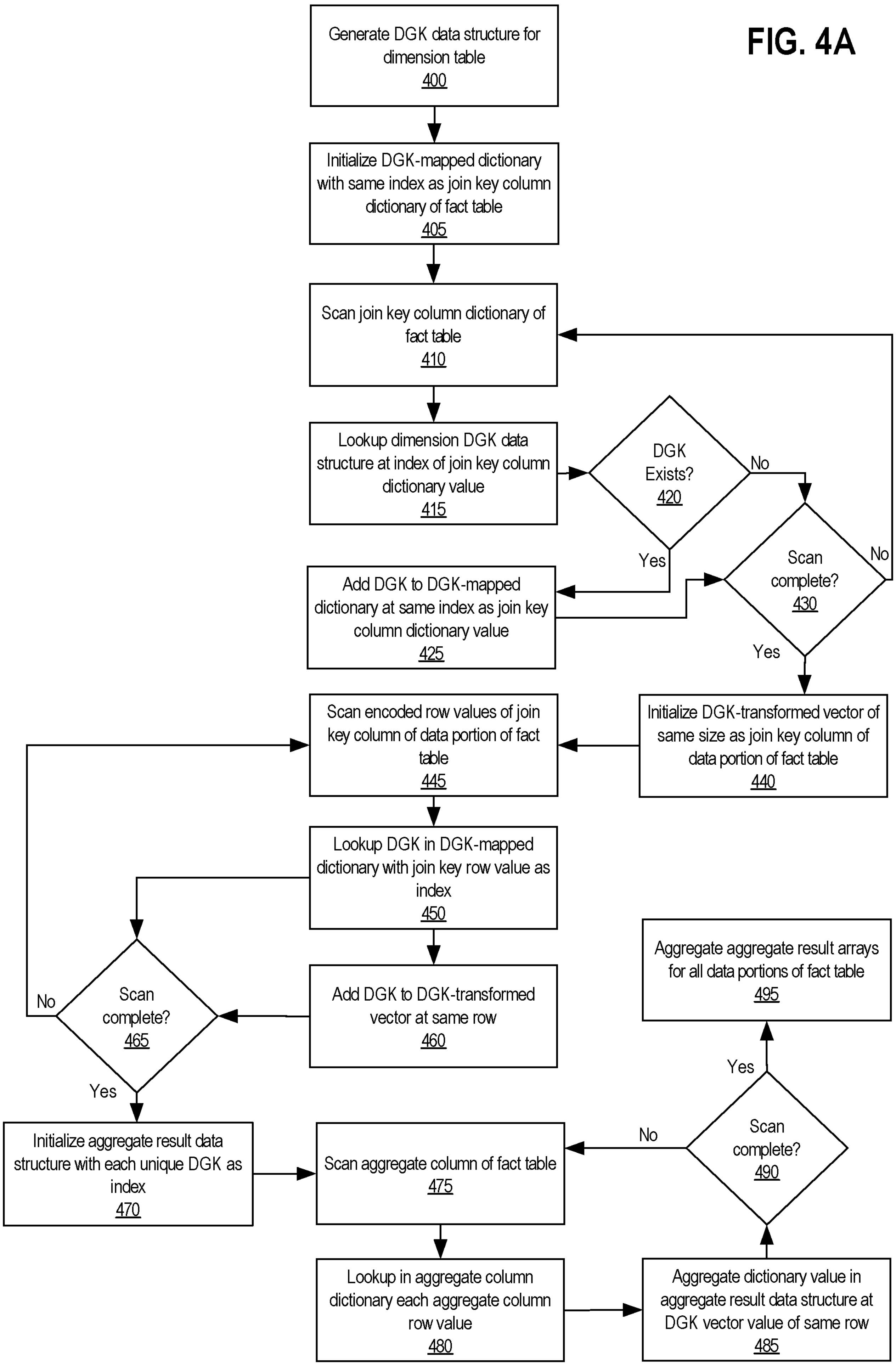

Generate DGK data structure for dimension table 400
Initialize DGK-mapped dictionary with same index as join key column dictionary of fact table 405
Scan join key column dictionary of fact table 410
Lookup dimension DGK data structure at index of join key column dictionary value 415
DGK Exists? 420
No
Yes
Add DGK to DGK-mapped dictionary at same index as join key column dictionary value 425
Scan complete? 430
No
Yes
Initialize DGK-transformed vector of same size as join key column of data portion of fact table 440
Scan encoded row values of join key column of data portion of fact table 445
Lookup DGK in DGK-mapped dictionary with join key row value as index 450
Add DGK to DGK-transformed vector at same row 460
Scan complete? 465
No
Yes
Initialize aggregate result data structure with each unique DGK as index 470
Scan aggregate column of fact table 475
Lookup in aggregate column dictionary each aggregate column row value 480
Aggregate dictionary value in aggregate result data structure at DGK vector value of same row 485
Scan complete? 490
No
Yes
Aggregate aggregate result arrays for all data portions of fact table 495

FIG. 4A

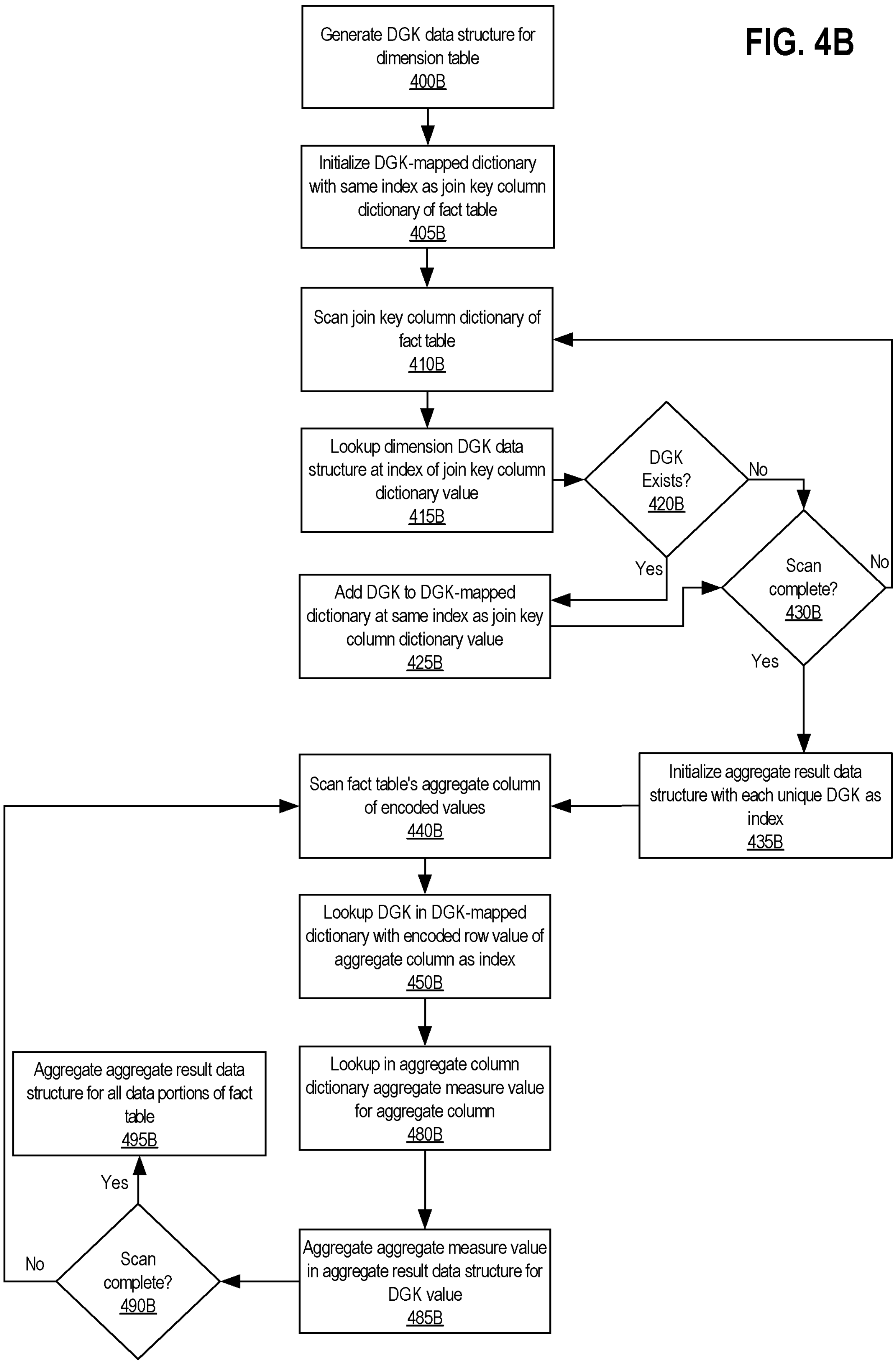

Generate DGK data structure for dimension table 400B
Initialize DGK-mapped dictionary with same index as join key column dictionary of fact table 405B
Scan join key column dictionary of fact table 410B
Lookup dimension DGK data structure at index of join key column dictionary value 415B
DGK Exists? 420B
No
Yes
Add DGK to DGK-mapped dictionary at same index as join key column dictionary value 425B
Scan complete? 430B
No
Yes
Initialize aggregate result data structure with each unique DGK as index 435B
Scan fact table's aggregate column of encoded values 440B
Lookup DGK in DGK-mapped dictionary with encoded row value of aggregate column as index 450B
Lookup in aggregate column dictionary aggregate measure value for aggregate column 480B
Aggregate aggregate measure value in aggregate result data structure for DGK value 485B
Scan complete? 490B
No
Yes
Aggregate aggregate result data structure for all data portions of fact table 495B

FIG. 4B

FIG. 5

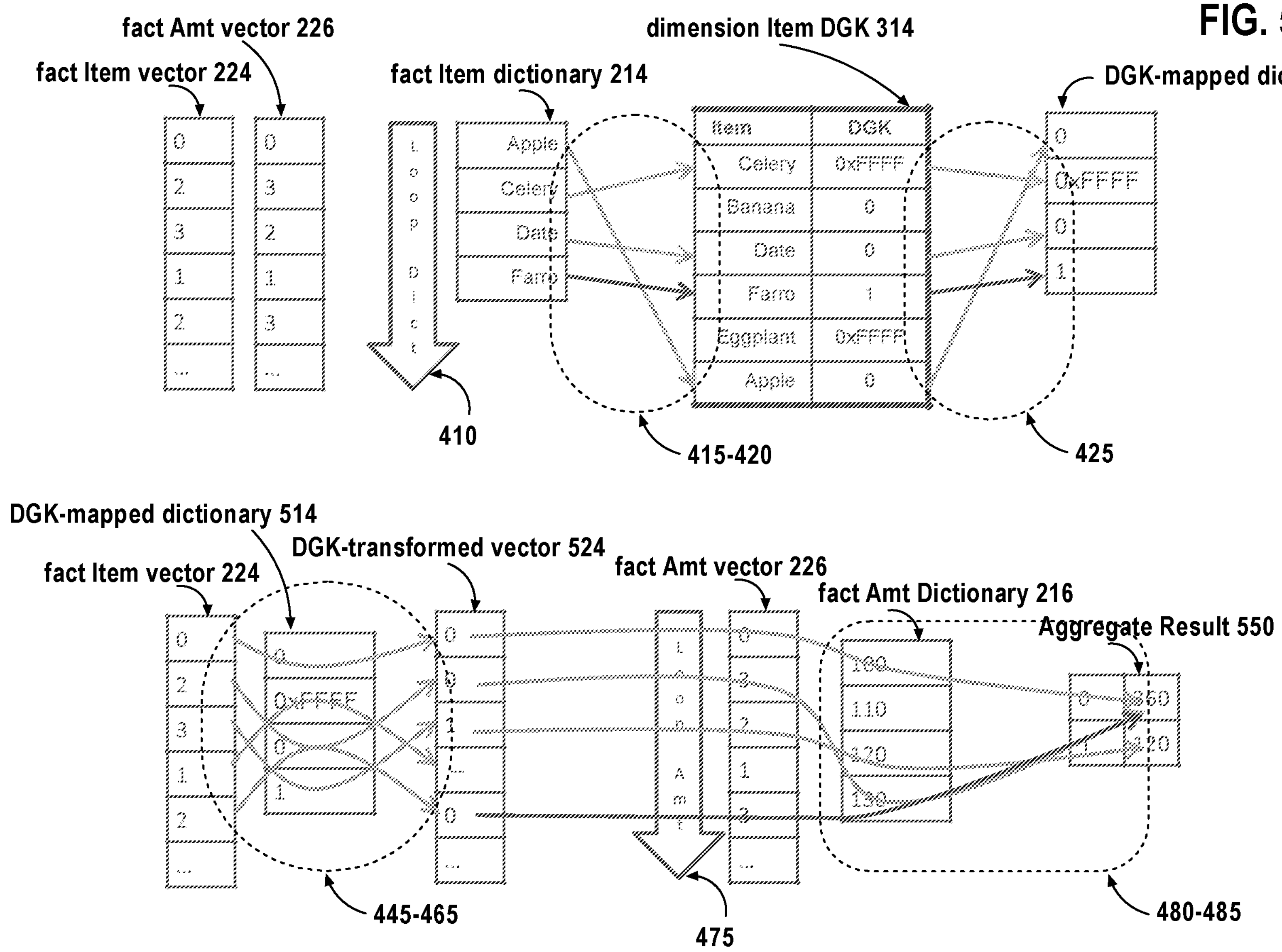
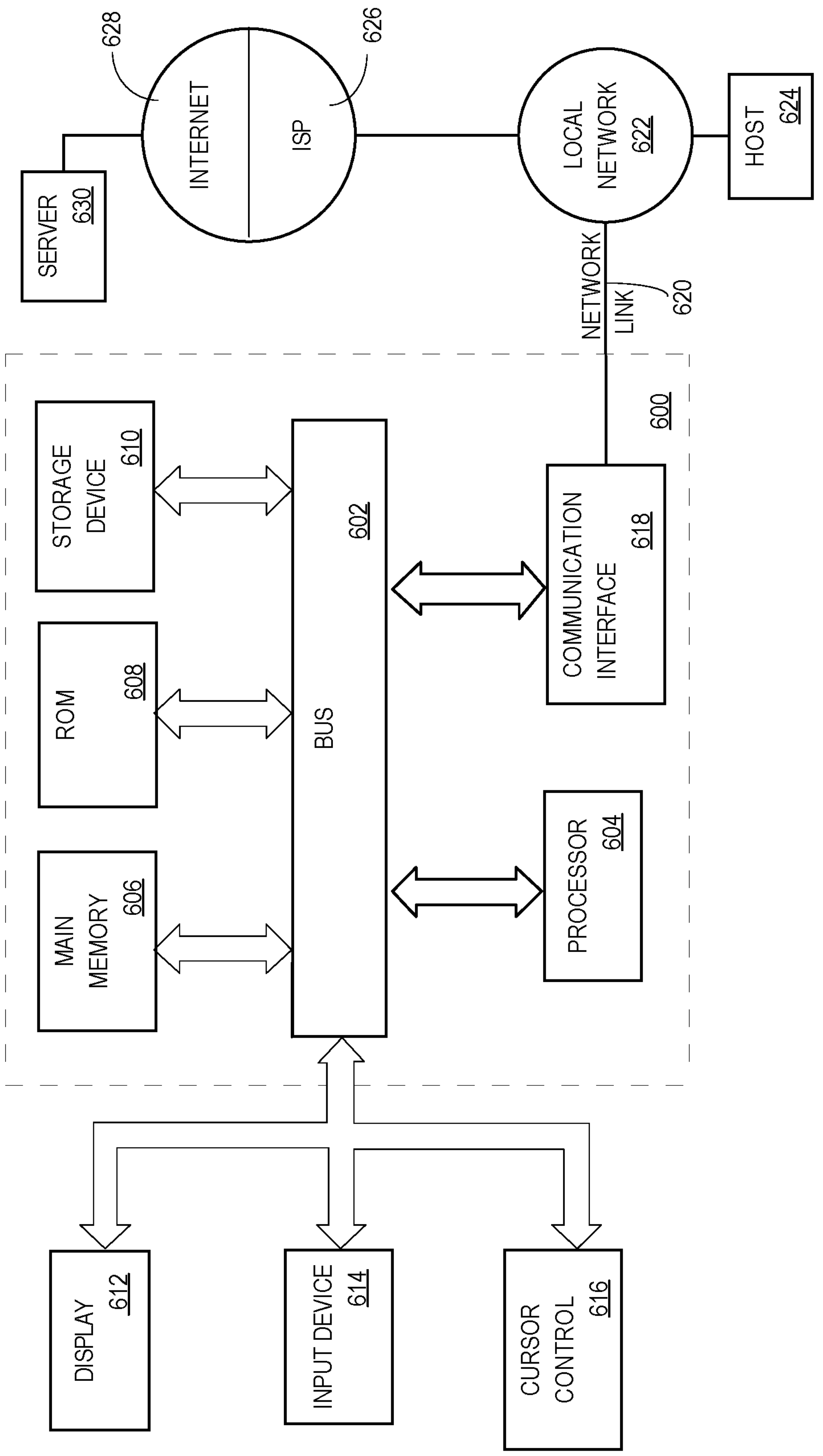

fact Amt vector 226
fact Item vector 224
fact Item dictionary 214
dimension Item DGK 314
DGK-mapped dictionary 514
0
2
3
1
2
...
0
3
2
1
3
...
Loop Dict
410
Apple
Celery
Date
Farro
Item
DGK
Celery
0xFFFF
Banana
0
Date
0
Farro
1
Eggplant
0xFFFF
Apple
0
415-420
0
0xFFFF
0
1
425
DGK-mapped dictionary 514
DGK-transformed vector 524
fact Item vector 224
fact Amt vector 226
fact Amt Dictionary 216
Aggregate Result 550
0
0xFFFF
0
1
0
0
1
...
0
...
Loop Amt
475
100
110
120
130
0
350
1
120
445-465
480-485

FIG. 6


SERVER 630
INTERNET
628
ISP
626
LOCAL NETWORK 622
HOST 624
NETWORK LINK
620
600
STORAGE DEVICE 610
ROM 608
MAIN MEMORY 606
BUS
602
COMMUNICATION INTERFACE 618
PROCESSOR 604
DISPLAY 612
INPUT DEVICE 614
CURSOR CONTROL 616

# INTERNATIONAL SEARCH REPORT

International application No
PCT/US2017/040612

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F17/30
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | LAHIRI TIRTHANKAR ET AL: "Oracle Database In-Memory: A dual format in-memory database", 2015 IEEE 31ST INTERNATIONAL CONFERENCE ON DATA ENGINEERING, IEEE, 13 April 2015 (2015-04-13), pages 1253-1258, XP032781164, DOI: 10.1109/ICDE.2015.7113373 [retrieved on 2015-05-26] page 1253, right-hand column, paragraph 2 - last paragraph; figure 1 page 1254, right-hand column, last paragraph page 1255, right-hand column, paragraph 5 page 1256, left-hand column, paragraph 3 - right-hand column, paragraph 2; figure 5 page 1257, left-hand column, paragraph 3 - last paragraph ----- | 1-16 |

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 8 September 2017 | 18/09/2017 |

| Name and mailing address of the ISA/ | Authorized officer |
|---|---|
| European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016 | Herri, Edmond |

1

# INTERNATIONAL SEARCH REPORT

International application No
PCT/US2017/040612

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| A | US 2015/088809 A1 (KOCIUBES ADAM [US] ET AL) 26 March 2015 (2015-03-26) abstract paragraph [0037] - paragraph [0051] paragraph [0055] - paragraph [0076] paragraph [0083] - paragraph [0090] ----- | 1-16 |
| A | DANIEL ABADI ET AL: "Integrating compression and execution in column-oriented database systems", ELECTRONIC PROCEEDINGS, ACM SIGMOD/PODS 2006 CONFERENCE : JUNE 26 - 29, 2006, CHICAGO, ILLINOIS, USA, ACM PRESS, NEW YORK, NY, 27 June 2006 (2006-06-27), pages 671-682, XP058263267, DOI: 10.1145/1142473.1142548 ISBN: 978-1-59593-434-5 page 675, right-hand column, paragraph 2 - last paragraph ----- | 1-16 |
| A | US 9 165 008 B1 (RAMESH BHASHYAM [IN] ET AL) 20 October 2015 (2015-10-20) abstract ----- | 1-16 |

1

# INTERNATIONAL SEARCH REPORT

**Information on patent family members**

International application No
PCT/US2017/040612

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| US 2015088809 A1 | 26-03-2015 | CN 105683956 A | 15-06-2016 |
| | | EP 3047399 A1 | 27-07-2016 |
| | | US 2015088809 A1 | 26-03-2015 |
| | | WO 2015042070 A1 | 26-03-2015 |
| US 9165008 B1 | 20-10-2015 | NONE | |