# Numbering Combinations for Compact Representation of Many-to-Many Relationship Sets

Savo Tomović[1*]

[1*]Faculty of Science, University of Montenegro, Cetinjska 2, Podgorica, 81000, Montenegro, Montenegro.

Corresponding author(s). E-mail(s): savot@ucg.ac.me;

**Abstract**

In this paper we propose an approach to implement specific relationship set between two entities called combinatorial relationship set. For the combinatorial relationship set $B$ between entity sets $G$ and $I$ the mapping cardinality is many-to-many. Additionally, entities from $G$ can be uniquely encoded with a pair of values *(h, k)* generated with the procedure for numbering combinations of entities from $I$. The encoding procedure is based on combinatorial number system that provides a representation of all possible $k$-combinations of a set of $n$ elements by a single number. In general many-to-many relationship sets are represented by a relation or table, while the combinatorial relationship is not physically stored as separate table. However, all information is encapsulated into a single column added to $G$. The new column is a candidate key in $G$. Additional operation named *Rank-Join* to fundamental relational-algebra is presented to combine information from $g$ and $i$ associated with a combinatorial relationship set. Motivation for combinatorial relationship originates from challenges in designing and implementing multivalued dimensions and bridge tables in data-warehouse models.

**Keywords:** data-warehouse modelling, multivalued dimensions, bridge tables, combinatorial numbering system

# 1 Introduction

In this paper we propose an approach to eliminate the table corresponding to many-to-many relationship set $B$ between entity sets $G$ and $I$. The presented technique can be employed to model any many-to-many relationship between tables $G$ and $I$ when a row from $G$ represents group containing elements or items from $I$. In that case, it is enough to extend the schema for $G$ by adding one new column that encodes group-item mapping based on combinatorial number system [1]. The added column uniquely identifies each row in $G$, so it can be declared as the primary key. In addition, the table corresponding to the many-to-many relationship between $G$ and $I$ can be eliminated from the database.

The motivation for this study originates from issues in conceptual and logical design of a data warehouse systems. In the data warehouse context relationship set with many-to-many cardinality is usually referred to as a bridge table [2–4].

Data warehouse conceptual schema can be expressed with entity-relationship diagram [5]. The design usually conforms to so called *star schema* [6]. There are two kinds of entities or tables in the star schema, namely *fact table* and *dimension tables*. The fact table is associated with each dimension table exclusively via a relationship set with a many-to-one cardinality [2–4]. The general star schema is illustrated in figure 1. Notice that the fact table can be considered as $n$-ary relationship set relating dimension tables.

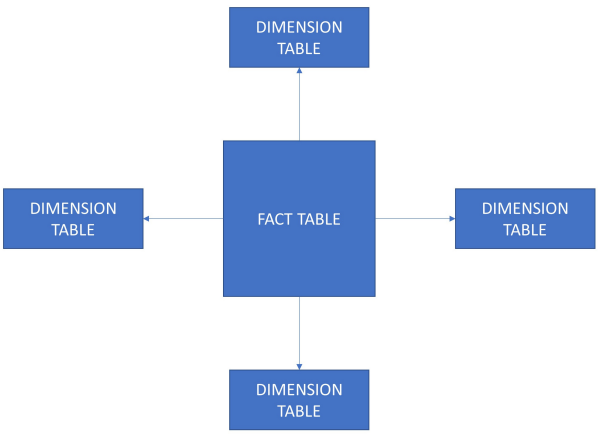


**Fig. 1** Star schema of a data warehouse

Many-to-one relationship between fact table and dimension tables means that dimensions are single valued [4]. In other words, the only one dimension value corresponds to one row or tuple from the fact table. Of course, it is possible to different dimension values appear in several rows in the fact table.

Multivalued dimension is dimension that has multiple values corresponding to one row in the fact table [4]. Situations with multivalued dimensions phenomenon can indicate that such dimension corresponds to a lower measurement grain than this contained in the fact table and consequently such dimension should be removed or redesigned [7]. However, there are several domains where multivalued dimensions are native and indivertible [3]. For example, in a healthcare system it is necessary to store all diagnosis and symptoms as well as medical procedures processed in a single patient visit. Unfortunately, it is often to have aged and hospitalized patients with more than 50 diagnosis [8, 9].

In general, introducing a bridge table is necessary to model multivalued dimensions [2–4], [10–15]. For example, the bridge table represents set of diagnosis that appear together in one patient visit. It contains one row for every diagnosis in a specific diagnosis group.

If a group has $n$ diagnosis, then the associated bridge table has $n$ rows for that group [4]. For example, let diagnosis $d_1, d_2, ..., d_n$ are encountered for one patient. They constitute a group that should be identified uniquely with the key $g_i$. All patients with the same medical assessment should be assigned to the same diagnosis group $g_i$. The corresponding bridge table between diagnosis table and diagnosis group table contains separate row for each diagnosis $d_j$ in the group $g_i$. More formally, for the group $g_i$ the bridge table contains $n$ rows of the following form $(g_i, d_j), 1 \leq j \leq n$.

The similar situation as with the bridge table in data warehouse modelling arises when multivalued attributes [5] from E-R diagram are mapped to relational model. For a multivalued attribute $I$ we create a table $B$ with the schema $B = (G_{PK}, I_{PK})$ [5]. The column $G_{PK}$ corresponds to the primary key of the entity set or relationship set of which $I$ is an attribute. The column $I_{PK}$ corresponds to $I$. Notice that the primary key in $B$ is $(G_{PK}, I_{PK})$.

In the rest of the paper we detail an approach to eliminate the table corresponding to many-to-many relationship set $B$ (bridge table) between entity $G$ playing the role *group* and entity $I$ playing the role *item* (or element of a group). All groups in $G$ are constituted exclusively from items in $I$. The main idea of the proposed approach is illustrated in figure 2. All information from the bridge table (i.e. which items belong to which group) can be encoded in only one column added to the relation schema corresponding to $G$. Additionally, this column is candidate key in $G$. Because of the fact that the new column can substitute an original primary key, we can say that this mapping does not require any additional storage. The new column can take over a role of the primary key in $G$. Also, we propose an extension to relational-algebra query language to support the previous implementation.

The paper is divided into several sections. The next section contains short overview of the literature. The third section enumerates main contributions of the paper. The fourth section introduces combinatorial bridge table and the algorithm for compact representation of a combinatorial bridge table. Algorithm is based on several procedures for representation of a given $k$-combination of a set of $n$ elements by a single number. Here $k$-combination

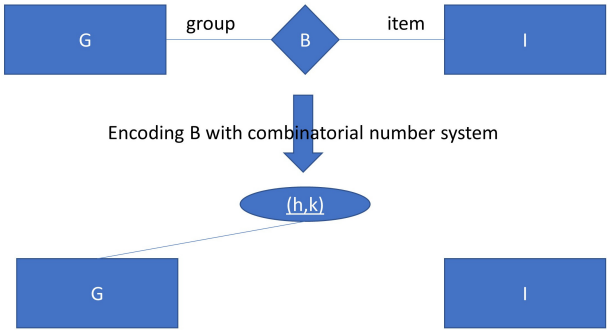


**Fig. 2** Compression of many-to-many relationship set

stands for a group constituted of elements from the corresponding combination. In the fifth section we propose additional operations *Rank-Join* and *Rank-Inverse-Join* to the relational algebra to support joins between group and item tables associated with combinatorial bridge table. The following section explains case study from healthcare domain that illustrates advances of proposed combinatorial bridge with respect to classic bridge. The last section contains final discussions and conclusion.

## 2 Related work

The fact table from a star schema mentioned in the first section contains measures (usually numeric) about the central subject. It is connected to dimension tables with keys, one for each dimension. Consequently, the fact table key consists of union of keys referencing dimension tables [16].

The most common relationship between the fact table and each dimension table is many-to-one [2–4]. That is, one row in the fact table should be connected to only one row in a dimension table, but one row in a dimension table can be associated with many rows in the fact table. Thus, the fact table can be considered as the table implementing many-to-many relation between any of two dimensions. Consequently, all dimensions are single valued [4].

But, real-world complexity makes it impossible to model problem in this manner. We will here mention just two challenges: [4]

- Multivalued dimension is dimension that has multiple values corresponding to one row in the fact table,
- Multivalued attribute occurs when a dimension row needs to capture more values for a single attribute.

For example, consider customer relationship management systems that constantly collect and store demographic and status data, contact information, professional and other skills about organization's customers. It is expectable that the number of different attributes regarding customer profile grows constantly. Consequently, this design will lead to potentially open-ended list of name-value pairs for storing customer data as it is presented in figure 3.

| Loan Application Data |
| --- |
| Photograph: <image> |
| Primary Income: $72345 |
| Other Taxable Income: $2345 |
| Tax-Free Income: $3456 |
| Alimony: $666 |
| Number Dependents: 4 |
| Years Since Bankruptcy: 8 |
| … 100 more name-value pairs… |

**Fig. 3** Name-value pair dimension

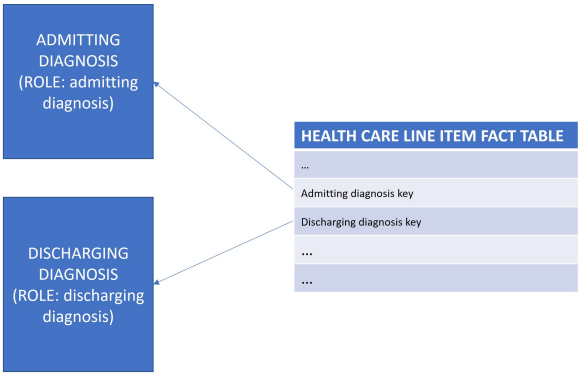


**Fig. 4** Roles played by the Diagnosis dimension

Fact table must have one column for each name-value pair that has ever appeared. Also, many of them will be filled with $NULL$ values, resulting in a very sparse representation [4].

Analogue situation occurs in other domains, such as healthcare system where it is necessary to store all diagnosis and symptoms as well as medical procedures processed in a single patient visit.

Multivalued dimensions can be modelled with positional design or bridge table design [2–4].

Positional design assumes to have one column for every dimension value. For example, when modelling patient diagnosis, we can design fact table to have separate column for every diagnosis. Situation where it is enough to store just admitting diagnosis and discharging diagnosis is illustrated in figure 4. There are two diagnosis dimension tables, one for each role.

Positional design is very attractive because the multivalued dimension is presented with named columns [3]. Also, BI tools can easily support such

design because the named columns are naturally presented in the tool. Query performance using these attributes can be significantly good if bitmap indexes are created on each column (because many columns contain low cardinality contents) [3]. On the other side, the positional design is not very scalable. One can easily run out of columns and it is awkward to add new columns. Positional designs can be scaled up to perhaps 100 or so columns before the databases and user interfaces become hard to maintain [2, 4].

Unfortunately, it is often to have aged and hospitalized patients with more than 100 diagnosis [8, 9]. Also, the number of different diagnosis can grow constantly.

Authors in [2–4], [7, 10–15] agreed that the solution to both these problems requires a new kind of table, called a bridge. Bridge table provides a wide variety of analytic possibilities, but in some cases occurrence of a bridge table can indicate errors in defining appropriate measurement grain in the fact table [7], producing inaccurate results (if such a table is not queried correctly, double-counting may result). The spectre of this risk is so great that many products will refuse to consider using a bridge at all [4]. However, there are a number of domains where multivalued dimensions are native and inevitable [2–4, 8, 9].

To conclude, there are a number of domains where multivalued dimensions are native and inevitable and also when the number of columns is above some limit and if new columns are added frequently, introducing the bridge table is necessary.

For example, the bridge table represents set of diagnosis that appear together in one patient visit. It contains one row for every diagnosis in a specific group. If a group has $n$ diagnosis, then the associated bridge table has $n$ rows for that group. For example, let diagnosis $d_1, d_2, ..., d_n$ are encountered for one patient. They constitute a group that should be identified uniquely with the key $g_i$. All patients with the same medical assessment would be assigned the same diagnosis group key $g_i$. The corresponding bridge table contains separate row for each diagnosis in a group. Formally, for the group $g_i$ the bridge table contains $n$ rows of the following form $(g_i, d_j), 1 \leq j \leq n$.

The concatenation of the group key and diagnosis key would uniquely identify each bridge table row. For example, in figure 5, d diagnosis with keys 100 and 300 constitute diagnosis group with key 1000, and diagnosis with keys 100, 200 and 300 constitute diagnosis group identified with 2000.

Further, the bridge table is connected to the fact table with many-to-many relationship. The fact table contains diagnosis group key as foreign key referencing diagnosis group bridge table. For example, the first row in the fact table (figure 5) holds the *DiagnosisGroup* key 1000. There are two rows in *Diagnosis Group Bridge* table for the co-occurrence of diagnosis 100 and 300. If the same diagnosis co-occur again, the same group number can be used.

In the literature, two methods for implementing a bridge table are presented [2–4]. The main difference between these methods is in the number of additional tables required to connect multivalued dimension to the fact table.

Consider multivalued dimension table represented with the following schema:

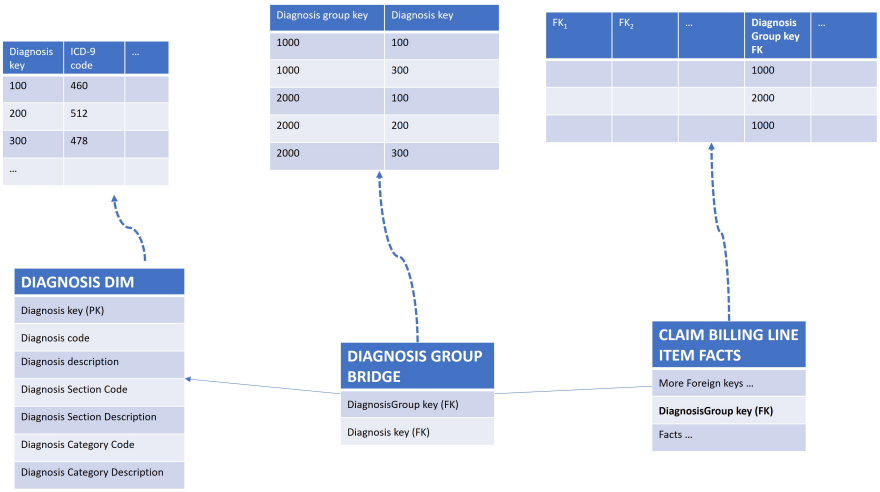


**Fig. 5** Multivalued diagnosis dimension table

$$Items = (Item_{PK}, ItemCode, ItemDescription, ItemType, ItemCategory, ...). \quad (1)$$

Let the schema for a fact table be as follows:

$$Facts = (FK_1, FK_2, ..., FK_p, Measure_1, Measure_2, ..., Measure_q). \quad (2)$$

Fact table contains foreign keys for every dimension and list of numeric facts. For multivalued dimension table $items(Items)$ it is necessary to join several $Item_{PK}$ values with the single record in the fact table.

Earlier in this section we explained why positional approach with adding new columns to the fact table, one for each $Item_{PK}$ value, is not scalable and generally acceptable solution.

The first approach for bridge table implementation is straightforward. Multivalued dimension table $Items$ is connected to $ItemGroup$ table via $ItemGroupBridge$ table. The $ItemGroup$ table is connected to the fact table in a way that the fact table is extended with just one column $ItemGroupForeingKey$. This column is foreign key from $Facts$ to the $ItemGroup$ table. According to this approach we end up with usual many-to-one relationship between the fact table $Facts$ and the $ItemGroup$ table that now plays role of an ordinary dimension table. On the other hand, items can be part of many different groups as well as one group can contain many different items. The bridge table $ItemGroupBridge$ implements many-to-many relationship between multivalued dimension $Items$ and $ItemGroup$. The previous implementation schema is presented in figure 6.

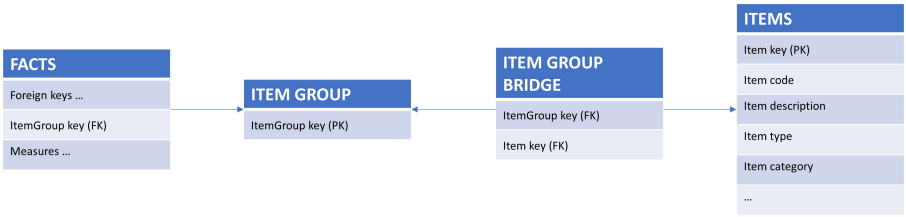


**Fig. 6** *ItemGroup* dimension to create many-to-one relationship with the fact table

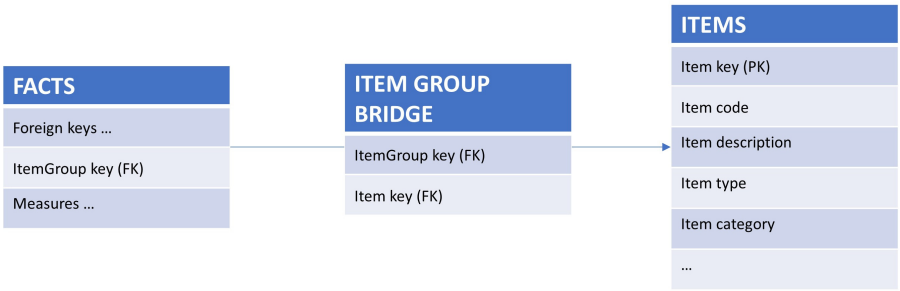


**Fig. 7** Many-to-many relationship between bridge table and facts

The second approach for bridge table implementation removes *ItemGroup* table and directly attaches *ItemGroupBridge* to the fact table. Now, *ItemGroupBridge* takes a role of a dimension table. However, relationship between fact table *Facts* and dimension table *ItemGroupBridge* is now many-to-many which means that assumption of foreign-key-primary-key relationship is broken. Implementation schema is shown in figure 7. This approach requires less tables (meaning less storage requirements and more efficient joins) but often it is not naturally supported in modelling tools [3].

In both approaches there is a bridge table with the schema $(ItemKey, ItemGroupKey)$. In the first approach the bridge is created between multivalued dimension *Items* and *ItemGroups* that represents a set of items associated with one row in the fact table. In the second approach the bridge table is created between multivalued dimension *Items* and the fact table.

Next section introduces a procedure for eliminating the bridge table (figure 6) or eliminating many-to-many relationship between bridge and fact tables (figure 7). The idea is to compress information from the bridge table by creating specific group key according to items included. That is where combinatorial numbering system is used for.

# 3 Main contributions

In a classic dimensional schema, each dimension is attached to a fact table with one-to-many relationship [2–4]. But there are a number of domains in which

a dimension occurs to be multivalued. For example, a patient in a hospital may have multiple simultaneous diagnosis and may receive multiple medical treatments and procedures. In this case, diagnosis and procedures appear to be multivalued dimensions. In addition, they must be connected to the fact table with corresponding bridge tables. Bridge table contains one row for each diagnosis/procedure in a group.

Apart healthcare, a number of situations in which a dimension is legitimately multivalued are recognized in the literature [2–4], [7, 10–15]: customer relationship management, human resources management, financial services, insurance etc. Different authors stated that it is not possible to simplify many-to-many relationship by decomposing it into a number of one-to-many relationships (positional approach) [2–4]. Briefly, this approach complicates reporting and supports only a finite number of relationships.

An alternative to the positional approach is to bridge the relationship between fact table and dimension table with a special table, which is fittingly referred to as a bridge table. This approach can accommodate groups of any size. For example, if three diagnosis are determined for one patient visit, a *diagnosis_group_key* is created for the co-occurrence and three rows are added to the diagnosis group table, one for each member. If a group of 15 diagnosis appears, a group is created and 15 rows are added. A group is also needed when only one diagnosis is registered. The group table will contain one row, associating the *diagnosis_group_key* with the single group member. Every fact will refer to a group.

Bridge table is used to store information about which values from dimension tables constitute together a specific group. Because of that, schema for a bridge table is of the form $(group\_key, item\_key)$.

In this paper we:

- propose **combinatorial bridge table**. Combinatorial bridge table is resulting from the procedure for compressing a classic bridge table by using combinatorial number system. The idea is to compress information from the bridge table by creating specific group key according to items included. The encoding procedure provides a representation of a group with $k$ items by a single number. Group of $k$ items can be considered as $k$-combination of a set of $n$ elements, where $n$ is total number of items. With this approach, we construct a combinatorial bridge table with single-column schema *combinatorial_group_key* that is significantly smaller than the classic bridge table. More precisely, if average number of items per group is $w$, combinatorial bridge table is $O(w)$ times smaller than corresponding classic bridge table. In other words, the classic bridge table will have $w$ rows for each group, while the combinatorial bridge table will use just one row for a group, regardless of number of items contained.
- introduce **combinatorial relationship set** to model any many-to-many relationship set $B$ between entity sets $G$ and $I$ in situations when $G$ is playing the role *group* containing *items* from $I$. Instead of creating a separate table for $B$ we propose to extend $G$ with one column that is encoded

with combinatorial number system generated by primary key domain of $I$ (combinatorial group key from the previous paragraph). The new column is candidate key in $G$.

- propose **relational algebra operations *Rank-Join* and *Rank-Inverse-Join*** to reconstruct all information from $g \bowtie b \bowtie i$. In other words, combinatorial encoding of many-to-many relationship set is without loss of information.

# 4 Compressing many-to-many relationship sets

In [17] authors pointed out that the bridge table size could increase significantly with increasing average group size, i.e. number of items assigned to each group. For example, when fact table contains 1000000 rows and in average 5 items constitute one group, the size of the bridge table can be as twice as the fact table size depending on the size and number of other fields in the fact table row.

In this section we propose the algorithm for efficient representation of the bridge table that encodes group-items mapping. The algorithm is based on combinatorial number system [1].

We will consider situations from figure 6 and figure 7 separately because of clarity, although both of them can be treated simultaneously. With symbol $\leftarrow$ we denote one-to-many relationship. With symbol $\rightarrow$ we denote many-to-one relationship.

In figure 6 we can recognize that the role *group* is played by a table $ItemGroup \equiv G$ and that a table $Items \equiv I$ plays the *item* role. These tables are associated with a bridge table, i.e. many-to-many relationship set $ItemGroupBridge \equiv B$. This approach is denoted by $G \leftarrow B \rightarrow I$.

In figure 7 we can recognize that table $G$ is not present. There are tables $Items \equiv I$ playing *item* role and $ItemGroupBridge \equiv B$ that provides group-item mapping. The table with *group* role is omitted because it is assumed that such bridge table can be attached directly to the fact table. In other words, for any group it is sufficient to know only group identifier stored in the bridge table. This design is denoted by $B \rightarrow I$.

Let $I = \{i_1, i_2, ..., i_n\}$ be set of item keys. Without loss of generality we can assume that all $i_j, 1 \leq j \leq n$ are integers. Additionally, we can assume that $i_1 < i_2 < ... < i_n$. In combinatorial number system, group of items $(i_{j_1}, i_{j_2}, ..., i_{j_k})$ can be represented with just one integer. The procedure named *RankGroup* is an implementation of the mapping [1]. It takes group of $k$ items $(i_{j_1}, i_{j_2}, ..., i_{j_k})$ as input and returns the ordinal position $h, 1 \leq h \leq \binom{n}{k}$ of the later. The result $h$ indicates that $(i_{j_1}, i_{j_2}, ..., i_{j_k})$ is lexicographically the $h^{th}$ combination of size $k$ of the set $I = \{i_1, i_2, ..., i_n\}$.

In other words, the *RankGroup* preserves lexicographical ordering in the set of $k$-combinations of $I = \{i_1, i_2, ..., i_n\}$. Thus, $RankGroup(i_1, i_2, ..., i_k) = 1$, $RankGroup(i_1, i_2, ..., i_{k-1}, i_{k+1}) = 2$ and finally $RankGroup(i_{n-k+1}, i_{n-k+2}, ..., i_n) = \binom{n}{k}$.

The pair $(h, k)$ is the key that uniquely identifies the group $(i_{j_1}, i_{j_2}, ..., i_{j_k})$. The complexity of the *RankGroup* method is $O(k^2)$. The pseudo-code of the *RankGroup* is given below. It is taken from [1].

**Algorithm 1** *RankGroup*$(i_{j_1}, i_{j_2}, ..., i_{j_k}, n)$

1: $sum \leftarrow 0$
2: **for** $l = 1..k$ **do**
3: $\quad sum \leftarrow sum + \binom{n - i_{j_l} - l + 1}{l}$
4: **end for**
5: $h \leftarrow \binom{n}{k} - sum$

The classic bridge table from the approach $G \leftarrow B \rightarrow I$ as well as from the approach $B \rightarrow I$ contains $k$ rows to represent group of items $g = (i_{j_1}, i_{j_2}, ..., i_{j_k})$. These rows are: $(g, i_{j_1}), (g, i_{j_2}), ..., (g, i_{j_k})$.

With *RankGroup* method the specific group will be represented with just one row $(h, k)$ regardless the number of items contained, where $h = RankGroup(i_{j_1}, i_{j_2}, ..., i_{j_k}, n)$. We need to store value for $k$ because we want to support groups of any size rather than fixing the number of items to any specific $k$. The pair $(h, k)$ is candidate key in a bridge table.

The discussed method reduces bridge table size by $O(group\_width\_avg)$ times where *group_width_avg* is number of items per group in average.

The procedure $RankGroupInverse(h, k, n)$ [1] takes as arguments group key $(h, k)$ and $n = \mathsf{I}$ and returns corresponding set of items $\{i_{j_1}, i_{j_2}, ..., i_{j_k}\}$ such that $h = RankGroup(n, i_{j_1}, i_{j_2}, ..., i_{j_k})$. The complexity of *RankGroupInverse* is $O(nk)$ [1]. The pseudo-code of the *RankGroupInverse* proposed in [1] is given below.

**Algorithm 2** $RankGroupInverse(h, n, k)$

1: $(c_{j_1}, c_{j_2}, ..., c_{j_k}) \leftarrow ORDERINV(n, k, \binom{n}{k} - h)$
2: $(i_{j_1}, c_{i_2}, ..., i_{j_k}) \leftarrow COMPLEMENT(n, c_{j_1}, c_{j_2}, ..., c_{j_k})$

The *RankGroupInverse* procedure ensures that compressing of a bridge table with the *RankGroup* method is without loss of information. In other words, group of items $g = (i_{j_1}, i_{j_2}, \ldots, i_{j_k})$ that is encoded with the pair $(h, k)$ by the *RankGroup* procedure can be reconstructed by the *RankGroupInverse* method.

For example, let two groups are defined. The group $G_1$ consists of items $I_1, I_2, I_3$. The group $G_2$ consists of items $I_1, I_2$. The classic bridge table is presented in figure 8 (left). It contains 5 rows, 3 for the group $G_1$ and 2 for the group $G_2$.

In the same figure, combinatorial bridge table is presented. The compressed bridge table contains just two rows, one per each group regardless of number of items it consists of. The row $(1, 3)$ represents the group $G_1$ because

**Algorithm 3** $ORDERINV(n, k, g = \binom{n}{k} - h)$

```
for l = k..1 do
    j ← n, c_l ← 0, t ← (n−1 choose l)
    while  do c_l == 0
        if ( then g ≥ t)
            c_l ← j
            t ← t × (j − l)/j
            j ← j − 1
            g ← g − binomc_l − 1l
        end if
    end while
end for
ORDERINV ← (c_1, c_2, ..., c_k)
```

**Algorithm 4** $COMPLEMENT(n, c_{j_1}, c_{j_2}, ..., c_{j_k})$

```
for l = 1..k do
    d_l ← (n + 1) − c_{k−j_l+1}
end for
```

| ItemGroupKey | ItemKey |
|---|---|
| G1 | I1 |
| G1 | I2 |
| G1 | I3 |
| G2 | I1 |
| G2 | I2 |

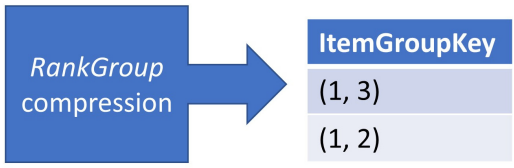


**Fig. 8** RankGroup compression of many-to-many relationship set

$RankGroup(I_1, I_2, I_3) = 1$. In other words, the combination $(I_1, I_2, I_3)$ is the $1^{st}$ combination of 3 elements of the set $I = \{I_1, I_2, ..., I_n\}$. Similarly, the row $(h = 1, k = 2)$ represents the group $G_2$ because $(I_1, I_2)$ is the $h = 1^{st}$ combination of $k = 2$ elements of the set $I = \{I_1, I_2, ..., I_n\}$.

Based on previous considerations we can propose implementation schemas for approaches $G \leftarrow B \rightarrow I$ (figure 6) and $B \rightarrow I$ (Figure 7).

Relational representation of the approach $G \leftarrow B \rightarrow I$ is as follows:

$$G = (Group_{PK}, groupAttrib_1, groupAttrib_2, ..., groupAttrib_p),$$
$$I = (Item_{PK}, itemAttrib_1, itemAttrib_2, ..., itemAttri_q),$$
$$B = (Group_{PK}, Item_{PK}).$$

.

We suggest to extend schema $G$ with a new column *groupRank* and obtain the schema $G_{rankc} = G \cup \{groupRank\}$. The *groupRank* values are populated with the *RankGroup* procedure in a way that $g[groupRank] = (h, k) = RankGroup(i_{j_1}, i_{j_2}, ..., i_{j_k})$.

In other words, set of items $\{i_{j_1}, i_{j_2}, ..., i_{j_k}\} \in I$ constitute one group that can be considered as $k$-combination of the set $\{1, 2, ..., n\}$. The *RankGroup* method provides unique encoding of each group, so the column *groupRank* is a candidate key or even can be taken as the primary key in $G$.

The method $RankGroupInverse(h, k, n)$ provides an inverse mapping from $g[groupRank] = (h, k)$ to the $k$-combination $(i_{j_1}, i_{j_2}, ..., i_{j_k})$.

Consequently, methods *RankGroup* and $RankGroupInverse$ provides us to delete a schema $B$ from database without loss of information. All information from $b(B)$ is encapsulated into *groupRank* column of the form $(h, k)$.

To conclude, relational representation of the approach $G \leftarrow B \rightarrow I$ with combinatorial bridge table is reduced to two schemas:

$$\begin{aligned} G_{rankc} = (&Group_{PK}, groupRank, groupAttrib_1, \\ &groupAttrib_2, ..., groupAttrib_p), \\ I = (&Item_{PK}, itemAttrib_1, itemAttrib_2, ..., itemAttri_q). \end{aligned} \tag{3}$$

.

Notice that the primary key $Group_{PK}$ in $G_{rankc}$ can be substituted with a new column *groupRank*.

Relational representation of the approach $B \rightarrow I$ is as follows:

$$\begin{aligned} I = (&Item_{PK}, itemAttrib_1, itemAttrib_2, ..., itemAttri_q), \\ B = (&Group_{PK}, Item_{PK}). \end{aligned} \tag{4}$$

.

With combinatorial bridge table the previous is reduced to two schemas:

$$\begin{aligned} I = (&Item_{PK}, itemAttrib_1, itemAttrib_2, ..., itemAttri_q), \\ B_{rankc} = (&groupRank). \end{aligned} \tag{5}$$

.

Notice that the primary key in $B_{rankc}$ is *groupRank*. Again, we emphasize that $b_{rankc}(B_{rankc})$ is *group_width_avg* times less than $b(B)$.

In Figure 9 we propose graphical representation of the combinatorial bridge table. Tables $G$ and $I$ are presented along with combinatorial bridge $B$. The combinatorial bridge is depicted with the dashed line. It is directed from $G$ to $I$. Direction indicates that $G$ plays *group* role and $I$ plays *item* role.

# 5 Extending Relational Algebra

In this section we introduce *combinatorial relationship set*. As we stated earlier relationship set with many-to-many cardinality is usually referred to as a

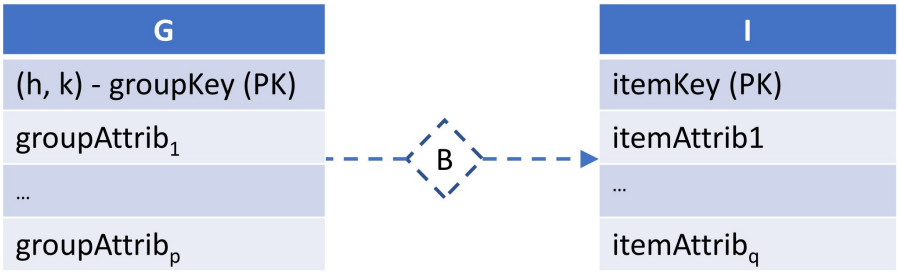


**Fig. 9** Combinatorial relationship set

bridge table in data-warehouse context. Previous section details on procedures to compress classic bridge table and obtain combinatorial bridge table.

The same procedures can be applied to relationship set $B$ with many-to-many cardinality between table $G$ with *group* role and table $I$ with *item* role resulting in *combinatorial relationship set*.

Some extensions to relational algebra in the data warehouse context are proposed in [18–20]. Following such ideas, we propose additional operation *Rank-Join* to the relational algebra to support join between group and item tables associated with combinatorial relationship set. We will use the symbol $\triangleright$ for the *Rank-Join*.

Notice that it is not feasible to write equivalent queries in SQL query language to join table with combinatorial relationship set. It is due to relational database servers do not implement *RankGroup* and *RankGroupInverse* methods that are essential for compressing combinatorial relationship set.

To combine information from relations $g = g(G), i = i(I)$ and $b(B)$, where $b$ is tabular representation of classic many-to-many relationship set between $g$ and $i$ the following relational algebra expression must be calculated

$$g \bowtie b \bowtie i \tag{6}$$

where natural-join operation is denoted by the symbol $\bowtie$.

In the rest of this section we define additional operations to relational algebra to obtain the same information from $g_{rankc} = g_{rankc}(G_{rankc})$ and $i = i(I)$ (resulted from the design $G \leftarrow B \rightarrow I$) and from $b_{rankc} = b_{rankc}(B_{rankc})$ and $i = i(I)$ (resulted from the design $B \rightarrow I$).

The *Rank-Inverse-Join* operation is unary operation that returns a relation containing rows that are result of applying *RankGroupInverse* function to specified column from its argument relation. *Rank-Inverse-Join* is denoted by Greek letter $\alpha$. Number of item keys in the table $I$ is subscript to $\alpha$. The column name on which *RankGroupInverse* transformation should be done is also subscript to $\alpha$. The argument relation as usual follows in parentheses.

The pseudo-code of the procedural implementation of the query $\alpha_{(n,groupRankColumn)}(g)$ can be expressed as follows:

**Algorithm 5** *RankInverseJoin*$(groupRankColumn, g, n)$

```
result ← {}
for (h, k) in g[groupRankColumn] do
    current_combination = RankGroupInverse(h, k, n)
    for item in current_combination do
        result ← result ⋃ ((h, k), item)
    end for
end for
RankInverseJoin ← result
```

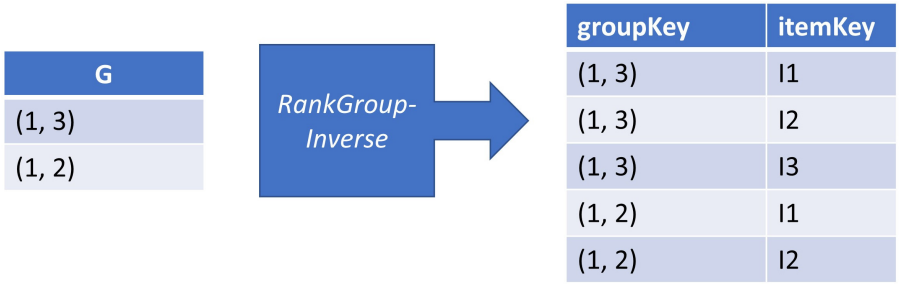


**Fig. 10** *Rank-Group-Inverse* operation

Thus, the relation that results from the query $\alpha_{n,groupRank}(g_{rankc})$ is shown in Figure 10. The similar results is obtained from the expression $\alpha_{n,groupRank}(b_{rankc})$ when the the design $B \rightarrow I$ is considered.

The *Rank-Join* operation is binary operation that allows us to combine *Rank-Group-Inverse* operation and a natural join into one operation as follows:

$$g_{rankc} \triangleright_{n,groupRank} i = g_{rankc} \bowtie \alpha_{n,groupRank}(g_{rankc}) \bowtie i \tag{7}$$

Subscripts to $\triangleright$ are $n$ and $groupRank$. The last expression provides the same information as the expression $g \bowtie b \bowtie i$.

Similarly, expression

$$b_{rankc} \triangleright_{n,groupRank} i = \alpha_{n,groupRank}(b_{rankc}) \bowtie i \tag{8}$$

provides the same information as the expression $b \bowtie i$.

# 6 Case study - Hospital Datawarehouse

In this section an illustration from real life problem is presented and usability of discussed approach is demonstrated. The motivation for this work originates from the challenges in designing and implementing data-warehouse system devoted to hospital management in American healthcare system.

The raw data was exported by the author directly from information system of one hospital acquisition and management company. The data was separated among several tables in a relational database. All records originated from

four different hospitals in California. All hospitals belong to the same hospital management company, so transactional data from every hospital are stored together in the same database. Specific data preprocessing procedures were necessary to be designed and implemented on the raw data to obtain usual star schema design.

The fact table about inpatients and diagnosis bridge table are taken as an example. Table 1 contains data about number of patients who were admitted to the hospital for further treatment (inpatients) and number of distinct diagnosis registered on a single patient. There are in total 700 different diagnosis coded in a standard International Classification of Disease (ICD-9) format. The covered period was 2013 - 2017.

**Table 1** Inpatients

| Number of distinct diagnosis per patient | Number of patients |
|---|---|
| ≥11 | 39342 |
| 10 | 9402 |
| 9 | 13506 |
| 8 | 20036 |
| 7 | 32723 |
| 6 | 53660 |
| 5 | 69580 |
| 4 | 83743 |
| 3 | 130781 |
| 2 | 185211 |
| 1 | 102747 |

From Table 1, it can be calculated that total number of records in the fact table is 740731. Average number of distinct diagnosis registered in one patient visit is 4.

Let us estimate a classic bridge size when it is implemented with the approach $B \rightarrow I$ (figure 7). There are 740731 facts joined to 4 (in average) distinct diagnosis in a diagnosis group yielding $740731 \times 4 = 23703392$ records. There are two key fields in the bridge table (Diagnosis Group key and Diagnosis key), causing that total bridge table size is $23703392 records \times 2 field \times 4 Bytes = 189,627,136 Bytes$.

With *RankGroup* method every diagnosis group can be represented with just one pair $(h, k)$ regardless the number of diagnosis contained. The resulting bridge table is $740731 facts \times 8 Bytes = 5,925,848 Bytes$. Such bridge table is $189,627,136/5,925,848 = 32$ times less than the corresponding classic bridge.

The analogous approach is applied on a bridge table that models medical procedures taken on single patient visit. It is usual for inpatients to get treatment consisting of a group of procedures. Every procedure group is encoded with combinatorial number system in the same way as it is done with diagnosis groups. Resulting combinatorial bridge table is several times less than the original classic bridge.

In addition, compressing with combinatorial numbering system is applied not only on inpatient data, but also on other fact tables including emergency room, laboratory, radiology, revenue, quality care, patient satisfaction etc. In all cases we were able to significantly reduce classic bridge table by compressing it with *RankGroup* method.

As future work we plan to create a number of larger datasets from this domain and confirm the expected performances of compressing bridge table with combinatorial number system.

# 7 Conclusion

In this paper we introduce combinatorial many-to-many relationship set $B$ between entities $G$ and $I$ when entities from $G$ play role of *group* composed of entities from $I$ that plays *item* role. Instead of creating separate table for $B$ we propose to extend $G$ with one column that is encoded with combinatorial number system generated by primary key domain of $I$. The new column is candidate key in $G$.

The motif for the proposed approach originates from design and implementation issues regarding to multivalued dimension modelling in data warehouses.

Combinatorial relationship can be considered as bridge table in data-warehouse context. Classic bridge table is used to store information about which values from dimension tables constitute together a specific group. Because of that, schema for a classic bridge table is of the form $(group\_key, item\_key)$.

Combinatorial bridge table is created by encoding $(group\_key, item\_key)$ mapping with combinatorial number system. Consequently, combinatorial bridge table is of single-column schema and it is significantly smaller than corresponding classic bridge table.

More precisely, if average number of items per group is $w$, combinatorial bridge table is $O(w)$ times smaller than the classic bridge table. In other words, classic bridge table will have $w$ rows for each group, while combinatorial bridge table will use just one row for a group regardless of number of items contained (i.e. pair of values $(h, k)$ generated by combinatorial numbering system).

This approach is not only applicable in data-warehouse modelling. The similar situation arises when multivalued attribute of an entity set from E-R diagram should be translated to relational model. Instead of creating new table it is possible to add new column to the entity set and encode corresponding collection of attribute values with combinatorial number system.

The previous encoding transformation is without loss of information. To provide access to all information behind such encoding, we propose relational algebra operations *Rank-Join* and *Rank-Inverse-Join*. These operations are sufficient to reconstruct all information from $g \bowtie b \bowtie i$.

# References

[1] Akl GS (1989): Generating Combinations Lexicographically. In: The Design of Parallel and Analysis Algorithms. Prentice Hall, New Jersey, pp 147-150

[2] Vaisman A, Zimanyi E (2014): Conceptual Data Warehouse Design. In: Data Warehouse Systems Design and Implementation. Springer, pp 89-116

[3] Kimball R, Ross M (2013): Advanced Dimension Techniques. In: The Data Warehouse Toolkit, Third Edition. Wiley, pp 62-67

[4] Adamson C (2010): Multi-Valued Dimensions and Bridges. In: Star Schema The Complete Reference. Mc Graw Hill, pp 549-610

[5] Silberschatz A, Korth HF, Sudarshan S (2011): Database Design and the E-R Model. In: Database system concepts. McGraw-Hill, pp 259-321

[6] Han J, Kamber M (2012): Data Warehousing and Online Analytical Processing. In: Data Mining Concepts and Techniques, 3rd Edition. The Morgan Kaufman Series in Data Management Systems, pp 125-184

[7] Mansmann S, Neumuth T, Scholl MH (2007): Multidimensional Data Modelling for Business Process Analysis. Proceedings of the 26th International Conference on Conceptual Modelling. 10.1007/978-3-540-75563-0_4

[8] Dutta R (2013): Health Care Data Warehouse System Architecture for Influenza (Flu) Diseases. National Conference on Advancement of Computing in Engineering Research. 10.5121/csit.2013.3208

[9] Visscher SL, et al. (2017): Developing a standardized healthcare cost data warehouse. BMC Health Services Research. https://doi.org/10.1186/s12913-017-2327-8

[10] Sharma D (2014): Dimension Modelling Techniques in Business Intelligence. International Journal of Emerging Trends & Technology in Computer Science.

[11] Ballard C, Herreman D, Schau D, Bell R, Kim E, Valencic A (1998): Data Analysis Techniques. In: Data Modeling Techniques for Data Warehousing. International Business Machines Corporation, pp 9-13

[12] Prasad RN, Acharya S (2013): Multidimensional Data Modeling. In: Fundamentals of Business Analytics. Wiley India Private Limited.

[13] Husemann B, Lechtenborger J, Vossen G (2000): Conceptual Data Warehouse Design. Proceedings Of International Workshop on Design and Management of Data Warehouses.

[14] Moody D, Kortink MR (2000): From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart design. Proceedings of International Workshop on Design and Management of Data Warehouses.

[15] Mansmann S, Neumuth T, Burgert O, Roger M, Scholl MH (2008): Conceptual Data Warehouse Design Methodology for Business Process Intelligence. In Complex Data Warehousing and Knowledge Discovery for Advanced Retrieval Development: Innovative Methods and Applications. 10.4018/978-1-60566-748-5

[16] Tryfona N, Busborg F, Christiansen JGB (1999): starER: A Conceptual Model for Data Warehouse Design. DOLAP. https://dl.acm.org/doi/pdf/10.1145/319757.319776

[17] Song IL, Rowen W, Medsker C, Ewen E (2001): An Analysis of Many-to-Many Relationships Between Fact and Dimension Tables in Dimensional Modeling. Proceedings of the International Workshop on Design and Management of Data Warehouses.

[18] Kuijpers B, Vaisman A (2017): An Algebra for OLAP. Intelligent Data Analysis. https://doi.org/10.3233/IDA-163161

[19] Ravat F, Teste O, Tournier R, Zurfluh G (2008): Algebraic and graphic languages for OLAP manipulations. International Journal of Data Warehousing and Mining. 10.4018/jdwm.2008010102

[20] Mansmann S, Scholl MH (2008): Visual OLAP (Online Analytical Processing). In Encyclopedia of Database Systems. https://doi.org/10.1007/978-1-4614-8265-9_447

# 8 Declarations

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript. The authors have no relevant financial or non-financial interests to disclose. Manuscript has no associated data.