Algorithms used to obtain aggregated value sets from relational databases

MIRELA-CATRINEL VOICU
Faculty of Economic Sciences,
"Nicholas Georgescu-Roegen" Interdisciplinary Platform
West University of Timișoara
ROMANIA

Abstract: - For n fields used for grouping we can obtain $2^n$ aggregation types. When n is not small, $2^n$ is a considerable value. With a new model for the specification of aggregation types, we present ways in which we can obtain any subsets of aggregation types, starting from n fields used for grouping. The algorithms concern cases in which:
- The specification of aggregation types is made after we insert the data in the tables.
- The specification of aggregation types is made before we insert the data in the tables. In order to obtain the results really fast, we also use hypercubes.
- The case of one table (the fields used for grouping and for aggregation are from the same table) where we use one hypercube and we can specify the aggregation types in any moment (after or before the data insertion in the table) - model which can be used for questionnaires.

These algorithms can be used in many economic analyses, in order to obtain complex reports. We will also present an example which concerns the sales in a supermarket chain. In our study we have worked with databases from Access and we have made the algorithm implementation in Delphi, but the implementation can also be made in other programming environments and we can also use other relational databases.

Key-Words: aggregated value sets, hypercube, economic analyses, relational databases, programming environment

1 Introduction

Data analysis is used in more departments or sectors like finance departments, marketing departments, manufacturing sector, sales departments etc. Data analysis applications typically aggregate data across many dimensions (n>=0). For aggregations, many tools are known. We recall some from these:

An SQL aggregate function (AF) produces one answer:
Select AF (attribute_value) from table
which corresponds to one aggregation type.

An SQL aggregate function (AF) and the Group by operator produce also one answer:
Select attribute_1,...,attribute_n, AF (attribute_value) from table group by attribute_1,...,attribute_n
which corresponds to one aggregation type.

The Rollup operator (from Oracle) – corresponds to $n+1$ aggregation types.

The Cube operator – corresponds to $2^n$ aggregation types (the maximal set possible).

In the case in which n is not small, $2^n$ is a considerable value. In the case in which the user wants to obtain (in the same result table) other subsets of aggregated values than the sets given by the known tools, we propose two algorithms.

In the beginning, we recall how we want to refer to the sets of aggregation types (see [6], [7]). In order to specify the aggregation types, we propose that the user make specifications, which contain combinations of “m” and/or “f” and/or “u”, where:

- $f$ – means one field used for grouping,
- $u$ – means one field not used for grouping,
- $m$ – means zero, one or more fields not used for grouping.

Now, we consider the table presented in Fig. 1. Here, the fields field1, field2, field3, field4, field5 form the maximal set used for grouping and the field fvalue is used for aggregation.
The specification $m f m$ produces the results presented in Fig. 2 (which correspond to five aggregation types).

The specification $m f u f m$ produces the results presented in Fig. 3 (which correspond to three aggregation types).

The specification $f m f m$ produces the results presented in Fig. 4 (which correspond to four aggregation types).

In such specifications we can also eliminate some fields for a certain $f$.

The user must specify the $n$ fields used for grouping. Using specifications, which are composed of “$f$”, “$m$” or “$u$”, the user can obtain any wanted subsets of aggregation types for the $n$ specified fields.

For starters, we consider a database with a snowflake scheme. We suppose that the user wants to obtain one result table (containing aggregated values) or many results tables having the same header. In the latter case, the difference between result tables is given by the used aggregation types.

The algorithm presented in this paper supposes that, before introducing the records in tables, we will specify: the fact table, dimension tables, the relationships of the tables used for aggregations and the aggregation types (see Fig. 6 and Fig. 8).

In the moment in which we will introduce a new record in the fact table, we also will construct a hypercube (see Fig. 11). Its records will be formed as the set of all subsets obtained from the fields used for grouping and the fields used for aggregations, corresponding to the new record (from the fact table) and the records from the dimension tables corresponding to the specified relationships.

From this hypercube we will save in certain tables (additionally used in database) only the records which form the sets which provide the values used for the specified aggregation types (see Fig. 12). After this moment the hypercube will be deleted. From these last additional tables, using sample SQL statements and the Group By operator we will obtain the desired result tables (containing the aggregated values), in a very short time as in Fig. 13.

2.1 Specification of aggregation types

2.1.1 Create a table

In the moment in which we create a table, we save some data (referring to all the fields of the new table) in an additional table (see Fig. 5). The additional table from the Fig. 5 contains the following fields:

2 Algorithm in which, before inserting records in tables, we will specify the desired aggregation types
The field `table_name` contains the name of the tables from database.
- The field `field_name` contains the name of the fields from database.
- The field `data_type` contains the data type of each field from database.
- The field `t` refers to the name of an additional table corresponding to a work table from the database.
- The field `tf` represents a code for the unique identification of each field from the database.

When we create a new table, the corresponding table (its name is given by application and it is saved in the field `t` from Fig. 5) is also created. For each field from the initial table it has as fields: the field code (given by `tf` from Fig. 5) and the field value (for such a table, see Fig. 5, Table 2 and Fig. 10). In addition, we have a field `norecord` for the number of each record in the initial table. This last field also exists in the initial tables and it is generated automatically by application.

In the moment in which we have created all the tables, we will pass to the following step: we will specify the fields used for grouping, for aggregation and the relationships between tables.

2.1.2 Tables, fields, relationships, aggregation functions, fact table

Now, we consider a database with seven tables (see the Table 1, `i` between 1 and 7), each table with six fields. The field `norecord` is used to generate automatically the number of a record from a table, for its unique identification.

<table>
<thead>
<tr>
<th><code>table_name</code></th>
<th><code>field_name</code></th>
<th><code>data_type</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>tabi</td>
<td>norecord</td>
<td>Integer</td>
</tr>
<tr>
<td>tabi</td>
<td>f1   PK</td>
<td>text(10)</td>
</tr>
<tr>
<td>tabi</td>
<td>f2</td>
<td>text(10)</td>
</tr>
<tr>
<td>tabi</td>
<td>f3</td>
<td>text(10)</td>
</tr>
<tr>
<td>tabi</td>
<td>f4</td>
<td>text(10)</td>
</tr>
<tr>
<td>tabi</td>
<td>f5</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Table 1 Database structure

The user must specify the tables, fields, relationships, aggregation functions like in Fig. 6. In Fig. 6 we must follow these steps:

1 – select the used tables;

2 – select the fields used for grouping (in order in which they form the header for the result table – these fields will be indexed from 0 to `n-1`, where `n` represents the number of used fields);

3 – here the user must specify the used tables, the relationships and the fact table.

4 – this step is used to allow the user to introduce the aggregation functions (one or more);

5 – at this step many tables will be created:

5.1. - a table which contains, for each field used for grouping, the following information: the field name, its table name and its field index.

5.2. - a table which contains information on the fields used for grouping (initial and additional tables).

5.3. – we will have a table (for the additional tables) which contains information on the relationships specified in Fig. 6.

5.4. – a table which contains the additional table name, corresponding to the fact table.

2.1.3 Specification of the aggregation types
Now, the user must specify the sets of aggregation types in the following way (see Fig. 8):
1 – “prepare” the form for a new specification of aggregation types (clear some components on the form);

2 – selection of \( m, f, u \) in the desired way;

3 – the user must select each \( f \) (from CheckListBox), step by step. For a selected \( f \), with a click on the field name, the user can eliminate the field (all possible fields are displayed in a ListBox, like in Fig. 8), which will not be used. In this moment a table will be created, which contains the field index, the table name and the field name for all selected fields for the corresponding \( f \).

4 – with a click on the command button “create aggregation type table” we will obtain two tables (presented in Fig. 9), which contain as records the aggregation types (for the initial and additional tables). Here we have a cartesian product between the records from the tables corresponding to each \( f \). Using the indexes, we can formulate conditions for where clauses (according to the presence of \( m \) or \( u \) at the left or right of each \( f \)).

In Fig. 9, in the left table, the first record - “\( tab3f31, tab3f32 \)” means that for grouping we will have the fields \( f31 \) and \( f32 \) from the table \( tab3 \).

In Fig. 9, in the right table, the first record - “\( t3f1t3f2t1f5t3f5 \)” means that for grouping we will have the fields \( f31 \) and \( f32 \) from the table \( t3 \). We use “\( t1f5t3f5 \)” corresponding to the fields used for aggregations (\( t1f5 \) from \( t1 \) and \( t3f5 \) from \( t3 \)).

Now, we repeat the steps 1-4 until the moment when we will have specified all aggregation types.

### 2.2 Insert records

We consider the records presented in Table 2. Here \( i \) is between 1 and 7. At this step we must consider two situations. The first situation is when we insert a new record in a different table than the fact table (i.e. in a dimension table).

<table>
<thead>
<tr>
<th>Table i</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
<th>f5</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>1</td>
</tr>
<tr>
<td>tab2</td>
<td>a2</td>
<td>a4</td>
<td>a5</td>
<td>b2</td>
<td>1</td>
</tr>
<tr>
<td>tab3</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td>b3</td>
<td>1</td>
</tr>
<tr>
<td>tab3</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td>b4</td>
<td>1</td>
</tr>
<tr>
<td>tab4</td>
<td>a3</td>
<td>a6</td>
<td>b4</td>
<td>c4</td>
<td>1</td>
</tr>
<tr>
<td>tab5</td>
<td>a4</td>
<td>b5</td>
<td>c5</td>
<td>d5</td>
<td>1</td>
</tr>
<tr>
<td>Tab6</td>
<td>a5</td>
<td>b6</td>
<td>c6</td>
<td>d6</td>
<td>1</td>
</tr>
<tr>
<td>Tab7</td>
<td>a6</td>
<td>b7</td>
<td>c7</td>
<td>d7</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 2 Records from tables**

In this moment, in the corresponding additional table, we will add two records, like in Fig. 10 (\( t1 \) is the additional table corresponding to \( tab1 \) – see also Fig. 5). The first record is ("", null, ",", null, ",", null, ",", null, "null, no_record") and the second record is (field_1_code, field_1_value, field_2_code, field_2_value, field_3_code, field_3_value, field_4_code, field_4_value, field_5_code, field_5_value, no_record).

The second situation is when we insert a new record in the fact table. Here we also insert the corresponding records in the additional table, but in addition we will make the following three operations:

Fig. 10 Records in the additional table
1 - Using the table (see also step 5.3 from Sub-subsection 2.1.2) which corresponds to the relationships from Fig. 6 (relationships on the additional tables), we extract from the additional tables (see field \( t \) from Fig. 5 and Fig. 10) the records which correspond to all the relationships. For each table used in the relationship specified in Fig. 6, we will obtain only two records (like the records from Fig. 10).

For the case presented in Fig. 6, we will obtain seven such tables corresponding to the new record from the fact table and to the records from the dimension tables which will respect all relationships.

2 – Now, we will generate a hypercube – a table which contains all subsets of the set formed with twelve fields (ten for grouping and two for aggregation), using:
- the last tables (which have the same structure as the table presented in Fig. 10),
- the table which contains information on the fields used for grouping (ten fields in our example presented in Fig. 6, see step 5.2 from Sub-subsection 2.1.2),
- the table which contains information on the fields used for aggregation (two fields in our example presented in Fig. 6, see step 5.2 from Sub-subsection 2.1.2).

In addition, the hypercube contains a field – \( f \) - which has as values a concatenation of all code fields corresponding to each record (see Fig. 11). This field contains, in fact, all possible aggregation types using the twelve fields.

3 – Here we use the hypercube and the tables which contain the aggregation types, like in the right table from Fig. 9. For each specification of aggregation types, we will create a table which contains the corresponding values from hypercube. For example, for the specification \( m f m \) we will obtain the table presented in Fig. 12.

When we have constructed all these tables, we will delete the hypercube. These tables (like the table from Fig. 12) contain only the records which are necessary in order to obtain the aggregated values for a certain specification (like \( m f m \) in Fig. 8).

In Fig. 12, the field \( n t i \) (here \( i \) is between 1 and 7) represents the record number from the table \( t_i \) (see Fig. 10). Using these fields, when we modify or delete a record in the initial tables (like \( t_i \)), the same changes affect the corresponding additional table and the tables like the one in Fig. 12.

2.3 Final result tables

Using the tables like the one presented in Fig. 12 (which provide the values corresponding to a specification of aggregation types), and the table from Fig. 7, now, we can obtain the final results. We will use, as example, the form presented in Fig. 13.
Here we can select if we want only one table which contains the results corresponding to all aggregation types (for which we have selected the corresponding checkbox), or we can create a result table for each specification. For the last case, in Fig. 14, we present such a result table for the specification $m_f m$.

3 One table, one hypercube and all aggregation types

In the case in which for aggregations we use fields from only a single table, we can construct a hypercube associated with this table, in order to specify the aggregation types in any moment (i.e., also when we have records in table). We present in this section such an algorithm.

3.1 Create a table and insert records

3.1.1 Create a table

In the moment in which we create a new table, we save some data (referring to all the fields of the new table) in an additional table (as the table from Fig. 5). This additional table contains the following fields:

- The field `table_name` contains the name of the tables from database.
- The field `field_name` contains the name of the fields from database.
- The field `data_type` contains the data type of each field from database.
- The field $t$ refers to the name of the additional table (hypercube) corresponding to a table from the database. The corresponding table hypercube has the same structure as the table from the Fig. 10 (see also Fig. 15).
- The field $tf$ represents a code for the unique identification of each field from the database.

3.1.2 Insert a record

Now, we consider, as example, a table named `tab1` which contains twelve fields (named $f_i$, where $i$ is between 1 and 12). We consider the first ten fields with `text(10)` data type and the last two fields with `integer` data type. Automatically, in each new table, the application adds a new field, named `norecord`, which is used for the unique identification of each record in table.

In table `tab1` we insert the following record: 

\[("a", "b", "c", "d", "e", "f", "g", "h", "i", "j", 1, 1)\].

When we insert a new record, the record number will be automatically generated in the initial table and in hypercube table.

In this moment we will also insert in the table `cube_11f1` (the additional table hypercube associate to the table `tab1`) the corresponding hypercube of the new record (see Fig. 15). Our record is a set with twelve elements and its hypercube contains the set of all subsets formed with these twelve values.

3.2 Specification of aggregation types

3.2.1 Table, fields, aggregation functions

The user must specify the tables, fields, relationships, aggregation functions as in Fig. 16. In this figure we must follow these steps:

1 – select the used table;
2 – select the fields used for grouping (in order in which they form the header for the result table – these fields will be indexed from 0 to $n-1$, where $n$ represents the number of used fields);
3 – this step is used to allow the user to introduce the
aggregation functions (one or more);

4 – at this step many tables will be created:
4.1. - a table which contains, for each field used for grouping, the following information: the field name, its table name and its field index.
4.2. - a table which contains information on the fields used for grouping (initial and additional table).
- a table which contains information on the fields used for aggregation (initial and additional table).
- a table which contains information on the fields which will not be used for grouping or for aggregation (initial and additional table);
For each field used for grouping or aggregation, in these new tables we have a corresponding field in which we save as records: the field name, its table name, the field code, the name of the additional table (see, as example, Fig. 7).

3.2.2 Aggregation types
Now, the user must specify the sets of aggregation types in the following way:
1 – “prepare” the form for a new specification of aggregation types (clear some components on the form);
2 – selection of \( m, f, u \) in the desired way;
3 – the user must select each \( f \) (from CheckListBox), step by step. For a selected \( f \), with a click on the field name, the user can eliminate the field (all possible fields are displayed in a ListBox, like in Fig. 17), which will not be used. In this moment a table will be created, which contains the field index, the table name and the field name for all selected fields for the corresponding \( f \).

![Fig. 17 Aggregation types](image)

In the moment in which we have the table for each \( f \) from a specification (see Fig. 17), we can pass to the following step:
4 – with a click on the command button “create aggregation type table” we will obtain a table (see, as example, such tables in Fig. 9), which contains as records the aggregation types (for the additional tables). Here we have a cartesian product between the records from the tables corresponding to each \( f \) from a specification of aggregation types.
Now, we repeat the steps 1-4 until the moment when we will have specified all aggregation types.

3.3 Final result tables
We can select if we want only one table which contains the results corresponding to all aggregation types (see, as example, Fig. 13), or we can create a result table for each specification.

![Fig. 18 Sub-hypercube used for aggregations](image)

Fig. 18 Sub-hypercube used for aggregations
For the last case, in Fig. 20, we present such a result table for the specification \( m f m f m \) from Fig. 17.

![Fig. 19 The set of records which provide the values used for an aggregation type specification (here, \( m f m f m \) from Fig. 17)](image)

Fig. 19 The set of records which provide the values used for an aggregation type specification (here, \( m f m f m \) from Fig. 17)

In both cases, first, we will extract from hypercube (see Fig. 18) the set which corresponds to the fields used for grouping and for aggregation (see step 4.2 from Sub-subsection 3.2.1). In addition, we will have a new field - \( f \) - which has as values a concatenation of all code fields corresponding to the each record. This field
contains, in fact, all possible aggregation types using the nine fields (seven for grouping and two for aggregation). The table from the Fig.18 is in fact a sub-hypercube of the hypercube from Fig.15 (corresponding only to the nine fields: seven used for grouping and two used for aggregation).

From the sub-hypercube from Fig.18, we will extract only the records which provide the values used for an aggregation type specification. For each specification of aggregation types we will have such a table, like one presented in Fig.19. In the moment in which we have obtained all these last tables, we will delete the sub-hypercube.

With the tables like the one in Fig. 19 and using simple SQL statements (see, as example, Fig.13), we will obtain the result tables (see Fig.20).

4 The algorithm used for any database

This algorithm supposes the construction of tables, which contain as records the aggregation types and the use of SQL statements (by application) for each such record. This means that if we have (for example) 100 aggregation types there are in use 100 SQL statements for aggregations.

We can make the connection with any database (here, from Access), because additional constructions on the databases are not necessary.

We consider, for example, the tables presented in Figure 21.

4.1 Tables, fields, relationships, aggregation functions

The user must specify the tables, fields, relationships, aggregation functions like in Figure 22. In this figure we must follow these steps (like in Sub-subsection 2.1.2):

1 – select the used tables;

2 – select the fields used for grouping (in order in which they form the header for the result table – these fields will be indexed);

3 – this step is used to allow the user to introduce the aggregation functions (one or more);

4 – here the user must specify the tables used and (if necessary) the relationships. Here a table will be created, which has as record the fields (and the corresponding indexes) used for grouping (see the Figure 23).

4.2 Specification of aggregation types

Fig. 24 Specification of aggregation types
Now, the user must specify the sets of aggregation types in the following way (like in Sub-subsection 2.1.3):

1 – “prepare” the form for a new specification of aggregation types (clear some components on the form);
2 – selection of \( m, f, u \) in the desired way;
3 – the user must select each \( f \) (from CheckListBox), step by step. For a selected \( f \), with a click on the field name, he can eliminate the field (all possible fields are displayed in a ListBox, like in Figure 24), which will not be used. At this moment a table will be created, which contains the field index, the table name and the field name for all selected fields for the corresponding \( f \). We present such tables in Figure 25 (for the specification \( m f m f m \)).

![Fig. 25 The corresponding tables for each \( f \) from the specification \( m f m f m \)](image)

4 – with a click on the command button “create aggregation type table” we will obtain a table, which contains as records the aggregation types (see Figure 26). Here we have a cartesian product between the records from the tables corresponding at each \( f \) (presented in Figure 25). Using the indexes, we can formulate conditions for where clauses (according to the presence of \( m \) or \( u \) at left or right of each \( f \)).

Now, we repeat the step 1-4 until the moment when we will have specified all aggregation types. We will obtain the results at the following step (see Figure 24):

5 – with a click on the command button “result tables”, for each table like the table presented in Figure 26 (these tables correspond to each specification of aggregation types), we will construct a corresponding result table, which contains the aggregated values, like in Figure 27.

![Fig. 26 A table, which contains as records the aggregation types](image)

![Fig. 27 A result table corresponding to the specification \( m f m f m \) of aggregation types](image)

For example, for the second record from the table \( interm1 \) from Figure 26, the application executes the following SQL statement:

```sql
insert aggregated values in a result table
Insert into r1
the fields used for grouping
Select Table1.field1 as "Table1_field1", NULL as "Table1_field2", Table1.field3 as "Table1_field3", NULL as "Table1_field5", NULL as "Table1_field4", NULL as "Table2_field1", NULL as "Table2_field5", NULL as "Table2_field4", NULL as "Table2_field3",
the aggregated data
min(Table1.fvalue) as min_Table1_fvalue,
max(Table2.fvalue) as max_Table2_fvalue
```

When we have the table for each \( f \) from a specification (see Figure 25), we can pass to the following step:

4 – with a click on the command button “create aggregation type table” we will obtain a table, which
From Table1, Table2
the relationships
Where Table1.field1=Table2.field2
the fields used for grouping
Group by Table1.field1, Table1.field3

This means that for the table presented in Figure 26, the application executes 36 SQL statements.

For this step the time can be a problem. When the number of aggregation types is increasing, the necessary time for obtaining the results is also increasing.

From the start, we can create only one result table which will contain the aggregated values, but we prefer to create a separate table for each specification (like in Figure 27), because, in this way, we can easily obtain the results only for some specifications presented before (not for all of them), without repeating the algorithm. We can also eliminate some fields from the result table. We present this possibility in the following subsection.

4.3 Final tables

Fig. 28 Final confirmations for result tables

Fig. 29 A result table for aggregated values

We confirm the desired specifications of aggregation types like in Figure 28. With a click on the field name we eliminate the field from the result table.

For the case presented in Figure 28, we will obtain the result table presented in Figure 29.

5 An example which concerns the sales in a supermarket chain

Data analysis is used in more departments or sectors like finance departments, marketing departments, the manufacturing sector, sales departments etc. Data analysis applications typically aggregate data across many dimensions (n>=0). Many analyses can be very complex and can concern a considerable number of aggregation types.

In order to illustrate the importance of this subject and its practical applicability, in this paper we consider as example data corresponding to the Monoprix supermarket chain. For analysis, we use information from the web site www.monoprix.fr. The database structure corresponding to the supermarkets chain is presented in Figure 30.

In the Monoprix supermarkets chain we find the following product categories: Mode & Accessories (subcategories: Skirts, Shirts, Tops, Accessories, Underwear, T-Shirt, Dress, Trousers, Jacket/Waistcoat & Winter coat); Beauty & Care (subcategories: Body Care, Face Care, Make-up); House & Leisure (subcategories: Pieces of furniture, Decoration Accessories, Kitchen Materials, Stationery); Food/Maintenance (subcategories: Soft drinks, Bread-pastry, Maintenance, Fruits, Vegetables, Breakfast, Cooked dishes, Fish, dairy product, Dry Products, Frozen, Meat, Alcoholic drinks, ice cream, sausages).

Starting from the database presented in Figure 30, we can make many analyses concerning the product sales. These analyses can concern the sales at country, department, city, district or shop level, etc. In order to simplify the presentation, we present a possible analysis type at shop level.

In order to easily observe the implementation of the algorithm from the Section 4 and the result tables, we fix the values for the following fields: Shop_name (eg. Monoprix Convention), Category (eg. Beauty&Care), Subcategory (Body_Care) and year (eg. 2006). In this case we can consider for grouping, only the fields Sub_subcategory, Brand, Consumer_type, Product_name, Promotion, Date, Month. Now we present the types of aggregations and their specifications (see the Table 3).

We select the tables and the fields, we specify the relationships, criteria and the aggregation functions like in Figure 31.

We specify the aggregation types and for each f we confirm the fields which will be used for grouping like in Figure 32.
Case 2.1. \textbf{f f m}
\begin{itemize}
  \item \textbf{f f m} Sub_subcategory, Brand
\end{itemize}

Case 3.1. \textbf{f f f m f m}
\begin{itemize}
  \item \textbf{f f m f m} Sub_subcategory, Brand, Consumer_type
  \item \textbf{f f m f m} Sub_subcategory, Brand, Product_name
  \item \textbf{f f m f m} Sub_subcategory, Brand, Promotion
  \item \textbf{f f m f m} Sub_subcategory, Brand, Date
  \item \textbf{f f m f m} Sub_subcategory, Brand, Month
\end{itemize}

Case 4.1. \textbf{f f f m f m}
\begin{itemize}
  \item \textbf{f f f m f m} Sub_subcategory, Brand, Consumer_type, Product_name
  \item \textbf{f f f m f m} Sub_subcategory, Brand, Consumer_type, Promotion
  \item \textbf{f f f m f m} Sub_subcategory, Brand, Consumer_type, Date
  \item \textbf{f f f m f m} Sub_subcategory, Brand, Consumer_type, Month
\end{itemize}

Case 4.2. \textbf{f f u f m f m}
\begin{itemize}
  \item \textbf{f f u f m f m} Sub_subcategory, Brand, Product_name, Promotion
  \item \textbf{f f u f m f m} Sub_subcategory, Brand, Product_name, Date
  \item \textbf{f f u f m f m} Sub_subcategory, Brand, Product_name, Month
\end{itemize}

Case 4.3. \textbf{f f u f f m f m}
\begin{itemize}
  \item \textbf{f f u f f m f m} Sub_subcategory, Brand, Promotion, Date
  \item \textbf{f f u f f m f m} Sub_subcategory, Brand, Promotion, Month
\end{itemize}

Case 5.1. \textbf{f f f f m f m}
\begin{itemize}
  \item \textbf{f f f f m f m} Sub_subcategory, Brand, Consumer_type, Product_name, Promotion
  \item \textbf{f f f f m f m} Sub_subcategory, Brand, Consumer_type, Product_name, Date
  \item \textbf{f f f f m f m} Sub_subcategory, Brand, Consumer_type, Product_name, Month
\end{itemize}

Case 5.2. \textbf{f f f f u m f m}
\begin{itemize}
  \item \textbf{f f f f u m f m} Sub_subcategory, Brand, Consumer_type, Promotion, Date
  \item \textbf{f f f f u m f m} Sub_subcategory, Brand, Consumer_type, Promotion, Month
\end{itemize}

Case 5.3. \textbf{f f u f f m f m}
\begin{itemize}
  \item \textbf{f f u f f m f m} Sub_subcategory, Brand, Product_name, Promotion, Date
  \item \textbf{f f u f f m f m} Sub_subcategory, Brand, Product_name, Promotion, Month
\end{itemize}

Case 6. \textbf{f f f f f m f m}
\begin{itemize}
  \item \textbf{f f f f f m f m} Sub_subcategory, Brand, Consumer_type, Product_name, Promotion, Date
  \item \textbf{f f f f f m f m} Sub_subcategory, Brand, Consumer_type, Product_name, Promotion, Month
\end{itemize}

\textbf{Table 3} Aggregation types

For obtaining result tables we can select all specifications of aggregation types or only some of them, like in Figure 33.

In the case in which we select all specifications of aggregation types, we will obtain the result presented in Figure 34.
Fig. 34 Result table corresponding to all specifications of aggregation types

If we select only the two first specifications of aggregation types (like in Figure 33), we will obtain the results presented in Figure 35. In both result tables (see Figures 34 and 35) the result presentation is affected by the order in which we introduce the specifications of aggregation types (see Figures 33). According to the situation, we can choose the intended order of specifications.

Fig. 35 Result table corresponding to the specifications \( ff \text{ } m \text{ } \) and \( ff \text{ } m \text{ } f \)

The field value from the table sales and the fields year, month, day and day_of_week from the table dates are additional fields. These fields are necessary only when we start the specification of the aggregation types, like in Figure 31, in order to use the algorithm proposed in the Section 4. When we have the result tables, like in the Figures 34 and 35, we can delete these fields.

6 Conclusion

The algorithms can be used for any type of databases. In the programming environment we use SQL statements to exploit the database. Here we have worked with databases from Access. In the case in which we want to work with another type of relational database, we must only modify (in the program) the SQL statement (if it is different from the corresponding SQL statement from Access).

We have presented the implementation in Delphi, but the implementation can also be made in other programming environments.

For \( n \) fields (used for grouping) from a database, we can obtain \( 2^n \) aggregation types. With our algorithms, we can obtain any subsets of aggregation types.

The algorithms from the Sections 2 and 3 concern a very short time for obtaining the results, but requires that the database be exploited only using the application.

We can propose this algorithm when it is often necessary to obtain various sets of aggregation values and the number of aggregation types is not very small.

Using hypercubes we can easily obtain any subsets of aggregation types, from the maximal possible set.
In the algorithm from the Section 2, before inserting records in tables, we must specify the desired aggregation types. We have presented the algorithm for a snowflake scheme. This algorithm is the same for a star scheme and can be extended to a constellation scheme or when the result tables have different headers. In the last two cases we must repeat all our constructions (see Figs. 6-9) separately for each case and we must use them when we introduce new records in fact table (see Figs. 11, 12).

In the algorithm from the Section 3 we can specify the desired aggregation types in any moment, but this algorithm is limited to the case in which we use fields from only one table (it can be used, for example, for questionnaires).

In the case in which a database is not exploited with the proposed applications from Sections 2 and 3, with the third algorithm, presented in the Section 4, we can also obtain the desired sets of aggregation types.

We must remember that if the number \( n \) of aggregation types is greater, the time for obtaining the results can be a problem (because \( n \) SQL statements are executed, instead of only one SQL statement - like in the first algorithm).

However, we can propose this algorithm when it is not often necessary to obtain such results or the number of aggregation types is not greater.

Our algorithms can be used in many other economic situations, as well. We have presented a situation at shop level but the analysis can be used in more departments or sectors such as finance departments, marketing departments, manufacturing sector, tourism etc.

References: