Proceedings of the 5th WSEAS Int. Conf. on APPLIED INFORMATICS and COMMUNICATIONS, Malta, September 15-17, 2005 (pp342-347)

Algorithm for obtaining aggregated value sets from multidimensional databases

MIRELA VOICU, GABRIELA MIRCEA
Faculty of Economic Sciences
West University of Timişoara
ROMANIA
http://www.fse.uvt.ro

Abstract: - For n fields (used for grouping) from a database, we can obtain 2^n aggregation types - the maximal set possible (obtained, for example, with the Cube operator from Oracle). In this paper we want to present one algorithm with which the user can obtain any subsets from this maximal set.

Key-Words: - SQL, aggregated value sets

1 Introduction

Data analysis applications typically aggregate data across many dimensions (n>=0).

A *SQL* aggregate function (*AF*) produces one answer:

Select AF (attribute_value) from table

which corresponds to one aggregation type.

A *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 one algorithm.

Firstly, we want to present how we want to refer the sets of aggregation types. 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 the *Figure 1*. Here, the fields *field1, field2, field3, field4, field5* form the maximal set used for grouping and the field *fvalue* is used for aggregation.

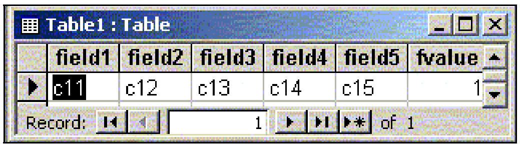


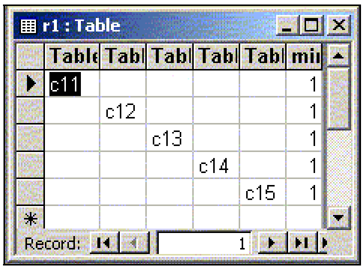
Table1 : Table

field1	field2	field3	field4	field5	fvalue
c11	c12	c13	c14	c15	1

Record: 1 of 1

Figure 1. An initial table.

The specification *m f m* produces the results presented in *Figure 2* (which correspond to five aggregation types).

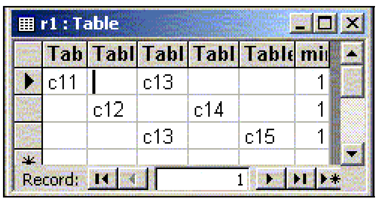

r1 : Table

Tabl	Tabl	Tabl	Tabl	Tabl	mi
c11					1
	c12				1
		c13			1
			c14		1
				c15	1

Record: 1

Figure 2. The result for *m f m*.

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

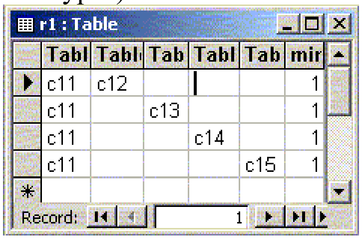

r1 : Table

Tab	Tabl	Tabl	Tabl	Table	mi
c11		c13			1
	c12		c14		1
		c13		c15	1

Record: 1

Figure 3. The result for *m f u f m*.

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


r1 : Table

Tabl	Tabl	Tab	Tabl	Tab	mir
c11	c12				1
c11		c13			1
c11			c14		1
c11				c15	1

Record: 1

Figure 4. The result for ***f m f m*.**

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

A database, generally, contains one or more tables. For aggregation, the user uses fields from one or more of these tables. He must specify the *n* (maximal number of fields used for grouping) fields. Using specifications, which are composed by "*f*" or/and "*m*" or/and "*u*", he can obtain any wanted subsets of aggregation types for the *n* fields specified.

We propose our original algorithm. The implementation is made in a programming environment (we work here, for example, in *Delphi*) and with a database (here we use databases from *Access*).

The algorithm supposes that

- any table is constructed by the application (here, proposed in *Delphi*) because, at the moment of construction, the application also constructs an additional table used for the aggregations which will concern the new table constructed;

- in the moment in which we insert (modify, delete) a new record in a table, the same changes affect the corresponding additional table;

- we construct a table, which contains the aggregation types. Using this new table and the additional tables from the database, executing (by our application) only one *SQL* statement, we can obtain the wanted result table.

It is very important that all actions on the database (create/delete tables, insert/delete/modify records, etc.) be made only with the proposed application.

2 Create a table

In the moment in which we create a table (like in the *Figure 5*), we save some data (referring to all the fields of the new table) in a certain table (see the *Figure 6*). The field *tf* from *Figure 6* wants to be a code for unique identification of each field from the database. The field *t* from *Figure 6* wants to be the name of the additional table corresponding to the new table (here, *tab3*).

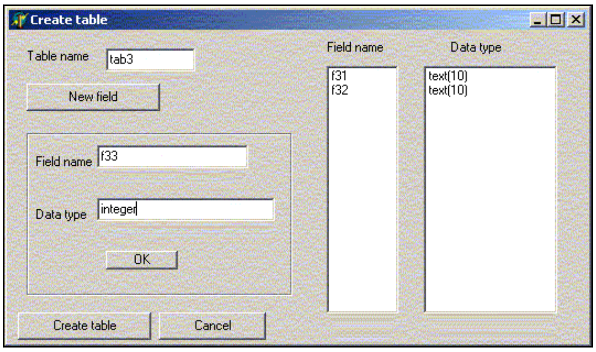
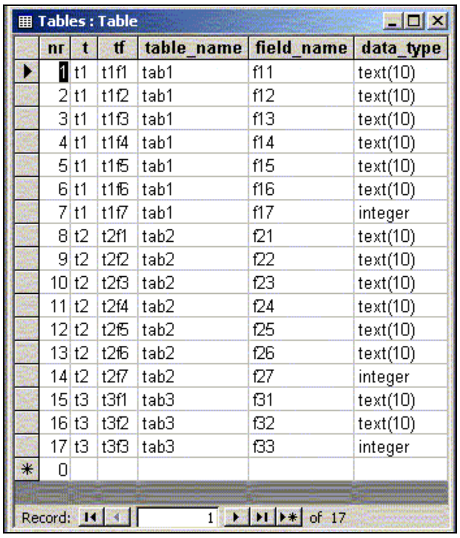


Figure 5. Create a table

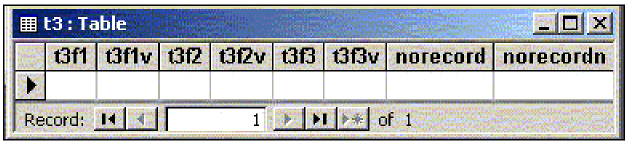
In this moment the corresponding table (in this case, *t3*) is also created, and it has as fields: the field code (given by *tf* from the *Figures 6*, for example *t3f1* in the *Figure 7*), the field from the initial table (in this case *t3f1v*, see *Figures 7, 8* and *9*) for each field from the initial table (in this case, *tab3*). In addition, we have a field *norecord* for the number of record in the initial table (in this case, *tab3*). The field *norecordn* refers to the same record number, but in text format. These two last fields exist also in the table *tab3*.


Tables : Table

nr	t	tf	table_name	field_name	data_type
1	t1	t1f1	tab1	f11	text(10)
2	t1	t1f2	tab1	f12	text(10)
3	t1	t1f3	tab1	f13	text(10)
4	t1	t1f4	tab1	f14	text(10)
5	t1	t1f5	tab1	f15	text(10)
6	t1	t1f6	tab1	f16	text(10)
7	t1	t1f7	tab1	f17	integer
8	t2	t2f1	tab2	f21	text(10)
9	t2	t2f2	tab2	f22	text(10)
10	t2	t2f3	tab2	f23	text(10)
11	t2	t2f4	tab2	f24	text(10)
12	t2	t2f5	tab2	f25	text(10)
13	t2	t2f6	tab2	f26	text(10)
14	t2	t2f7	tab2	f27	integer
15	t3	t3f1	tab3	f31	text(10)
16	t3	t3f2	tab3	f32	text(10)
17	t3	t3f3	tab3	f33	integer
0					

Record: 1 of 17

Figure 6. Data for a table


t3 : Table

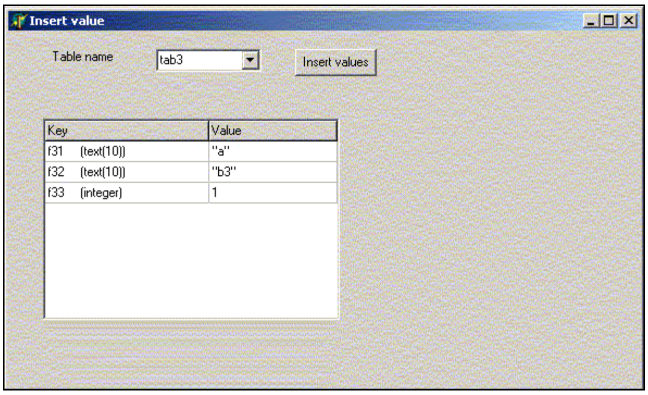
t3f1	t3f1v	t3f2	t3f2v	t3f3	t3f3v	norecord	norecordn

Record: 1 of 1

Figure 7. The corresponding additional table

3 Insert a new record

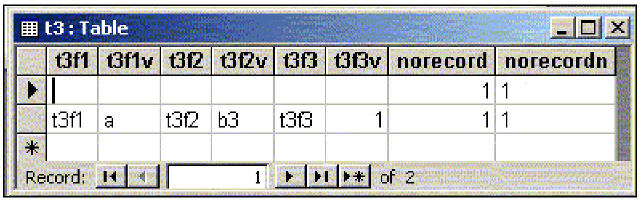
We insert new records in a table like in the *Figure 8*.



Key	Value
f31 (text(10))	"a"
f32 (text(10))	"b3"
f33 (integer)	1

Figure 8. Insert a new record

In this moment, in the corresponding table (in this case, *t3*), two new records corresponding to the new record (here, from *Figure 8*) are also inserted, like in the *Figure 9*.



t3 : Table

t3f1	t3f1v	t3f2	t3f2v	t3f3	t3f3v	norecord	norecordn
						1	1
t3f1	a	t3f2	b3	t3f3	1	1	1

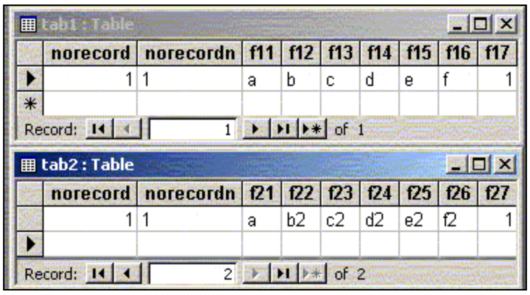
Record: 1 of 2

Figure 9. The corresponding records from the additional table

In *Figure 9*, the first record is ("", null, "", null, "", null, no_record, "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, no_record, "no_record"). And this happen for each record like the record from the *Figure 8*.

4 Algorithm presentation

We consider now the initial tables presented in *Figure 10* and the corresponding additional tables in *Figure 11*. We prefer to present the study for tables, which contain only one record, because in this way, we can easily present the result images.



tab1 : Table

norecord	norecordn	f11	f12	f13	f14	f15	f16	f17
1	1	a	b	c	d	e	f	1

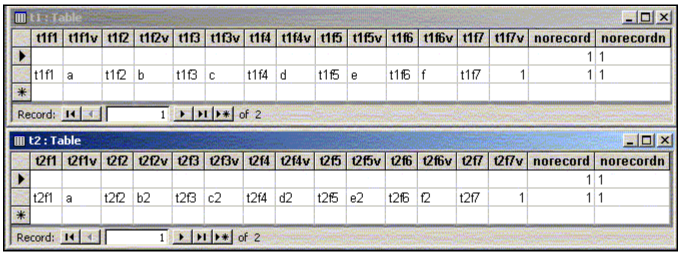
Record: 1 of 1

tab2 : Table

norecord	norecordn	f21	f22	f23	f24	f25	f26	f27
1	1	a	b2	c2	d2	e2	f2	1

Record: 2 of 2

Figure 10. Initial tables



t1 : Table

t1f1	t1f1v	t1f2	t1f2v	t1f3	t1f3v	t1f4	t1f4v	t1f5	t1f5v	t1f6	t1f6v	t1f7	t1f7v	norecord	norecordn
														1	1
t1f1	a	t1f2	b	t1f3	c	t1f4	d	t1f5	e	t1f6	f	t1f7	1	1	1

Record: 1 of 2

t2 : Table

t2f1	t2f1v	t2f2	t2f2v	t2f3	t2f3v	t2f4	t2f4v	t2f5	t2f5v	t2f6	t2f6v	t2f7	t2f7v	norecord	norecordn
														1	1
t2f1	a	t2f2	b2	t2f3	c2	t2f4	d2	t2f5	e2	t2f6	f2	t2f7	1	1	1

Record: 1 of 2

Figure 11. The corresponding tables for the tables presented in the *Figure 10*

Now, we present the algorithm used for aggregations. This is constructed in a number of steps. We want to present each step.

4.1 Tables, fields, relationships, aggregation functions

The user must specify the tables, fields, relationships, aggregation functions like in *Figure 12*. In *Figure 12* 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);

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 13*).

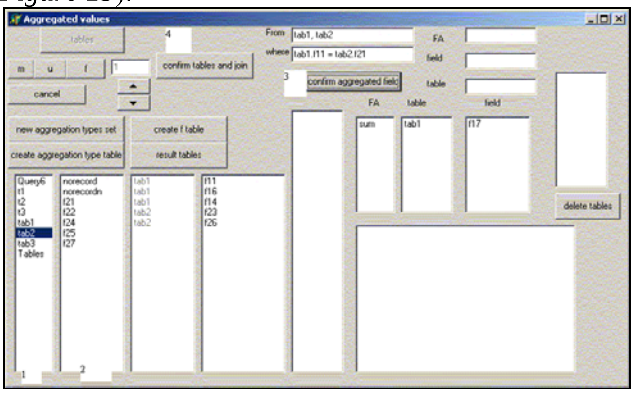
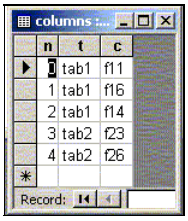


Figure 12. Tables, fields, relationships, aggregation functions



columns

n	t	c
0	tab1	f11
1	tab1	f16
2	tab1	f14
3	tab2	f23
4	tab2	f26

Figure 13. Indexes for the fields used for grouping

4.2 Specification of aggregation types

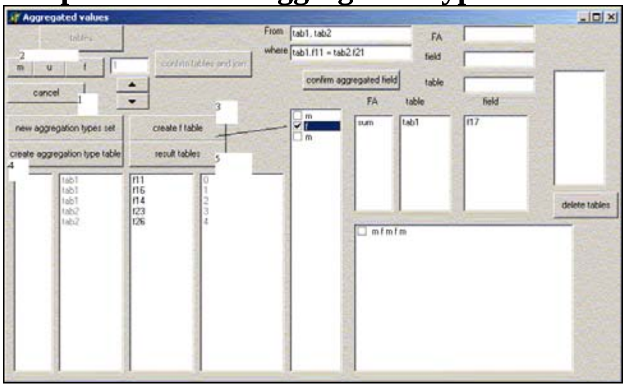


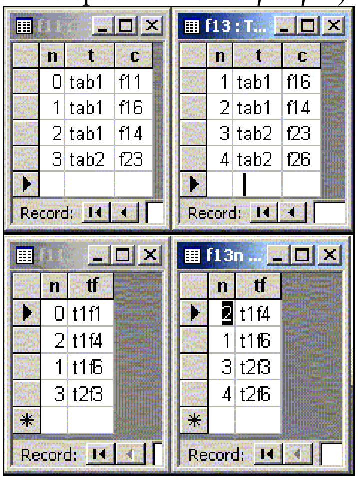
Figure 14. Specification of aggregation types

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, he can eliminate the field (all possible fields are displayed in a *ListBox*, like in *Figure 14*), 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*. This last table is transformed in a table, which for each record contains, the field index and the field code. We present such tables in *Figure15* (for the specification *m f m f m*).



n	t	c
0	tab1	f11
1	tab1	f16
2	tab1	f14
3	tab2	f23

n	t	c
1	tab1	f16
2	tab1	f14
3	tab2	f23
4	tab2	f26

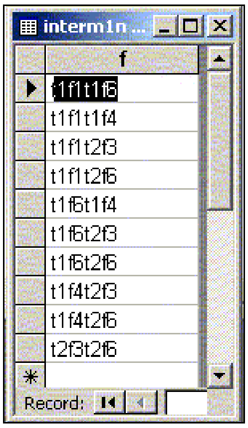
n	tf
0	t1f1
2	t1f4
1	t1f6
3	t2f3

n	tf
2	t1f4
1	t1f6
3	t2f3
4	t2f6

Figure 15. The corresponding tables for each *f* from the specification *m f m f m*

In the moment in which we have the table for each *f* from a specification (see *Figure 14*), 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 contains as records the aggregation types (see *Figure 16*). Here we have a cartesian product between the records from the tables corresponding to each *f* (presented in the *Figure 15)*. 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*).

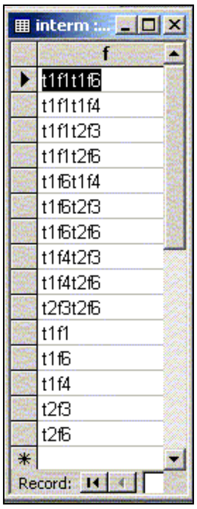


f
t1f1t1f6
t1f1t1f4
t1f1t2f3
t1f1t2f6
t1f6t1f4
t1f6t2f3
t1f6t2f6
t1f4t2f3
t1f4t2f6
t2f3t2f6

Figure 16. A table, which contains as records the aggregation types

Now, we repeat the step 1-4 until the moment when we will have specified all aggregation types. We will obtain the results in the following step (see *Figure 14*):

5 – with a click on the command button “result tables”, for the tables like the table presented in the *Figure 16* (these tables correspond at each specification of aggregation types), we will construct a unique table, which contains all the aggregation types, like in the *Figure 17*.



f
t1f1t1f6
t1f1t1f4
t1f1t2f3
t1f1t2f6
t1f6t1f4
t1f6t2f3
t1f6t2f6
t1f4t2f3
t1f4t2f6
t2f3t2f6
t1f1
t1f6
t1f4
t2f3
t2f6

Figure 17. The table, which contains as records all the aggregation types

In *Figure 17*, for example, *t1f1* means the code for the field *tab1.f11* and *t1f1t1f6* means the code for the field *tab1.f11* concatenated with the code for the field *tab1.f16.*

We will obtain the result table presented in *Figure 18*. In this case, using only one *SQL* statement we can obtain the result table.

tab1_f11	tab1_f16	tab1_f14	tab2_f23	tab2_f26	sum_tab1_f17
				f2	1
			c2		1
			c2	f2	1
		d			1
		d		f2	1
		d	c2		1
	f				1
	f			f2	1
	f		c2		1
	f	d			1
a					1
a				f2	1
a			c2		1
a		d			1
a	f				1

Figure 18. The result table of aggregations

The *SQL* statement is formulated (by application), for the presented case, in the following way:

the fields used for grouping (from the additional tables, here t1 *and* t2*)*
select tt1f1.t1f1v as tab1_f11, tt1f6.t1f6v as tab1_f16, tt1f4.t1f4v as tab1_f14,
tt2f3.t2f3v as tab2_f23, tt2f6.t2f6v as tab2_f26,

the aggregated data(from the additional table)
sum(tt1f7.t1f7v) as sum_tab1_f17

the result table
into r1

the additional tables are used for aggregations
from t1 tt1f1, t1 tt1f6, t1 tt1f4, t2 tt2f3, t2 tt2f6, t1 tt1f7

where

the aggregation types (the table interm *is presented in* Figure 17*)*
(tt1f1.t1f1+ tt1f6.t1f6+ tt1f4.t1f4+ tt2f3.t2f3+ tt2f6.t2f6+ tt1f7.t1f7 in (select f+"t1f7" from interm))

and (tt1f1.norecord=tt1f6.norecord) and
(tt1f1.norecord=tt1f4.norecord) and
(tt1f1.norecord=tt1f7.norecord) and
(tt2f3.norecord=tt2f6.norecord)

corresponding to the relationship from Figure 14, *the application creates a table norec (presented in the* Figure 19*) for the records (which respect these relationships) from the additional tables*
and("n"+tt1f1.norecordn+"n"+tt2f3.norecordn in (select nor from norec))

the fields used for grouping
group by tt1f1.t1f1v, tt1f1.t1f6v, tt1f1.t1f4v, tt2f3.t2f3v, tt2f6.t2f6v

nor
n1n1

Figure 19. The records which will be used for aggregations, from the additional tables

From the result table, presented in the *Figure 18*, eliminating fields or specifications for aggregation types (like in the *Figure 20*), we can obtain subsets for the maximal set specified.

5 Final tables

We confirm the desired specifications of aggregation types like in the *Figure 20*. With a click on the field name, we eliminate the field from the result table.

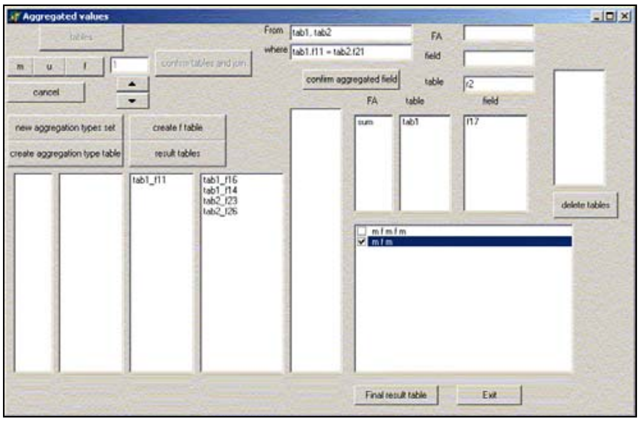


Figure 20. Final confirmations for result tables

For the case presented in the *Figure 20*, we will obtain the result table presented in the *Figure 21*.

tab1_f16	tab1_f14	tab2_f23	tab2_f26	sum_tab1_f17
			f2	1
		c2		1
	d			1
f				1

Figure 21. Final result table

6 Conclusions

The algorithm can be used for any type of databases. We have presented the implementation in *Delphi*, but the implementation can be made also in other programming environments.

For *n* fields (used for grouping) from a database, we can obtain 2^n aggregation types. With our algorithm, we can easily obtain any subsets of aggregation types and in a very short time.

References:

[1] Borland Delphi 6 for Windows, Developer'Guide, 2001.

[2] Chatziantoniou D., Ross K. – Querying Multiple Features in Relational Databases – VLDB 1996

[3] Chaudhuri S., Smith K. – Including Group By in Query Optimization – Proc. Of VLDB 1994.

[4] Gray J. et all – Data Cube: A relational aggregation operator generalizing Group-By, Cross-Tab, and Sub-Totals – Technical Report MSR-TR-95-22, Microsoft Corporation, October 1995.

[5] Murlaikrishna M., - Improved Unnesting Algorithms for Join Aggregate SQL Queries - Proc. VLDB Conf. 1994.

[6] http://www.olapcouncil.org.

[7] http://sgbd.developpez.com/ (general documentation on SQL statements).