

CSC 630: INDEPENDENT STUDY REPORT

Implementation details of an

ALGORITHM FOR
SELECTING MATERIALIZED VIEWS
FOR
DATA WAREHOUSING ENVIRONMENT

By

Gayathri TK
[gtambar@ncsu.edu]

under the guidance of

Dr. Rada Chirkova

Selecting Materialized Views

1. Introduction

An important issue in data warehouse development is the selection of a set of views to materialize in order to accelerate On-line analytical processing queries, given certain space and maintenance time constraints. [1] presents a framework for analyzing issues in selecting views to materialize so as to achieve the best combination of good query performance and low view maintenance.

Main goal of this project is to implement the algorithm for selecting materialized views as explained in [1]. The algorithm presented in the paper uses a two step approach for selecting views to be materialized:

1. Generate Multi-View Processing Plan (MVPP)
2. Selection of Views to be materialized from an MVPP

The paper provides two algorithms to generate MVPP(s): one can generate a feasible solution expeditiously, the other can provide an optimal solution by mapping the optimal MVPP generation problem as a 0-1 integer programming problem. Latter algorithm is implemented in this project.

2. Syntax

The algorithm is modeled as a dynamic shared library, which is stored under 'lib' folder in postgres installation directory. The library generated by this project is named 'matViews.so'. The algorithm can be invoked by creating a user defined function (UDF) as follows:

```
CREATE OR REPLACE FUNCTION
SelectMatViews (TEXT, TEXT) RETURNS INTEGER AS
'matViews.so', 'SelectMatViews' LANGUAGE 'C'
WITH (ISSTRICT);
```

Once the UDF is created, it can be called from the database client as follows:

```
select SelectMatViews ('\usr/local/queries.txt', '\usr/local/output.txt');
```

where

SelectMatViews – UDF
/usr/local/queries.txt – input file in specific format
/usr/local/output.txt – output file contains the views that are selected to be materialized

3. Input/Output

Input to the algorithm is a set of global queries and their access frequencies, and a set of base relations and their update frequencies. Output is a set of views to be materialized.

4. Algorithm

1. From the input file, read the set of base relations along with their update frequencies. Create a map table where each relation is identified by a unique integer.
2. For each Query, read its access frequency and the query statement.
3. Retrieve the set of join trees for this query from the planner. This is done by executing the query and saving the intermediate results generated by the planner.
4. In Join Tree, each relation is mapped to an integer value, which is the index of that relation in the relation list generated by the planner. This relation-id mapping is local to this query. We need to have a mapping that is consistent across all the queries. Hence the relation id in the join tree needs to be changed to be consistent with the map table created in step 1.
5. Once the relation-id mapping is updated in the join tree, it is saved locally for future processing.
6. For each join tree, generate a list of join patterns [all possible subtrees of the join tree] used by that tree. Also calculate the estimated cost of pattern as mentioned in section 4.3 of [1].
7. Create two matrices which are input to integer programmer solver as follows:

Let

l – no of join plan trees

k – no of global queries

m – no of join patterns

matrix A [k][l] :

$a[i][j] = 1$ if query q_i can be answered by join plan tree p_j

matrix B [m][l] :

$b[i][j] = 1$ if pattern s_i is contained in join plan tree p_j

8. Write the matrices to a file, in the format required by integer programmer solver.
9. Invoke the solver to get a set of join plan trees which form the optimal MVPP.
10. Once MVPP is generated, find a set of materialized views such that total cost of query processing and view maintenance is minimized. This is done as mentioned in section 4.1 of [1].

5. Notes

- $C_a^q(v)$: cost of access for query q using view v is randomly generated.
- $C_m^r(v)$: cost of maintenance of view v based on changes to base relation r is randomly generated.
- $Ecost(v)$: benefits of sharing a view among multiple views is randomly generated.
- Current implementation uses AMPL and CPLEX to solve Integer programming problem.

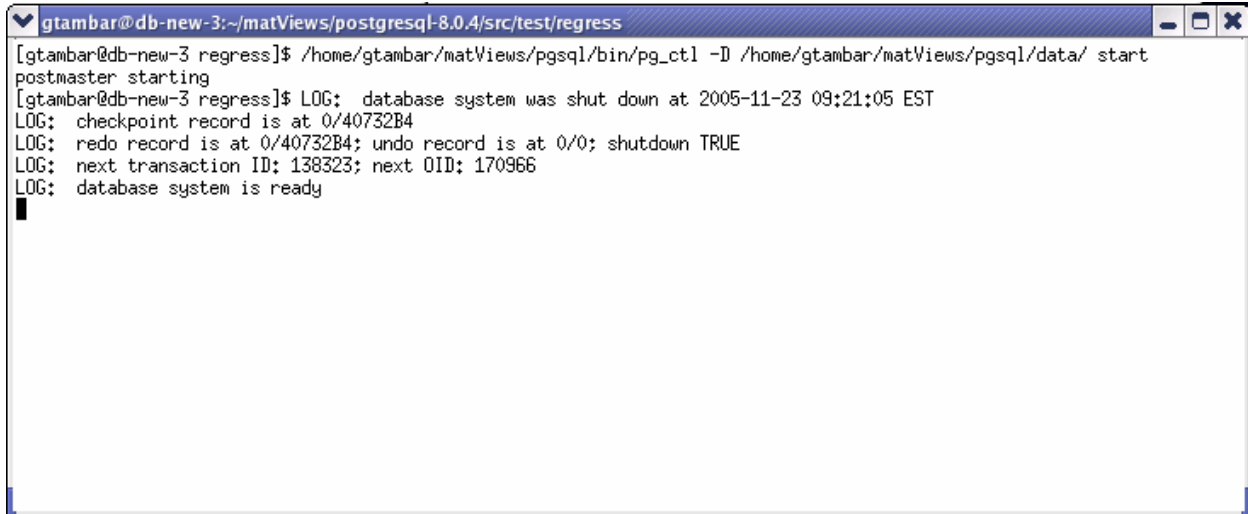
6. Files Created or Modified

- `/src/test/viewDesign/matViews.c` : This has the implementation of the algorithm
- `/src/include/commands/matViews.h` : This defines all the data structures used by the algorithm
- `/src/backend/optimizer/path/allpaths.c` : Modified to return the join plan trees of a query to the algorithm

Appendix:

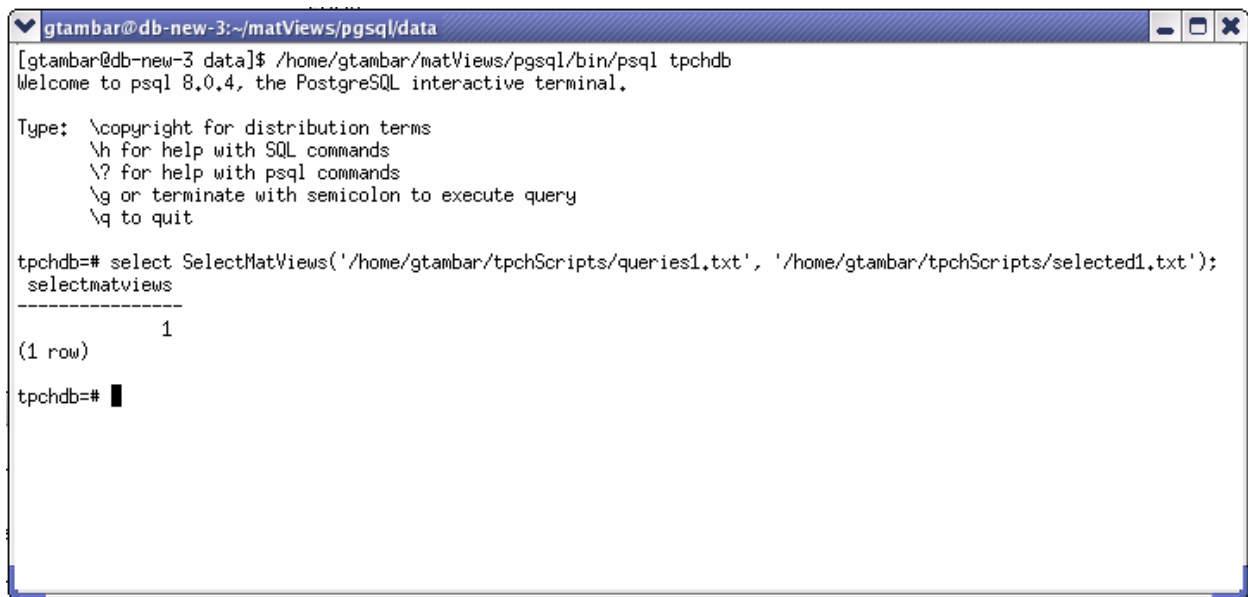
Sample program execution

1. Start the server



```
gtambar@db-new-3:~/matViews/postgresql-8.0.4/src/test/regress
[gtambar@db-new-3 regress]$ /home/gtambar/matViews/pgsql/bin/pg_ctl -D /home/gtambar/matViews/pgsql/data/ start
postmaster starting
[gtambar@db-new-3 regress]$ LOG:  database system was shut down at 2005-11-23 09:21:05 EST
LOG:  checkpoint record is at 0/40732B4
LOG:  redo record is at 0/40732B4; undo record is at 0/0; shutdown TRUE
LOG:  next transaction ID: 138323; next OID: 170966
LOG:  database system is ready
█
```

2. Start the database client and run the User Defined function.



```
gtambar@db-new-3:~/matViews/pgsql/data
[gtambar@db-new-3 data]$ /home/gtambar/matViews/pgsql/bin/psql tpchdb
Welcome to psql 8.0.4, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

tpchdb=# select SelectMatViews('/home/gtambar/tpchScripts/queries1.txt', '/home/gtambar/tpchScripts/selected1.txt');
selectmatviews
-----
1
(1 row)

tpchdb=# █
```

Input File Used:

```
BASE RELATIONS:
lineitem 50383021
orders 762909
customer 5442633
supplier 887262
part 842109
nation 4435623
partsupp 5426836
region 542
END BASE RELATIONS
```

```
QUERY:
54462573
SELECT
    l.returnflag,
    l.linestatus,
    SUM(l.extendedprice),
    COUNT(l.orderkey)
FROM
    lineitem l,
    orders o,
    customer c,
    supplier s,
    part p
WHERE
    l.shipdate >= '1992-03-15'
    AND l.shipdate <= '1998-12-01'
    AND l.orderkey = o.orderkey
    AND c.custkey = o.custkey
    AND l.suppkey = s.suppkey
    AND l.partkey = p.partkey
GROUP BY
    l.returnflag,
    l.linestatus;
```

```
QUERY:
65565468
SELECT
    o.orderkey,
    l.linestatus,
    SUM(l.extendedprice)
FROM
    lineitem l,
    orders o,
    customer c,
    supplier s,
    part p
WHERE
    l.shipdate < '1992-12-12'
    AND l.discount >= 0.01
    AND l.discount <= 0.05
```

```
AND l.returnflag = 'R'  
AND l.orderkey = o.orderkey  
AND c.custkey = o.custkey  
AND l.suppkey = s.suppkey  
AND l.partkey = p.partkey
```

```
GROUP BY  
  l.linestatus,  
  o.orderkey;
```

QUERY:

7669082

SELECT

```
  n1.nationkey AS supp_nation,  
  n2.nationkey AS cust_nation,  
  SUM(l.extendedprice) AS revenue,  
  SUM(o.totalprice)
```

FROM

```
  lineitem l,  
  orders o,  
  customer c,  
  supplier s,  
  part p,  
  nation n1,  
  nation n2
```

WHERE

```
  n1.name = 'INDIA'  
  AND n2.name = 'GERMANY'  
  AND l.shipdate >='1990-01-01'  
  AND l.shipdate <= '1999-12-12'  
  AND s.suppkey = l.suppkey  
  AND o.orderkey = l.orderkey  
  AND c.custkey = o.custkey  
  AND l.partkey = p.partkey  
  AND s.nationkey = n1.nationkey  
  AND c.nationkey = n2.nationkey
```

GROUP BY

```
  supp_nation,  
  cust_nation;
```

Output File Generated:

```
SELECT *
FROM lineitem l, orders o, customer c, supplier s, part p
WHERE l.orderkey = o.orderkey
AND c.custkey = o.custkey
AND l.suppkey = s.suppkey
AND l.partkey = p.partkey
```

README

All the files in the project are located in :
postgresql-8.0.4/src/test/viewDesign/

Input Files:

```
postgresql-8.0.4/src/test/viewDesign/queries.txt
postgresql-8.0.4/src/test/viewDesign/tpchScripts/queries1.txt
postgresql-8.0.4/src/test/viewDesign/tpchScripts/queries2.txt
postgresql-8.0.4/src/test/viewDesign/tpchScripts/queries3.txt
```

README for matViews.c

=====

1. InputFile Format:

=====

Input File should start with "BASE RELATIONS:" in a single line, followed by each of the base relations along with their update frequencies in a separate line. This section should end with "END BASE RELATIONS" string in a separate line.

Each Query starts with "QUERY:" string on a separate line, followed by access frequency of the query on the next line. SQL query is then entered after this.

This is how input file looks:

```
BASE RELATIONS:
table1 <update freq of table1>
table2 <update freq of table2>
table3 <update freq of table3>
END BASE RELATIONS
```

```
QUERY:
<access freq of query>
<query stmt...
....
...
...
...>
```

```
QUERY:
<access freq of query>
<query stmt..
....
....
```

....
....>

2. Compiling and running the code

=====

- the path AMPL executable and input files are defined as macros in matViews.so. Make sure that the path is correct.
- give *make* to compile the source files
- copy 'matViews.so' file to postgres lib directory with the following command: `cp matViews.so `pg_config --pkglibdir``

3. Calling SelectMatViews() User defined function

=====

- start DB server with the following command:
`<pgsql_dir>/bin/pg_ctl -D <pgsql_dir>/data start`
where `pgsql_DIR` is where postgres is installed, then
- start client by issuing the following command:
`<pgsql_dir>/bin/psql <database name>`
- create a user defined functions by executing the following command:

```
CREATE OR REPLACE FUNCTION
SelectMatViews(TEXT, TEXT) RETURNS INTEGER AS
'matViews.so', 'SelectMatViews' LANGUAGE 'C'
WITH (ISSTRICT);
```

- to select Materializ Views execute the following command:
`select SelectMatViews(<InputFilename>, <outputFileName>);`

4. Output File

=====

- the output file contains the selected views that need to be materialized.

AMPL/CPLEX: Integer Programmer Solver

Integer programmer solver is invoked from matViews.c file using UNIX `system()` call. The solver requires two input files:

mvpp.mod: (this file is not changed)

```
gtambar@db-new-3:~/ampl
set M;
set N;
set K;

param A{i in M, j in N};
param B{i in K, j in N};
param S{i in K};

var x{j in N} binary;

minimize cost: sum{i in K} S[i]*sum{j in N} B[i,j]*x[j];
subject to constraint1 {i in M}: sum{j in N} A[i,j]*x[j]=1;

~
```

mvpp.dat : generated by the algorithm

```
gtambar@db-new-3:~/ampl
set M := 1 2 3 ;

set N := 1 2 3 4 ;

set K := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ;

param A: 1 2 3 4 :=
1 1 0 0 0
2 0 1 0 0
3 0 0 0 0 ;

param B: 1 2 3 4 :=
1 1 0 0 0
2 0 1 0 0
3 0 1 0 0
4 0 1 0 0
5 0 1 0 0
6 0 0 1 0
7 0 0 1 0
8 0 0 1 0
9 0 0 1 0
10 0 0 1 0
11 0 0 1 0
12 0 0 0 1
13 0 0 0 1
14 0 0 0 1
15 0 0 0 1
16 0 0 0 1
17 0 0 0 1 ;

param S :=1 780 2 0 3 749 4 228 5 799 6 584 7 520 8 805 9 21 10 226 11 644 12 503 13 97 14 242 15 823 16 234 17 492 ;

~
```

AMPL script file

```
gtambar@db-new-3:~/ampl
reset; reset;

model "/home/gtambar/ampl/mvpp.mod";
data "/home/gtambar/ampl/mvpp.dat";

option solver "/home/gtambar/ampl/cplex";

solve;

display {j in N:x[j]>0} x[j];

quit;

~
```

References

- [1] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of the 23th VLDB Conference*, pages 136–145, 1997.