Providing Simple Materialized Views in PostGreSQL

CS601: Course Project

for

CS-631: Implementation techniques for RDBMS

by

Aishwarya Singh (163050060)
Ashish Mithole (163050010)
Awisha Makwana (163050079)
Khushboo Agarwal(163050066)

Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai 400 076

Contents

List of Figures

1 Aim

Providing Simple Materialized Views in PostGreSQL

1. Create view statement is extended as:
   create material view V as (one-table-query)
   The view is restricted to come from only one base table and may contain aggregation and grouping, but no having clause. Internally, postgreSQL creates a new table definition and initializes it by executing the query defining the view.

2. Update statements (insert/delete/update) are extended appropriately by associating triggers with them which update the materialized view.

2 Objective

Materialized view implemented in PostgreSQL use the rule system like static views do. The materialized view cannot subsequently be directly updated and that the query used to create the materialized view is stored in exactly the same way that a view's query is stored, so that fresh data can't be generated for the materialized view. Therefore, our objective is:

- Extending materialized view query so as to create a table in backend for every "create material view statement".
- Extending grammar to deny complex queries i.e. queries that include JOIN statements, HAVING clause and NESTED subqueries.
- Defining trigger functions for every materialized view so as to handle insert, delete or update events on the base table.
- Build and test the project for different insert/update/delete queries on base table.

3 Introduction

PostgreSQL is an open source, object-relational database (ORDBMS) written in C language. Five main components of PostgreSQL:

1. The parser - parse the query string
2. The rewriter - apply rewrite rules
3. The optimizer - determine an efficient query plan
4. The executor - execute a query plan
5. The utility processor - process DDL like CREATE TABLE

In Lex and parse the query string submitted by the user: parser/gram.y has the grammar and entry point is **parser/parser.c**
Produces a raw parsetree: a linked list of parse nodes.
Parse nodes are defined in **include/nodes/parsenodes.h**
There is usually a simple mapping between grammar productions and parse node structure.

Materialized view: It is a snapshot of a query saved into a table. It is very effective at speeding up queries, and is increasingly being supported by commercial databases and data warehouse systems. As a separate table is created for a materialized view, we can insert, update or delete tuples in a view present as a table by adding triggers, which can be further used to optimise the complex queries.

Triggers: A trigger is a set of actions that are run automatically when a specified change operation (SQL INSERT, UPDATE, DELETE or TRUNCATE statement) is performed on a specified table.

4 Approach

Changes in **gram.y**

- Extended the grammar for materialized view in file *gram.y* with a new name "MATERIAL VIEW".
- Changed the *Ctas– >relkind* from *OBJECT_MATVIEW* to *OBJECT_TABLE.*
- Defined MATERIAL as a new keyword in file *kwlist.h* and other related files.
- Added some new clauses and modified some existing ones to the grammar to deny complex queries.

Changes in **postgres.c**

- Examined each query to check whether its a **CREATE MATERIAL VIEW** query.
- Tokenized the query to extract the *Table name, View name and Select statement* used to construct the view.
- The trigger functions on the table using appropriate SQL statements are appended to the query.
- Executed those triggers by piggybacking them with original **CREATE MATERIAL VIEW** query.

5 Pseudo Code

The following grammar constructs are added to **'gram.y'** to extend the functionality of materialized view.
File name: gram.y
Line number: 3425-3472
Code snippet:

```
MatCreateMatViewStmt:
CREATE OptNoLog MATERIAL VIEW mat_create_mv_target AS MatSelectStmt
{
    CreateTableAsStmt *ctas = makeNode(CreateTableAsStmt);
ctas->query = $7;
ctas->into = $5;
ctas->relkind = OBJECT_TABLE;
ctas->is_select_into = false;
/* cram additional flags into the IntoClause */
$5->rel->relpersistence = $2;
$5->skipData = !($8);
$$ = (Node *) ctas;
};

MatSelectStmt: mat_simple_select    %prec UMINUS
            ;
mat_create_mv_target:
qualified_name
{
$$ = makeNode(IntoClause);
    $$->rel = $1;
$$->onCommit = ONCOMMIT_NOOP;
$$->viewQuery = NULL; /* filled at analysis time */
$$->skipData = false; /* might get changed later */
};
```

The following code is implemented in **'postgres.c'** to implement update/delete/insert triggers on materialized view.
File name: postgres.c
Line number: 4062-4144
Code snippet:

```
case 'Q':   /* simple query */
{
    const char *query_string;

    /* Set statement_timestamp() */
    SetCurrentStatementStartTimestamp();

    query_string = pq_getmsgstring(&input_message);
    pq_getmsgend(&input_message);
```

```
if (am_walsender)
    exec_replication_command(query_string);
else{
    exec_simple_query(query_string);
    //backup of original query to avoid unintentional
    //change of original query
    char MyQuery[1000];
    strcpy(MyQuery, query_string);
    int i;
    //conversion of query to lower case
    for(i = 0; MyQuery[i]; i++)
    {
        MyQuery[i] = tolower(MyQuery[i]);
    }
    //if query is a "create material view" statement
    if(strstr(MyQuery, "create material view")){
    //extract the select query of material view
    char select[100];
    strcpy(select, strstr(MyQuery, "select"));
    //extract the table name
    char fromTable[100];
    strcpy(fromTable, strstr(MyQuery, "from"));
    //extract the view name
    char view[100];
    strcpy(view, strstr(MyQuery, "view"));
    //tokenizing to get the view name
    char * MyView;
    MyView = strtok(view, " ");
    MyView = strtok(NULL, " ");
    //tokenizing to get the table name
    char * MyTable;
    MyTable = strtok(fromTable, " ");
    MyTable = strtok(NULL, " ");

    if(select[strlen(select)-3] ==  ';')
    {
```

```
    select[strlen(select)-3] = '\0';
}
else if(select[strlen(select)-2] ==  ';')
{
    select[strlen(select)-2] = '\0';
}
if(MyTable[strlen(MyTable)-2] ==  ';')
{
    MyTable[strlen(MyTable)-2] = '\0';
}
else if(MyTable[strlen(MyTable)-1] ==  ';')
{
    MyTable[strlen(MyTable)-1] = '\0';
}
//apply trigger
char trigger1[1000];
sprintf(trigger1, "CREATE OR REPLACE FUNCTION insert_%s()RETURNS
trigger AS $$ BEGIN EXECUTE format('delete from %s; INSERT INTO
%s(%s)') USING NEW; RETURN NULL; END; $$LANGUAGE 'plpgsql';",
MyView, MyView, MyView, select);

char trigger2[1000];
sprintf(trigger2, "CREATE OR REPLACE FUNCTION delete_%s()RETURNS
trigger AS $$ BEGIN EXECUTE format('delete from %s; INSERT INTO
%s(%s)'); RETURN NULL; END; $$LANGUAGE 'plpgsql';",MyView, MyView,
MyView, select);

//update insert and delete trigger
char ins_trig[1000];
sprintf(ins_trig, "DROP TRIGGER IF EXISTS insert_%s ON %s;
CREATE TRIGGER insert_%s AFTER INSERT ON %s FOR EACH ROW
EXECUTE PROCEDURE insert_%s();", MyView, MyTable, MyView);

char up_trig[1000];
sprintf(up_trig, "DROP TRIGGER IF EXISTS update_%s ON %s;
CREATE TRIGGER update_%s AFTER UPDATE ON %s FOR EACH ROW
```

```
        EXECUTE PROCEDURE insert_%s();", MyView, MyTable, MyView);

        char del_trig[1000];
        sprintf(del_trig, "DROP TRIGGER IF EXISTS delete_%s ON %s;
        CREATE TRIGGER delete_%s AFTER DELETE ON %s FOR EACH ROW
        EXECUTE PROCEDURE delete_%s();", MyView, MyTable, MyView);

        exec_simple_query(trigger1);
        exec_simple_query(trigger2);
        exec_simple_query(ins_trig);
        exec_simple_query(del_trig);
        exec_simple_query(up_trig);
    }
}
```

6 Test scenarios and Test queries

The following are the test queries and the action performed by creating a table called **mytable** and inserting few values in it.(Refer Fig:1) The materialized view to be tested will be based upon this table.

```
CREATE TABLE mytable (CourseID int, Name varchar(255), Marks int);

INSERT INTO mytable VALUES(631, 'Khushboo', 52);
INSERT INTO mytable VALUES(631, 'Aishwarya', 72);
INSERT INTO mytable VALUES(631, 'Awisha', 62);
INSERT INTO mytable VALUES(631, 'Ashish', 70);
INSERT INTO mytable VALUES(641, 'Khushboo', 58);
INSERT INTO mytable VALUES(641, 'Aishwarya', 79);
INSERT INTO mytable VALUES(641, 'Awisha', 61);
INSERT INTO mytable VALUES(641, 'Ashish', 76);
```

Now we create the extended materialized view here using **CREATE MATERIAL VIEW**. The query creates a table for the materialized view called **myview**. Now we can perform insert, delete and update operations on the base table, and the changes will get reflected in the view. (Refer Fig:3 and Fig:4)

```
CREATE MATERIAL VIEW myview AS SELECT courseid, AVG(Marks)
FROM mytable GROUP BY CourseId;
SELECT * FROM myview;

INSERT INTO mytable VALUES(601, 'Khushboo', 82);
SELECT * FROM myview;

DELETE FROM mytable where courseid=601;
SELECT * FROM myview;

UPDATE mytable set marks=marks+10 where courseid=631;
SELECT * FROM myview;
```

If a relation called "myview" already exists then the query below gives an ERROR saying relation "myview" already exists. To execute the same, we need to drop the existing table and again create the relation "myview".(Refer Fig:4)

```
CREATE MATERIAL VIEW myview AS SELECT  * FROM mytable;
DROP MATERIAL VIEW myview;
```

The new Material View should deny statements as per the modofied grammar(e.g. query involving multiple tables, having clause, aliases etc) in its select statement. The following test queries were used to check the same. (Refer Fig:5)

This query is used to test that table aliases are denied.

```
create material view matview as select b.courseid, b.name from mytable b;
```

This query is used to test that table columns cannot be renamed

```
create material view matview(a,b) as select courseid, name from mytable b;
```

This query is used to test that table aliases cannot be used.

```
create material view matview as select b.courseid, b.name from mytable;
```

This query is used to test that having clause is not permitted.

```
create material view matview as select sum(courseid) from mytable
where courseid < 4 having sum(courseid) > 10;
```

This query is used to test that aggregate queries with group by clause, but without having clause are executed successfully.

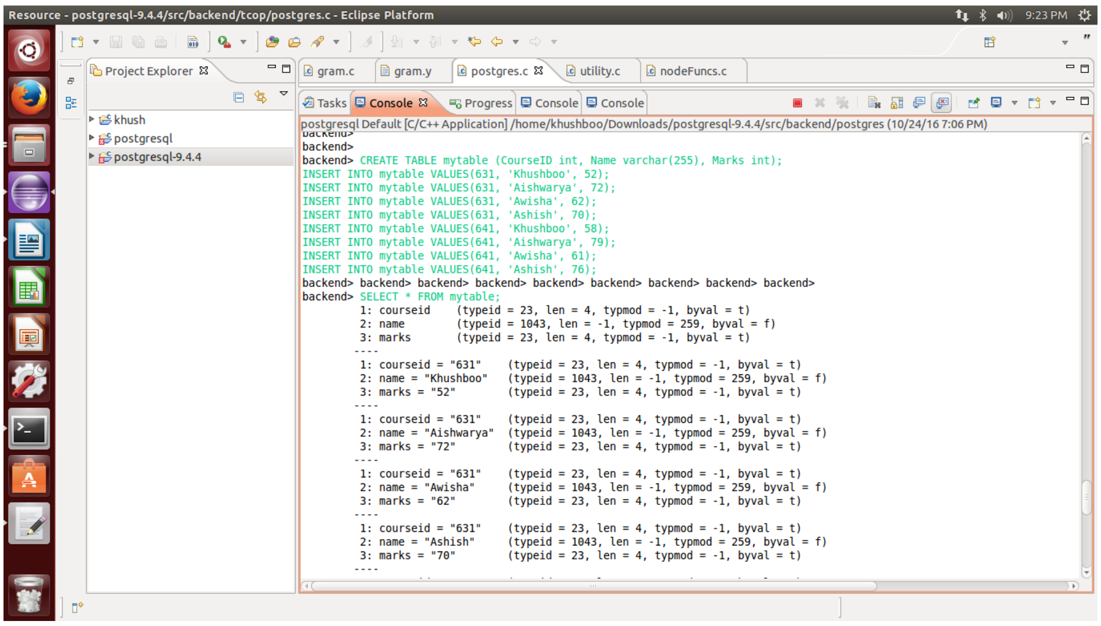
```
create material view matview as select sum(courseid) from mytable
where courseid < 4;
```

This query is used to test that multiple tables are not allowed in from clause.

```
create material view matview as select c.courseid, d.courseid
from mytable c, mytable d;
```

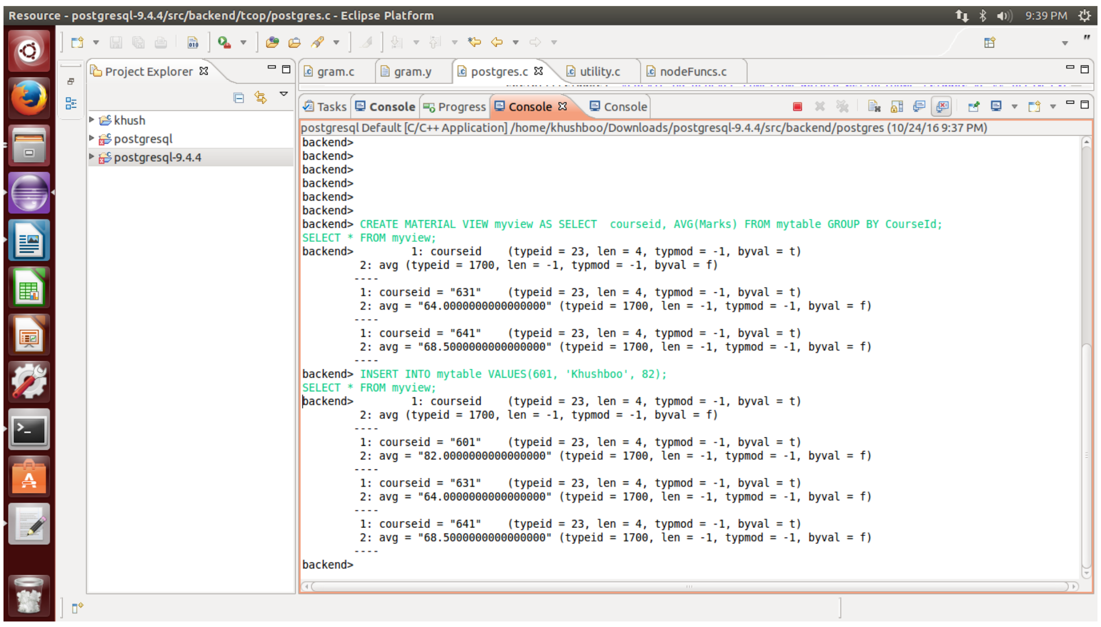
7 Results and Screenshots

Below are screen shots of our test data and the corresponding outputs:



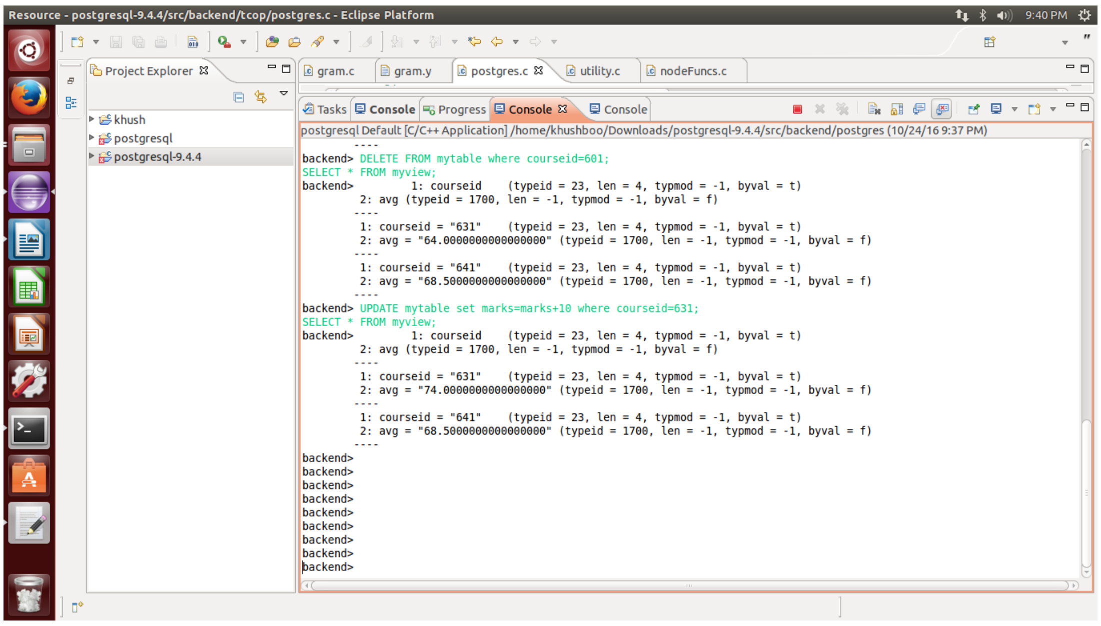
```
postgresql Default [C/C++ Application] /home/khushboo/Downloads/postgresql-9.4.4/src/backend/postgres (10/24/16 7:06 PM)
backend>
backend>
backend> CREATE TABLE mytable (CourseID int, Name varchar(255), Marks int);
INSERT INTO mytable VALUES(631, 'Khushboo', 52);
INSERT INTO mytable VALUES(631, 'Aishwarya', 72);
INSERT INTO mytable VALUES(631, 'Awisha', 62);
INSERT INTO mytable VALUES(631, 'Ashish', 70);
INSERT INTO mytable VALUES(641, 'Khushboo', 58);
INSERT INTO mytable VALUES(641, 'Aishwarya', 79);
INSERT INTO mytable VALUES(641, 'Awisha', 61);
INSERT INTO mytable VALUES(641, 'Ashish', 76);
backend> backend> backend> backend> backend> backend> backend> backend> backend>
backend> SELECT * FROM mytable;
         1: courseid    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: name        (typeid = 1043, len = -1, typmod = 259, byval = f)
         3: marks       (typeid = 23, len = 4, typmod = -1, byval = t)
        ----
         1: courseid = "631"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: name = "Khushboo"   (typeid = 1043, len = -1, typmod = 259, byval = f)
         3: marks = "52"        (typeid = 23, len = 4, typmod = -1, byval = t)
        ----
         1: courseid = "631"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: name = "Aishwarya"  (typeid = 1043, len = -1, typmod = 259, byval = f)
         3: marks = "72"        (typeid = 23, len = 4, typmod = -1, byval = t)
        ----
         1: courseid = "631"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: name = "Awisha"     (typeid = 1043, len = -1, typmod = 259, byval = f)
         3: marks = "62"        (typeid = 23, len = 4, typmod = -1, byval = t)
        ----
         1: courseid = "631"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: name = "Ashish"     (typeid = 1043, len = -1, typmod = 259, byval = f)
         3: marks = "70"        (typeid = 23, len = 4, typmod = -1, byval = t)
        ----
```

Figure 1: Creating the base table ”mytable” for the metarilized view



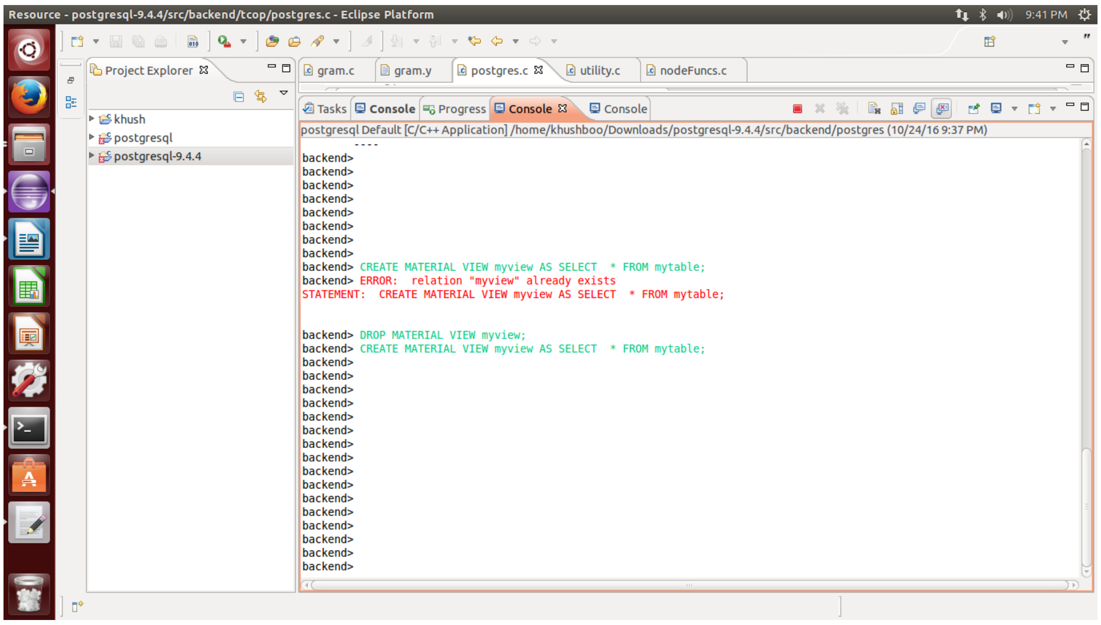
```
postgresql Default [C/C++ Application] /home/khushboo/Downloads/postgresql-9.4.4/src/backend/postgres (10/24/16 9:37 PM)
backend>
backend>
backend>
backend>
backend>
backend>
backend> CREATE MATERIAL VIEW myview AS SELECT  courseid, AVG(Marks) FROM mytable GROUP BY CourseId;
SELECT * FROM myview;
backend>          1: courseid    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
         1: courseid = "631"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg = "64.0000000000000000" (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
         1: courseid = "641"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg = "68.5000000000000000" (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
backend> INSERT INTO mytable VALUES(601, 'Khushboo', 82);
SELECT * FROM myview;
backend>          1: courseid    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
         1: courseid = "601"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg = "82.0000000000000000" (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
         1: courseid = "631"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg = "64.0000000000000000" (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
         1: courseid = "641"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg = "68.5000000000000000" (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
backend>
```

Figure 2: Creating the materialized view "myview" using **"create material view as"**



```
postgresql Default [C/C++ Application] /home/khushboo/Downloads/postgresql-9.4.4/src/backend/postgres (10/24/16 9:37 PM)
        ----
backend> DELETE FROM mytable where courseid=601;
SELECT * FROM myview;
backend>          1: courseid    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
         1: courseid = "631"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg = "64.0000000000000000" (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
         1: courseid = "641"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg = "68.5000000000000000" (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
backend> UPDATE mytable set marks=marks+10 where courseid=631;
SELECT * FROM myview;
backend>          1: courseid    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
         1: courseid = "631"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg = "74.0000000000000000" (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
         1: courseid = "641"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: avg = "68.5000000000000000" (typeid = 1700, len = -1, typmod = -1, byval = f)
        ----
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
```

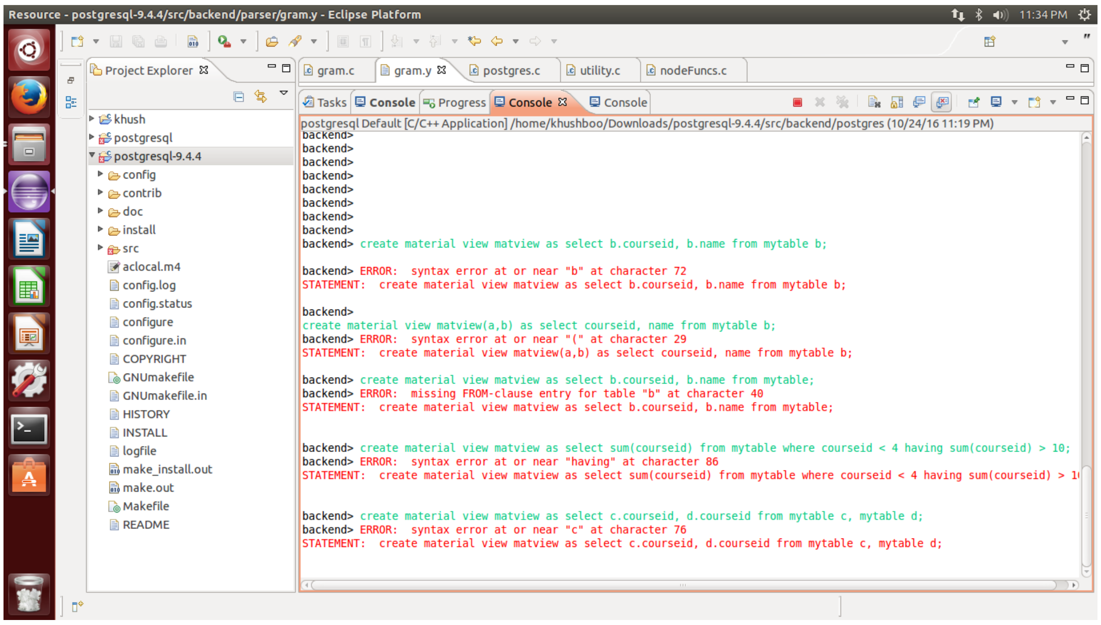
Figure 3: Testing materialized view for delete and update operations



```
postgresql Default [C/C++ Application] /home/khushboo/Downloads/postgresql-9.4.4/src/backend/postgres (10/24/16 9:37 PM)
        ----
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend> CREATE MATERIAL VIEW myview AS SELECT  * FROM mytable;
backend> ERROR:  relation "myview" already exists
STATEMENT:  CREATE MATERIAL VIEW myview AS SELECT  * FROM mytable;


backend> DROP MATERIAL VIEW myview;
backend> CREATE MATERIAL VIEW myview AS SELECT  * FROM mytable;
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
```

Figure 4: The query gives ERROR when materialized view with same name already exists



```
postgresql Default [C/C++ Application] /home/khushboo/Downloads/postgresql-9.4.4/src/backend/postgres (10/24/16 11:19 PM)
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend>
backend> create material view matview as select b.courseid, b.name from mytable b;

backend> ERROR:  syntax error at or near "b" at character 72
STATEMENT:  create material view matview as select b.courseid, b.name from mytable b;

backend>
create material view matview(a,b) as select courseid, name from mytable b;
backend> ERROR:  syntax error at or near "(" at character 29
STATEMENT:  create material view matview(a,b) as select courseid, name from mytable b;

backend> create material view matview as select b.courseid, b.name from mytable;
backend> ERROR:  missing FROM-clause entry for table "b" at character 40
STATEMENT:  create material view matview as select b.courseid, b.name from mytable;


backend> create material view matview as select sum(courseid) from mytable where courseid < 4 having sum(courseid) > 10;
backend> ERROR:  syntax error at or near "having" at character 86
STATEMENT:  create material view matview as select sum(courseid) from mytable where courseid < 4 having sum(courseid) > 1


backend> create material view matview as select c.courseid, d.courseid from mytable c, mytable d;
backend> ERROR:  syntax error at or near "c" at character 76
STATEMENT:  create material view matview as select c.courseid, d.courseid from mytable c, mytable d;
```

Figure 5: ERROR when the select clause inside materialized view invloves complex clauses

8 Summary

We extended the **Create materialized view statement** as **create material view V**. Internally, A new table definition is created at the backend for each **create material view V** query.

Internally, postgreSQL has created a new table definition and it has been initialized by executing the query defining the view.

Update statements (insert/delete/update) are extended appropriately by associating triggers with them which update the materialized view.

The new materialized view works fine with simple queries containing one base table. It does not supports complex queries (join, having clause, nested queries). The query may contain aggregation and grouping functions.