CS 631 Project
Functional Dependency Support in PostgreSQL

Hemanth Kumar Naradasu - 22M0777
Piyush Sawarkar - 22M0772
P S V N Bhavani Shankar - 22M0743

Problem Description:

Functional dependency is a relationship that exists between two set of attributes X and Y (X -> Y) where X is the determinant and Y is the dependent. Functional dependencies are not supported by pretty much any database today. Given a set of FDs on a specific relation of type X -> Y (where X and Y are single attributes), the FD Check is not being done by a majority of databases during insertion or updation.

Goal:

Our Goal is to check for any FD Violation during Insertion or Updation. The assumption is that both the determinant and dependant are both single attributes. For a given FD, say X -> Y; defined for a relation say R, we perform a check on the relation while INSERT whether there are 2 or more distinct values of Y for the same value of X. Similarly, for UPDATE, we check whether updating X given a condition on Y leads to violation of X -> Y.

Files Modified :

postgres.c , fd_violation.c (new file created for our code in src/backend/tcop), fd_violation.h (new file in src/include/tcop), makefile in src/backend/tcop.

Proposed Methodology:

First, we store the information about functional dependencies of all the relations in a tablke called **fd_table.** This table has three attributes, namely Relation, det_col (determinant of the FD), dep_col (dependant of the FD). Whenever a user creates a table, he/she inserts the FD information into this table. When FD Information is populated into the fd_table, an index is automatically created on the relation on the determinant attribute.

INSERT:

Assumption: The violation check is being done on FDs X->Y, where X and Y are single attributes.

We are handling two types of Insert Queries.

- INSERT INTO table(column1, column2, column3...) VALUES(value1, value2, value3...)
- INSERT INTO table VALUES(value1, value2, value3....)

First a check is made to determine if the command being executed is insert command. If the query is an insert query then a function call is made to **check_insert_fd_violation()** in the **fd_violation.c** file.

The query string is parsed to identify the type of insert query. If the insert query is of type INSERT INTO table(column1, column2, column3....) VALUES(value1, value2, value3....) then the query string is parsed to get the column names and column values.

If the insert query is of type INSERT INTO table VALUES(value1, value2, value3....) then the column values are obtained by parsing the query string. To get the column names execute a SQL query to fetch the column names from the information_schema and using **SPI** (server programming interface)

After getting the column names and column values, we then identify the table of which the insertion is being done. If the table into which insertion is being done is fd_table then a function call to **handle_fd_table_insertion()**. In this function we create an index on the relation on the determinant column with name of the index as index_relationname_determinantname. This is done by using SPI and executing a query for creation of index.

If the insertion is being done on table other than the fd_table then we do a function call in **handle_other_table_insertion().** The method for checking the fd violation is specified below. If there is any functional dependency violation then the insertion is aborted.

Algorithm 1: Insertion Violation Check Algorithm

Data: $n \geq 0$, Consistent Relation R, Set FD_Table F containing Functional Dependencies of the form $x->y$ denoted hence forth as fd and tuple t denoted by $< x, y >$ to insert in to R

Result: Valid Insertion of a tuple t based on existing Functional Dependencies in F over R

```
for each fd on R in F do
    xcol ←fd[x]
    ycol ←fd[y]
    y* ← get all t[ycol] values from R corresponding to
     t[xcol] ; /* Will be unique */
    if y* is NULL then
     | PERMIT
    else
        if y* exists and y*!=t[ycol] then
         | ABORT
        end
    end
    PERMIT
end
```

UPDATE:

Assumption: The violation check is being done on FD X -> Y, where X and Y are single attributes. And the value that is updated is X (determinant attribute) and the where condition is on Y (dependent column).

We are handling update query of type :

- UPDATE TABLE SET X=value WHERE Y=value

First a check is made to determine if the command being executed is update command. Then a function call is made to **check_update_fd_violation()** in fd_violation.c file. The work flow in the check_update_fd_violation() is as follows.

Update R
set X
where Y;
Assumption X, Y are single column.
Case 1: X does not belong to FD, Y does not belong to FD
PERMIT
Case 2: X belong to FD, Y does not belong to FD
a) X belong to Left[FD]
OR
b) X belong to Right[FD]
2 a)
FD's of the form:
X->alpha
X->beta
X->gamma
select alpha, beta, gamma where X=Xnew;
select alpha', beta', gamma' where X=Xold;
Either for X=Xnew, No tuple existed: PERMIT
OR
if existed then, check if for all cols mentioned above, there corresponding cols' values are same then: PERMIT
if not then : ABORT
2 b)
FD's of the form:
alpha->X
beta->X
gamma->X
To PERMIT, whatever tuples are satisfying for y, for every determiner fd check if all tuples for each value of left handside of fd, is getting changed and NOT any proper subset of it is getting changed.
Case 3: X does not belong to FD and Y belong to FD
PERMIT
Case 4: X belongs to FD and Y belongs to FD
So either X->Y OR Y->X FD exists.
Cases with Counter Examples.
4 a) Counter Cases:
(1,a), (1,a), (1,a), (2,a), (3,a)
if a change is made X is changed to something, which already existed in the table then thats a conflict.
NOT PERMIT
if X was new value then also PERMIT
if X is present and same then PERMIT
4 b)
(1,a), (1,a), (1,a) -> all subsets fetched and every y changed so wont conflict. PERMIT

Results :

The fd_table is shown below with attributes .

```
postgresql Default [C/C++ Application]  [pid: 17056]

PostgreSQL stand-alone backend 14.5
backend> select * from fd_table;
         1: relation_name       (typeid = 1043, len = -1, typmod = 104, byval = f)
         2: det_col     (typeid = 1043, len = -1, typmod = 104, byval = f)
         3: dep_col     (typeid = 1043, len = -1, typmod = 104, byval = f)
        ----
         1: relation_name = "student"   (typeid = 1043, len = -1, typmod = 104, byval = f)
         2: det_col = "id"      (typeid = 1043, len = -1, typmod = 104, byval = f)
         3: dep_col = "name"    (typeid = 1043, len = -1, typmod = 104, byval = f)
        ----
         1: relation_name = "student"   (typeid = 1043, len = -1, typmod = 104, byval = f)
         2: det_col = "name"    (typeid = 1043, len = -1, typmod = 104, byval = f)
         3: dep_col = "branch"  (typeid = 1043, len = -1, typmod = 104, byval = f)
        ----
```

Insert Violation Example:

```
backend> truncate student;
backend> insert into student values(1,'a','cse');
backend> insert into student values(2,'b','ece');
backend> insert into student values(1,'c','mech');
backend> 2022-11-28 16:40:49.331 IST [17056] ERROR:  Insertion Aborted, Functional Dependency Violation: this value '1' in 'id' has multiple values in 'name'
2022-11-28 16:40:49.331 IST [17056] STATEMENT:  insert into student values(1,'c','mech');


backend> insert into student values(3,'a','mech');
backend> 2022-11-28 16:41:10.328 IST [17056] ERROR:  Insertion Aborted, Functional Dependency Violation: this value ''a'' in 'name' has multiple values in 'branch'
2022-11-28 16:41:10.328 IST [17056] STATEMENT:  insert into student values(3,'a','mech');
```

Update Violation Example:

```
backend> select * from student;
         1: id  (typeid = 23, len = 4, typmod = -1, byval = t)
         2: name        (typeid = 1043, len = -1, typmod = 34, byval = f)
         3: branch      (typeid = 1043, len = -1, typmod = 34, byval = f)
        ----
         1: id = "1"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: name = "a"  (typeid = 1043, len = -1, typmod = 34, byval = f)
         3: branch = "cse"      (typeid = 1043, len = -1, typmod = 34, byval = f)
        ----
         1: id = "2"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: name = "b"  (typeid = 1043, len = -1, typmod = 34, byval = f)
         3: branch = "ece"      (typeid = 1043, len = -1, typmod = 34, byval = f)
        ----
         1: id = "3"    (typeid = 23, len = 4, typmod = -1, byval = t)
         2: name = "c"  (typeid = 1043, len = -1, typmod = 34, byval = f)
         3: branch = "mech"     (typeid = 1043, len = -1, typmod = 34, byval = f)
        ----
backend> update student set name='c' where branch='ece';
backend> 2022-11-28 16:57:13.916 IST [4523] ERROR:  Updation Aborted, Functional Dependency Violation: this value ''c'' in 'name' has multiple values in 'branch'
2022-11-28 16:57:13.916 IST [4523] STATEMENT:  update student set name='c' where branch='ece';


backend>
```

Challenges:

The most challenging part was to identify the files where the code needed to be modified. The other part which was challenging was to find the flow of code when an insertion or an update is being done. Also designing an algorithm to identify functional dependency violation on various test cases during insertion and updation was also challenging.

Future scope:

The current project was only done for handling functional dependencies which have only single attributes on determinant and dependent side. The project can be extended ti handle multiple attributes on determinant and dependent side as well. And also our project only handles a subset of insertion and updation cases so the project can be extended to check for functional dependency violation during all cases.

Conclusion:

We have implemented the check for functional dependency violation during insertion and updation. Though the functional dependency violation check we have coded checks for only few cases, the project does have more enhancements to be done to handle all cases.