

Applying Constraint Logic Programming to SQL Semantic Analysis

CSE-505 Project Report

Report By :- Siddhi Mundada & Annuj Jain

1 Abstract

Aiding programmers during compile time in execution of SQL queries helps improve productivity as well as save time. Consider a scenario wherein a person writes a query which is syntactically correct but inconsistent semantically. In this case, the Query processor will take the query as an input, do some processing and produce an output. Some of the steps in the processing phase include parsing the SQL query, checking for syntax, basic semantic checks and finally produce an optimized query that can be given to the execution engine to produce the output. In our case, since our query is syntactically correct, it'll pass through the syntax and semantic parsers and reach the execution engine to finally produce an unexpected output, null outputs, missing values, joins etc. Thus a presence of a more advanced semantic analyser will help improve productivity. The main motive of this project is to aid programmers with an advance semantic analysis during compile time. This paper aims to learn more about different types of SQL semantic errors, Constraint Logic Programming and it's use to find plausible faults in a given query.

2 Introduction

We start off by understanding some of the common terminologies that will be used over the course of this project. They are mentioned below.

SQL:

SQL stands for Structured Query Language. It is used to communicate with a database. SQL is a declarative language used to create, maintain and query relational databases. It can be used to store and retrieve large amounts of data from a database. It is particularly used to handle structured data. It is a standard language for relational database management systems .

There is so much data being produced every day that handling this data becomes one of the most significant challenge in today's era. SQL plays a very vital role in performing this task as it is particularly effective at data manipulation, and it is also very reliable. Also the data stored in a relational database can be easily queried, modified and manipulated with the help of simple SQL queries.

SQL queries can contain errors and these errors can be classified into two parts:

1. Run time errors:

These type of errors occur during the execution of a successfully compiled program. Some of the examples are division by zero, target object has not been properly allocated, collection is null, etc.

2. Compile time errors:

These type of errors occur due to violation of the rules of the syntax. These errors are supposed to be fixed before the compilation of the code.

a. Inconsistent conditions:

These messages report an inconsistency while attempting to execute a query. Some examples of inconsistent conditions are unavailable data, inconsistent page, internal inconsistency etc.

b. Uncorrelated relations in joins:

A query that does not contain references to objects in a parent statement is called an uncorrelated relation. Uncorrelated EXISTS-sub-queries are simply missing join conditions.

c. Unused tuple variables:

It is a simple syntax check to ensure that all tuple variables declared under FROM are actually accessed somewhere in the query.

d. Constant output columns:

An output column is unnecessary if it contains a single value that is constant and can be derived from the query without any knowledge about the database.

e. Duplicate output columns:

We should eliminate the duplicate columns to remove such errors.

Processing a given SQL query consists of phases like parsing/translating query to point out syntax, semantic errors and then using an optimizer, find out the most optimal representation and send it the execution engine. However during compile time, there are certain semantic errors, pointing to inconsistent queries, that are not caught and sent forward to execution.

Example of a typical semantic error:

```
SELECT * FROM employee WHERE dept = 'IT' AND dept = 'HR'
```

The query passes the syntax checking stage with ease. But we can see that the where clause is probably written in an incorrect manner, specifically the AND condition which should have been OR to return the expected results. This query is then sent for processing and gives incorrect results. Our purpose is to point out such possibly incorrect queries during compile time and thus save time.

The motivation behind our topic to aid programmers to improve the productivity of their task. A usual modern SQL system has a syntax checking stage which parses data to do syntax and semantic analysis but fails to include more advance techniques pointing possible incorrect statements.

Constraint Logic Programming:

In regular logic programming ,programs are queried to prove the goal but in constraint logic programming it contains constraints in addition to literals. Constraints are relations that we want to be satisfied.They are declarative i.e. they let the user express what is wanted rather than how to do it. Constraints can be embedded in a fundamental way in a programming language which is called constraint logic programming. Execution of CLP is performed by an interpreter which starts with the goal and recursively scan the clauses.Constraints encountered are placed in a set called constraint store. If this set is found to fail then it backtracks trying to prove the goal.

Datalog Program:

The Data Educational System converts the SQL query into an intermediate representation, i.e. the Datalog program. This program is then converted into a CLP program. The paper describes the grammar used for defining the datalog program and rules concerning its conversion. One of the key points in selecting and using a CLP language is to understand the different domain solvers constraint logic programming offers. Given a single query, two different domain solvers operate differently. This is because different solves prune the search tree in different manner under specific constraints.

3 Examples

Following are problem definitions of some example Queries that we will be using for demonstration purpose.

Example 1:

```
CREATE TABLE gas_products(name VARCHAR(20) PRIMARY KEY,
butane FLOAT CHECK butane BETWEEN 0.0 AND 100.0,
propane FLOAT CHECK propane BETWEEN 0.0 AND 100.0,
olefins FLOAT CHECK olefins BETWEEN 0.0 AND 100.0,
diolefins FLOAT CHECK diolefins BETWEEN 0.0 AND 100.0,
CHECK butane+propane+olefins+diolefins = 100.0);
```

Query: SELECT name FROM *gas_products* WHERE *butane* > 60.0 AND *propane* > 50.0

Example 2:

```
CREATE TABLE dept(id CHAR(10) PRIMARY KEY,  
name CHAR(20),  
location CHAR(20));  
  
CREATE TABLE emp(name CHAR(20) PRIMARY KEY, dept CHAR(10)  
REFERENCES dept(id), salary INT);
```

Query: SELECT emp.name, dept.name FROM emp,dept WHERE
emp.dept = dept.id

Example 3:

Query: SELECT * FROM emp WHERE salary < 2000 AND salary > 5000

4 Applying Constraint Logic Programming to SQL Semantic Analysis

The paper that we have selected uses Constraint Logic programming (CLP) to do SQL semantic analysis. The idea is to use CLP to model SQL queries in an abstract layer during compile time to signal possible semantic errors and point out the error before sending the query to the execution engine. In particular, the paper focuses on SQL semantic errors that can be caught independently of the database instance. After converting a condition occurring in the Datalog program into its equivalent CLP term, including the domain(χ), with which we send this query to a available domain solver. Thus, with the help the specific domain solvers, semantic errors are pointed out.

A flowchart briefly explaining the flow of operations is shown below:

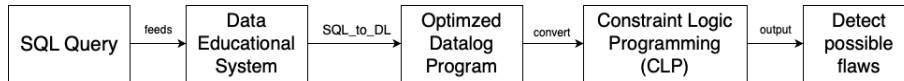


Figure 1: Flow of Operations

4.1 SQL to DataLog

The first step for the given approach is to convert the given SQL query into an equivalent DataLog program. A DataLog program is a Declarative Logic Programming language which consists of a set of DataLog rules. We use the Data Educational System (DES) to generate a datalog program. This software takes an SQL query and produces its equivalent datalog query. The equivalent datalog programs for our use cases are:

Example1:

```
INSERT into gas_products
VALUES('a',25.0,25.0,25.0,25.0);
```

```
DES> SELECT name FROM gas_products WHERE butane>60.0 AND propane>50.0;
Warning: [Sem] Inconsistent condition.
Info: SQL statement compiled to:

answer(A) :-
    gas_products(A,B,C,_D,_E),
    B>60.0,
    C>50.0.

answer(gas_products.name:varchar(20)) ->
{
}
Info: 0 tuples computed.
```

Figure 2: SQL to Datalog for example 1

Example2:

```
INSERT into emp VALUES('aron','1',100);
INSERT into dept VALUES('1','aron','newyork');
```

```
DES> SELECT emp.name, dept.name FROM emp, dept WHERE emp.dept=dept.id
;
Info: SQL statement compiled to:

answer(A,B) :-
    emp(A,C,_D),
    dept(C,B,_E).

answer(emp.name:char(20),dept.name:char(20)) ->
{
    answer(aron,sales)
}
Info: 1 tuple computed.
```

Figure 3: SQL to Datalog for example 2

Example3:

```
INSERT into emp VALUES('aron','1',100);
INSERT into dept VALUES('1','aron','newyork');
```

```

DES> select * from emp where salary<2000 and salary>5000
;
Warning: [Sem] Inconsistent condition.
Info: SQL statement compiled to:

answer(A,B,C) :-
    emp(A,B,C),
    C<2000,
    C>5000.

answer(emp.name:char(20),emp.dept:char(10),emp.salary:int) ->
{
}
Info: 0 tuples computed.

```

Figure 4: SQL to Datalog for example 3

4.2 DataLog to CLP

After finding a DataLog query equivalent of our SQL query, we try to optimize the program to get a refined datalog program. This is done with the help of predefined grammar rules, some pre-processing on SQL query etc. After optimizing, the next step is the translation of DL to CLP which is given as:

$DL.to.CLP((head_1 = goal_1, goal_2...goal_n)) = (head_1 = goal'_1, goal'_2...goal'_n)$,
where $goal'_1 = goal_1, goal'_2 = goal_2, ..., goal'_n = goal_n$

The following set of rules and conditions will help us convert the given datalog program into it's equivalent CLP program.

a. A given Datalog program Π_{DL} is said to be converted to Π_{CLP} if Π_{CLP} represents the actual meaning of Π_{DL} . Given that meaning of a Datalog program is $\lceil \Pi_{DL} \rceil$ and that of a CLP is $\lceil \Pi_{CLP} \rceil$, we say that Π_{CLP} represents Π_{DL} if $\lceil \Pi_{DL} \rceil \subseteq \lceil \Pi_{CLP} \rceil$. This will be more clear with the following example:

Say that we have $\Pi_{DL} = \{r(X) : -X = 1, X = 2\}$.

The meaning of this program is $\lceil \Pi_{DL} \rceil = \{r(1), r(2)\}$

Now, we can have

$\Pi_{CLP}^1 = \{r(X) : -X > 0, X < 3\}$

The meaning of this program will be $\{r(1), r(2)\}$

$\Pi_{CLP}^2 = \{r(X) : -X > 0\}$

The meaning of this program will be $\{r(1), r(2), r(3), r(4)...\}$

From the above examples of CLP programs we can see that the given datalog program can be represented by both the CLP programs, more accurately with Π_{CLP}^1 though. Thus, we first need to find the accurate representation of Π_{DL} .

b. A given base relation R can be converted to its equivalent clp program by simply doing a conjunction of the user defined constraints. For example, consider the base relation given in example 1 in the introduction section. The CLP program for this will be conjunction of all the user-defined constraints.

gas_products(A):-

ctr(butane >= 0, float), ctr(butane <= 100, float), ctr
(propane >= 0, float), ctr(propane = 100, float), ctr(olefins = 0, float),
ctr(olefins = 100, float), ctr(diolfins = 0, float), ctr(diolfins = 100, float),
ctr(butane + propane + olefins + diolfins = 100, float).

c. If there is a clause which contains of a relation call to some other clause, then we simply omit the function call and replace that with 'true' so that our program runs independently. After this, however, we include the required clauses in the relation that was mentioned in the function-call. Consider the above mentioned gas related clauses:

$\{r(X) : -t(X), X > 10\}$, where t is a call to a base relation t.

Let t be defined as,

CREATE TABLE t(x INT CHECK $x > 0$ AND $x < 1000$).

Thus our datalog query will be converted as follows:

$\{r(X) : -true, X > 10\}$, omit the function call by replacing it with 'true'
 $\{r(X) : -X > 0, X < 1000, X > 10\}$, including the constraints(if any) from the base relation t.

These are some of the definitions of the grammar concerning DL_{to}CLP. The equivalent CLP programs for the two examples taken are as follows:

Example 1: SELECT name FROM gas_products WHERE butane>60.0 AND propane>50.0

answer(B,P):-

ctr(B >= 0, float), ctr(B <= 100, float), ctr(P >= 0, float), ctr(P <= 100, float),
ctr(O >= 0, float), ctr(O <= 100, float), ctr(D >= 0, float), ctr(D <= 100, float),
ctr(B + P + O + D = 100, float), ctr(B > 60, float), ctr(P > 50, float).

Example 2: SELECT emp.name, dept.name FROM emp, dept WHERE emp.dept = dept.id

answer(A,B):-

ctr(A=B,CHAR(10)).

Example 3: SELECT * FROM emp WHERE salary< 2000 AND salary> 5000

answer(N,D,S):-

ctr(S< 2000,integer), ctr(S> 5000,integer).

4.3 Query Execution

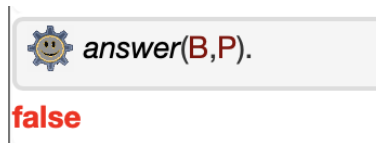
After getting the equivalent datalog program, we need to find out the domain of the given query, so that we can send that query to that particular query solver, CLP(X), where X is the domain of the given query. X can be FD(Finite domain of Integers), B(Boolean), Q(rational numbers) and R(numeric approximate). Note that we need to find out the specific domains of the query since each domain solver will follow different constraints and different pruning methodology to get different answers. In the given approach, every query is converted into a CLP program which includes the target domain type. This information can be used to send this particular query to the target domain solver. After finding the concerned domain solver, a given query is sent to the query solver during CLP evaluation time. The approach defined by the paper is that of a solve function $\text{solve}(\phi, \Pi_{CLP})$ where ϕ takes a goal that is to be solved in the context of Π_{CLP} . This will return a success substitution or a failure.

The output for the given queries is:

Example 1:

```
:-use_module(library(clpfd)).  
  
answer(B,P):-  
    B in 1..100,  
    P in 1..100,  
    O in 1..100,  
    D in 1..100,  
    B + P + O + D #= 100,  
    B#>60,  
    P#>50.
```

Figure 5: CLP-FD solver for example 1



```
answer(B,P).  
false
```

Figure 6: CLP-FD solver result for example 1

Example 2:

This code fragment runs correctly and gives the desired output i.e. the employees from all the departments.

Example 3:


```

:-use_module(library(clpfd)).

answer(N,D,S):-
    S#<2000,
    S#>5000.

```

Figure 7: CLP-FD solver for example 3

```

⚙️ answer(N,D,S).
🔗 Singleton variables: [N,D]
false

```

Figure 8: CLP-FD solver result for example 3

4.4 Conclusion

In this phase of the project, we have tried to gain a good understanding of the first two steps, namely SQL to DataLog and DataLog to CLP translations. We have tried to implement the approach mentioned in the paper for grammar conversion. From the given use cases we can see that the results match the desired outputs. These queries also take much lesser time as compared to other popular SQL query solvers. The DataLog to SQL was an important part of the project since it is not an automated step and requires grammar definitions to work on the translation. These grammar definitions will help find the efficiency of the system. We have worked on the approach used in the paper to reach the results. However, these findings can be extended to explore more in this field and come with more concrete solutions that can fully automated and thus be used at a large scale.

4.5 References

1. Fernando Saenz-Perez, 2019. Applying Constraint Logic Programming to SQL Semantic Analysis.
2. http://www3.cs.stonybrook.edu/~pfodor/courses/CSE505/L12_CLP.pdf
3. <https://courses.cs.washington.edu/courses/cse544/lectures/ppt>
4. <http://des.sourceforge.net>
5. Brass, C. and Goldberg, 2006. Semantic Errors in SQL: A Quite Complete List
6. Estevez-Martin, S., Fernandez, A. J., Sáenz-Pérez, F., Hortaz Gonzalez, T., Rodriguez-Artalejo, M., and del Vado Virseda, 2009. On the Cooperation of the Constraint Domains H, R and FD in CFLP.
7. https://en.wikipedia.org/wiki/Constraint_logic_programming