R-SQL: An SQL Database System with Extended Recursion

Gabriel Aranda-López, Susana Nieva, Fernando Saénz-Pérez
and Jaime Sánchez-Hernández
Universidad Complutense, Madrid

XVIII Jornadas de Programación y Lenguajes
Madrid, 20 de Septiembre, 2013
The Structured Query Language (SQL)

SQL is a programming language designed for managing data in relational database management systems (DBMS) and it became an ANSI standard in 1986, and an ISO standard in 1987.

It is widely considered a declarative language, but most of the DBMS include 4th generation languages.

Since then, the standard has been enhanced several times with added features.
Limited Expression of Recursion of SQL

- Recursion was included in SQL-99 standard, but:
  - Only linear recursion is allowed.
  - Mutual recursion is not supported.
  - Cycles in graphs make SQL systems go into an infinite loop.
- In this work, we present R-SQL system to solve these limitations.
Plan of the talk

- Introduction.
- System Overview.
- Database definition.
  - Syntax.
  - Some examples.
- Fixpoint computation.
  - Dependency Graph and Stratification.
  - Python Code Generation
  - An algorithm to compute the fixpoint.
- Integration into a DBMS.
- System Demostration
System Structure

- R-SQL
  - Database
  - Definition: `definition.rsq1`
  - PostgreSQL
- SWI Prolog
  - Parser
  - Dependency Graph
  - Stratification
  - Python Code Generation
- Python
  - Script: `script.py`
Database Definition

R-SQL

postgres SQL

Python

Database definition.rsq1

Database

Parser

Dependency Graph

Stratification

Python Code Generation

script.py
Main Idea

To define recursive relations as:

\[ R \text{ Sch} := \text{SELECT} \ldots \text{FROM} \ R_1 \ldots R \ldots \ R_n \ldots \]
R-SQL Syntax (II)

\[
\text{sql_db} ::= \text{R sch} := \text{sel_stm}; \\
\text{...} \\
\text{R sch} := \text{sel_stm}; \\
\text{sel_stm} ::= \text{SELECT exp,..,exp [FROM R,..,R [WHERE c]]} \\
\text{ | sel_stm UNION sel_stm} \\
\text{ | sel_stm EXCEPT R} \\
\text{sch} ::= (A T,\ldots,A T) \\
\text{exp} ::= C \mid R.A \mid \text{exp opm exp} \mid - \text{exp} \\
\text{c} ::= \text{TRUE} \mid \text{FALSE} \mid \text{exp opc exp} \mid \text{NOT (c)} \\
\text{ | c [ AND | OR ] c}
\]
Example 1: Mutual Recursion

Even/Odd Numbers in 0...100

even(x float) :=
    SELECT 0
    UNION
    SELECT odd.x+1 FROM odd WHERE odd.x<100;

odd(x float) :=
    SELECT even.x+1 FROM even WHERE even.x<100;
Example 2: Non-Linear Recursion

Fibonacci Numbers

\[
\begin{align*}
\text{fib1}(n \text{ float, } f \text{ float}) & := \\
& \quad \text{SELECT fib.n, fib.f FROM fib;}
\end{align*}
\]

\[
\begin{align*}
\text{fib2}(n \text{ float, } f \text{ float}) & := \\
& \quad \text{SELECT fib.n, fib.f FROM fib;}
\end{align*}
\]

\[
\begin{align*}
\text{fib}(n \text{ float, } f \text{ float}) & := \\
& \quad \text{SELECT 0,1 UNION} \\
& \quad \text{SELECT 1,1 UNION} \\
& \quad \text{SELECT fib1.n+1,fib1.f+fib2.f FROM fib1,fib2} \\
& \quad \quad \text{WHERE fib1.n=fib2.n+1 AND fib1.n<10;}
\end{align*}
\]
Example 3: Cycle in Graph

arc(ori varchar(1), des varchar(1)) :=
    SELECT 'a','b' UNION
    SELECT 'b','c' UNION
    SELECT 'c','a';

path(ori varchar(1), des varchar(1)) :=
    SELECT arc.ori, arc.des FROM arc UNION
    SELECT arc.ori, path.des FROM arc,path
    WHERE arc.des=path.ori;
Fixpoint Computation
Analogously to Datalog:

- A dependency graph is defined using relation definitions. An `EXCEPT` clause plays the role of negation.
- A stratification of the database relations is obtained from the dependency graph.
- The stratification is used to compute the least fixpoint interpretation, stratum by stratum.
Example 3: Extended Graph

arc(ori varchar(1), des varchar(1)) :=
   SELECT 'a','b' UNION
   SELECT 'b','c' UNION
   SELECT 'c','a';

path(ori varchar(1), des varchar(1)) :=
   SELECT arc.ori, arc.des FROM arc UNION
   SELECT arc.ori, path.des FROM arc,path
   WHERE arc.des=path.ori;

bPath(ori varchar(10), des varchar(10)) :=
   SELECT path.ori, path.des FROM path
   WHERE (path.ori = 'b' OR path.des = 'b');

avoid_b(ori varchar(10), des varchar(10)) :=
   SELECT path.ori, path.des FROM path
   EXCEPT bPath;
Dependency Graph and Stratification

arc := SELECT 'a', 'b' UNION SELECT 'b', 'c' UNION SELECT 'c', 'a';
path := SELECT * from arc UNION SELECT ... FROM arc, path ...;
bPath := SELECT path.ori, path.des FROM path WHERE ...;
avoid_b := SELECT path.ori, path.des FROM path EXCEPT bPath;

Stratification

[(arc, 1), (path, 1), (bPath, 1), (avoid_b, 2)]
System Structure

R-SQL

- database
- Postgres SQL

- Database definition.rsqu
- SWI Prolog
  - Parser
  - Dependency Graph
  - Stratification
  - Python Code Generation

- Python
  - script.py
Interpretation of Select Statements

Given an interpretation $I: \{R_1, \ldots, R_n\} \rightarrow \mathcal{P}(\mathcal{T})$, we define the semantics of a `sel_stm` in the context of $I$ as:

- $[\text{SELECT } \text{exp}_1, \ldots, \text{exp}_k]^I = \{(\text{exp}_1, \ldots, \text{exp}_k)\}$.
- $[\text{SELECT } \text{exp}_1, \ldots, \text{exp}_k \text{ FROM } R_1, \ldots, R_m \text{ WHERE } c]^I = \{(\text{exp}_1[\bar{a}/A], \ldots, \text{exp}_k[\bar{a}/A]) | \bar{a} \in I(R_1) \times \ldots \times I(R_m) \text{ and } c[\bar{a}/A] \text{ is satisfied}\}$.
- $[\text{sel_stm}_1 \text{ UNION } \text{sel_stm}_2]^I = [\text{sel_stm}_1]^I \cup [\text{sel_stm}_2]^I$.
- $[\text{sel_stm \ EXCEPT } R]^I = [\text{sel_stm}]^I \setminus I(R)$.
Example 1: Even/Odd Numbers

Given $I(\text{even}) = \{(0), (2)\}$ and $I(\text{odd}) = \emptyset$.

Semantics of odd in the context of $I$

$\left[ \text{SELECT even.x+1 FROM even WHERE even.x<100} \right]^I =$

$\{(\text{even.x+1})[a/\text{even.x}] \mid (a) \in I(\text{even}) \text{ and } (\text{even.x}<100) \text{ [a/\text{even.x}] is satisfied} \} =$

$\{(1), (3)\}$. 
Example 1: Even/Odd Numbers, Intended Semantics

$I(\text{even}) = \{(0), (2), \ldots, (100)\} = \\
[\text{SELECT 0 UNION SELECT odd.x+1 FROM odd WHERE x<100}]^I$

$I(\text{odd}) = \{(1), (3), \ldots, (99)\} = \\
[\text{SELECT even.x+1 FROM even WHERE even.x<100}]^I$
We define an operator over the set of interpretations of stratum $i$. $T_i$ is successively applied to minimal interpretations to obtain the interpretation fixpoint.

For a definition of the form $R_{sch} := \text{sel}_{stm}$, in stratum $i$:

$$T_i(I)(R) = [\text{sel}_{stm}]^I$$

The database semantics is a fixpoint

$$fix_i = \bigcup_{n \geq 0} T_i^n (fix_{i-1})$$

where $i$ is the number of strata.
Example of the Computation of the Fixpoint

\begin{center}
\begin{tabular}{|c|c|}
\hline
$T_1^n(\emptyset)(\text{path})$ & Added tuples at each iteration \\
\hline
$T_1^1(\emptyset)(\text{path})$ & \{('a', 'b'), ('b', 'c'), ('c', 'a')\}, \\
$T_1^2(\emptyset)(\text{path})$ & \{('a', 'c'), ('b', 'a'), ('c', 'b')\}, \\
$T_1^3(\emptyset)(\text{path})$ & \{('a', 'a'), ('b', 'b'), ('c', 'c')\}, \\
\hline
\end{tabular}
\end{center}
Example of the Computation of the Fixpoint

\[
\begin{array}{|c|c|}
\hline
T_1^n(\emptyset)(\text{path}) & \text{Added tuples at each iteration} \\
\hline
T_1^1(\emptyset)(\text{path}) & \{(\text{'a'}, \text{'b'}), (\text{'b'}, \text{'c'}), (\text{'c'}, \text{'a'})\}, \\
T_1^2(\emptyset)(\text{path}) & \{(\text{'a'}, \text{'c'}), (\text{'b'}, \text{'a'}), (\text{'c'}, \text{'b'})\}, \\
T_1^3(\emptyset)(\text{path}) & \{(\text{'a'}, \text{'a'}), (\text{'b'}, \text{'b'}), (\text{'c'}, \text{'c'})\}, \\
\hline
\end{array}
\]

All these tuples are \( \text{fix}_1(\emptyset)(\text{path}) \).
Example of the Computation of the Fixpoint

\[
\begin{array}{|c|c|}
\hline
 T_1^n(\emptyset)(\text{path}) & \text{Added tuples at each iteration} \\
\hline
 T_1^1(\emptyset)(\text{path}) & \{ ('a','b'), ('b','c'), ('c','a') \}, \\
 T_1^2(\emptyset)(\text{path}) & \{ ('a','c'), ('b','a'), ('c','b') \}, \\
 T_1^3(\emptyset)(\text{path}) & \{ ('a','a'), ('b','b'), ('c','c') \}, \\
\hline
\end{array}
\]

All these tuples are \( \text{fix}_1(\emptyset)(\text{path}) \).

Similarly, we obtain tuples for \( \text{arc} \) y \( \text{bPath} \) to complete \( \text{fix}_1 \).
Example of the Computation of the Fixpoint

<table>
<thead>
<tr>
<th>$T^n_1(\emptyset)(\text{path})$</th>
<th>Added tuples at each iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T^1_1(\emptyset)(\text{path})$</td>
<td>${(\text{'a'}, \text{'b'}), (\text{'b'}, \text{'c'}), (\text{'c'}, \text{'a'})}$,</td>
</tr>
<tr>
<td>$T^2_1(\emptyset)(\text{path})$</td>
<td>${(\text{'a'}, \text{'c'}), (\text{'b'}, \text{'a'}), (\text{'c'}, \text{'b'})}$,</td>
</tr>
<tr>
<td>$T^3_1(\emptyset)(\text{path})$</td>
<td>${(\text{'a'}, \text{'a'}), (\text{'b'}, \text{'b'}), (\text{'c'}, \text{'c'})}$,</td>
</tr>
</tbody>
</table>

All these tuples are $\text{fix}_1(\emptyset)(\text{path})$.
Similarly, we obtain tuples for $\text{arc y bPath}$ to complete $\text{fix}_1$. Then, $\text{fix}_2(\text{fix}_1)(\text{avoid}_b)$ is computed to obtain $\text{fix}_2$, which is the database fixpoint.
Algorithm for Computing the Fixpoint

\[ \text{str} := 1 \]

while \( \text{str} \leq \text{numstr} \) do

    for each \( R_i \in R_{\text{str}} \) do
        CREATE TABLE \( R_i \) sch\( i \)
    end for

    change := true

    while change do

        size := rel_size_sum(\( R_{\text{str}} \))

        for each \( R_i \in R_{\text{str}} \) do
            INSERT INTO \( R_i \) SELECT * FROM sel_stm\( i \);
            EXCEPT SELECT * FROM \( R_i \);
        end for

        change := (size \neq rel_size_sum(\( R_{\text{str}} \)))
    end while

end while

str := str + 1
Example of Code Generation (I)

1\textsuperscript{st} stratum

\texttt{arc(ori varchar(1), des varchar(1)) :=}
\begin{verbatim}
    SELECT 'a','b' UNION SELECT 'b','c' UNION SELECT 'c','a';
\end{verbatim}

\texttt{path(ori varchar(1), des varchar(1)) :=}
\begin{verbatim}
    SELECT arc.ori, arc.des FROM arc UNION
    SELECT arc.ori, path.des FROM arc,path
    WHERE arc.des=path.ori;
\end{verbatim}

\texttt{bPath(ori varchar(10), des varchar(10)) :=}
\begin{verbatim}
    SELECT path.ori, path.des FROM path
    WHERE (path.ori = 'b' OR path.des = 'b');
\end{verbatim}

2\textsuperscript{nd} stratum

\texttt{avoid_b(ori varchar(1), des varchar(1)) :=}
\begin{verbatim}
    SELECT path.ori, path.des FROM path
    EXCEPT bPath;
\end{verbatim}
Example of Code Generation (I)

Python code produced by R-SQL for Stratum 2

cursor.execute("create table avoid_b (...);")
ch = True
while ch:
    size1=0 size2=0 ch=False
cursor.execute("select * from avoid_b;")
res=cursor.fetchall()
size1= size1 + len(res)

cursor.execute("insert into avoid_b
   (((select * from path) except (select * from bPath))
    except (select * from avoid_b));")

cursor.execute("select * from avoid_b;")
res=cursor.fetchall()
size2= size2 + len(res)
ch = (size1 != size2)
System Structure

R-SQL

Database

PostgreSQL SQL

Python

script.py

SWI Prolog

Parser

Dependency Graph

Stratification

Python Code Generation

database definition.rsql
Integrating R-SQL into a DBMS

Once a R-SQL database definition has been processed, the tables obtained are stored as a database instance of PostgreSQL and the user can:

- Formulate queries that will be solved using those tables.
- Define views that can be used in conjunction with other regular views that can either computed on demand or can be materialized.
Integrating R-SQL into a DBMS

Once a R-SQL database definition has been processed, the tables obtained are stored as a database instance of PostgreSQL and the user can:

- Formulate queries that will be solved using those tables.
- Define views that can be used in conjunction with other regular views that can either computed on demand or can be materialized.

But, the main goal of the proposal is to allow less-restricted recursive relation definitions in existing SQL engines.
System Demostration

The implementation is available at:
https://gpd.sip.ucm.es/trac/gpd/wiki/GpdSystems/RSQL.
Thanks!