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.

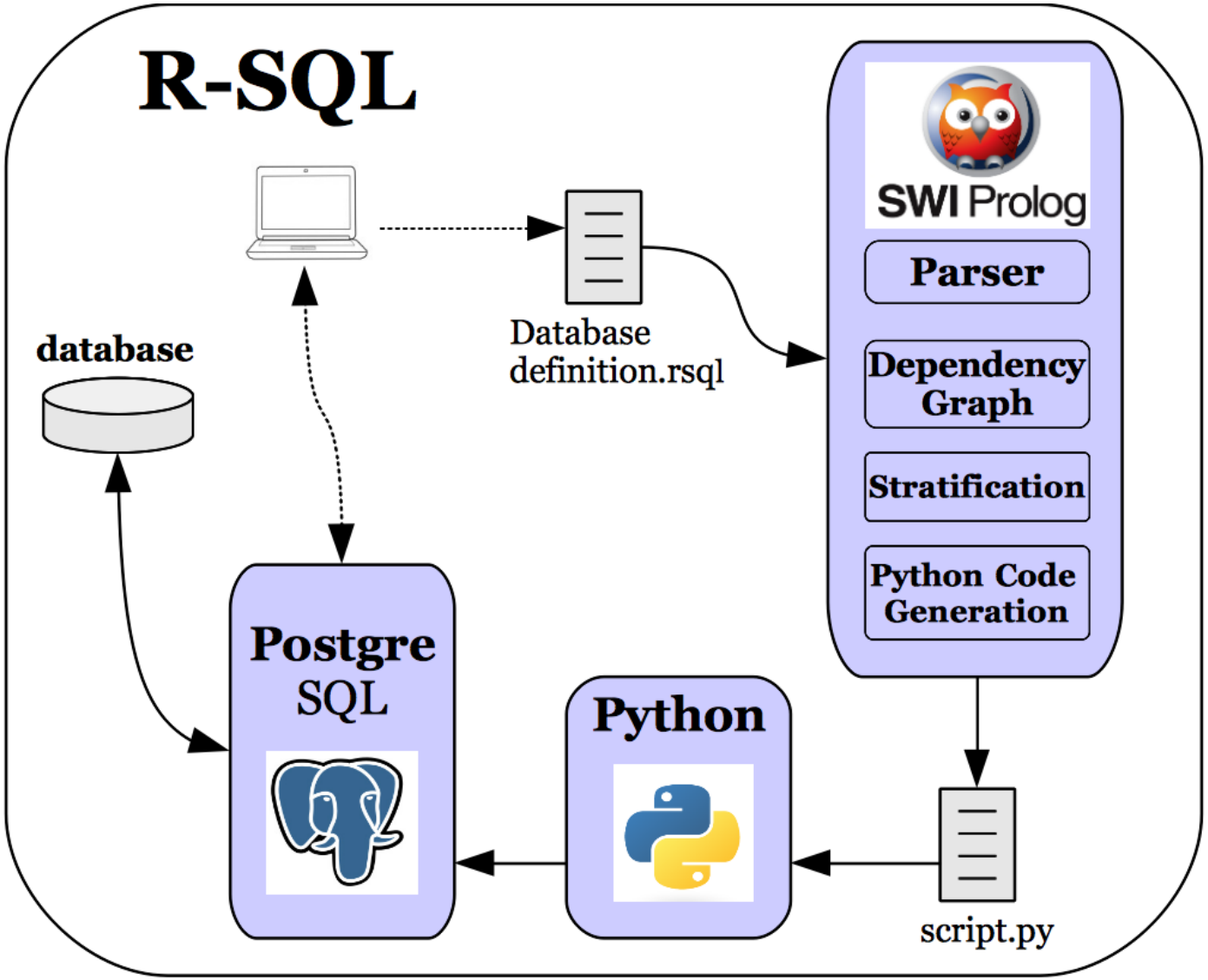
Limited Expression of Recursion of SQL

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

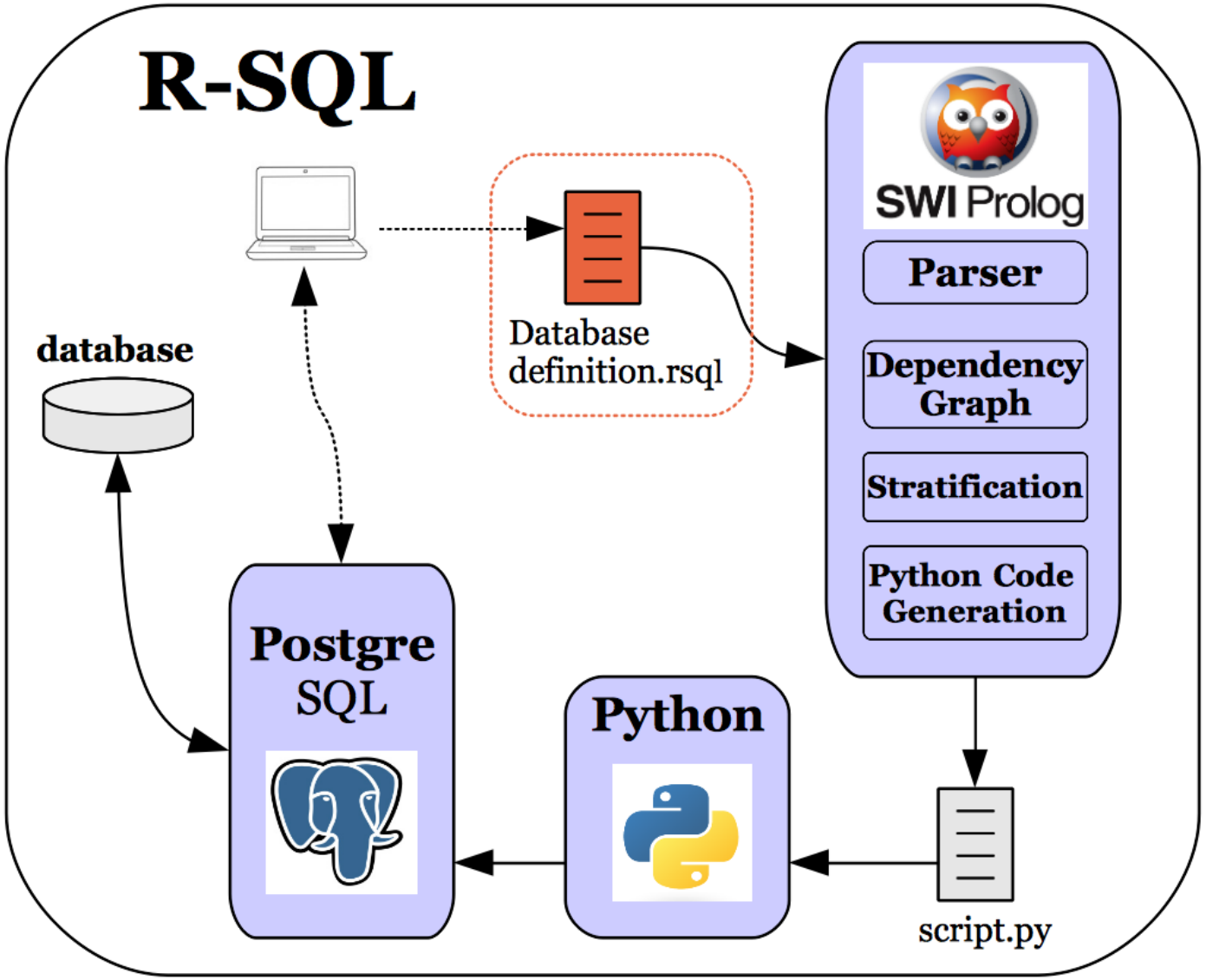
Plan of the talk

- 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



Database Definition



R-SQL Syntax (I)

Main Idea

To define recursive relations as:

```
R Sch := SELECT ... FROM R1... R ..... Rn ...
```

R-SQL Syntax (II)

```
sql_db    ::=  R sch := sel_stm;
               ...
               R sch := sel_stm;

sel_stm   ::=  SELECT exp,..,exp [FROM R,..,R [WHERE c]]
               | sel_stm UNION sel_stm
               | sel_stm EXCEPT R

sch       ::=  (A T,...,A T)

exp       ::=  C | R.A | exp opm exp | - exp

c         ::=  TRUE | FALSE | exp opc exp | NOT (c)
               | c [ AND | OR ] c
```

Example 1: Mutual Recursion

Even/Odd Numbers in 0...100

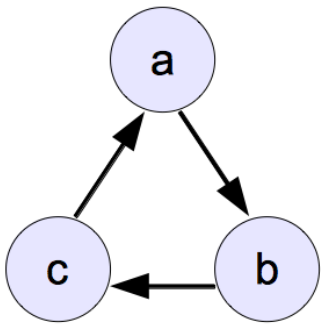
```
even(x float) :=
  SELECT 0
  UNION
  SELECT odd.x+1 FROM odd WHERE odd.x<100;


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

Example 2: Non-Linear Recursion

Fibonacci Numbers

```
fib1(n float, f float) :=
  SELECT fib.n, fib.f FROM fib;

fib2(n float, f float) :=
  SELECT fib.n, fib.f FROM fib;

fib(n float, f float) :=
  SELECT 0,1 UNION
  SELECT 1,1 UNION
  SELECT fib1.n+1,fib1.f+fib2.f FROM fib1,fib2
    WHERE fib1.n=fib2.n+1 AND fib1.n<10;
```

Example 3: Cycle in Graph



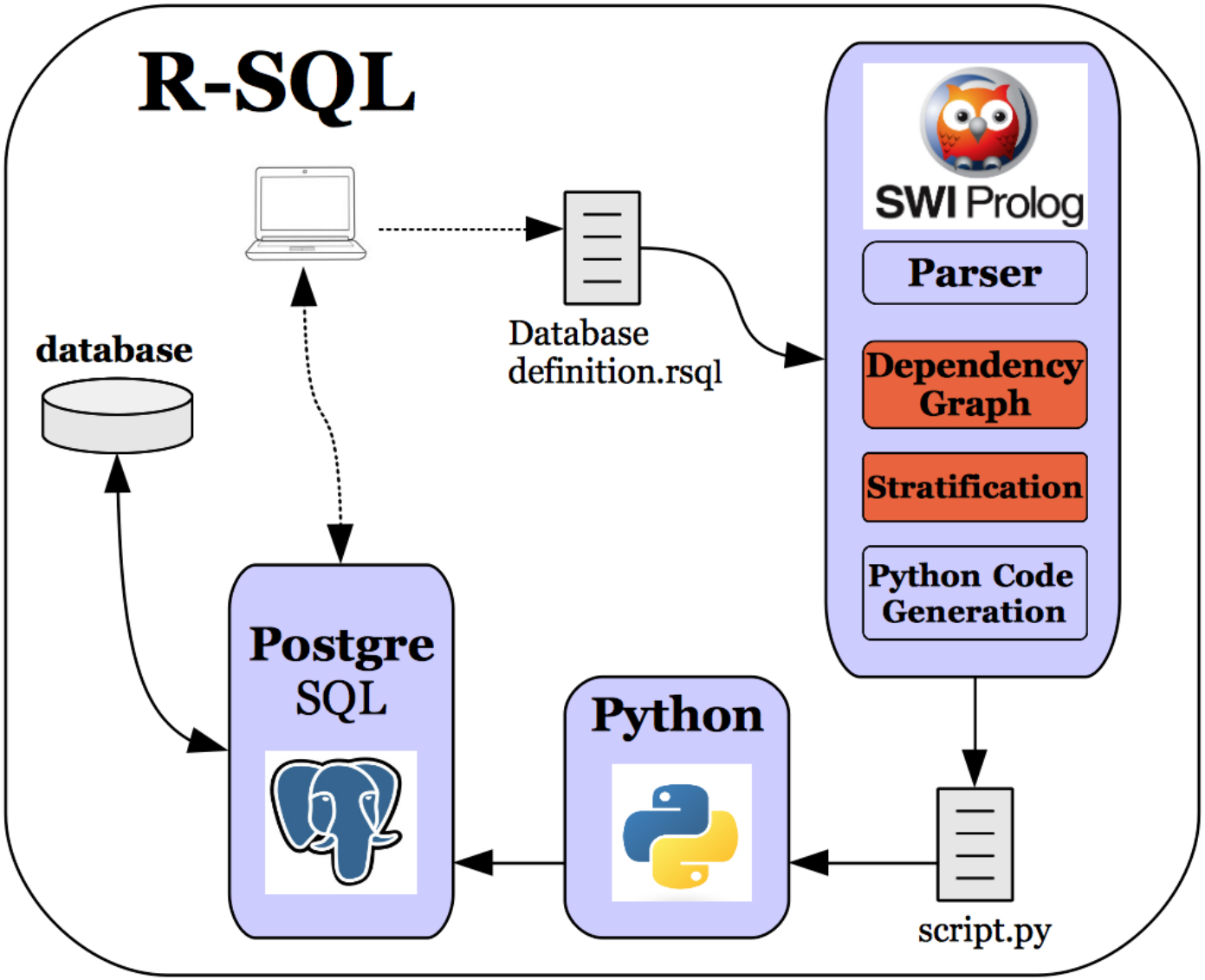

```
arc(ori varchar(1), des varchar(1)) :=
  SELECT 'a','b' UNION
  SELECT 'b','c' UNION
  SELECT 'c','a';

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

Stratified Fixpoint Semantics for R-SQL

- Analogously to Datalog:
  - A depencency 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.

System Structure

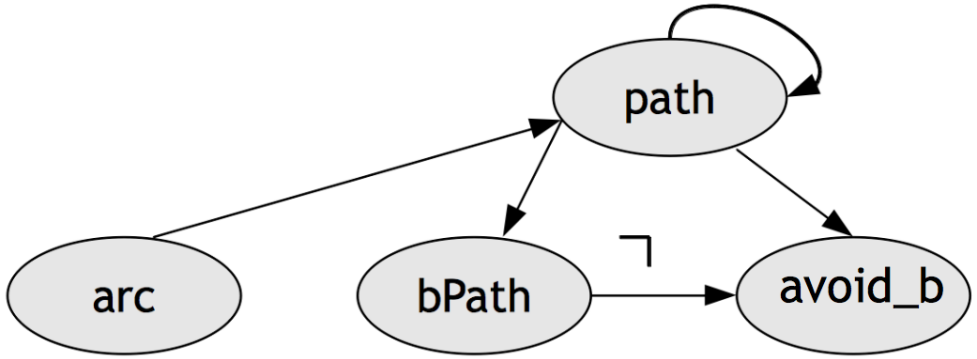


Example 3: Extended Graph

```
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

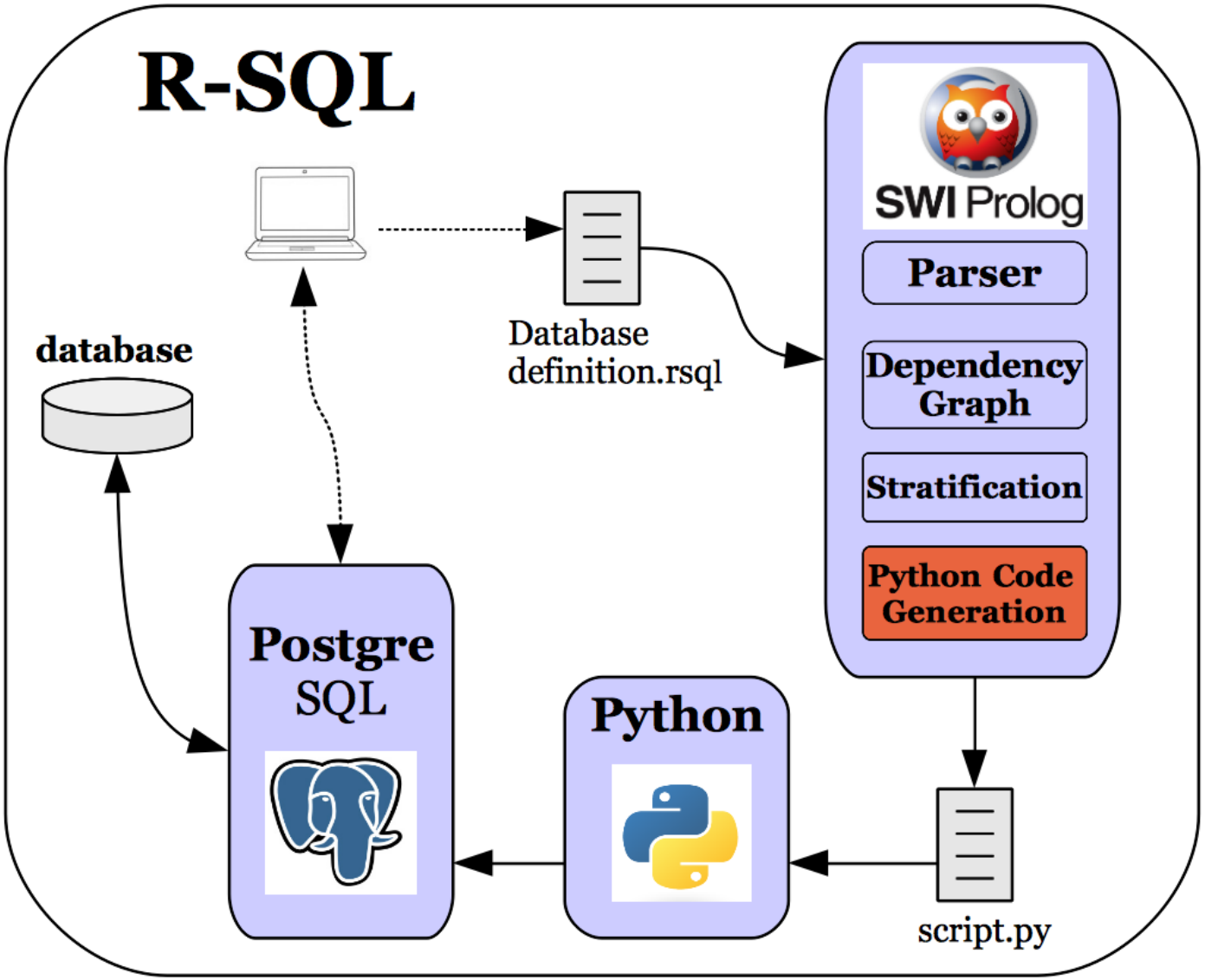
```
arc := SELECT 'a','b' UNION SELECT 'b','c' UNION SELECT 'c','a';
path := SELECT * from arc UNION SELECT ... FROM arc, path ...;
bPath := SELECT path.ori, path.des FROM path WHERE ...;
avoid_b := SELECT path.ori, path.des FROM path EXCEPT bPath;
```



Stratification

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

System Structure



Interpretation of Select Statements

Given an interpretation $\mathtt{I}$: $\{\mathtt{R1}, ..., \mathtt{Rn}\} \rightarrow \mathcal{P}(\mathcal{T})$, we define the semantics of a `sel_stm` in the context of $\mathtt{I}$ as:

- $[\![\text{SELECT } \mathtt{exp}_1, \ldots, \mathtt{exp}_k]\!]^{\mathtt{I}} = \{(exp_1, \ldots, exp_k)\}$.
- $[\![\text{SELECT } \mathtt{exp}_1, \ldots, \mathtt{exp}_k \text{ FROM } \mathtt{R}_1, \ldots, \mathtt{R}_m \text{ WHERE } \mathtt{c}]\!]^{\mathtt{I}} =$ $\{(exp_1[\overline{a}/\overline{\mathtt{A}}], \ldots, exp_k[\overline{a}/\overline{\mathtt{A}}]) |$ $\overline{a} \in \mathtt{I}(\mathtt{R}_1) \times \ldots \times \mathtt{I}(\mathtt{R}_m)$ and $c[\overline{a}/\overline{\mathtt{A}}]$ is satisfied$\}$.
- $[\![\mathtt{sel\_stm}_1 \text{ UNION } \mathtt{sel\_stm}_2]\!]^{\mathtt{I}} = [\![\mathtt{sel\_stm}_1]\!]^{\mathtt{I}} \bigcup [\![\mathtt{sel\_stm}_2]\!]^{\mathtt{I}}$.
- $[\![\mathtt{sel\_stm} \text{ EXCEPT } \mathtt{R}]\!]^{\mathtt{I}} = [\![\mathtt{sel\_stm}]\!]^{\mathtt{I}} \setminus \mathtt{I}(\mathtt{R})$.

Example 1: Even/Odd Numbers

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

Semantics of odd in the context of I

$[\![\mathtt{SELECT\ even.x+1\ FROM\ even\ WHERE\ even.x<100}]\!]^{\mathtt{I}} =$

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

$\{(1),(3)\}$.

Example 1: Even/Odd Numbers, Intended Semantics

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

$\mathtt{I}(\mathtt{odd}) = \{(1), (3), \ldots, (99)\} =$
$[\![\mathtt{SELECT\ even.x+1\ FROM\ even\ WHERE\ even.x<100}]\!]^{\mathtt{I}}$

Fixpoint Interpretation

- 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 := sel_stm`, in stratum i:

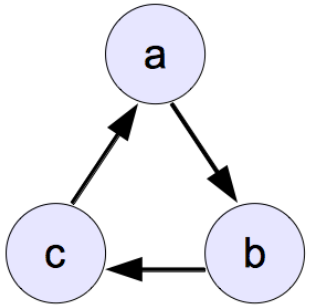
$$T_i(\mathtt{I})(\mathtt{R}) = [\![\mathtt{sel\_stm}]\!]^{\mathtt{I}}$$

- The database semantics is a fixpoint

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

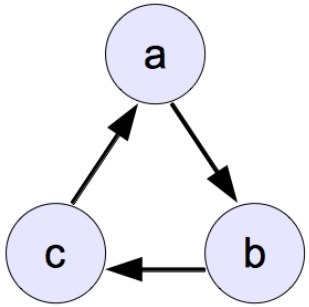
where i is the number of strata.

Example of the Computation of the Fixpoint



$T_1^n(\emptyset)(\texttt{path})$	Added tuples at each iteration
$T_1^1(\emptyset)(\texttt{path})$	{('a','b'), ('b','c'), ('c','a') },
$T_1^2(\emptyset)(\texttt{path})$	{('a','c'), ('b','a'), ('c','b') },
$T_1^3(\emptyset)(\texttt{path})$	{('a','a'), ('b','b'), ('c','c') },

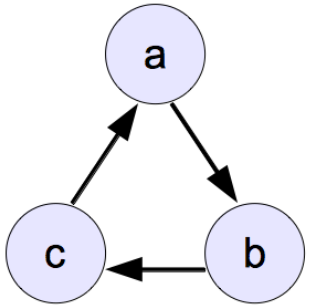
Example of the Computation of the Fixpoint



$T_1^n(\emptyset)(\texttt{path})$	Added tuples at each iteration
$T_1^1(\emptyset)(\texttt{path})$	{('a','b'), ('b','c'), ('c','a') },
$T_1^2(\emptyset)(\texttt{path})$	{('a','c'), ('b','a'), ('c','b') },
$T_1^3(\emptyset)(\texttt{path})$	{('a','a'), ('b','b'), ('c','c') },

All these tuples are $fix_1(\emptyset)(\texttt{path})$.

Example of the Computation of the Fixpoint

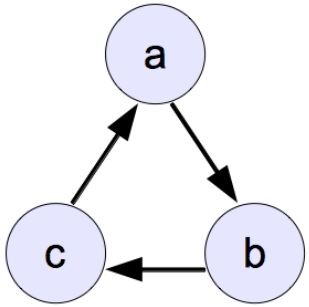


$T_1^n(\emptyset)(\texttt{path})$	Added tuples at each iteration
$T_1^1(\emptyset)(\texttt{path})$	{('a','b'), ('b','c'), ('c','a') },
$T_1^2(\emptyset)(\texttt{path})$	{('a','c'), ('b','a'), ('c','b') },
$T_1^3(\emptyset)(\texttt{path})$	{('a','a'), ('b','b'), ('c','c') },

All these tuples are $fix_1(\emptyset)(\texttt{path})$.

Similarly, we obtain tuples for `arc` y `bPath` to complete fix_1.

Example of the Computation of the Fixpoint



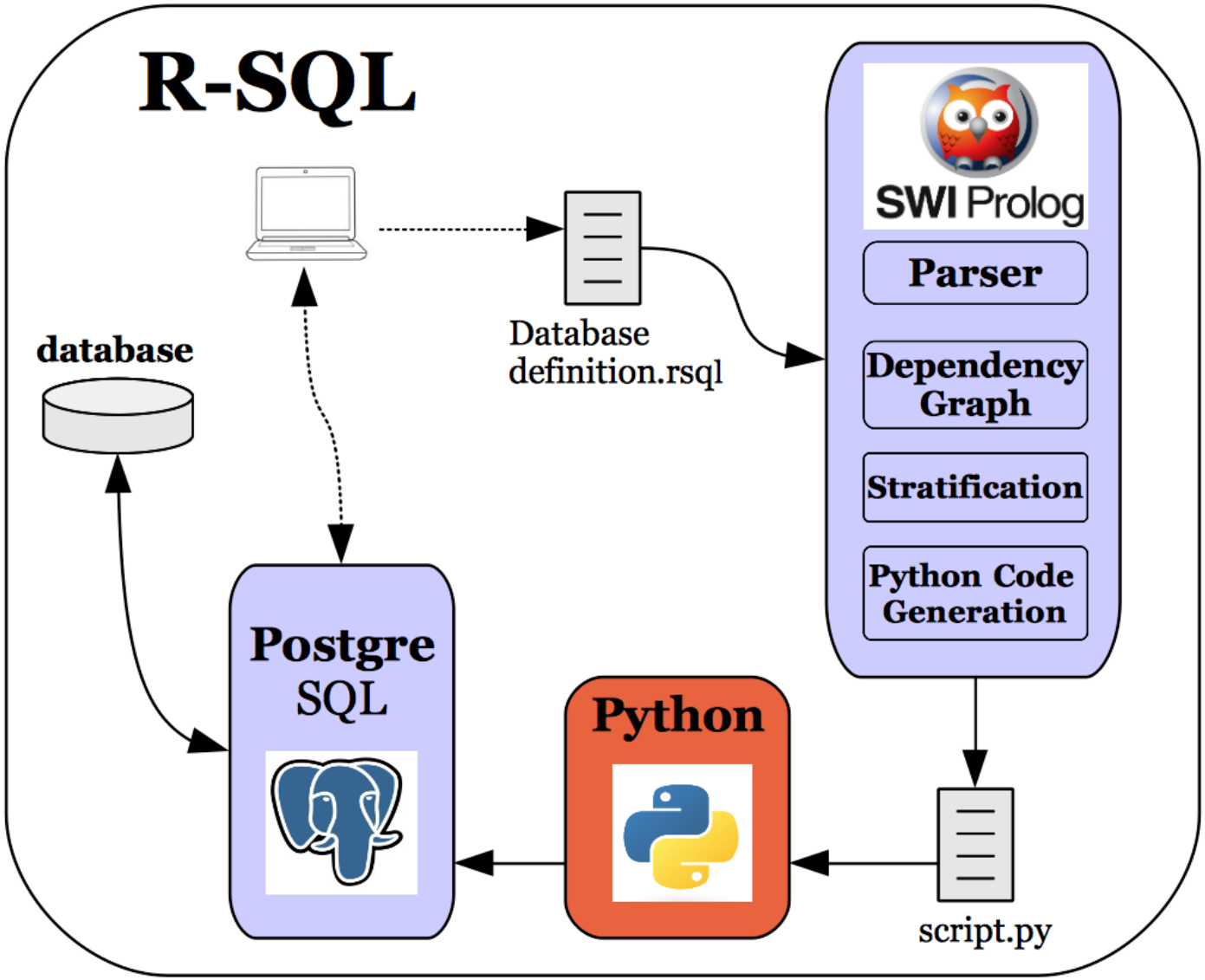
$T_1^n(\emptyset)(\texttt{path})$	Added tuples at each iteration
$T_1^1(\emptyset)(\texttt{path})$	{('a','b'), ('b','c'), ('c','a') },
$T_1^2(\emptyset)(\texttt{path})$	{('a','c'), ('b','a'), ('c','b') },
$T_1^3(\emptyset)(\texttt{path})$	{('a','a'), ('b','b'), ('c','c') },

All these tuples are $fix_1(\emptyset)(\texttt{path})$.
Similarly, we obtain tuples for `arc` y `bPath` to complete fix_1.
Then, $fix_2(fix_1)(\texttt{avoid\_b})$ is computed to obtain fix_2, which is the database fixpoint.

Algorithm for Computing the Fixpoint

```
str:=1
while str ≤ numstr do
    for each R_i ∈ R_str do CREATE TABLE R_i sch_i
    change := true
    while change do
        size := rel_size_sum(R_str)
        for each R_i ∈ R_str do
        INSERT INTO R_i SELECT * FROM sel_stm_i
            EXCEPT SELECT * FROM R_i;
        change = (size ≠ rel_size_sum(R_str))
    end while
    str:=str+1
end while
```

System Structure



Example of Code Generation (I)

1st stratum

```
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');
```

2nd stratum

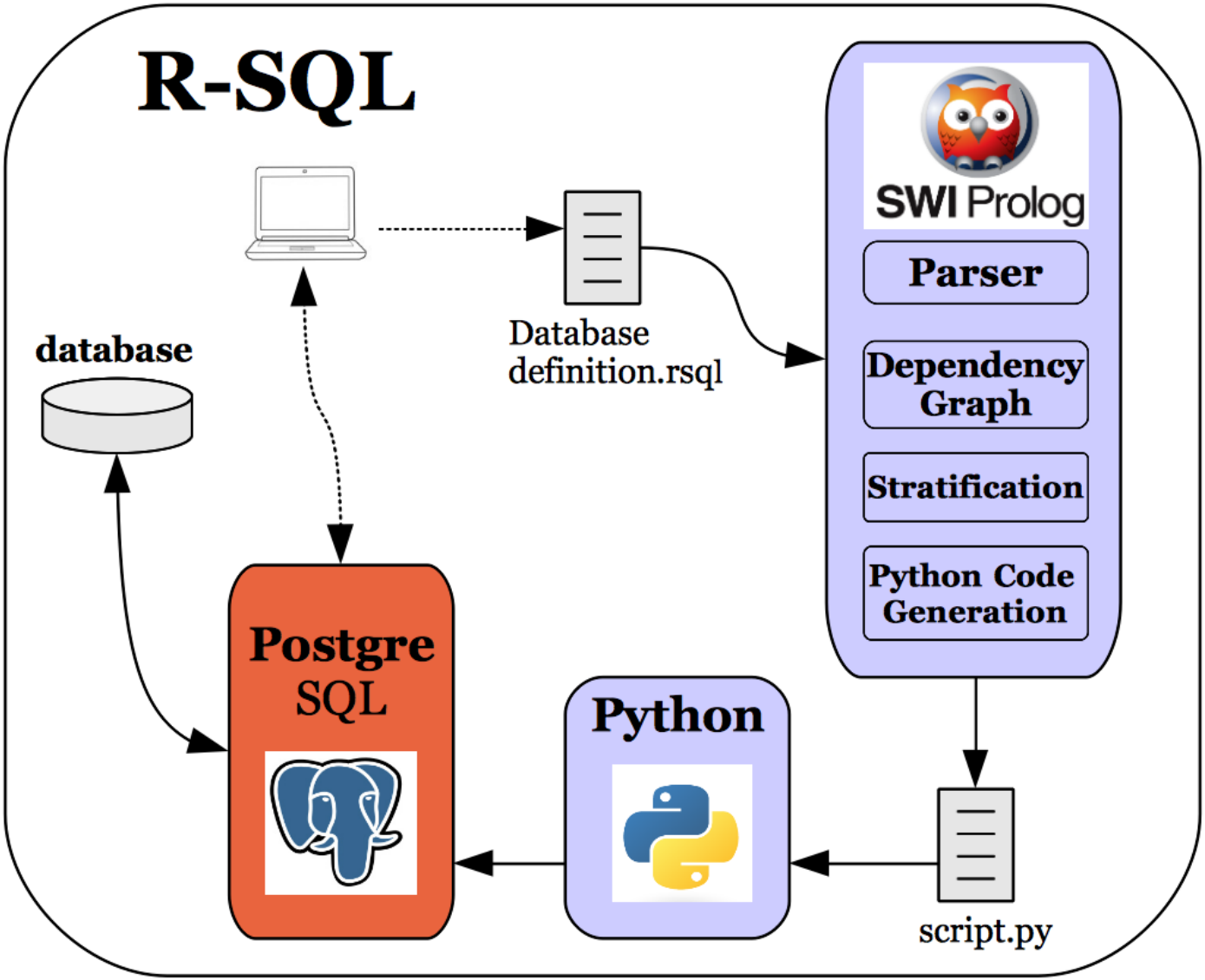
```
avoid_b(ori varchar(1), des varchar(1)) :=
  SELECT path.ori, path.des FROM path
    EXCEPT bPath;
```

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



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!