Incorporating Hypothetical Views and Extended Recursion into SQL Database Systems *

Gabriel Aranda-López[1], Susana Nieva[1], Fernando Sáenz-Pérez[2], and Jaime Sánchez-Hernández[1]

[1] Dept. Sistemas Informáticos y Computación, UCM, Spain
[2] Dept. Ingeniería del Software e Inteligencia Artificial, UCM, Spain
garanda@fdi.ucm.es, {nieva,fernan,jaime}@sip.ucm.es

Abstract

Current database systems supporting recursive SQL impose restrictions on queries such as linearity, and do not implement mutual recursion. In a previous work we presented the language and prototype R-SQL to overcome those drawbacks. Now we introduce a formalization and an implementation of the database system HR-SQL that, in addition to extended recursion, incorporates hypothetical reasoning in a novel way which cannot be found in any other SQL system, allowing both positive and negative assumptions. The formalization extends the fixpoint semantics of R-SQL. The implementation improves the efficiency of the previous prototype and is integrated in a commercial DBMS.

1 Introduction

Current relational database systems provide limited support for the ANSI/ISO standard language SQL w.r.t. recursion. In [2] we proposed a new approach, called R-SQL, aimed to overcome some of such limits. We developed a formal framework, borrowing techniques from the deductive database field, such as stratified negation [15], and following the original relational data model [7], so avoiding both duplicates and nulls (as encouraged by [8]). But in addition to recursion, several applications require predictive and historical analysis over large amounts of data [10], typically making some sort of assumptions to deduce conclusions. Hypothetical queries, also known as "what-if" queries, can help managers to take decisions on scenarios that are somewhat changed with respect to a current state. Such queries are used, for instance, for deciding what resources must be added, changed or removed to optimize some criterion. Current applications include OLAP environments business intelligence, and e-commerce.

So, driven by these needs, with the work proposed in this paper we face the inclusion of hypothetical queries and views in the recursive SQL setting based on [2]. To this end, we extend a subset of standard SQL to embody both recursive definitions and hypothetical views in the language HR-SQL. We summarize the syntax and semantics of the definition language in Section 2, and introduce a novel syntax and semantics of queries and view definitions in sections 3 and 4, respectively. An assumption (hypothetical reasoning) can be either overloading the relation (with the new clause ASSUME `query` IN `relation`) or restricting it (ASSUME `query` NOT IN `relation`). For supporting our approach, we propose a stratified fixpoint semantics which is an extension of the semantics presented in [2] to give meaning to hypothetical queries and view definitions.

Since our targets are current state-of-the-art relational database systems (DBMS's), we adhere to stratification in order to return a single answer set [15], which is a natural expectation

*This work has been partially supported by the Spanish projects S2009/TIC-1465 (PROMETIDOS) and UCM-BSCH-GR58/08-910502 (GPD-UCM).

from current database users. In Section 5, we propose an implementation of the HR-SQL language for the concrete system IBM DB2 (although it is easily adaptable to any other system), which improves the prototype introduced in [2] by factoring out those fragments of SQL queries that can be computed out of the fixpoint operator loop. In addition, we propose an efficient query solving procedure by generating SQL PL scripts and temporary tables to avoid locks and logging, therefore providing memory scalability and performance. Moreover, we provide a shell in which users can submit regular, hypothetical, and extended recursive SQL queries. Whereas regular queries are directly sent to the host DBMS, hypothetical and recursive queries are processed by such SQL PL scripts.

Related Work. To the best of our knowledge, there have been neither a formalization nor a system for SQL combining recursion and hypothetical queries as we do. However, we list some related works in both the relational and logic programming fields. With respect to hypothetical relational databases, the very first work was presented in [13], where hypotheses were stated by replacing actual data with a REPLACE operator, and assumed data persist until the query is finished. In that early work, recursion was not considered. Works as [9] present extensions of RA to support hypothetical queries by means of updates and with no recursion. Also, the educational system DES [12] includes hypothetical SQL queries, but neither hypothetical views nor negative assumptions are supported. On the logic programming arena, Hypothetical Datalog [3, 5] fits into intuitionistic logic programming, an extension of logic programming including both embedded implications and negation, and integrates atomic assumptions as hypothetical queries in the inference system. It has been a proposal thoroughly studied from semantic and complexity point-of-views, allowing to assume atoms in order to prove goals. Transaction logic [4] allows a database to be updated by transactions with elementary updates, and the transaction base is immutable. If bulk updates are needed, the transaction base must account for them. In [1] we developed a more expressive setting for constraint deductive databases based on Hereditary Harrop formulas. In particular, it provides support for assuming rules as hypothetical queries. Our current work can be understood as porting this feature to relational databases by adding the assume clause involving assumptions over relations which intensionally add new tuples (i.e., with select statements) to such relations. As a surplus, in this work, we allow to intensionally remove tuples from such relations by negative assumptions.

2 The Definition Database Language of HR-SQL

This language is oriented to provide definition for databases using a SQL-like language, which allows to define recursive definitions of relations. The formal syntax for a database definition is described by the following grammar:

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

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

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

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

cond     ::=  TRUE | FALSE | exp b_op exp | NOT cond | cond [AND|OR] cond

m_op     ::=  + | - | / | *

b_op     ::=  = | <> | < | > | >= | <=
```

Uppercase identifiers denote terminal symbols and lowercase ones denote grammar productions, `R` stands for relation names, `A` for attribute names, `T` for standard SQL types (as INTEGER, FLOAT, VARCHAR(N)), `cond` for Boolean conditions, `m_op` and `b_op` for mathematical and Boolean operators respectively, and `C` for constants of a valid SQL type.

A database `db` is a (non-empty) sequence of relation definitions. A relation definition assigns a select statement to the relation, that is identified by its name `R` and its schema `sch`, that is a tuple of attribute names with their corresponding types. As syntactic sugar, we admit `*` in the projection list of SQL statements. The HR-SQL definition language coincides with the R-SQL introduced in [2], but the syntax of the EXCEPT operator allows now any select statement in the right part, instead of a simple relation name (as it was the case in [2]).

Example 1. *The travel database definition below (inspired on an example of [6]) represents the information sketched in Figure 1. This database includes the relations* `flight`, `bus` *and* `boat` *with schema* `(ori varchar(10), des varchar(10), time float)` *to store information about origin (*`ori`*), destination (*`des`*) and time (*`time`*), for traveling around the Canary Islands.*

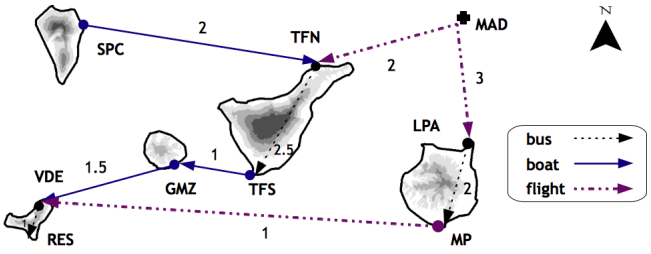

Figure 1: Travel Database for the Canary Islands.

The relation `link` *collects all the possible transports. The relation* `travel` *is the transitive closure of* `link`*, i.e., it provides all the possible travels of the database, maybe concatenating any of the available transports. Their respective definitions written in HR-SQL syntax are:*

```
link(ori varchar(10), des varchar(10), time float):=
   SELECT * FROM flight UNION SELECT * FROM boat UNION
   SELECT * FROM bus;
travel(ori varchar(10), des varchar(10), time float):=
   SELECT * FROM link UNION
   SELECT link.ori, travel.des, link.time + travel.time
          FROM link, travel WHERE link.des=travel.ori;
```

From now on, $\mathtt{RN}_{\mathtt{db}}$ stands for the set of relations names $\{\mathtt{R}_1, \ldots, \mathtt{R}_n\}$ defined in a database `db`. We write $\mathtt{RN}_{\mathtt{sel\_stm}}$ for the set of relation names occurring in a select statement `sel_stm`. For the case of a select statement of the form $\mathtt{sel\_stm} = \mathtt{sel\_stm}_1$ EXCEPT $\mathtt{sel\_stm}_2$ we also define $\mathtt{RN}^{\neg}_{\mathtt{sel\_stm}}$ as the set of relation names occurring in $\mathtt{sel\_stm}_2$ (notice that $\mathtt{RN}^{\neg}_{\mathtt{sel\_stm}} \subseteq \mathtt{RN}_{\mathtt{sel\_stm}}$). We assume that for every `R sch:= sel_stm` defined in `db` it holds that $\mathtt{RN}_{\mathtt{sel\_stm}} \subseteq \mathtt{RN}_{\mathtt{db}}$.

2.1 Fixpoint Semantics

The meaning of every relation defined in a database `db` corresponds to the set of tuples that "satisfies" the relation definition. In [2] a stratified fixpoint semantics for the language R-SQL was introduced. Here, we recapitulate the main concepts in order to facilitate the understanding of the following sections. In addition, we introduce the semantics of the extended EXCEPT select statement.

The stratified fixpoint theory holds on the notion of dependency graph for a database. The *dependency graph* associated to db, denoted by $DG_{\mathtt{db}}$, is a directed graph whose nodes are the elements of $\mathtt{RN}_{\mathtt{db}}$, and the edges (which can be negatively labeled) are determined as follows. For any relation definition R sch := sel_stm there is an edge from every relation name $\mathtt{R}' \in \mathtt{RN}_{\mathtt{sel\_stm}}$ to R. Those edges produced by the relation names belonging to $\mathtt{RN}^{\neg}_{\mathtt{sel\_stm}}$ are *negatively labeled*. Then, for every pair of relations $\mathtt{R}_1, \mathtt{R}_2 \in \mathtt{RN}_{\mathtt{db}}$, we say that $\mathtt{R}_2$ *depends* on $\mathtt{R}_1$ if there is a path from $\mathtt{R}_1$ to $\mathtt{R}_2$ in $DG_{\mathtt{db}}$. And $\mathtt{R}_2$ *negatively depends* on $\mathtt{R}_1$ if there is a path from $\mathtt{R}_1$ to $\mathtt{R}_2$ in $DG_{\mathtt{db}}$ with at least one negatively labeled edge. The previous concepts are needed to characterize the stratifiable databases.

Definition 1. *A* stratification *of a database* db *defining n relations is a mapping* $str : \mathtt{RN}_{\mathtt{db}} \to \{1, \ldots, n\}$*, such that:* $str(\mathtt{R}_i) \leq str(\mathtt{R}_j)$*, if* $\mathtt{R}_j$ *depends on* $\mathtt{R}_i$*, and* $str(\mathtt{R}_i) < str(\mathtt{R}_j)$*, if* $\mathtt{R}_j$ *negatively depends on* $\mathtt{R}_i$*.*

The database db *is* stratifiable *if there exists a stratification for it. In this case, for every* $\mathtt{R} \in \mathtt{RN}_{\mathtt{db}}$*, we say that* $str(\mathtt{R})$ *is the* stratum *of* R*. And for a select statement* sel_stm*, we define* $str(\mathtt{sel\_stm}) = max\{str(\mathtt{R}_i) \mid \mathtt{R}_i \in \mathtt{RN}_{\mathtt{sel\_stm}}\}$*.*

From now on, we consider a fixed stratifiable database db and a stratification *str* for it. In order to give meaning to a relation R ($\mathtt{A}_1$ $\mathtt{T}_1, ..., \mathtt{A}_r$ $\mathtt{T}_r$), we assume that every type $\mathtt{T}_i$, $i = 1..r$, denotes a domain D_i. We also assume a *universal domain* $\mathcal{D}$, which is the union of the family of the considered domains. Since different relations can have different arities, we use the set $\mathcal{T} = \bigcup_{n \geq 1} \mathcal{D}^n$. Interpretations are defined as functions that associate an element of $\mathcal{P}(\mathcal{T})$ to each element of $\mathtt{RN}_{\mathtt{db}}$, and they are classified by strata, as we formalize next.

Definition 2. *Let* $i \geq 1$*, an* interpretation I *for* db*, over the stratum* i*, is a function* $I : \mathtt{RN}_{\mathtt{db}} \to \mathcal{P}(\mathcal{T})$*, such that for every* $\mathtt{R} \in \mathtt{RN}_{\mathtt{db}}$ *with schema* sch*:*

- *If* $\mathtt{sch} \equiv (\mathtt{A}_1\ \mathtt{T}_1, \ldots, \mathtt{A}_r\ \mathtt{T}_r)$*, and* $D_1, \ldots, D_r$ *are, respectively, the domains denoted by* $\mathtt{T}_1, \ldots, \mathtt{T}_r$*, then* $I(\mathtt{R}) \subseteq D_1 \times \ldots \times D_r$*,*
- $I(\mathtt{R}) = \emptyset$*, if* $str(\mathtt{R}) > i$*.*

The set of interpretations for db *over the stratum* i *is denoted by* $\mathcal{I}_i^{\mathtt{db}}$*. Let* $I_1, I_2 \in \mathcal{I}_i^{\mathtt{db}}$*.* I_1 is less than or equal to I_2 at stratum i*, denoted by* $I_1 \sqsubseteq_i I_2$*, if the following conditions are satisfied for every* $\mathtt{R} \in \mathtt{RN}_{\mathtt{db}}$*:* $I_1(\mathtt{R}) = I_2(\mathtt{R})$*, if* $str(\mathtt{R}) < i$*, and* $I_1(\mathtt{R}) \subseteq I_2(\mathtt{R})$*, if* $str(\mathtt{R}) = i$*.*

It is straightforward to check that for any i, $(\mathcal{I}_i^{\mathtt{db}}, \sqsubseteq_i)$ is a poset. The main question is that when an interpretation over a stratum i increases, the set of tuples associated to the relations whose stratum is i can increase, but the sets associated to relations of smaller strata remain invariable. In addition, $(\mathcal{I}_i^{\mathtt{db}}, \sqsubseteq_i)$ is a complete partially ordered set: If $\{I_n\}_{n \geq 0}$ is a chain in $(\mathcal{I}_i^{\mathtt{db}}, \sqsubseteq_i)$, then $\hat{I}$, defined as $\hat{I}(\mathtt{R}) = \bigcup_{n \geq 0} I_n(\mathtt{R})$, $\mathtt{R} \in \mathtt{RN}_{\mathtt{db}}$, is the least upper bound of $\{I_n\}_{n \geq 0}$.

The following definition formalizes the meaning of a select statement sel_stm in the context of a concrete interpretation I.

Definition 3. *Let* $i \geq 1$*,* $I \in \mathcal{I}_i^{\mathtt{db}}$*. Let* sel_stm *be a select statement, such that* $str(\mathtt{sel\_stm}) \leq i$*. We recursively define the* interpretation of sel_stm w.r.t. I *for* db*, denoted by* $[\![\mathtt{sel\_stm}]\!]^I$*, as follows:*

- $[\![\mathtt{sel\_stm}_1 \text{ UNION } \mathtt{sel\_stm}_2]\!]^I = [\![\mathtt{sel\_stm}_1]\!]^I \cup [\![\mathtt{sel\_stm}_2]\!]^I$*.*
- $[\![\mathtt{sel\_stm}_1 \text{ EXCEPT } \mathtt{sel\_stm}_2]\!]^I = [\![\mathtt{sel\_stm}_1]\!]^I \setminus [\![\mathtt{sel\_stm}_2]\!]^I$*.*
- $[\![\text{SELECT } \mathtt{exp}_1, \ldots, \mathtt{exp}_k]\!]^I = \{(exp_1, \ldots, exp_k)\}$*, where* exp_i *denotes the mathematical evaluation of* $\mathtt{exp}_i$*.*

- $[\![\text{SELECT } \texttt{exp}_1,\ldots,\texttt{exp}_k \text{ FROM } \texttt{R}_1,\ldots,\texttt{R}_m \text{ WHERE } \texttt{cond}]\!]^I =$
  $\{(exp_1[\overline{a}/\overline{\texttt{A}}],\ldots,exp_k[\overline{a}/\overline{\texttt{A}}]) \mid \overline{a} \in I(\texttt{R}_1)\times\ldots\times I(\texttt{R}_m), cond[\overline{a}/\overline{\texttt{A}}] \textit{ is satisfied}\}$,
  where $\overline{\texttt{A}}$ *represents a sequence of attributes prefixed with their corresponding relation names, i.e., if* $\texttt{A}_1^j,\ldots,\texttt{A}_{r_j}^j$ *are the attributes of* $\texttt{R}_j$, $1 \le j \le m$, *then* $\overline{\texttt{A}}$ *is the complete sequence* $\texttt{R}_1.\texttt{A}_1^1,\ldots,\texttt{R}_1.\texttt{A}_{r_1}^1,\ldots,\texttt{R}_m.\texttt{A}_1^m,\ldots,\texttt{R}_m.\texttt{A}_{r_m}^m$; *the notation* $exp_j[\overline{a}/\overline{\texttt{A}}]$, $1 \le j \le k$, *stands for the mathematical evaluation of* $\texttt{exp}_j$, *after replacing the tuple* $\overline{a}$ *by* $\overline{\texttt{A}}$; *and* $cond[\overline{a}/\overline{\texttt{A}}]$ *denotes the evaluation of the Boolean expression* cond, *with the previous substitution.*

Next, for every i, an operator $T_i^{\texttt{db}}$ over the set $\mathcal{I}_i^{\texttt{db}}$ of interpretations of stratum i for db is defined. $T_i^{\texttt{db}}$ is continuous, as stated in [2]. The least fixpoint of $T_i^{\texttt{db}}$ is the interpretation giving meaning to the relations of db in the stratum i.

Definition 4. *The operator* $T_i^{\texttt{db}} : \mathcal{I}_i^{\texttt{db}} \longrightarrow \mathcal{I}_i^{\texttt{db}}$ *transforms interpretations over* i *as follows. For every* $I \in \mathcal{I}_i^{\texttt{db}}$ *and for every* $\texttt{R} \in \texttt{RN}_{\texttt{db}}$*:*

- $T_i^{\texttt{db}}(I)(\texttt{R}) = I(\texttt{R})$, *if* $str(\texttt{R}) < i$.
- $T_i^{\texttt{db}}(I)(\texttt{R}) = [\![\texttt{sel\_stm}]\!]^I$, *if* $str(\texttt{R}) = i$ *and* sel_stm *is the definition of* R *in* db.
- $T_i^{\texttt{db}}(I)(\texttt{R}) = \emptyset$, *if* $str(\texttt{R}) > i$.

Proposition 1 (Continuity of $T_i^{\texttt{db}}$)**.** Let $i \ge 1$ and $\{I_n\}_{n\ge0}$ be a chain of interpretations in $\mathcal{I}_i^{\texttt{db}}$ ($I_0 \sqsubseteq_i I_1 \sqsubseteq_i I_2 \sqsubseteq_i \ldots$). Then, $T_i^{\texttt{db}}(\bigsqcup_{n\ge0} I_n) = \bigsqcup_{n\ge0} T_i^{\texttt{db}}(I_n)$.

Therefore, the existence of a least fixpoint stratum by stratum is a direct consequence of the Knaster-Tarski fixpoint theorem [14].

Theorem 1. There is a fixpoint interpretation $fix^{\texttt{db}} : \texttt{RN}_{\texttt{db}} \to \mathcal{P}(\mathcal{T})$, such that for every R $\in \texttt{RN}_{\texttt{db}}$, if sel_stm is the definition of R in db, then $fix^{\texttt{db}}(\texttt{R}) = [\![\texttt{sel\_stm}]\!]^{fix^{\texttt{db}}}$.

The interpretation $fix^{\texttt{db}}$ defines the semantics of db. The construction of this fixpoint is stratum by stratum as follows:

The operator $T_1^{\texttt{db}}$ has a least fixpoint, called $fix_1^{\texttt{db}}$, which is $\bigsqcup_{n\ge0}(T_1^{\texttt{db}})^n(\emptyset)$, the least upper bound of the sequence $\{(T_1^{\texttt{db}})^n(\emptyset)\}_{n\ge0}$, where $(T_1^{\texttt{db}})^n(\emptyset)$ is the result of n successive applications of $T_1^{\texttt{db}}$ to the empty interpretation.

Consider now the sequence $\{(T_2^{\texttt{db}})^n(fix_1^{\texttt{db}})\}_{n\ge0}$ of interpretations in $(\mathcal{I}_2^{\texttt{db}}, \sqsubseteq_2)$ greater than $fix_1^{\texttt{db}}$. Using the definition of $T_i^{\texttt{db}}$ and the fact that $fix_1^{\texttt{db}}(\texttt{R}) = \emptyset$ for every R such that $str(\texttt{R}) \ge 2$, it is easy to prove (as for the stratum 1) that such sequence is a chain, $fix_1^{\texttt{db}} \sqsubseteq_2 T_2^{\texttt{db}}(fix_1^{\texttt{db}}) \sqsubseteq_2 T_2^{\texttt{db}}(T_2^{\texttt{db}}(fix_1^{\texttt{db}})) \sqsubseteq_2 \ldots, \sqsubseteq_2 (T_2^{\texttt{db}})^n(fix_1^{\texttt{db}}), \ldots$ with least upper bound in $(\mathcal{I}_2^{\texttt{db}}, \sqsubseteq_2)$, $\bigsqcup_{n\ge0}(T_2^{\texttt{db}})^n(fix_1)$, that is the least fixpoint of $T_2^{\texttt{db}}$ containing $fix_1^{\texttt{db}}$, called $fix_2^{\texttt{db}}$.

Now, if $k = max\{str(\texttt{R}) \mid \texttt{R} \in \texttt{RN}_{\texttt{db}}\}$, by proceeding successively, for every i, $1 < i \le k$, a chain, $\{(T_i^{\texttt{db}})^n(fix_{i-1}^{\texttt{db}})\}_{n\ge0}$ can be defined, and a fixpoint of $T_i^{\texttt{db}}$, $fix_i^{\texttt{db}} = \bigsqcup_{n\ge0}(T_i^{\texttt{db}})^n(fix_{i-1}^{\texttt{db}})$, can be found. In addition, $fix_1^{\texttt{db}} \sqsubseteq_k \ldots \sqsubseteq_k fix_k^{\texttt{db}}$. We call $fix^{\texttt{db}}$ to $fix_k^{\texttt{db}}$, since it contains the information of the whole database.

3 The Query Language of HR-SQL

As usual in SQL, users of an HR-SQL database can formulate queries by means of select statements. The novelty of the HR-SQL language w.r.t. R-SQL is the incorporation of hypothetical queries. The syntax of queries is defined as:

```
query    ::=  sel_stm | sel_hyp
sel_hyp  ::=  ASSUME hypo,...,hypo sel_stm
hypo     ::=  sel_stm [NOT] IN R
```

Example 2. *Consider the database of Example 1, and the query: how long does it take to arrive in Valverde from Madrid, if boat links that take more than one hour are not considered? It can be expressed in HR-SQL as:*

```
ASSUME SELECT * FROM boat WHERE boat.time > 1 NOT IN link
SELECT travel.time FROM travel
     WHERE travel.ori = 'MAD' and travel.des = 'VDE'
```

From the logical point of view, a hypothetical query can be interpreted as an intuitionistic implication: it represents the value of the consequent assuming the antecedent. Next we formalize this idea.

3.1 The Semantics of a Query

As usual, the answer of a query is identified with the set of tuples that satisfy such a query. So, for a stratifiable database definition db, this answer corresponds to the interpretation of the query w.r.t. the fixpoint of db. The following definition formalizes this concept for the different cases of queries. In the case of a hypothetical query, to reflect the changes introduced in the current database assuming the hypothesis, we will use the notation db[R sch := sel_stm′/R sch := sel_stm] to denote the database definition that results from the database db by replacing the relation definition R sch := sel_stm by R sch := sel_stm′. In addition, *sel*(query) denotes the select statement of query. More precisely *sel*(sel_stm) = sel_stm and *sel*(ASSUME $\mathtt{hypo}_1, \ldots, \mathtt{hypo}_k$ sel_stm) = sel_stm.

For readability, we give the definition only for the case of one assumption; for a sequence of assumptions it is obtained as a simple sequential extension, considering a sequence of such replacements, as shown in Example 3 later.

Definition 5. *Let* query *be a query for* db. *Its* answer w.r.t. db, *denoted by* $[\![\mathtt{query}]\!]_{\mathtt{db}}$, *is defined by cases:*

Simple query: $[\![\mathtt{sel\_stm}]\!]_{\mathtt{db}} = [\![\mathtt{sel\_stm}]\!]^{fix^{\mathtt{db}}}$.

Hypothetical query: If R sch := $\mathtt{sel\_stm}_{\mathtt{R}}$ *is the definition of* R *in* db, *then:*

- $[\![$ASSUME sel_stm′ IN R sel_stm$]\!]_{\mathtt{db}} = [\![\mathtt{sel_stm}]\!]^{fix^{\mathtt{db}'}}$, *where* $\mathtt{db}' = \mathtt{db}[\mathtt{R\ sch := sel\_stm}_{\mathtt{R}}$ UNION sel_stm′/ R sch := $\mathtt{sel\_stm}_{\mathtt{R}}]$.
- $[\![$ASSUME sel_stm′ NOT IN R sel_stm$]\!]_{\mathtt{db}} = [\![\mathtt{sel_stm}]\!]^{fix^{\mathtt{db}'}}$, *where* $\mathtt{db}' = \mathtt{db}[\mathtt{R\ sch := sel\_stm}_{\mathtt{R}}$ EXCEPT sel_stm′/ R sch := $\mathtt{sel\_stm}_{\mathtt{R}}]$.

Example 3. *Let* db *be the following database definition (for simplicity, we omit the schema* A int *for all the relations):*

```
R1:= SELECT 1 UNION SELECT 2 UNION SELECT 3;
R2:= sel_stmR2
     where sel_stmR2 ≡ SELECT 1 UNION SELECT 3 UNION SELECT 5
          EXCEPT SELECT R1.A FROM R1 WHERE R1.A=1 OR R1.A=2;
R3:= SELECT R2.A FROM R2 UNION SELECT R3.A*2 FROM R3 WHERE R3.A<5;
```

Consider the following hypothetical query:

```
query ≡ ASSUME SELECT R1.A FROM R1 WHERE R1.A < 3 IN R2,
                SELECT 3 NOT IN R2
        SELECT R3.A FROM R3
```

Then $[\![\mathtt{query}]\!]_{\mathtt{db}} = [\![$SELECT R3.A FROM R3$]\!]^{fix^{\mathtt{db}'}}$, *where* $\mathtt{db}' = (\mathtt{db})\theta\sigma$ *being:*

$\theta = [\texttt{R2 := sel\_stm}'_{\texttt{R2}}/\ \texttt{R2 := sel\_stm}_{\texttt{R2}}]$,
$\sigma = [\texttt{R2 := sel\_stm}'_{\texttt{R2}}$ EXCEPT SELECT 3$/\texttt{R2 := sel_stm}'_{\texttt{R2}}]$,
$\texttt{sel\_stm}'_{\texttt{R2}} \equiv \texttt{sel\_stm}_{\texttt{R2}}$ UNION SELECT R1.A FROM R1 WHERE R1.A < 3.

Therefore db′ *is the following database:*

```
R1:= SELECT 1 UNION SELECT 2 UNION SELECT 3;
R2:= ((SELECT 1 UNION SELECT 3 UNION SELECT 5
       EXCEPT SELECT R1.A FROM R1 WHERE R1.A=1 OR R1.A=2)
      UNION SELECT R1.A FROM R1 WHERE R1.A<3) EXCEPT SELECT 3;
R3:= SELECT R2.A FROM R2 UNION SELECT R3.A*2 FROM R3 WHERE R3.A<5;
```

The computation of a simple query for an existing database is easy, because the value of $[\![\texttt{sel\_stm}]\!]_{\texttt{db}}$ is $[\![\texttt{sel\_stm}]\!]^{fix^{\texttt{db}}}$, and $fix^{\texttt{db}}$ is known and coincides with the instance of the database. The case of a hypothetical query sel_hyp requires additional explanation, its meaning is the interpretation of a select statement w.r.t. a new database db′, where some relations have changed because the assumptions are incorporated to the corresponding relations. db′ must be a stratifiable database in order to define the interpretation $fix^{\texttt{db}'}$. By taking advantage of the stratified semantics, the computation of $fix^{\texttt{db}'}$ can be simplified:

First, the dependency graph $DG_{\texttt{db}'}$ is an extension of $DG_{\texttt{db}}$, because $\texttt{RN}_{\texttt{db}'} = \texttt{RN}_{\texttt{db}}$, and every relation definition of db′ is in db, but the new relation definition R sch := $\texttt{sel\_stm}_{\texttt{R}}$ UNION|EXCEPT sel_stm′. The edges from the relations inside $\texttt{sel\_stm}_{\texttt{R}}$ to R are already in $DG_{\texttt{db}}$. So $DG_{\texttt{db}'}$ can be built from $DG_{\texttt{db}}$ as follows: For every $\texttt{R}' \in \texttt{RN}_{\texttt{sel\_stm}'}$, an edge from R′ to R is added; it is negatively labeled in the EXCEPT case or if $\texttt{R}' \in \texttt{RN}^{\neg}_{\texttt{sel\_stm}'}$. A stratification for db′, $str' : \texttt{RN}_{\texttt{db}'} \to \{1, \ldots, n\}$, if it exists, satisfies $str'(\texttt{R}) \geq str(\texttt{R})$, since (as we have remarked already) the select statement that defines R in db′, contains the select statement $\texttt{sel\_stm}_{\texttt{R}}$, which defines R in db.

Second, in order to obtain $[\![sel(\texttt{sel\_hyp})]\!]^{fix^{\texttt{db}'}}$, it is only necessary to compute $fix^{\texttt{db}'}(\texttt{R}')$ for the relations R′ such that the relations in $\texttt{RN}_{sel(\texttt{sel\_hyp})}$ depend on R′. In addition, $fix^{\texttt{db}'}$ has not to be computed from stratum 1, as we will see. Let $i = str'(\texttt{R})$ ($i = min\{str'(\texttt{R}_j) | 1 \leq j \leq k\}$ in the general case, if assumptions for the relations $\texttt{R}_1, \ldots, \texttt{R}_k$ are considered), then $fix^{\texttt{db}'}(\texttt{R}') = fix^{\texttt{db}}(\texttt{R}')$, for every R′ with $str'(\texttt{R}') < i$. And let $S = \{\texttt{R}'' \mid \texttt{R}' \in \texttt{RN}_{sel(\texttt{sel\_hyp})}$ and R′ depends on $\texttt{R}''\}$, then $fix^{\texttt{db}'}$ can be obtained from $fix^{\texttt{db}}$ in the following way:

1. Compute $fix_i^{\texttt{db}'}(\texttt{R}')$ from $fix_{i-1}^{\texttt{db}}$ for every relations $\texttt{R}' \in S$ and $str'(\texttt{R}') = i$.
2. Compute $fix_j^{\texttt{db}'}(\texttt{R}')$ from $fix_{j-1}^{\texttt{db}'}$ for the relations $\texttt{R}' \in S$ and $str'(\texttt{R}') = j$, $j = i + 1$ .. $str'(sel(\texttt{sel\_hyp}))$.

Example 4. *Consider the stratifiable database* db *and the query of Example 3. Let str be a stratification for* db*, such that* $str(\texttt{R1}) = 1$, $str(\texttt{R2}) = 2$, $str(\texttt{R3}) = 3$. *In this case, str is also a stratification for the modified database* db′*, detailed in Example 3, needed to answer to* query*. It is easy to check that:*

$fix^{\texttt{db}}(\texttt{R1}) = \{(1), (2), (3)\}$, $fix^{\texttt{db}}(\texttt{R2}) = \{(3), (5)\}$, $fix^{\texttt{db}}(\texttt{R3}) = \{(3), (5), (6)\}$.

In order to obtain $fix^{\texttt{db}'}$*, notice that* $\texttt{RN}_{sel(\texttt{query})} = \{\texttt{R3}\}$. *So* $S = \{\texttt{R}'' \mid \texttt{R}' \in \{\texttt{R3}\}$ *and* R′ *depends on* $\texttt{R}''\} = \{\texttt{R1}, \texttt{R2}, \texttt{R3}\}$, *but the computation can start at stratum* $2 = str(\texttt{R2})$, *with* $fix_1^{\texttt{db}'} = fix_1^{\texttt{db}}$. R2 *is the only relation in S in stratum 2.*

$fix_2^{\texttt{db}'}(\texttt{R2}) = \{(1), (2), (5)\}$.

Similarly, for stratum 3, only $fix_3^{\texttt{db}'}(\texttt{R3})$ *must be computed to get the answer, even in the case that* db *had other relations in this stratum.*

$fix_3^{\texttt{db}'}(\texttt{R3}) = \{(1), (2), (4), (5), (8)\} = [\![$SELECT R3.A FROM R3$]\!]^{fix^{\texttt{db}'}} = [\![\texttt{query}]\!]_{\texttt{db}}$.

4 The View Definition Language of HR-SQL

In this section we extend the definition language by allowing the definition of views, which essentially consists of assigning names to queries in order to use them as relation names inside other queries, or inside itself to express recursive queries. The syntax is as follows:

```
vd    ::=  view ...  view
view  ::=  V sch := sel_stm; | HV sch := sel_hyp;
```

We use `V` for names of views that are defined by a non hypothetical query, and `HV` for hypothetical views. From now on, those symbols can be considered as elements of the set $\mathtt{RN}_{\mathtt{db}}$ as relation names.

We say that `vd` is *a definition of views for* `db` if the involved names in it are relation names of `db` or view names defined in `vd`. Mutual recursive definitions are allowed among non hypothetical views. Then their names can occur inside the definition of any view (hypothetical or not). Every hypothetical view can be recursive but its name cannot appear inside the definition of other views, which means that in a definition of views of the form:

$\mathtt{V}_1\ \mathtt{sch}_1\ \mathtt{:=}\ \mathtt{sel\_stm}_1\mathtt{;}\ \ldots\quad \mathtt{V}_m\ \mathtt{sch}_m\ \mathtt{:=}\ \mathtt{sel\_stm}_m\mathtt{;}$
$\mathtt{HV}_1\ \mathtt{sch}_1\ \mathtt{:=}\ \mathtt{sel\_hyp}_1\mathtt{;}\ \ldots\quad \mathtt{HV}_r\ \mathtt{sch}_r\ \mathtt{:=}\ \mathtt{sel\_hyp}_r\mathtt{;}$

for every $j = 1..m$, $\mathtt{V}_j$ can occur everywhere; for every $j = 1..r$, $\mathtt{HV}_j$ can occur inside the expression $sel(\mathtt{sel\_hyp}_j)$, but not in $\mathtt{sel\_stm}_1, \ldots, \mathtt{sel\_stm}_m$, $\mathtt{sel\_hyp}_k$, if $k \neq j$, nor in the assumption part of $\mathtt{sel\_hyp}_j$.

Example 5. *Referring to Example 1, assume there is a volcanic eruption in El Hierro and the airspace must be closed in the archipelago, as well as the bus in this island. But a boat from El Hierro to La Palma is added. The hypothetical view below can be defined in HR-SQL to represent the reachable cities in the Canary islands in this situation.*

```
reachable(ori varchar(10),des varchar(10)) := ASSUME
    (SELECT * FROM bus WHERE bus.ori = 'VDE'
     UNION SELECT * FROM flight) NOT IN link,
     SELECT 'RES','SPC',1.5 IN boat
  SELECT link.ori, link.des FROM link UNION
  SELECT link.ori, reachable.des FROM link, reachable
   WHERE link.des = reachable.ori;
```

4.1 The Semantics of a Definition of Views

A view name identifies a query, so the meaning of a definition of views `vd` sets the correspondence between every view name in `vd` and the interpretation of the corresponding query. But this interpretation must consider the original database definition extended with the views defined in `vd` as new relations. As we will show, stratification must be extended to assign a stratum to every view name. Next, these ideas are formalized. First we consider the definition of a simple view, then we will generalize it to the definition of a sequence of views.

Definition 6. *Let* `V sch := sel_stm` *be the definition of a non hypothetical view for* `db`*. The* meaning of `V` w.r.t. `db`*, denoted by* $[\![\mathtt{V}]\!]_{\mathtt{db}}$*, is equal to* $[\![\mathtt{sel\_stm}]\!]_{\mathtt{db}'}$*, where* $\mathtt{db}'$ *is the result of extending* `db` *with* `V sch := sel_stm` *as a new relation.*

Let `HV sch := sel_hyp` *be the definition of a hypothetical view for* `db`*. The* meaning of `HV` w.r.t. `db`*, denoted by* $[\![\mathtt{HV}]\!]_{\mathtt{db}}$*, is equal to* $[\![\mathtt{sel\_hyp}]\!]_{\mathtt{db}'}$*, where* $\mathtt{db}'$ *is the result of extending* `db` *with* `HV sch :=` $sel(\mathtt{sel\_hyp})$ *as a new relation.*

In V sch := sel_stm, the value $[\![\texttt{V}]\!]_{\texttt{db}} = [\![\texttt{sel\_stm}]\!]_{\texttt{db}'} = [\![\texttt{sel\_stm}]\!]^{fix^{\texttt{db}'}}$ depends on the fixpoint of a new database definition which should be stratifiable. db′ is equal to db extended with V sch := sel_stm, it will be non stratifiable if V appears in an EXCEPT clause inside sel_stm, but in the other case, the fixpoint of the new database will be equal to the one of db, except for the new relation V. Notice that $\texttt{RN}_{\texttt{db}'} = \texttt{RN}_{\texttt{db}} \cup \{\texttt{V}\}$, and no relation defined in db may depend on V. So, if k is the maximum stratum of db (with n relations), then a stratification str' for db′ can be defined as $str' : \texttt{RN}_{\texttt{db}'} \to \{1, \ldots, n+1\}$, with $str'(\texttt{R}) = str(\texttt{R})$ for every $\texttt{R} \in \texttt{RN}_{\texttt{db}}$, and $str'(\texttt{V}) = k+1$. Hence, for $i = 1..k$, $fix_i^{\texttt{db}'} = fix_i^{\texttt{db}}$. Therefore: $fix^{\texttt{db}'} = fix_{k+1}^{\texttt{db}'} = \bigsqcup_{m \geq 0}(T_{k+1}^{\texttt{db}'})^m(fix^{\texttt{db}})$, so $fix^{\texttt{db}'}$ is an extension of the known $fix^{\texttt{db}}$, and only the last stratum $k+1$ for the relation V (the only one in this stratum) must be calculated to find $[\![\texttt{sel\_stm}]\!]^{fix^{\texttt{db}'}} = fix_{k+1}^{\texttt{db}'}(\texttt{V})$.

The semantics for the case HV sch := ASSUME sel_stm′ [NOT] IN R sel_stm requires to modify the original database in two ways:

1. $[\![\texttt{HV}]\!]_{\texttt{db}} = [\![\texttt{sel\_hyp}]\!]_{\texttt{db}'}$, according to Definition 6, where db′ results from extending db with HV sch := sel_stm.
2. $[\![\texttt{sel\_hyp}]\!]_{\texttt{db}'} = [\![\texttt{sel\_stm}]\!]^{fix^{\texttt{db}''}}$, in accordance with Definition 5, where $\texttt{db}'' = \texttt{db}'[\texttt{R sch := sel\_stm}_{\texttt{R}}$ UNION | EXCEPT $\texttt{sel\_stm}'/\texttt{R sch := sel\_stm}_{\texttt{R}}]$.

In 1, the original database is extended with a new relation, HV. Notice that, considering HV as a relation identifier, the added definition, HV sch := sel_stm, is syntactically correct (however HV sch := sel_hyp is not allowed as a relation definition). In 2, the assumption is incorporated to the corresponding relation, as explained in Section 3.1. So, the new relation definitions in db″ are:

HV sch := sel(sel_hyp); R sch := $\texttt{sel\_stm}_{\texttt{R}}$ UNION | EXCEPT sel_stm′;

Then, the dependency graph $DG_{\texttt{db}''}$ can be built from $DG_{\texttt{db}}$ adding new edges to the relation R, as explained before. But there is also a new node HV an new edges: For every $\texttt{R}' \in \texttt{RN}_{sel(\texttt{sel\_hyp})}$, an edge from R′ to HV, that is negatively labelled if $\texttt{R}' \in \texttt{RN}^{\neg}_{sel(\texttt{sel\_hyp})}$.

As for the non hypothetical case, a stratification of db″, $str' : \texttt{RN}_{\texttt{db}''} \to \{1, \ldots, n+1\}$, if it exists, may assign the stratum $k+1$ to HV. $fix_k^{\texttt{db}''}$ can be computed as explained in Section 3.1 for hypothetical queries, and the computation of $fix_{k+1}^{\texttt{db}''}$ will consider only HV, and $fix_{k+1}^{\texttt{db}''}(\texttt{HV}) = [\![\texttt{sel\_stm}]\!]^{fix^{\texttt{db}''}}$.

Example 6. *Consider the database* db *of Example 3 and the hypothetical view:*

```
HV A int := ASSUME
     SELECT R1.A FROM R1 WHERE R1.A < 3 IN R2,  SELECT 3 NOT IN R2
  SELECT R3.A FROM R3 UNION SELECT HV.A*3 FROM HV WHERE HV.A < 3;
```

Following Definition 6, $[\![\texttt{HV}]\!]_{\texttt{db}} = [\![$SELECT R3.A FROM R3 UNION SELECT HV.A*3 FROM HV WHERE HV.A<3$]\!]^{fix^{\texttt{db}''}}$*, where* db″ *is an extension of the database* db′ *of Example 3 with:*

```
HV A int := SELECT R3.A FROM R3 UNION
            SELECT HV.A*3 FROM HV WHERE HV.A < 3;
```

A function str' extending str in such a way that $str'(\texttt{HV}) = 4$ is a stratification of the new database. For $1 \leq i \leq 3$, $fix_i^{\texttt{db}''} = fix_i^{\texttt{db}'}$, which appear in Example 4. Then, since $[\![\texttt{HV}]\!]_{\texttt{db}}$ coincides with $fix^{\texttt{db}''}(\texttt{HV})$, it is only necessary to calculate $fix_4^{\texttt{db}''}(\texttt{HV}) = (\bigsqcup_{m \geq 0}(T_4^{\texttt{db}''})^m(fix_3^{\texttt{db}''}))(\texttt{HV}) = \{(1), (2), (3), (4), (5), (6), (8)\}$.

Next we deal with the case of simultaneous view definitions for a database db. The idea is that the semantics of vd associates to every view name in vd, the interpretation of the query

that defines the view. But, if there is more than one non hypothetical view definition in vd, it is not valid to identify $[\![\mathtt{V}]\!]_{\mathtt{db}}$ with $[\![\mathtt{sel\_stm}]\!]_{\mathtt{db}'}$, being db′ the result of extending db with `V sch := sel_stm`. This is because other names defined in vd distinct of V can occur inside sel_stm, while they are not defined in db′. Then the semantics of vd is defined as follows:

Definition 7. *Let* db *be a database and let*

$\mathtt{vd} ::= \mathtt{V}_1\ \mathtt{sch}_1\ :=\ \mathtt{sel\_stm}_1\,;\ \ldots\ ;\ \mathtt{V}_m\ \mathtt{sch}_m\ :=\ \mathtt{sel\_stm}_m\,;$
$\mathtt{HV}_1\ \mathtt{sch}_1\ :=\ \mathtt{sel\_hyp}_1\,;\ \ldots\ ;\ \mathtt{HV}_r\ \mathtt{sch}_r\ :=\ \mathtt{sel\_hyp}_r\,;$

be a definition of views for db*. The semantics of* vd *is defined as the mapping that associates* $\mathtt{V}_j$ *to* $[\![\mathtt{V}_j]\!]_{\mathtt{db}'}$*, for* $j = 1..m$*, and* $\mathtt{HV}_j$ *to* $[\![\mathtt{HV}_j]\!]_{\mathtt{db}'}$*, for* $j = 1..r$*, where* db′ *is the result of extending* db *with:*

$\mathtt{V}_1\ \mathtt{sch}_1\ :=\ \mathtt{sel\_stm}_1\,;\ \ldots\ ;\ \mathtt{V}_m\ \mathtt{sch}_m\ :=\ \mathtt{sel\_stm}_m\,;$

Notice that, according to Definition 6, $[\![\mathtt{V}_j]\!]_{\mathtt{db}'} = [\![\mathtt{sel\_stm}_j]\!]_{\mathtt{db}''}$ for every $j = 1..m$, where db″ is the result of extending db′ with $\mathtt{V}_j\mathtt{sch}_j := \mathtt{sel\_stm}_j$, but this definition is already in db′, so db″ = db′. Since, $\mathtt{HV}_1, \ldots, \mathtt{HV}_r$ do not appear in $\mathtt{sel\_stm}_j$, their definitions are not required in db′. But for every $1 \leq j \leq r$, $[\![\mathtt{HV}_j]\!]_{\mathtt{db}'} = [\![\mathtt{sel\_hyp}_j]\!]_{\mathtt{db}''}$, where db″ is the result of extending db′ with $\mathtt{HV}_j\ \mathtt{sch}_j\ :=\ sel(\mathtt{sel\_hyp}_j)$, allowing $\mathtt{HV}_j$ to be recursive.

In order to compute the answer of every view included in a simultaneous definition, hypothetical views can be relegated to process the others. As in the simple case, db′ must be stratifiable. In the practice, if db′ is stratifiable, a stratification str' for db′, such that $str'(\mathtt{V}_j) > n$ for every $1 \leq j \leq m$ can be found. Then the interpretation $fix^{\mathtt{db}'}$ can be obtained stratum by stratum, starting from $fix^{\mathtt{db}}$, as in the simple case. Now, every hypothetical view can be treated separately, starting each time with $fix^{\mathtt{db}'}$ as initial interpretation, and processing each view as in the simple case.

5 The HR-SQL System

We present a SWI-Prolog implementation for the HR-SQL language adapted for IBM DB2. The system, with a bundle of examples, is available at `https://gpd.sip.ucm.es/trac/gpd/wiki/GpdSystems/HR-SQL`. The structure of the system is depicted in Figure 5. The interface consists of a prompt '`hr-db2 =>`' which works as an extension of the command interpreter of DB2. The user can submit any DB2 input to manage an existing database (label A in Figure 5), and also the following ones provided by HR-SQL (label B in Figure 5):

- `load_db` <*db_file*> loads an HR-SQL database definition from a file and computes the corresponding fixpoint. The resulting tuples for the relations are stored as DB2 tables.
- `load_vd` <*vd_file*> loads an HR-SQL definition of views from a file, computes the values for each view, and materializes them as DB2 tables.
- A hypothetical query written in HR-SQL syntax (`sel_hyp`), which is submitted to the system and recognized as such because it starts with ASSUME.

These new statements are preprocessed by the SWI-Prolog module as shown in Figure 5. After parsing, the dependency graph is built, a stratification is generated (if it exists; an error is thrown otherwise). The current algorithm to compute the stratification tries to minimize the number of relations in each strata. This allows to improve the efficiency of the fixpoint computation w.r.t. [2], because now each stratum i contains only those mutually recursive relations, avoiding to process the rest of them in each iteration of the fixpoint operator at stratum i. After the stratification, an SQL PL script is produced as will be explained in

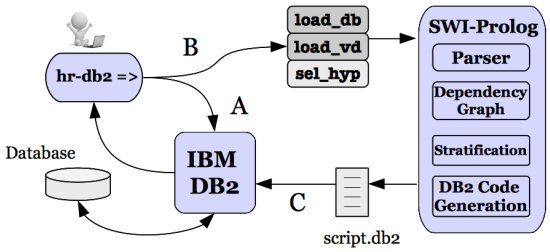

Figure 2: The HR-SQL System.

Section 5.1. This output is executed by DB2 (label C in Figure 5). The code generation for hypothetical views needs an additional process which is shown in Section 5.2.

5.1 Computing the Fixpoint

Figure 3 shows the algorithm for generating the DB2 database corresponding to the fixpoint of an HR-SQL database definition. It produces the SQL statements (CREATE and INSERT) needed to build such a database. This version enhances the one in [2] with the functions *in* and *out* which will be explained later.

```
1   for all R ∈ RN_db do CREATE TABLE R sch;
2   i := 1
3   while i ≤ numStr do
4     for all R ∈ RN_i do INSERT INTO R out(sel_stm_R);
5     repeat
6       size := rel_size(RN_i)
7       for all R ∈ RN_i do
8         INSERT INTO R in(sel_stm_R) EXCEPT SELECT * FROM R;
9     until size = rel_size(RN_i)
10    i := i + 1
```

Figure 3: Algorithm to Compute the Fixpoint

The algorithm considers a concrete stratification for the database where *numStr* denotes the number of strata and $\mathtt{NR}_i$ the set of relations of stratum i. First of all, a table is created for each relation $\mathtt{R\ sch\ :=\ sel\_stm_R}$ of the database (line 1). Then, the external *while* (lines 3-10) computes successively the fixpoints $fix_1^{\mathtt{db}}, fix_2^{\mathtt{db}}, \ldots, fix_{numStr}^{\mathtt{db}}$. According to the theory, each $fix_i^{\mathtt{db}}$ is calculated for every relation of $\mathtt{NR}_i$, by iterating the operators $T_i^{\mathtt{db}}$ of Definition 3, i.e., the *repeat* (lines 5-9) at iteration n computes $(T_i^{\mathtt{db}})^n(fix_{i-1})$. The loop is iterated while some tuple is added to the tables of the current stratum; the variable *size* is used to check if some tuple is added to some relation of the current stratum.

This algorithm improves the efficiency of the introduced in [2] by reducing the work in the iterations of the *repeat* with the functions *in* and *out*. The idea is that the iteration of the operator $T_i^{\mathtt{db}}$ is only needed for recursive relations; in fact, only for the recursive fragment of the select statements defining those relations. The functions *in* and *out* split each `sel_stm` into the (recursive) fragment that must be used in the INSERT statements inside the loop (line 8), and the fragment that can be processed before the loop, as the base case of the recursive definition (line 4). The *in* and *out* fragments of a `sel_stm` can be easily determined using the

stratum of its components because, as mentioned before, the stratification is such that if a relation R in stratum i depends on another relation R′, then the stratum of R′ is lower than i, so it must be previously computed, or it is exactly i (if they are mutually recursive) and both relations must be computed simultaneously. Therefore, if for instance R := $\texttt{sel\_stm}_1$ UNION $\texttt{sel\_stm}_2$, $str(\texttt{R}) = i$, and $str(\texttt{sel\_stm}_1) < i$, then $\texttt{sel\_stm}_1$ will be part of the *out* fragment, and the corresponding tuples can be inserted before the loop, because the involved relations are already computed in the computation of a previous stratum. Functions *in* and *out* are defined by recursion on the structure of sel_stm. For example, if $\texttt{sel\_stm} \equiv \texttt{ss}_1$ EXCEPT $\texttt{ss}_2$, and $str(\texttt{sel\_stm}) = i$, then $str(\texttt{ss}_2) < i$, so:

$in(\texttt{sel\_stm}) = in(\texttt{ss}_1)$ EXCEPT $\texttt{ss}_2$; $\quad out(\texttt{sel\_stm}) = out(\texttt{ss}_1)$ EXCEPT $\texttt{ss}_2$.

5.2 Computing Hypothetical Views

The SQL PL script generated to process views follows the ideas of Section 4.1. We use the view reachable of Example 5 to illustrate the system steps to solve a hypothetical view definition. It is interesting as it is a recursive definition containing positive and negative assumptions.

First of all, the system extends the original dependency graph with the new edges due to hypothetical assumptions: two negatively labeled edges to link, one from bus, and another from flight. Due to the expanded form of stratification we have defined, the stratification for the original database is also a stratification for the new one. Following the explanations of Section 3.1, the system looks for those relations that must be recomputed to obtain the tuples of the view reachable, in this case only boat and link. The algorithm that generates the SQL statements, for computing these relations and the new view, is quite similar to that presented in Figure 3 to compute the fixpoint of a database. Next we explain the differences following the example.

The relations needed to compute the view are locally created and recomputed using temporary tables, and the computation will start at stratum $i = min\{str(\texttt{boat}), str(\texttt{link})\}$:

```
DECLARE GLOBAL TEMPORARY TABLE link AS link;
DECLARE GLOBAL TEMPORARY TABLE boat LIKE boat;
INSERT INTO SESSION.boat
 ((SELECT 'TFS','GMZ',1) UNION (SELECT 'GMZ','VDE',1.5) UNION
  (SELECT 'SPC','TFN',2 l) UNION (SELECT 'RES','SPC',1.5));
INSERT INTO SESSION.link
 (SELECT * FROM flight UNION SELECT * FROM SESSION.boat UNION
  SELECT * FROM bus EXCEPT (SELECT * FROM bus WHERE bus.ori = 'VDE')
  UNION SELECT * FROM flight);
```

Temporary tables are prefixed with SESSION. For processing a hypothetical view of the form $\texttt{HV} := \texttt{sel\_hyp}_{\texttt{HV}}$, the script to compute the tuples of HV will consider the definition HV := sel_stm, where sel_stm results from replacing R by SESSION.R in $sel(\texttt{sel\_hyp}_{\texttt{HV}})$. The tuples for reachable are materialized and stored, then the temporary tables are discarded. Temporary tables are adequate as they are in-memory data structures.

The computation of hypothetical queries follows the same steps, but instead of creating tables, a cursor is used to obtain the answer without materializing it.

6 Conclusions and Future Work

We have designed a practical, formally-supported SQL system, porting some techniques from the deductive database field to the relational one. Thus, we provide an original way to give semantics to SQL languages supporting recursion. In addition our system allows both less-limited

recursion (w.r.t. current SQL systems) and hypothetical reasoning (as a novel addition to such systems), acting as a front-end to DB2. Although targeted to this system, our work can be straightforwardly applied to any other SQL system. However, it can be improved in a number of ways: With respect to recursion, in-memory indexing can be applied for small search keys. These keys can be identified as the candidate keys derived from explicit functional dependencies (as already allowed in DB2) and primary keys. Also, both general and particular optimization methods can be applied to our work. For the first, the differential semi-naïve algorithm [15] allows to save tuples in recursive joins along fixpoint iterations. For the second sort of method, already-known linear recursion optimizations [11] can also be applied by analyzing the dependency graph and easily detecting such cases. With respect to hypothetical queries and views, we plan to extend the definition language, allowing mutual recursion in hypothetical views. Finally, we can extend this work by allowing not only materialized views, but also regular views. For this, table functions (cf. IBM DB2 concepts) can be used as a natural construction to build HR-SQL query results on-the-fly.

References

[1] G. Aranda-López, S. Nieva, F. Sáenz-Pérez, and J. Sánchez-Hernández. An extended constraint deductive database: Theory and implementation. *The Journal of Logic and Algebraic Programming*, 2013.

[2] G. Aranda-López, S. Nieva, F. Sáenz-Pérez, and J. Sánchez-Hernández. Formalizing a Broader Recursion Coverage in SQL. In *Symposium on Practical Aspects of Declarative Languages (PADL'13)*, volume 7752 of *LNCS*, 2013. In Press.

[3] A. J. Bonner. Hypothetical datalog: Negation and linear recursion. In *The ACM Symposium on the Principles of Database Systems (PODS)*, pages 286–300, 1989.

[4] A. J. Bonner and M. Kifer. Transaction logic programming. In *ICLP*, pages 257–279, 1993.

[5] A. J. Bonner and L. T. McCarty. Adding negation-as-failure to intuitionistic logic programming. In E. L. Lusk and R. A. Overbeek, editors, *Logic Programming, Proc. of the North American Conference*, pages 681–703. The MIT Press, 1989.

[6] H. Christiansen and T. Andreasen. A Practical Approach to Hypothetical Database Queries. In *Transactions and Change in Logic Databases*, volume 1472 of *LNCS*, pages 340–355. Springer, 1998.

[7] E. Codd. A Relational Model for Large Shared Databanks. *Communications of the ACM*, 13(6):377–390, June 1970.

[8] C. J. Date. *SQL and relational theory: how to write accurate SQL code*. O'Reilly, Sebastopol, CA, 2009.

[9] T. Griffin and R. Hull. A framework for implementing hypothetical queries. In *SIGMOD Conference*, pages 231–242, 1997.

[10] W. H. Inmon. *Building the data warehouse*. QED Information Sciences, Inc., Wellesley, MA, USA, 2005.

[11] C. Ordonez. Optimization of Linear Recursive Queries in SQL. *IEEE Transactions on Knowledge and Data Engineering*, 22(2):264–277, 2010.

[12] F. Sáenz-Pérez. Datalog Educational System, October 2011. `http://des.sourceforge.net/`.

[13] M. Stonebraker and K. Keller. Embedding expert knowledge and hypothetical data bases into a data base system. In *The 1980 ACM SIGMOD International Conference on Management of Data*, SIGMOD '80, pages 58–66. ACM, 1980.

[14] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[15] J. Ullman. *Principles of Database and Knowledge-Base Systems Vols. I (Classical Database Systems) and II (The New Technologies).* Computer Science Press, 1989.