DOI: 10.21276/sjet

Scholars Journal of Engineering and Technology (SJET) **ISSN 2321-435X (Online)**
Sch. J. Eng. Tech., 2017; 5(6):254-257 **ISSN 2347-9523 (Print)**

(An International Publisher for Academic and Scientific Resources)
www.saspublisher.com

Research Article

Join Operations in Relational Databases with Automatic Attributes Renaming

Poliakov S[1], Buy D[1], Mohmmed Karam[1], Israa Jasim AL.kalafa[2]
[1]Taras Shevchenko National University of Kyiv, Kyiv, Ukraine
[2]Ministry of Science and technology, Ukraine

***Corresponding author**
Poliakov S
Email: sergey.a.polyakov@gmail.com

Abstract: The paper considers inner and outer join of relation operations, as well as set-theoretical operations on relations. Unlike join operations in Codd's relational algebra the join operations under consideration are based on Kleene and Priest three-valued logic used in SQL, that allows to give the semantics of join operations more precisely. In relational algebra, any relation that does not contain tuples is represented as an empty set. It is impossible to define outer joins for such relations provided we do not know what columns contain an empty relation. In the paper relations with schemas are introduced. That allows the operations to be defined even if one or both operands are empty relations unlike classical relational algebra.
Keywords: relational algebra, relational databases, program theory, programming language semantics, SQL, table join operations.

INTRODUCTION

Join operations play an important role in relational algebra as they allow to restore complex objects, joining many tables. Initially Codd introduced set-theoretic operations on relation and relation join operation in his article [1]. Then relational algebra was extended with operations of outer join. Now there exists a big set of join operations. See, for example, [2-4].

In [5, 6] another approach to SQL algebra was presented. In these works programming algebra of higher-order functions called operators in other programming languages was built for SQL. We describe the improved version of join operations of this program algebra. To avoid column name collision the join operators that are introduced in this article make renaming automatically. The relation definition was extended to schemes as well.

BASIC OPERATIONS ON RELATIONS

Let us clarify relations in terms of nominative sets. Fix two following sets: *Atr*, whose elements are called attribute names, and *Dom* - a universal domain of primitive data. We suppose that the special element *null* belongs to universal domain and is called an undefined value. Thus, we call an arbitrary finite attribute set $S \subseteq Atr$ a scheme. The tuple of scheme *S* is called a set of pairs (a, d) where $a \in Atr$, $d \in Dom$ (i.e. a set of pairs (attribute, value)), whose projection by the first component is equal to *S*. Let's denote operation of tuple projection of t by first component as $pr_1(t)$ and scheme of tuple t as $S(t)$. For given tuple t it is easy to calculate its scheme making projection by the first component $S(t) = pr_1(t)$. Tuples can also be defined as finite mapping from attribute names to universal domain.

The relation of the scheme S is called pair (r, S), where r is finite set of tuples of scheme *S* which is called *a relation state* or *state.* Note, that in the most cases a relation scheme can be obtained from schemes of its tuples. But for an empty state it is impossible. The set of all tuples (relations) of scheme S is denoted by $Tup(S)$ (respectively $Rel(S)$), and set of all tuples (relations) - *Tup* (respectively *Rel*). Thus $Tup(S) \stackrel{\text{def}}{=} \{t | pr_1(t) = S\}$, $Rel(S) \stackrel{\text{def}}{=} \{(r, S) | r \in 2^{Tup(S)}\}$, $Tup \stackrel{\text{def}}{=} \bigcup_{S \subseteq Atr} Tup(S)$, $Rel \stackrel{\text{def}}{=} \bigcup_{S \subseteq Atr} Rel(S)$

The scheme may be empty $\emptyset$. For such scheme there exists exactly one tuple denoted by ε and two states - $\{\varepsilon\}$ and $\emptyset$. The tuples will be denoted by $t, t_1, t_2, \ldots$, relations by $-R, R_1, R_2, \ldots$, relation's states by $r, r_1, r_2, \ldots$, and schemes by $S, S_1, S_2, \ldots$

Two relations R_1 and R_2 are supposed to be *consistent by scheme* if their schemes are equal. We will denote such relation by $R_1 \cong R_2$. Similarly,: $t_1 \cong t_2 \stackrel{def}{=} S(t_1) = S(t_2)$ is defined to be consistent by scheme tuples.

The binary relation of *tuple compatibility* $t_1 \approx t_2$ is defined as $\forall a(a \in S(t_1) \& a \in S(t_2) \Longrightarrow t_1(a) = t_2(a))$. In other words, two tuples are compatible if their attributes with the equal names have equal values. Tuples without equal attribute name are always compatible. The binary operation $\overline{\cup}: Tup \times Tup \longrightarrow Tup$ of the compatible tuple union is defined as follows: $t_1 \,\overline{\cup}\, t_2 = \{(a,d)|(a,d) \in t_1 \vee (a,d) \in t_2\}$. Its definition domain is $dom\,\overline{\cup} \stackrel{def}{=} \{(t_1,t_2)|t_1,t_2 \in Tup \vee t_1 \approx t_2\}$

Along with elementary attribute names we will consider complex names such as $a_1.a_2$. These names will be used to avoid name ambiguity in tuples. Let's introduce special function $a \Rightarrow: Tup \longrightarrow Tup$ that is parametrized by attribute name a. It is defined by formula $a \Rightarrow (t) = \{(a.a_i, d_i)|(a_i, d_i) \in t\}$. This function will be called the *prefixation*.

The operation of *total union* of tuples $\cup_{a_1,a_2}: Tup \times Tup \longrightarrow Tup$ is total function parametrized by attribute names a_1 and a_2. It is defined as follows: $t_1 \cup_{a_1,a_2} t_2 = \{t_1' \cup t_2' | t_1' = (a_1 \Rightarrow t_1) \& t_2' = (a_2 \Rightarrow t_2)\}$

For schemes the prefixation is defined as follows: $S_1 \cup_{a_1,a_2} S_2 = a_1 \Rightarrow (S_1) \cup a_2 \Rightarrow (S_2)$, where $a \Rightarrow (S) \stackrel{def}{=} \{a.a_i | a_i \in S\}$

We will write $t(a)$ to receive the value of a

We will say that a tuple belongs to a relation if it belongs to the relation' state $t \in R \stackrel{def}{=} \exists r \exists S(R = (r,S) \wedge t \in r)$.

SET-THEORETIC RELATION OPERATIONS

Let's R_1 be a relation (r_1, S_1) and R_2 be a relation (r_2, S_2) where $S_1 = S_2 = S$.
Union of relations $R_1 \Cup R_2 = (r_1 \cup r_2, S)$
Intersection of relations $R_1 \Cap R_2 = (r_1 \cap r_2, S)$
Difference of relations $R_1 \backslash\backslash R_2 = (r_1 \backslash r_2, S)$.

The domain of these operation definition is a set of consistent pairs of relations $\{(R_1,R_2)|R_1 \cong R_2\}$. The resulting relation state is calculated, respectively, as theoretical-set union, intersection and the difference of arguments states. Obviously, the scheme of resulting relation coincides with input relation schemes. We can say that these operations are restriction of the theoretical-set operation on pairs of consistent relations.

INNER JOIN OPERATIONS

We define *cross join* operation $\times_{a_1,a_2}$ as total binary function of a type $\times_{a_1,a_2}: Rel \times Rel \longrightarrow Rel$ parametrized by attribute names a_1, a_2, $a_1 \neq a_2$.

Let R_1 be relation (r_1, S_1) and R_2 be relation (r_2, S_2). Then, cross join is defined by formula $\times_{a_1,a_2}(R_1,R_2) = (r_3, S_1 \cup_{a_1,a_2} S_2)$, where $r_3 = \{t_1 \cup_{a_1,a_2} t_2 | t_1 \in r_1 \& t_2 \in r_2\}$

Natural join operation $\otimes_{a_1,a_2}$ or equijoin is total function of a type $\otimes_{a_1,a_2}: Rel \times Rel \to Rel$. Let R_1 be relation (r_1, S_1) and R_2 be relation (r_2, S_2). Then, natural join is defined by formula $\otimes_{a_1,a_2}(R_1,R_2) = (r_3, S_1 \cup_{a_1,a_2} S_2)$, where $r_3 = \{t_1 \cup_{a_1,a_2} t_2 | t_1 \in r_1 \& t_2 \in r_2 \& t_1 \approx t_2\}$,

Next operation is *join by attributes* $\otimes^B_{a_1,a_2}$ of a type $Rel \times Rel \to Rel$, where B is a set of attribute names and a_1, a_2 are names such that $a_1 \neq a_2$. This is partial parametric function with domain of definition $dom\ \otimes^B_{a_1,a_2} \stackrel{def}{=} \{(R_1,R_2)|B \subseteq S(R_1) \cap S(R_2)\}$

It is defined by formula $\otimes^B_{a_1,a_2}(R_1,R_2) = (r_3, S_1 \cup_{a_1,a_2} S_2)$, where $r_3 = \{t_1 \cup_{a_1,a_2} t_2 | t_1 \in r_1 \& t_2 \in r_2 \& t_1(B) = t_2(B)\}$

Let $p: Tup \to Bool$ be, as generally accepted, a partial binary predicate. Under *join on predicate* p we understand a partial operation $\otimes^p_{a_1,a_2}$ of a type $Rel \times Rel \to Rel$, where p is a predicate and a_1, a_2, such that $a_1 \neq a_2$ are attribute names. Let $dom\ p$ be domain of definition of a predicate p. The domain of definition of operation join on a predicate is $dom \otimes^p_{a_1,a_2} \stackrel{def}{=} \{R|R = (r,S)\ \&\ r \subseteq dom\ p\}$. The operation is defined by formula $\otimes^p_{a_1,a_2}(R_1,R_2) = (r_3, S_1 \cup_{a_1,a_2} S_2)$, where $r_3 = \{t | t = t_1 \cup_{a_1,a_2} t_2 \& t_1 \in r_1 \& t_2 \in r_2 \& p(t) = True\}$

Given above-mentioned join operation specification, we can see that cross-join, natural-join and equijoin are derived from join on predicate operation.

Let p be always true predicate $p \stackrel{def}{=} true$, then $\times_{a_1,a_2}(R_1,R_2) \equiv \otimes^{true}_{a_1,a_2}(R_1,R_2)$

Let A be set of attribute names $(a_1, a_2, \ldots, a_n)$ and c_1 and c_2, $c_1 \neq c_2$ be attribute prefixes. Then, predicate $p^A_{c_1,c_2}$ is defined as follows

$$p^A_{c_1,c_2}(t) = \begin{cases} true, & if\ t(c_1.a_i) = t(c_2.a_i), i = 1..n \\ false, & otherwise \end{cases}$$

For given relations $R_1 = (r_1, S_1)$, $R_2 = (r_2, S_2)$ and attribute prefixes c_1, c_2 let's designate

$E = S_1 \cap S_2$ and $p_\cap = p^E_{c_1,c_2}$. Then ,natural join is defined as $\otimes_{c_1,c_2}(R_1,R_2) = \otimes^{p_\cap}_{c_1,c_2}(R_1,R_2)$.

For given relations $R_1 = (r_1,S_1)$, and $R_2 = (r_2,S_2)$, attribute prefixes c_1, and c_2 let's denote $E \subseteq S_1 \cap S_2$ and $p_\subseteq = p^E_{c_1,c_2}$. Then, equijoin is defined as follows :$\otimes^{p}_{c_1,c_2}(R_1,R_2) = \otimes^{p_\subseteq}_{c_1,c_2}(R_1,R_2)$.

OUTER JOIN OPERATIONS

All outer join operations are subordinate to their inner join operations and are defined by the same logic scheme. We'll describe this scheme.

Let $\varphi_{a_1,a_2}: Rel \times Rel \nrightarrow Rel$ be a partial operation on a set of relations, the inclusion $\varphi_{a_1,a_2}(R_1,R_2) \subseteq R_1 \times_{a_1,a_2} R_2$ being performed for all $R_1,R_2 \in dom\ p$. Recall that the operations of inner join satisfy this inclusion.

Suppose $R_1 = (r_1,S_1)$ and $R_2 = (r_2,S_2)$ are relations from domain of definition of operation φ_{a_1,a_2}. Then, relation R_1 assumes the following breaking: $R_1 = (R_1 \cap^{\varphi}_{a_1,a_2} R_2) \Cup (R_1 -^{\varphi}_{a_1,a_2} R_2)$. The intersection $R_1 \cap^{\varphi}_{a,a} R_2 = (r',S')$ is defined as follows: $r' = \{t_1 | t_1 \in r_1 \& \exists t_2 (t_2 \in r_2 \& (t_1 \cup_{a_1,a_2} t_2) \in \varphi_{a_1,a_2}(R_1,R_2))\}$, and $S' = S_1$. The difference $R_1 -^{\varphi}_{a_1,a_1} R_2 = (r'',S'')$ is defined as follows: $r'' = \{t_1 | t_1 \in r_1 \& \forall t_2 (t_2 \in r_2 \Rightarrow (t_1 \cup_{a_1,a_2} t_2) \notin \varphi_{a_1,a_2}(R_1,R_2))\}$

Without the loss of generality it can be assumed that the tuples from a relation $R_1 \cap^{\varphi}_{a_1,a_2} R_2$ are used in forming join result, there being no continuation in join result, and the tuples from relation $R_1 -^{\varphi}_{a_1,a_1} R_2$ are not used, there being no continuation in join results. The operations of left, right and full outer join are just designed to consider relations-arguments tuples, that were not included in the results of input outer join. But here the question arises how to expand the tuples to upperschemes, To do this, SQL uses a special element of universal domain $null$.

Below by t^{null}_S we'll define the following constant tuple $t^{null}_S: S \to \{null\}$. Given directly outer join operations are induced by an inner join operation φ_{a_1,a_2}. This will require the following natural joins, where $R_1 = (r_1,S_1)$ and $R_2 = (r_2,S_2)$: $(R_1 -^{\varphi}_{a_1,a_2} R_2) \times_{a_1,a_2} (\{t^{null}_{S_2}\}, S_2)$, $(\{t^{null}_{S_1}\}, S_1) \times_{a_1,a_2} (R_2 -^{\varphi}_{a_1,a_2} R_1)$. Resulting relations states are calculated by expressions:

$\{t_1 \cup_{a_1,a_2} t^{null}_{S_2} | t_1 \in R_1 -^{\varphi}_{a_1,a_2} R_2\}$, and $\{t^{null}_{S_1} \cup_{a_1,a_2} t_2 | t_2 \in R_2 -^{\varphi}_{a_1,a_2} R_1\}$

Results schemes are obtained by input relation schemes union $S_1 \cup_{a_1,a_2} S_2$.

Under *outer left join* induced by operation φ, we understand the operation of a type $\varphi_l: Rel \times Rel \nrightarrow Rel$, where $dom\ \varphi_l \stackrel{def}{=} dom\ \varphi_{a_1,a_2}$, that is calculated by a formula $\varphi_l(R_1,R_2) = \varphi_{a_1,a_2}(R_1,R_2) \Cup (R_1 -^{\varphi}_{a_1,a_2} R_2) \times_{a_1,a_2} (\{t^{null}_{S_2}\}, S_2)$.

Under *outer right join* we consider the operation of a type $\varphi_r: Rel \times Rel \nrightarrow Rel$, where $dom\ \varphi_r \stackrel{def}{=} dom\ \varphi_{a_1,a_2}$, that is calculated by a formula $\varphi_r(R_1,R_2) =$ $\varphi_{a_1,a_2}(R_1,R_2) \Cup (\{t^{null}_{S_1}\}, S_1) \times_{a_1,a_2} (R_2 -^{\varphi}_{a_1,a_2} R_1)$

Under *outer full join* we consider the operation of a type $\varphi_f: Rel \times Rel \nrightarrow Rel$, where $dom\ \varphi_f \stackrel{def}{=} dom\ \varphi_{a_1,a_2}$, that is calculated by a formula $\varphi_f(R_1,R_2) =$ $\varphi_{a_1,a_2}(R_1,R_2) \Cup (R_1 -^{\varphi}_{a_1,a_2} R_2) \times_{a_1,a_2} (\{t^{null}_{S_2}\}, S_2) \Cup (\{t^{null}_{S_1}\}, S_1) \times_{a_1,a_2} (R_2 -^{\varphi}_{a_1,a_2} R_1)$.

Under *union join* we consider the operation of a type $\varphi_f: Rel \times Rel \nrightarrow Rel$, where $dom\ \varphi_f \stackrel{def}{=} dom\ \varphi_{a_1,a_2}$, that is calculated by a formula $\varphi_f(R_1,R_2) = (R_1 -^{\varphi}_{a_1,a_2} R_2) \times_{a_1,a_2} (\{t^{null}_{S_2}\}, S_2) \Cup (\{t^{null}_{S_1}\}, S_1) \times_{a_1,a_2} (R_2 -^{\varphi}_{a_1,a_2} R_1)$.

Resulting relation schemes are still being built by input relation schemes union.

Thus, the left join is designed to consider the first(left) relation tuples that do not participate in inner join, the right join – to consider the tuples of the second right relation that do not participate in inner join, full join - to consider for both arguments not being involved in inner join. Finally, union join, unlike previous operations do not replenish the result of inner union and only builds appropriately extended tuples of both arguments that are not involved in union join.

JOIN OPERATION STRUCTURE

Let us return to the inner join operation SQL. For cross join there is no sense in introducing outer join because the left, right and full joins coincide with it, and union join will always be an empty table (because all table-arguments strings are involved in forming the results of cross join,. Moreover, in modern SQL, including the latest versions, outer union join is supported only for natural join. Family operation structure of SQL-like languages is given.

Note, that a preliminary design should be modified to consider the special element *null* role while determining compatible tuples: the tuples are

compatible if common attributes have the same values in tuples, the values differing from *null*.

The same objective can be achieved by considering the following clarification in three-valued logic.

$$d_1 =_t d_2 \stackrel{\text{def}}{=} \begin{cases} true, \ if\ d_1 = d_2, d_1 \neq null, d_2 \neq null \\ false\ if\ d_1 \neq d_2, d_1 \neq null, d_2 \neq null \\ unknown\ if\ d_1 = null\ \vee d_2 = null \end{cases}$$

Where all $d_1, d_2 \in D$; here *unknown* – an undefined Boolean value. Then, *the compatibility relation* that already meets the three-valued predicate is modified as follows: $t_1 \approx t_2 \stackrel{\text{def}}{=} \&_{a \in S_1 \cap S_2} t_1(a) =_t t_2(a)$, where S_i is a scheme of t_i, $i = 1,2$. Note, that in SQL-like languages the equality is specified as a predicate $=_t$. In addition, a given definition is the manifestation of the overall situation in extending predicates on *null*- values: predicates keep *null* values, i.e. if at least one argument is equal to *null*, the result is Boolean value *unknown* .

In the same way, functions in extending on *null*-value keep it: if, at least, one argument coincides with *null* then the result is *null*. Thus, the situation is similar to natural extension of partial functions but in SQL extended functions are considered as the partial (i.e, division).

Table 1: Operation join structure

Inner join	**Join operation predicate by predicate**	**Outer join**			
		left	**right**	**full**	**union**
Cross join	$true$	–	–	–	–
Natural join	$p_{\cap}(t)$	+	+	+	+
Join by attribute	$p_{\subseteq}(t)$	+	+	+	–
Join on predicate	$p(t)$	+	+	+	–

CONCLUSION

The join operations can be considered as a special type of Cartesian product on the case when arguments are set of tuples called relations. We want the result of the operation to be a relation too. To achieve this, an operation of join must concatenate two tuples into new tuple. It is easy to do when names of tuple attributes are not intersected. Otherwise we will have the ambiguity of names. To avoid this the join operations introduced in the paper make attributes renaming automatically.

REFERENCES

1. Codd EF. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM. 1970; 13 (6): 377–387.
2. Maier D. The Theory of Relational Databases. Computer Press, Rockville, Md., 1983.
3. Date CJ. SQL and Relational Theory. O'Reilly Media, Inc.,, 3rd edition, 2015.
4. Garcia-Molina H, Ullman JD, Widom J. Database Systems. Pearson Education Inc., 2nd edition, 2009.
5. Brona J, Buy D, Zagorsky S, Poliakov S. Compositional Semantisc of SQL. Proc. of the Fourth International Scientific Conference. Electronic Computers and Informations'2000. Kosice. 2000; 287-292.
6. Редько ВН, Брона ЮЙ, Буй ДБ, Поляков СА. Реляційні бази даних: табличні алгебри та SQL-подібні мови. Київ: Видавничий дім "Академперіодика"; 2001.