

Lesson-4

Relational Algebra-Part II

Hi, I would like to extend the discussion on relational algebra and joins.

In general, a **join** is an operation that glues relations together. There are several kinds of joins. The join operation is very important for any relational database with more than a single table as it allows us to process more than one table at a time. A join Operation matches data from 2 or more tables, based on the values of one or more columns in each table.

The **join** is the method whereby two tables are combined so data from two or more tables can be used to extract information. In Relational Algebra, Codd defined the idea of a **natural join**. The following description describes a natural join

To do a natural join of two relations, you examine the relations for common attributes (columns with the same name and domain). For example, look at the following abstract tables: R (A, B, C, D) and Q (B, E, F)

Q : Table			
	B	E	F
	b1	e2	f3
	b2	e7	f2
	b3	e2	f3
	b4	e1	f2
▶			

R : Table				
	A	B	C	D
	a1	b2	c1	d1
	a2	b1	c3	d2
	a3	b3	c3	d2
	a4	b3	c1	d3
	a5	b2	c2	d4
▶				

Notice that relations R and Q both have attribute B with the same domain. Note also that there are no other attributes common to both. So ... to do a **natural join** of R and Q, you match rows from R and rows of Q with the same B value.

Next, you must know that the result of a join is another table that contains the same attributes (columns) as both of the tables being joined. The attributes in Q are B, E, and

F. The attributes in R are A, B, C, and D. The attributes in the join of Q and R are B, E, F, A, C, D. Notice that B only appears once in this list. For example, let us calculate the join of Q and R and call the resulting table T.

Join Q and R giving T:

Starting with the Q table (B is the primary key in the Q table), choose the first tuple (row). What tuple in R matches the tuple (b1, e2, f3) in Q? Well, it appears that only one tuple in Q fills the qualification that B = 'b1'. This tuple is (a2, b1, c3, d2). This result is the row in the joined table...Notice how the rows from the two tables make up one row in the result and that the B value only occurs once.

QJR : Table						
	B	E	F	A	C	D
	b1	e2	f3	a2	c3	d2
*						

The tuple (a2, b1, c3, d2) is the only tuple in R that matches the tuple (b1, e2, f3) in Q. Let us try the next tuple in Q:

Q : Table			
	B	E	F
	b1	e2	f3
	b2	e7	f2
	b3	e2	f3
	b4	e1	f2

R : Table				
	A	B	C	D
	a1	b2	c1	d1
	a2	b1	c3	d2
	a3	b3	c3	d2
	a4	b3	c1	d3
	a5	b2	c2	d4

This is (b2, e7, f2). It matches every tuple in R with B = 'b2'. These are (a1, b2, c1, d1) and (a5, b2, c2, d4). Here you create two rows in the join.

QJR : Table						
	B	E	F	A	C	D
	b1	e2	f3	a2	c3	d2
	b2	e7	f2	a1	c1	d1
	b2	e7	f2	a5	c2	d4

The next tuple in Q is (b3, e2, f3). There are two matching tuples in R where B = 'b3'. They are (a3, b3, c3, d2) and (a4, b3, c1, d4). So, we create two more rows in the join:

QJR : Table						
	B	E	F	A	C	D
	b1	e2	f3	a2	c3	d2
	b2	e7	f2	a1	c1	d1
	b2	e7	f2	a5	c2	d4
	b3	e2	f3	a3	c3	d2
	b3	e2	f3	a4	c1	d4
*						

The next and last tuple in Q is (b4, e1, f2). Note that there no tuples in R that match this tuple in Q with B = 'b4'. This means that no tuples are added to the natural join (to the **natural join** – in some of the other types of join, this is not true). So, the QJR table above is the final table in the join.

These three commands allow us to perform powerful and useful queries of databases. They are generally used together to obtain the results. Consider the following set of tables:

Sponsor : Table			Team : Table			
	SponsorId	SponsorName		TeamId	TeamName	SponsorId
►	S1	First National Bank	►	T1	Chargers	S1
	S2	House of Comets		T2	Streakers	S3
	S3	Swimmers Paradise		T3	Predators	S2
	S4	Dime Store		T4	Strike	S3
				T5	Extreme	S2
*			*			

Player : Table					
	PlayerId	TeamId	PlayerName	PlayerPhone	PlayerParent
►	P1	T1	Jill	444-2323	Sally
	P2	T1	Cathy	333-5555	Joan
	P3	T2	Samantha	222-4565	Susan
	P4	T2	Shelly	321-4321	Sarah
	P5	T4	Karen	432-5432	Liz
	P6	T1	Joyce	543-5432	Martha
	P7	T3	Sandy	654-3456	Jan
	P8	T3	Lois	989-8998	Carol
*					

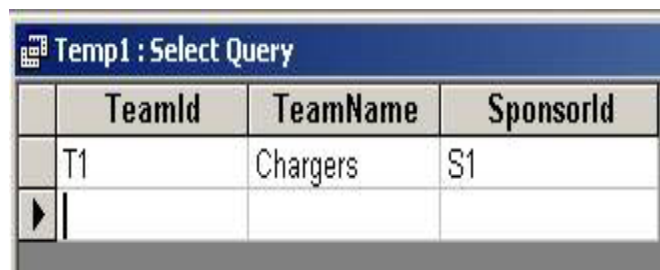
Let us consider the following query:

What are the names of the players that play for the Chargers?

What can be done to determine all the players on the Chargers? What would have to be done? First we would have to determine the TeamID of the Chargers. That would be 'T1'. Next, we would look down through the player table printing out all players with TeamId='T1'.

Now, think in terms of **select**, **project**, and **join**. Look at the following sequence of actions

Select Team where TeamName = "Chargers" giving Temp1;



	TeamId	TeamName	SponsorId
	T1	Chargers	S1
▶			

Join Temp1 and Player Giving Temp2;



	TeamId	TeamName	SponsorId	PlayerId	PlayerName	PlayerPhone	PlayerParent
	T1	Chargers	S1	P1	Jill	444-2323	Sally
	T1	Chargers	S1	P2	Cathy	333-5555	Joan
	T1	Chargers	S1	P6	Joyce	543-5432	Martha
▶							

Project Temp2 over PlayerName Giving Temp3;



PlayerName
Jill
Cathy
Joyce

By looking at the original tables, these are obviously the correct girls.

Note that using the alternative notation that we have addressed above, we can combine the above statements to solve the query

What are the names of the players that play for the Chargers?

in one Relational Algebra statement. Look at how the following 3 statements

Temp1 = Team where TeamName = "Chargers";

Temp2 = Temp1 nJoin Player;

Temp3 = Temp2[PlayerName];

can be combined into

Temp3 = ((Team where TeamName = "Chargers") nJoin Player)

Select Player where PlayerName = "Sandy" Giving Temp4;

Let's try another...

What is the name of the team that Sandy plays for?

Select Player where PlayerName = "Sandy" Giving Temp4;



PlayerId	TeamId	PlayerName	PlayerPhone	PlayerParent
P7	T3	Sandy	654-3456	Jan

Join Temp4 and Team Giving Temp5;

Temp5 : Select Query							
	PlayerId	TeamId	PlayerName	PlayerPhone	PlayerParent	TeamName	SponsorId
	P7	T3	Sandy	654-3456	Jan	Predators	S2
▶							

Project Temp5 over TeamName Giving Temp6;

Temp6 : Select Query	
	TeamName
	Predators
▶	

This is the correct name as can be seen by examining the tables.

Once again, we can combine the three Relational Algebra statements

Temp4 = Player where PlayerName = "Sandy";
Temp 5 = Temp4 nJoin Team;
Temp6 = Temp5[TeamName];into

Temp6 =
((Player where PlayerName = "Sandy") nJoin Team) [TeamName];

Let us try one more...

What are the names of all the girls on teams sponsored by Swimmers Paradise;

How about...

Select Sponsor where SponsorName = "Swimmers Paradise" giving Temp7;

Temp7 : Select Query	
SponsorId	SponsorName
S3	Swimmers Paradise
▶	

Join Temp7 and Team Giving Temp8;

Temp8 : Select Query			
SponsorId	SponsorName	TeamId	TeamName
S3	Swimmers Paradise	T2	Streakers
S3	Swimmers Paradise	T4	Strike
▶			

Join Temp8 and Player Giving Temp9;

Temp9 : Select Query							
SponsorId	SponsorName	TeamId	TeamName	PlayerId	PlayerName	PlayerPhone	PlayerParent
S3	Swimmers Paradise	T2	Streakers	P3	Samantha	222-4565	Susan
S3	Swimmers Paradise	T2	Streakers	P4	Shelly	321-4321	Sarah
S3	Swimmers Paradise	T4	Strike	P5	Karen	432-5432	Liz
▶							

Project Temp9 over Playername giving Temp10;

Temp10 : Select Query	
PlayerName	
Samantha	
Shelly	
Karen	
▶	

If you will look closely at the original tables, these are the correct girls...

Once more, let us combine the statements into one statement:

```
Temp7 = Sponsor where SponsorName = "Swimmers Paradise";
Temp8 = Temp7 nJoin Team;
Temp9 = Temp8 nJoin Player;
Temp10 = Temp9 [PlayerName];
```

can become

```
Temp10 = (((Sponsor where SponsorName = "Swimmers Paradise")
nJoin Team) nJoin Player) [PlayerName];
```

Now we will see the other joins.

SET Operations Union, Intersection and Set Difference

Consider two relations R and S.

The Union Operation

UNION of R and S the union of two relations is a relation that includes all the tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

The **union** operation is denoted \cup as in set theory. It returns the union (set union) of two compatible relations.

For a union operation $R \cup S$ to be legal, we require that

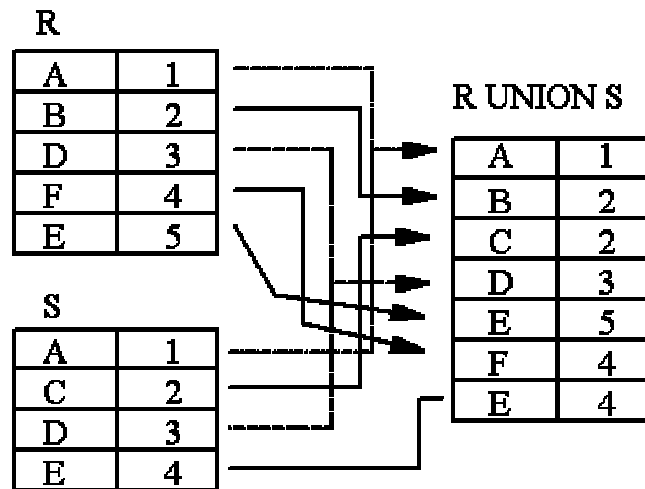
- R and S must have the same number of attributes.
- The domains of the corresponding attributes must be the same.

To find all customers of the SFU branch, we must find everyone who has a loan or an account or both at the branch.

We need both *borrow* and *deposit* relations for this:

$$\Pi_{cname}(\sigma_{branch="SFU"}(borrow)) \cup \Pi_{cname}(\sigma_{branch="SFU"}(deposit))$$

UNION Example :-



INTERSECTION

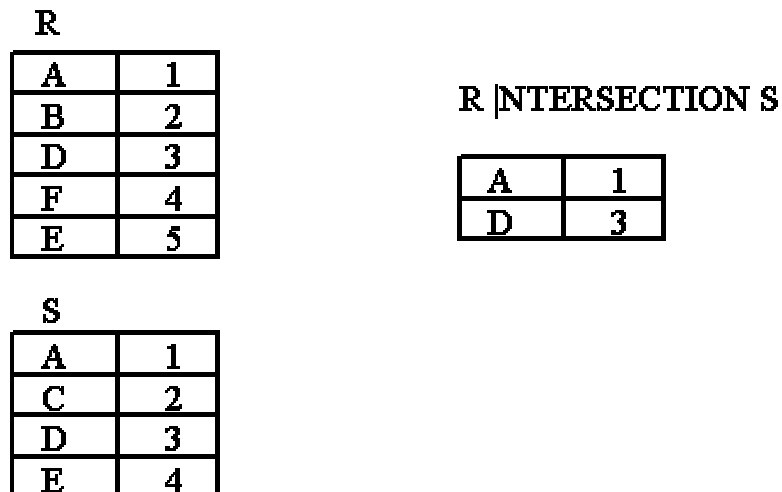
The **interference** of R and S includes all tuples that are both in R and S. The set intersection is a binary operation that is written as $R \cap S$ and is defined as the usual set intersection in set theory:

$$R \cap S = \{ t : t \in R, t \in S \}$$

The result of the set intersection is only defined when the two relations have the same headers. This operation can be simulated in the basic operations as follows:

$$R \cap S = R - (R - S)$$

INTERSECTION Example :-



The Set Difference

DIFFERENCE of R and S is the relation that contains all the tuples that are in R but that are not in S. For set operations to function correctly the relations R and S must be union compatible. Two relations are union compatible if

- they have the same number of attributes
- the domain of each attribute in column order is the same in both R and S.

The set difference is a binary operation that is written as $R - S$ and is defined as the usual set difference in set theory:

$$R - S = \{ t : t \in R, \neg t \in S \}$$

The result of the set difference is only defined when the two relations have the same headers.

DIFFERENCE Example :-

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R DIFFERENCE S

B	2
F	4
E	5

S DIFFERENCE R

C	2
E	4

Consider the following relation

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

List all cities where there is a branch office but no properties for rent.

$\Pi_{\text{city}}(\text{Branch}) - \Pi_{\text{city}}(\text{PropertyForRent})$

city
Bristol

Relational Calculus

An operational methodology, founded on predicate calculus, dealing with descriptive expressions that are equivalent to the operations of relational algebra. Codd's reduction algorithm can convert from relational calculus to relational algebra. Two forms of the relational calculus exist: the tuple calculus and the domain calculus. Codd proposed the concept of a relational calculus (applied predicate calculus tailored to relational databases).

Why it is called relational calculus?

It is founded on a branch of mathematical logic called the predicate calculus. Relational calculus is a formal query language where we write one **declarative** expression to specify a retrieval request and hence there is no description of how to evaluate a query; a calculus expression specifies *what* is to be retrieved rather than *how* to retrieve it. Therefore, the relational calculus is considered to be a **nonprocedural** language. This differs from relational algebra, where we must write a *sequence of operations* to specify a retrieval request; hence it can be considered as a **procedural** way of stating a query. It is possible to nest algebra operations to form a single expression; however, a certain order among

the operations is always explicitly specified in a relational algebra expression. This order also influences the strategy for evaluating the query.

It has been shown that any retrieval that can be specified in the relational algebra can also be specified in the relational calculus, and vice versa; in other words, the **expressive power** of the two languages is *identical*. This has led to the definition of the concept of a relationally complete language. A relational query language L is considered **relationally complete** if we can express in L any query that can be expressed in relational calculus. Relational completeness has become an important basis for comparing the expressive power of high-level query languages. However certain frequently required queries in database applications cannot be expressed in relational algebra or calculus. Most relational query languages are relationally complete but have *more expressive power* than relational algebra or relational calculus because of additional operations such as aggregate functions, grouping, and ordering.

Tuple calculus

The **tuple calculus** is a calculus that was introduced by Edgar F. Codd as part of the relational model in order to give a declarative database query language for this data model. It formed the inspiration for the database query languages QUEL and SQL of which the latter, although far less faithful to the original relational model and calculus, is now used in almost all relational database management systems as the ad-hoc query language. Along with the tuple calculus Codd also introduced the domain calculus which is closer to first-order logic and showed that these two calculi (and the relational algebra) are equivalent in expressive power. The SQL language is based on the tuple relational calculus which in turn is a subset of classical predicate logic. Queries in the TRC all have the form:

$$\{QueryTarget \mid QueryCondition\}$$

The *QueryTarget* is a tuple variable which ranges over tuples of values. The *QueryCondition* is a logical expression such that

- It uses the *QueryTarget* and possibly some other variables.

- If a concrete tuple of values is substituted for each occurrence of the *QueryTarget* in *QueryCondition*, the condition evaluates to a boolean value of *true* or *false*.

The result of a TRC query with respect to a database instance is the set of all choices of values for the query variable that make the query condition a true statement about the database instance. The relation between the TRC and logic is in that the *QueryCondition* is a logical expression of classical first-order logic.

The tuple relational calculus is based on specifying a number of **tuple variables**. Each tuple variable usually **ranges over** a particular database relation, meaning that the variable may take as its value any individual tuple from that relation. A simple tuple relational calculus query is of the form

$$\{t \mid \text{COND}(t)\}$$

where t is a tuple variable and $\text{COND}(t)$ is a conditional expression involving t . The result of such a query is the set of all tuples t that satisfy $\text{COND}(t)$. For example, to find all employees whose salary is above \$50,000, we can write the following tuple calculus expression:

$$\{t \mid \text{EMPLOYEE}(t) \text{ and } t.\text{SALARY} > 50000\}$$

The condition $\text{EMPLOYEE}(t)$ specifies that the *range relation* of tuple variable t is EMPLOYEE . Each EMPLOYEE tuple t that satisfies the condition $t.\text{SALARY} > 50000$ will be retrieved. Notice that $t.\text{SALARY}$ references attribute SALARY of tuple variable t ; this notation resembles how attribute names are qualified with relation names or aliases in SQL. The above query retrieves all attribute values for each selected EMPLOYEE tuple t . To retrieve only *some* of the attributes—say, the first and last names—we write

$$\{t.\text{FNAME}, t.\text{LNAME} \mid \text{EMPLOYEE}(t) \text{ and } t.\text{SALARY} > 50000\}$$

This is equivalent to the following SQL query:

Select T.FNAME, T.LNAME from EMPLOYEE AS T Where T.SALARY>50000;

Informally, we need to specify the following information in a tuple calculus expression:

1. For each tuple variable t , the **range relation** R of t . This value is specified by a condition of the form $R(t)$.
2. A condition to select particular combinations of tuples. As tuple variables range over their respective range relations, the condition is evaluated for every possible combination of tuples to identify the **selected combinations** for which the condition evaluates to TRUE.
3. A set of attributes to be retrieved, the **requested attributes**. The values of these attributes are retrieved for each selected combination of tuples.

Observe the correspondence of the preceding items to a simple SQL query: item 1 corresponds to the FROM-clause relation names; item 2 corresponds to the WHERE-clause condition; and item 3 corresponds to the SELECT-clause attribute list. Before we discuss the formal syntax of tuple relational calculus, consider another query we have seen before.

Retrieve the birthdate and address of the employee (or employees) whose name is ‘John B. Smith’.

Q0 : { t .BDATE, t .ADDRESS | EMPLOYEE(t) **and** t .FNAME=‘John’ **and** t .MINIT=‘B’ **and** t .LNAME=‘Smith’}

In tuple relational calculus, we first specify the requested attributes t .BDATE and t .ADDRESS for each selected tuple t . Then we specify the condition for selecting a tuple following the bar (|)—namely, that t be a tuple of the EMPLOYEE relation whose FNAME, MINIT, and LNAME attribute values are ‘John’, ‘B’, and ‘Smith’, respectively.

Domain Calculus

There is another type of relational calculus called the domain relational calculus, or simply, **domain calculus**. The language QBE that is related to domain calculus was developed almost concurrently with SQL at IBM Research, Yorktown Heights. The

formal specification of the domain calculus was proposed after the development of the QBE system.

The domain calculus differs from the tuple calculus in the *type of variables* used in formulas: rather than having variables range over tuples, the variables range over single values from domains of attributes. To form a relation of degree n for a query result, we must have n of these **domain variables**—one for each attribute. An expression of the Domain calculus is of the form

$$\{x_1, x_2, \dots, x_n \mid \text{COND}(x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m})\}$$

where $x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m}$ are domain variables that range over domains (of attributes) and COND is a **condition** or **formula** of the domain relational calculus.

A formula is made up of **atoms**

As in tuple calculus, atoms evaluate to either TRUE or FALSE for a specific set of values, called the **truth values** of the atoms. In case 1, if the domain variables are assigned values corresponding to a tuple of the specified relation R , then the atom is TRUE. In cases 2 and 3, if the domain variables are assigned values that satisfy the condition, then the atom is TRUE.

In a similar way to the tuple relational calculus, formulas are made up of atoms, variables, and quantifiers, so we will not repeat the specifications for formulas here. Some examples of queries specified in the domain calculus follow. We will use lowercase letters l, m, n, \dots, x, y, z for domain variables.

QUERY 0

Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

$$Q0 : \{uv \mid (\exists q) (\exists r) (\exists s) (\exists t) (\exists w) (\exists x) (\exists y) (\exists z)$$

(EMPLOYEE(qrstuvwxyz) and $q = \text{'John'}$ and $r = \text{'B'}$ and $s = \text{'Smith'}$)}

We need ten variables for the EMPLOYEE relation, one to range over the domain of each attribute in order. Of the ten variables q, r, s, \dots, z , only u and v are free. We first specify

the *requested attributes*, BDATE and ADDRESS, by the domain variables u for BDATE and v for ADDRESS. Then we specify the condition for selecting a tuple following the bar (|)—namely, that the sequence of values assigned to the variables qrstuvwxyz be a tuple of the EMPLOYEE relation and that the values for q (FNAME), r (MINIT), and s (LNAME) be ‘John’, ‘B’, and ‘Smith’, respectively. For convenience, we will quantify only those variables *actually appearing in a condition* (these would be q, r, and s in Q0) in the rest of our examples.

An alternative notation for writing this query is to assign the constants ‘John’, ‘B’, and ‘Smith’ directly as shown in Q0A, where all variables are free:

Q0A : {uv | EMPLOYEE(‘John’, ‘B’, ‘Smith’, t, u, v, w, x, y, z) }

QUERY 1

Retrieve the name and address of all employees who work for the ‘Research’ department.

Q1 : {qsv | (∃z) (∃l) (∃m) (EMPLOYEE(qrstuvwxyz) **and** DEPARTMENT(lmno) **and** l=‘Research’ **and** m=z)}

A condition relating two domain variables that range over attributes from two relations, such as m = z in Q1, is a **join condition**; whereas a condition that relates a domain variable to a constant, such as l = ‘Research’, is a **selection condition**.

QUERY 2

For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, birthdate, and address.

Q2 : {iksuv | (∃j) (∃m)(∃n) (∃t)(PROJECT(hijk) **and** EMPLOYEE(qrstuvwxyz) **and** DEPARTMENT(lmno) **and** k=m **and** n=t **and** j=‘Stafford’)}

QUERY 6

Find the names of employees who have no dependents.

Q6 : {qs | (∃t) (EMPLOYEE(qrstuvwxyz) **and** (**not**(∃l) (DEPENDENT(lmnop) **and** t=l))))}

Query 6 can be restated using universal quantifiers instead of the existential quantifiers, as shown in Q6A:

Q6A : $\{qs \mid (\exists t) (\text{EMPLOYEE}(qrstuvwxyz) \text{ and } ((\forall l) (\text{not}(\text{DEPENDENT}(lmnop)) \text{ or } \text{not}(t=l))))\}$

QUERY 7

List the names of managers who have at least one dependent.

Q7 : $\{sq \mid (\exists t) (\exists j) (\exists l)(\text{EMPLOYEE}(qrstuvwxyz) \text{ and } \text{DEPARTMENT}(hijk) \text{ and } \text{DEPENDENT}(lmnop) \text{ and } t=j \text{ and } l=t)\}$

As we mentioned earlier, it can be shown that any query that can be expressed in the relational algebra can also be expressed in the domain or tuple relational calculus. Also, any safe expression in the domain or tuple relational calculus can be expressed in the relational algebra.

Review Questions

1. What do you mean by Relational Algebra?
2. Explain the Database Specific Relational Algebraic Expressions?
3. Explain the various set operations?
4. Explain Tuple and domain relational Calculus?

References

Date, C.J., Introduction to Database Systems (7th Edition) Addison Wesley, 2000

Leon, Alexis and Leon, Mathews, Database Management Systems, LeonTECHWorld.