Relational Algebra

1. Introduction

Data manipulation within a RDBMS is done, mainly, using Relational Algebra (RA). In short, relational algebra operates on entities and relationships as conventional algebra operates on numbers. Typically, **each relational operation defines new entities or new relationships (new tables) on which one can carry out editing, searching, aggregation and consultation activities**.

We can say that, basically, the key roles played by RA are the following two:

- Data modelling and data analysis;
- Define the basis of query languages needed to interrogate a RDB.

The relational algebra operators can be classified into **basic and derivative operators**. The first ones make it possible to perform almost all the main operations needed to filter and to select data within a relational scheme. The second ones are used to simplify the execution of complex operations that, otherwise, would require the joint use of different basic operator.

Basic Operators

- *Projection*
- *Selection*
- *Cartesian Product*
- *Renaming*
- *Union*
- *Difference*

Main Derivative Operators

- *Inner Join*
- *Semi Join*
- *Outer Join*
- *Intersection*
- *Division*

2. Basic Operators

Projection

Let us consider a Table T made of *r* records of *n* fields. A projection based on a set (*m* < *n*) of attributes performed on T returns a table T_P with the same number (r) of records, but with a reduced number (*m* < *n*) of fields. Specifically, **only the m fields (or attributes) specified by the projection operator will be shown in the records of T_P**. For example, by operating a projection based on the attributes {Surname, Job}, Table 3.1 (a) is converted in Table 3.1 (b), as shown below:

We note that Table obtained using relational algebra operators does not have to be normalized. For instance, in Table 3.1(b) the primary key is missing.

Tab. 3.1 (a) *Original Table*

Id_Employee (PK)	Surname	Job	…	Respond To (FK)
1	Red	Nurseryman	…	r
2	Green	Nurseryman	…	r
3	Beagle	Clerk	…	4
4	Hart	Owner		Null
…	…	…	…	…
R	Doe	Chief Nurseryman	…	4

Tab. 3.1 (b)
Derived table using {Surname, Job} as projection attributes

Surname	Job
Red	Nurseryman
Green	Nurseryman
Beagle	Clerk
Hart	Chief Nurseryman
…	…
Doe	Owner

Selection

Let us consider a Table T made of r records of *n* fields. A selection based on m logical condition performed on T returns a table T_S with a reduced number ($k \leq r$) of records, each one with the original number *n* of fields. Specifically, **only the *k* records that meet all the *m* logical conditions will be displayed in the derived table T_S**. For example, if we consider Tab. 3.1 (a) once again, a selection based on the following logical condition {Respond to = Null} generates Tab. 3.1 (c).

Tab. 3.1 (c)
Derived table using {Respond To = Null} as logical criteria of a selection operation

Id_Employee (PK)	Surname	Job	…	Respond To (FK)
4	Hart	Owner		Null

As it can be seen, in this case, the output coincides with a single record (i.e., k = 1), because there is only one boss (the owner) of the shop.

Cartesian product

The Cartesian product is a **binary operation**, in that *it operates on two different input tables and returns a third table* as output. Let us consider a Table T_1 made of r_1 records of n fields and a Table T_2 made of r_2 records of m fields. By operating a Cartesian product between T_1 and T_2 a new table T_{CP} with $r_1 \times r_2$ records of $(n + m)$ fields is obtained. Specifically, the values of the first *n* fields correspond to the values of a record of T_1 and the value of the last *m* fields correspond to the values of a record of T_2. In other word by **performing a Cartesian product each record of T_1 is joined with each record of T_2**. An example concerning an AUTHORS and BOOKS table follows.

Tab. 3.2 (a) *AUTHORS Table*

Id _Author	Surname	Nationality	Language
1	Dante	Italian	Vulgar Italian
2	Collodi	Italian	Tuscan Italian
3	Joyce	Irish	English

Tab. 3.2 (b) *BOOKS table*

Id _Author	Id_Book	Title
1	1	Divina Commedia
1	2	De Vulgari Eloquentia
3	3	Ulisse
3	4	Finnegans Wake
2	5	Le avventure di Pinocchio

Tab. 3.2 (c)
Cartesian Product performed on AUTHORS and BOOKS tables

Id_Author	Surname	Nationality	Language	Id_Author	Id_Book	Title
1	Dante	Italian	Vulgar Italian	1	1	Divina Commedia
2	Collodi	Italian	Tuscan Italian	1	1	Divina Commedia
3	Joyce	Irish	English	1	1	Divina Commedia
1	Dante	Italian	Vulgar Italian	1	2	De Vulgari Eloquentia
2	Collodi	Italian	Tuscan Italian	1	2	De Vulgari Eloquentia
3	Joyce	Irish	English	1	2	De Vulgari Eloquentia
1	Dante	Italian	Vulgar Italian	3	3	Ulisse
2	Collodi	Italian	Tuscan Italian	3	3	Ulisse
3	Joyce	Irish	English	3	3	Ulisse
1	Dante	Italian	Vulgar Italian	3	4	Finnegans Wake
2	Collodi	Italian	Tuscan Italian	3	4	Finnegans Wake
3	Joyce	Irish	English	3	4	Finnegans Wake
1	Dante	Italian	Vulgar Italian	2	5	Le avventure di Pinocchio
2	Collodi	Italian	Tuscan Italian	2	5	Le avventure di Pinocchio
3	Joyce	Irish	English	2	5	Le avventure di Pinocchio

As it can be seen, since r_1 = 3 and r_2 = 5, a total of 15 records have been generated. Also, since n = 4 and m = 3, the new generated records are made of 7 fields.

It is important to note that this is just a relational operation, there is no logic in the obtained table, as **most of the generated records do not have any sense at all** (for instance Dante did not write either Finnegans Wake or Le avventure di Pinocchio). Yet, as we will see later, the Cartesian Product is extremely useful to perform queries operating on more than a single table.

Renaming

Renaming allows one **to modify the columns' headings** of a certain Table. To this aim, it is sufficient to pass, as input, a list of attributes to be used as new column headings and to specify the table and the columns that must be renamed.

Union

Union is a **binary operation**, in that it operates on two different input tables and returns a third table as output. Let us consider a Table T_1 and a Table T_2. The union performed on T_1 and on T_2 **returns a third Table T_3 obtained making the union of all the records of the original tables**. In other words, if T_1 has r_1 records and T_2 has r_2 records, then T_3 will have $r_3 = r_1 + r_2$ records, the first r_3 are those of T_1 and the last r_2 are those of T_2.
Obviously, for this operation to be possible, it is necessary **that the records of T_1 have the same number of fields as the records of T_2**. Also, each field of T_1 and T_2 must contain the same data type - defined on the same domain (or on the same eligibility subsets, if there are conditions and validation rules) - and there must be no duplication of the primary keys, in full compliance with referential integrity constraints.
A simple example is shown below:

Tab. 3.3 (a) *Table T_1*

ID_A (PK)	Value
A1	12000
A2	15000
A3	10000
…	….
A100	500

Tab. 3.3 (b) *Table T_2*

ID_B (PK)	Value
B1	22000
B2	35000
…	…
B15	20000

Tab. 3.3 (c) *Table $T_{2,bis}$*

ID_B (PK)	Date
B1	01/01/2000
B2	10/10/2010
…	…
B15	05/05/2020

Tab. 3.3 (d) *Table $T_{2,ter}$*

ID_B (PK)	Value
A1	50
A2	11111
…	…
A10	2000

Tab. 3.3 (e) *Union of T_1 and T_2*

ID_Union (PK)	Value
A1	12000
A2	15000
…	10000
A100	500
B1	22000
…	…
B15	20000

Note that an union can be performed only on Table T_1 and T_2 and the result is shown in Tab. 3.3 (e)[1]. Table T_1 and $T_{2,bis}$ cannot be combined using the Union operator because values stored in the second field are of different and incompatible types (i.e., Integer and Date). Similarly, Table T_1 and $T_{2,ter}$ cannot be combined using the Union operator because the resulting table would have duplicate primary keys and thus it would violate referential integrity constraints.

Difference

Also the Difference is a Union is a **binary operation**. Let us consider a Table T_1 and a Table T_2; the difference performed on T_1 and on T_2 **returns a third Table T_3 with all the record of T_1 except those ones that are also included in T_2.**

As for the Union, to perform a Difference, T1 and T2 must have the same structure (in terms of number and type of fields). A simple example follows:

Tab. 3.4 (a) *Table T_1*

Name	Surname
Maggie	Blond
Ronald	Regan
George	Busch
Bill	Clinton
Tom	Clay

[1] *Actually, also T_2 and $T_{2,Ter}$ could be combined using the Union operator*

Tab. 3.4 (b) *Table T_2*

Name	Surname
Maggie	Blond
Ronald	Regan
George	Phillips
Harry	Clinton
Tomas	Jefferson

Tab. 3.4 (c)
Table T_3 the Difference

Name	Surname
George	Busch
Bill	Clinton
Tom	Clay

3. Derivative Operators

Inner or Natural Join

Among all the relational algebra operators, the Inner (also called Natural[2]) Join is surely the most important one, as it allows one to generate very complex queries operating on two or more tables having common data.

Let us consider two tables T_1, made of n_1 records with r_1 fields, and T_2 made of n_2 records with r_2 fields. If m records of T_1 and T2 have some common data (i.e., same values of one or more fields), then the **Inner Join returns a table T_3 containing new records obtained as the combination of all the records of T_1 and T_2 that have some common data, without duplicated fields**.

More precisely, for each one of the m couples of records $R_{1,i}$ and $R_{2,j}$ (of T_1 and T_2, respectively) having k common data, the resulting table T_3 will contain a record with (n_1 + n_2 - k) fields, of which the first n_1 fields are those of $R_{1,i}$, the last ones are those of $R_{2,j}$, without the k common fields.

Most of the times, an Inner Join is made on two tables in OTM relation. In this peculiar, but frequent, condition, the join is made on the FK (i.e., the common field is limited to the value of the PK of the father table and to the value of the FK of the son table). To better clarify this concept, let us consider once again the AUTHORS and the BOOKS tables (i.e., Tab. 3.2 (a) and 3.2 (b), respectively). In this case, am Inner Join will return a table with five records of six fields. This is because:

- All the five records of the BOOKS table are linked with one of the three records of the AUTHORS table (i.e., there is no record with a null value in the FK field);
- The BOOKS table has records with four fields and the BOOKS table has records with three fields, but one of them is the Forward Key that, being a common field, will be erased by the natural join.

[2] *A small difference exists, but we will use the term Inner and Natural as synonyms.*

Thus, the result is the one shown by Tab. 3.5 below:

Tab. 3.5

Inner Join performed on AUTHORS and BOOKS Tables

Id _Author	Surname	Nationality	Language	Id_Book	Title
1	Dante	Italian	Vulgar Italian	1	Divina Commedia
2	Collodi	Italian	Tuscan Italian	5	Le avventure di Pinocchio
3	Joyce	Irish	English	3	Ulisse
1	Dante	Italian	Vulgar Italian	2	De Vulgari Eloquentia
3	Joyce	Irish	English	4	Finnegans Wake

As it can be seen, the table is not normalized; as above mentioned this is not a problem. Only the original table of a RDB must be normalized, whereas those one obtained by performing relational algebra may also be non-normalized (most of the times they are non-normalized).

As above mentioned the INNER JOIN is a derivative operator; as such it can be obtained by properly combining some basic operators. Specifically, the **inner join can be obtained** in the following way:

- A Cartesian product is made on T_1 and T_2;
- A Selection is made to filter data, so as to keep only the records that have the same value in the common field (typically those that have the same value of the Primary and of the Forward key);
- A Projection is made, to get rid of the duplicated values (i.e., the common data, typically the FK)

This is shown by Tab. 3.6 where the records and the duplicated fields that must be deleted are highlighted in light grey.

Tab. 3.6.

Cartesian Product, Selection and Projection performed on AUTHORS and BOOKS tables

Id_Author	Surname	Nationality	Language	Id_Author	Id_Book	Title
1	Dante	Italian	Vulgar Italian	1	1	Divina Commedia
2	Collodi	Italian	Tuscan Italian	1	1	Divina Commedia
3	Joyce	Irish	English	1	1	Divina Commedia
1	Dante	Italian	Vulgar Italian	1	2	De Vulgari Eloquentia
2	Collodi	Italian	Tuscan Italian	1	2	De Vulgari Eloquentia
3	Joyce	Irish	English	1	2	De Vulgari Eloquentia
1	Dante	Italian	Vulgar Italian	3	3	Ulisse
2	Collodi	Italian	Tuscan Italian	3	3	Ulisse
3	Joyce	Irish	English	3	3	Ulisse
1	Dante	Italian	Vulgar Italian	3	4	Finnegans Wake
2	Collodi	Italian	Tuscan Italian	3	4	Finnegans Wake
3	Joyce	Irish	English	3	4	Finnegans Wake
1	Dante	Italian	Vulgar Italian	2	5	Le avventure di Pinocchio
2	Collodi	Italian	Tuscan Italian	2	5	Le avventure di Pinocchio
3	Joyce	Irish	English	2	5	Le avventure di Pinocchio

Semi Join

The **Semi Join operator is a particular instance of the Natural Join** described above. Let us consider two tables T_1 and T_2 with some common fields. In this case, the result of a Semi Join between T_1 and T_2 is analogous to that of an Inner Join with the exception that, in this case, **the records of the output table T_3 will contains only the fields of the records of T_1**.

For instance, considering once again the AUTHORS and the BOOKS tables, performing a semi join instead of an Inner Join, then Table 3.5 would change as in Table 3.7 (where the records of the BOOKS table, in white and light grey, have been erased).

Tab. 3.7

Semi Join performed on AUTHORS and BOOKS Tables

Id _Author	Surname	Nationality	Language	Id_Book	Title
1	Dante	Italian	Vulgar Italian	1	Divina Commedia
2	Collodi	Italian	Tuscan Italian	5	Le avventure di Pinocchio
3	Joyce	Irish	English	3	Ulisse
1	Dante	Italian	Vulgar Italian	2	De Vulgari Eloquentia
3	Joyce	Irish	English	4	Finnegans Wake

Also the SEMI JOIN is a derivative operator; as such it can be obtained by properly combining some basic operators. Specifically, the **inner join can be obtained** in the following way:

- An Inner Join is made on T_1 and T_2;
- A Projection is made, to get rid of the fields that belongs to the records of T_2, i.e., only the data of T_1 are displayed.

Outer Join

At this point it should be clear that **performing an Inner or a Semi Join on two tables T_1 and T_2 having some common data, some of the original data may get lost in the output table T_3**. More precisely, records of T_1 that do not have a correspondence in the records of T_2 and vice versa, will not be displayed in T_3. To better clarify this idea, let us suppose that two new authors (i.e., Asimov and Camilleri) have been inserted in the AUTHORS table, but that there are no books written by them in the BOOKS table. Let us also suppose that a new title has been inserted in the BOOKS table (i.e., Il fu mattia Pascal), but that the author of this new book (i.e., Luigi Pirandello) has not been inserted in the AUTHORS table, yet.

A possible case is shown in Table 3.8 (a) and 3.8 (b).

If we performed an Inner Join on these tables, we would get, once again, Table 3.5 as result. Evidently, in the five records of Table 3.5 there is trace neither of the new authors, nor of the new book. These data 'have been lost' because they belong to uncoupled records.

Tab. 3.8 (a)
New AUTHORS Table

Id _Author	Surname	Nationality	Language
1	Dante	Italian	Vulgar Italian
2	Collodi	Italian	Tuscan Italian
3	Joyce	Irish	English
4	**Asimov**	**Polish**	**English**
5	**Camilleri**	**Italian**	**Italian**

Tab. 3.8 (b)
New BOOKS table

Id _Author	Id_Book	Title
1	1	Divina Commedia
1	2	De Vulgari Eloquentia
3	3	Ulisse
3	4	Finnegans Wake
2	5	Le avventure di Pinocchio
Null	6	Il Fu Mattia Pascal

This problem can be solved using an OUTER JOIN instead of the classical INNER JOIN. Indeed, by performing an OUTER JOIN on two tables T_1 and T_2 with some common field we obtain a third table T_3 that contains:

- All the records of T_1 concatenated with the records of T_2 having common data (i.e., the same records that are returned by means of an INNER JOIN performed on T_1 and T_2)
- All the records of T_1 that do not have any correspondence in T2, concatenated with a record of T_2 with all null values;
- All the records of T_2 that do not have any correspondence in T_1, concatenated with a record of T_1 with all null values;

For instance, in the previous case, {5, Camilleri, Italian, Italian} is a record of T_1 that is not linked to any record of T_2. Performing an OUTER JOIN, this record will be concatenated with a record of T_2 made of null values {Null, Null, Null}. Consequently, the resulting table will also contain the following new record: {5, Camilleri, Italian, Italian, Null, Null, Null}.

The overall result is shown in Table 3.9, where the (erased) common field are shown in light grey.

Tab. 3.9 (b)
The OUTER JOIN performed on AUTHORS AND BOOKS

Id _Author	Surname	Nationality	Language	Id_Authots	Id_Book	Title
1	Dante	Italian	Vulgar Italian	1	1	Divina Commedia
2	Collodi	Italian	Tuscan Italian	2	5	Le avventure di Pinocchio
3	Joyce	Irish	English	3	3	Ulisse
1	Dante	Italian	Vulgar Italian	1	2	De Vulgari Eloquentia
3	Joyce	Irish	English	3	4	Finnegans Wake
4	Asimov	Polish	English	Null	Null	Null
5	Camilleri	Italian	Italian		Null	Null
Null	Null	Null	Null	Null	6	Il Fu Mattia Pascal

We conclude this section noting that there are also two particular type of OUTER JOIN, these are the **<u>LEFT OUTER JOIN and the RIGHT OUTER JOIN</u>**. Specifically, in case of LEFT OUTER JOIN, in addition to the records of the Inner Join, in the output table we will find also the records of T_1 that are not linked to any record of T_2 (see Table 3.10 (b)); conversely, in case of a RIGH OUTER JOIN, in addition to the records of the Inner Join, in the output table we will find also the records of T_2 that are not linked to any record of T_1 (see Table 3.10 (a)).

Tab. 3.10 (a)
The LEFT OUTER JOIN performed on AUTHORS AND BOOKS

Id _Author	Surname	Nationality	Language	Id_Book	Title
1	Dante	Italian	Vulgar Italian	1	Divina Commedia
2	Collodi	Italian	Tuscan Italian	5	Le avventure di Pinocchio
3	Joyce	Irish	English	3	Ulisse
1	Dante	Italian	Vulgar Italian	2	De Vulgari Eloquentia
3	Joyce	Irish	English	4	Finnegans Wake
4	**Asimov**	**Polish**	**English**	**Null**	**Null**
5	**Camilleri**	**Italian**	**Italian**	**Null**	**Null**

Tab. 3.10 (b)
The LEFT OUTER JOIN performed on AUTHORS AND BOOKS

Id _Author	Surname	Nationality	Language	Id_Book	Title
1	Dante	Italian	Vulgar Italian	1	Divina Commedia
2	Collodi	Italian	Tuscan Italian	5	Le avventure di Pinocchio
3	Joyce	Irish	English	3	Ulisse
1	Dante	Italian	Vulgar Italian	2	De Vulgari Eloquentia
3	Joyce	Irish	English	4	Finnegans Wake
Null	**Null**	**Null**	**Null**	**6**	**Il Fu Mattia Pascal**

Thus, we could say that, in case of two tables in OTM relation:

- The <u>LEFT OUTER JOIN also returns the records of the “Father table that do not have a son”;</u>
- The <u>RIGHT OUTER JOIN also returns the records of the “Son table that do not have a father”.</u>

Clearly, also the SEMI JOIN, being a derivative operator can be obtained by properly combining some basic operators. Specifically, the **Outer join can be obtained** in the following way:

- An Inner Join is made on T_1 and T_2 and a table $T_{3,1}$ is obtained
- A Cartesian Product is made on all the records of T_1 that are not linked to any records of T_2 and a single record of T_2 having all null values. This operation generates, as output, a new table $T_{3,2}$
- A Cartesian Product is made on all the records of T_2 (that are not linked to any records of T_1) and a single record of T_1 having all null values. This operation generates, as output, a new table $T_{3,3}$
- The OUTER JOIN Table T_3 is obtained as the Union of $T_{3,1}$, $T_{3,2}$ and $T_{3,3}$.

Intersection

Given two tables T_1 and T_2 with records having the same attributes, the **Intersection returns as results a third table T_3 that contains all the records that are common to T_1 and T_2**.

It is easy to see that, to obtain the Intersection it is **sufficient to use the Difference operator twice**. Indeed, letting $T_4 = (T_1 - T_2)$, then we have that $T_3 = (T_1 - T_4)$.

This is graphically shown below:

Tab. 3.11 (a)

Table T_1

ID_A (PK)	Value
A1	12000
A2	15000
A3	10000
B1	22000
B5	20000

Tab. 3.11 (b)

Table T_2

ID_B (PK)	Value
B1	22000
B2	35000
B4	1500
B5	20000

Tab. 3.11 (c)
Table $T_4 = T_1 - T_2$

ID_B (PK)	Value
A1	12000
A2	15000
A3	10000

Tab. 3.11 (d) *Table $T_3 = T_1 - T_4$*

ID_B (PK)	Value
B1	22000
B5	20000

Division

Let us consider two tables T_1, with rows of r_1 fields, and T_2, with rows of $r_2 < r_1$ fields. Let us also suppose that a certain the r_2 fields of T_2 are compatible (i.e., has the same type of data) with an equal number of fields of T_1. In this case, a Division performed on T_1 and T_2 will return a third table T_3 with as many record as the number of records of T_1 that, relatively to the compatible fields, perfectly match the values of the records of T_2. Also, the records of T_3 will contain only the fields of T_1 that do not appear in T_2.

An example is shown below.

Tab. 3.12 (a)
Table T_1

a1	a2	a3	a4
a11	a12	a13	a14
a21	a22	a23	a24
a31	a32	a33	a34
a41	a42	a43	a44

Tab. 3.12 (b)
Table T_2

a1	a2
a11	a12
a41	a42

Tab. 3.12 (c)
Table T_3 Divison

a3	a4
a13	a14
a43	a44