Relational algebra operators – Cross product & natural join

Relational algebra is the mathematical basis for performing queries against a relational database. Operations are performed against relations – resulting in relations. Because the result of relational algebra operation is a relation, operations can be stacked up against each other. More on this as we go forward.

For the following examples, we are going to be using relations Employees & Parking with the following schemas:

Employee (Emp_id:int, Emp_name:string, Emp_office:int)
Parking (Emp_id:int, Parking_lot:string, Parking_space:int)

Cross product

Cross product is a way of combining two relation instances. The resulting relation has a schema that contains each of the attributes in both relations being combined. A cross product is represented by the following notation:

Employee x Parking

And would result in the following schema:

(Emp_id:int, Emp_name:string, Emp_office:int, Emp_id:int, Parking_lot:string, Parking_space:int)

Note there are two columns with the same header. We will look into how to deal with that shortly, but it is an artifact of the way cross product works.

Let's look at the following relation instances of Employee & Parking:

Employee		
Emp_id	Emp_name	Emp_office
1001	Bob	10
1002	Alice	11
1003	Sandy	10
1004	Larry	11
1005	Susan	11

Parking		
Emp_id	Parking_lot	Parking_space
1001	A	6
1002	A	14
1003	B	17
1004	B	6
1005	A	12

If we apply the cross product operation to these two instances of Employee & Parking, we are saying we that we want all combinations of rows (tuples) between the two. The statement:

Employee x Parking

is saying, build a new relation that contains all the attributes of the two original relations, and for each row in Employee give be an instance of the rows in Parking. The result of this operation is a relation instance that looks like this:

Emp_id	Emp_name	Emp_office	Emp_id	Parking_lot	Parking_space
1001	Bob	10	1001	A	6
1001	Bob	10	1002	A	14
1001	Bob	10	1003	B	17
1001	Bob	10	1004	B	6
1001	Bob	10	1005	A	12
1002	Alice	11	1001	A	6
1002	Alice	11	1002	A	14
1002	Alice	11	1003	B	17
1002	Alice	11	1004	B	6
1002	Alice	11	1005	A	12
1003	Sandy	10	1001	A	6
1003	Sandy	10	1002	A	14
1003	Sandy	10	1003	B	17
1003	Sandy	10	1004	B	6
1003	Sandy	10	1005	A	12
1004	Larry	11	1001	A	6
1004	Larry	11	1002	A	14
1004	Larry	11	1003	B	17
1004	Larry	11	1004	B	6
1004	Larry	11	1005	A	12
1005	Susan	11	1001	A	6
1005	Susan	11	1002	A	14
1005	Susan	11	1003	B	17
1005	Susan	11	1004	B	6
1005	Susan	11	1005	A	12

As you can see, this grows quickly, there are N * M rows in the resulting relation (N = number in Employees, M = number in Parking), and the resulting relation has X + Y columns (attributes) in it (X = number of attributes in Employees, Y = number of attributes in Parking). When you start to build queries, it is important to keep this in mind – perform your selection and projection criteria as early as logically possible to limit the amount of work being performed by the query processor.

Natural Join

If the original question being asked was: What is the list of employee names and their associated parking spaces, the cross product operation isn't very useful. Instead we look to the natural join operation, represented by the symbol |X|. This operation says: Create a new relation where the fact that there are common attributes in the two relations is recognized, and the new relation only contains those rows where the values in the two common attributes are equal to each other.

Because of the concept of foreign keys, this becomes very powerful – and is one of the primary tools in your arsenal going forward.

Let's look at our original two relations:

Employee		
Emp_id	Emp_name	Emp_office
1001	Bob	10
1002	Alice	11
1003	Sandy	10
1004	Larry	11
1005	Susan	11

Parking		
Emp_id	Parking_lot	Parking_space
1001	A	6
1002	A	14
1003	B	17
1004	B	6
1005	A	12

If we perform the operation:

Employee |X| Parking

We have recognized there is a common field between the two relations, Emp_id. The operation we are requesting is to create a new relation that contains all the attributes from Employee and Parking, but combine the common fields. The resulting relation schema looks like this:

(Emp_id:int, Emp_name:string, Emp_office:int, Parking_lot:string, Parking_space:int)

The operation selects those rows from Parking to combine with the row from Employee where Emp_id in Employee is equal to the Emp_id in Parking.   The resulting relation looks like this:

Emp_id	Emp_name	Emp_office	Parking_lot	Parking_space
1001	Bob	10	A	6
1002	Alice	11	A	14
1003	Sandy	10	B	17
1004	Larry	11	B	6
1005	Susan	11	A	12