Natural join

Natural join says r ~~⋈~~ ⋈ S, two tupples tr ∈ r and ts ∈ S and both have a common attribute, then we can use natural join. If both tupples have a null value, then the resultant relation will not have or consider that ~~tod~~ ~~tupp~~ tupple.

emp.

emp_name	street	city
Coyote	Toon.	Hollywood.
Rabbit	Tunnel	Carrotville.
Smith	Revolver	Death valley
Williams	Seaview	Seattle.

St-works.

emp_name	branch-name	salary
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
William.	Redmond	1816.

emp ⋈ St-works

emp_name	street	city	branch name	Sal.
Coyote	Toon	Hollywood	Mesa	1500.
Rabbit	Tunnel	Carrot ville.	Mesa	1300
Williams.	Seaview	Seattle	Redmond	1000.

✓ values existing with all the 2 relations.

emp natural join St-works on emp.emp_name = St.works.emp_name;

Outer join :- It is an extension of join operation to deal with missing information. Suppose we have 2 relations:-

Outer join results with null values.

⟕   N⋈

Left Outer :-

It takes all tupples in the left rotation that didnot match with any tupple in the right ~~rotation~~ relation, pads the tupples with null values for all other attributes from the right relation and adds them to the result of natural join.

e.g. Tables.

emp:

emp-name	street	city
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrot Villey
Smith	Revolver	Dean Valley
Williams	Sea View	Seattle

ft-works

emp-name	branch-name	salary
Coyote	Mesa	1500
Rabbit	Mesa	1300
Crates	Redmond	5300
Williams	Redmond	1500

emp ⟕ ft-works

emp-name	street	city	branch-name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrot Villey	Mesa	1300
Smith	Revolver	Death Valley	null	null
Williams	Seaview	Seattle	Redmund	1500

Right outer join (⟖).

It is symmetric to left outer join. It pads the tupples from the right relation those that were not matched with left relation and pads with null and adds them to the result of natural join. Thus all information from the right relation is present in the result of right outer join.

e.g.

emp_name	street	city	branch_name	salary.
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrot valley	Mesa.	1300.
Gates	null	null.	Redmond	5300
Williams	Seaview	Seattle.	Redmond	1500.

In cartesian product we will not get the distinct one. For that 'distinct' keyword is used, but in join no data is duplicated.

Full outer join (⟗)

It does both of left and right outer join, padding tupples from left relation, those were not matched by from right relation, as well as tupples from right relation those didn't match any from left relation. All are padded with null and adding them to the result of full outer join.

So,
E.g., of the same sum.

emp ⟗ ft_works.

emp_name	Street	City	branch_name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300.
Smith	Revolver	Deathvalley	null.	null.
Williams	Seaview	Seattle	Redmond	1500
Crates.	null.	null.	Redmond.	5300

Theta join (θ). i.e. ~~Cond~~ $A \bowtie_{\theta} B$. θ = condition.

θ is a comparison operator which compares with these operators i.e. {=, <, >, ≤, ≥, ≠}. A join operation with such a general join condition is called Theta join. Tupples whose join attributes are null or for which the join condition is false, do not appear in the result set.

Inner join

The join operation is used to combine data from multiple relations, so that related information can be presented in a single table. These operations are known as Inner join. It is a type of match and merge operation defined formally as a cartesian product and selection.

Equi Join

The most common use of join involves join conditions with equality comparisons only. Such a join where the only comparison operator is '=' ~~equal to~~ called an Equi Join. Here we always remember, the result of an equi join always have one or more pairs of attributes that have identical values in every tupple.