1 Relational Algebra (RA)

Relation (R): Schema that has a set of attributes A_k. Then:

$$R = A_1, A_2, \ldots, A_n \tag{1}$$

Tuple (t): Set of pairs $(attribute, value)$ where each of them gives the value of an attribute A_k for a value v_k of domain(A_k) for $k = 1, 2, \ldots, n$.

1.1 Projection (π)

Select all t and some attributes $A_1, A_2, ..., A_n$ from a relation R. Then,

$$\pi_{A_1,A_2,...,A_n}(R) = \{t[A_1, A_2, ..., A_n] \ : t \in R\} \tag{2}$$

RA	SQL
$\pi_{A_1,A_2,...,A_n}(R)$	select $A_1, A_2, ..., A_n$ from R

Table 1: Equivalence RA and SQL

1.2 Selection (σ)

Select all tuples t that satisfies the condition in the relation R. Then,

$$\sigma_{condition}(R) = \{t \in R : condition(t) \quad is \quad true\} \tag{3}$$

RA	SQL
$\sigma_{condition}(R)$	select $*$ from R where *condition*

Table 2: Equivalence RA and SQL

1.3 Composition (π) and (σ)

Select attributes $A_1, A_2, ..., A_n$ and all tuples t that satisfies the condition from a relation R. Then,

$$\pi_{A_1,A_2,...,A_n}(\sigma_{condition}(R)) = \{t \in R : t[A_1, A_2, ..., A_n] \ \ \& \ \ condition(t) \quad is \quad true\} \tag{4}$$

RA	SQL
$\pi_{A_1,A_2,...,A_n}(\sigma_{condition}(R))$	select $A_1, A_2, ..., A_n$ from R where *condition*

Table 3: Equivalence RA and SQL

1.4 Tuples without duplicate information (δ)

Select all tuples t that satisfies the condition and $t_a \neq t_c$ from a relation R. Then,

RA	SQL
$\delta(R)$	select DISTINCT $*$ from R

Table 4: Equivalence RA and SQL

1.5 Cartesian Product (x)

Set of tuples obtained when we combine two relation A, B where the tuples $a \in A$ and $b \in B$. Then,

$$AxB = \{(a, b) : a \in A \ \ \& \ \ b \in B\} = (a_1, b_1), ..., (a_m, b_1), ..., (a_m, b_2), ..., (a_m, b_n) \tag{5}$$

RA	SQL
AxB	select $*$ from A,B select $*$ from A cross join B

Table 5: Equivalence RA and SQL. Take account that we go to obtain $m * n$ tuples

1.6 Inner Join or Join ($\bowtie_k$)

Combines two relations A, B by an attribute that has different name ($A_{.name1}, B_{.name2}$) and same value ($A_n = B_m$). If the value appears in only one table then the tuple is not taken account.

RA	SQL
$\sigma_{A.name1=B.name2}(AxB)$ $A \bowtie_{A.name1=B.name2} B$	select $*$ from A INNER JOIN B ON $A.name1 = B.name2$ select $*$ from A JOIN B ON $A.name1 = B.name2$ select $*$ from A, B where $A.name1 = B.name2$

Table 6: Equivalence RA and SQL.

1.7 Natural Join ($\bowtie$)

Combines two relations A, B by attributes that has same name ($A_{.name} = B_{.name}$) and same value ($A_n = B_m$). If the value appears in only one table then the tuple is not taken account.

RA	SQL
$\sigma_{A.name=B.name}(AxB)$ $A \bowtie B$	select $*$ from A NATURAL JOIN B

Table 7: Equivalence RA and SQL. Be careful if there are more than one attribute with the same name.

1.8 Left Join ($A ⟕_k B$)

Combines two relations A, B by attributes that has different name ($A_{.name1} = B_{.name2}$) and same value ($A_n = B_m$). If the value appears in only table A then the other values go to be null.

RA	SQL
$A ⟕_{A.name1=B.name2} B$	select $*$ from A LEFT JOIN B on $A.name1 = B.name2$ select $*$ from A LEFT OUTER JOIN B on $A.name1 = B.name2$

Table 8: Equivalence RA and SQL.

1.9 Right Join ($A ⟖_k B$)

Combines two relations A, B by attributes that has different name ($A_{.name1} = B_{.name2}$) and same value ($A_n = B_m$). If the value appears in only table B then the other values go to be null.

RA	SQL
$A ⟖_{A.name1=B.name2} B$	select $*$ from A RIGHT JOIN B on $A_{.name1} = B_{.name2}$ select $*$ from A RIGHT OUTER JOIN B on $A_{.name1} = B_{.name2}$

Table 9: Equivalence RA and SQL.

1.10 Full Join ($A ⟗_k B$)

Combines two relations A, B by attributes that has different name ($A_{.name1} = B_{.name2}$) and same value ($A_n = B_m$). If the value appears in only table A then the other values go to be null and if the value appears in only table B then the other values go to be null.

RA	SQL
$A ⟗_{A.name1=B.name2} B$	select $*$ from A FULL OUTER JOIN B on $A_{.name1} = B_{.name2}$ select $*$ from A FULL JOIN B on $A_{.name1} = B_{.name2}$

Table 10: Equivalence RA and SQL.

1.11 Rename (ρ)

Variable used to rename a relation $\rho_{new\_name}(R)$ or rename an attribute $\rho_{new\_name,(A_1,A_2,...,A_n)}(R)$ where $A_1, A_2, ..., A_n$ could be new names.

RA	SQL
$\rho_{R1}(R)$	select $*$ from R AS $R1$
$\rho_{R2(AA_1,AA_2,...,AA_n)}(R)$	select $A1$ AS AA1, ..., An AS AAn from R AS $R2$

Table 11: Equivalence RA and SQL.

1.12 Union ($\cup$)

If we have two relations A, B with same *length* and *types* where $type_{A_m} = type_{B_m}$. Then, $R_{C_1,...,C_m} := A \cup B$.

RA	SQL
$A \cup B$	select $*$ from A UNION select $*$ from B;

Table 12: Equivalence RA and SQL.

1.13 Intersection ($\cap$)

If we have two relations A, B with same *length* and *types* where $type_{A_m} = type_{B_m}$. Then, $R_{C_1,...,C_m} := A \cap B$.

RA	SQL
$A \cap B$	select $*$ from A INTERSECT select $*$ from B;

Table 13: Equivalence RA and SQL.

1.14 Difference (-)

If we have two relations A, B with same *length* and *types* where $type_{A_m} = type_{B_m}$. Then, $R_{C_1,...,C_m} := A - B$.

RA	SQL
$A \cap B$	select $*$ from A EXCEPT select $*$ from B;

Table 14: Equivalence RA and SQL.

1.15 Division ($\div$)

If we have two relations A, B where $A \cap B$ and B have the attributes $A_t = [A_{l+1}, A_{l+2}, ..., A_m]$. Then, $A \div B$ returns tuples with attributes $A_{sol} = A - A_t$ where for each tuple of B there are a same tuple in $A \cap B$ with same tuple in A_{sol}.

If we have two tables A, B where the name of the attributes of A are 1,2 and the name for attribute B is 2. Then, $A \div B$ in the relational algebra could be found using select $A.1$ from A, B where $A.2 = B.2$ group by $A.1$ having $count(*) =$ (select $count(*)$ from B);

If we have two tables A, B where the name of the attributes of A are 1,2,3,4, and the name for attributes B are 3,4. Then, $A \div B$ in the relational algebra could be found using

select $A.1, A.2$ from A, B where $A.3 = B.3$ and $A.4 = B.4$ group by $A.1, A.2$ having $count(*) =$ (select $count(*)$ from B);

Using this deduction, we could use the function "divide." that returns a "div" temporal table. Then, we could obtain a division of two tables with

RA	SQL
$A \div B$	select DIVIDE('A','B'); select $*$ from DIV;

Table 15: Equivalence RA and SQL.

1.16 Assignation (:=)

It gives a relational expression name. For instance: $R' := \pi_{A_1}(R)$.

RA	SQL
$R := R1$	alter table R to $R1$
$R_{A1} := R_{AA1}$	alter table R rename column A1 to AA1;

Table 16: Equivalence RA and SQL.

1.17 Aggregation Function ($\mathscr{F}$)

$\mathscr{F}_{function\_name(A_1,A_2,...,An)}(R)$ execute a function over attributes $A_1, A_2, ..., A_n$ into a R relation.

RA	SQL
$\mathscr{F}_{function\_name(A_1,A_2,...,An)}(R)$	select $Function\_Name$ $(A_1, A_2, ..., A_n)$ from R;

Table 17: Equivalence RA and SQL.

RA FUNCTIONS	SQL FUNCTIONS	Description
$\mathscr{F}_{max(A_1)}(R)$	select max (A_1) from R;	maximum value of tuples
$\mathscr{F}_{min(A_1)}(R)$	select min (A_1) from R;	minimum value of tuples
$\mathscr{F}_{count(A_1)}(R)$	select $count$ (A_1) from R;	tuples sum
$\mathscr{F}_{avg(A_1)}(R)$	select avg (A_1) from R;	tuples average
$\mathscr{F}_{concat(A_1,`\quad',A_2)}(R)$	select $concat$ $(A_1, `\quad', A_2)$ from R;	tuples concatenated
$\mathscr{F}_{(A_1\|\|`\quad'\|\|A_2)}(R)$	select $A_1\|\|`\quad'\|\|A_2$ from R;	tuples concatenated
$\mathscr{F}_{generate\_series(1,5)}(R)$	select $generate\_series$ $(1, 5)$;	tuples with int numbers 1:5
$\mathscr{F}_{generate\_series(1,5)}(R)$	select $*$ from $generate\_series$ $(1, 5)$;	tuples with int numbers 1:5

Table 18: Equivalence RA and SQL.

2 Query Optimization

2.1 Sigma Cascade

$$\sigma_{c_1\&c_2\&\ldots\&c_n}(R) = \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots)) \tag{6}$$

2.2 Commutative of Sigma

$$\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R)) \tag{7}$$

2.3 Pi Cascade

$$\pi_1(\pi_2(\ldots(\pi_n(R))\ldots)) = \pi_1(R) \tag{8}$$

2.4 Commutativity of Sigma by Pi

$$\pi_{A_1,A_2,\ldots,A_n}(\sigma_c(R)) = \sigma_c(\pi_{A_1,A_2,\ldots,A_n}(R)) \tag{9}$$

2.5 Commutativity of Join

$$R \bowtie_k S = S \bowtie_k R \tag{10}$$

2.6 Commutativity of Product

$$Rx_kS = Sx_kR \tag{11}$$

2.7 Commutativity of Sigma by Join (or Product)

$$\sigma_c(R \bowtie S) = (\sigma_c(R)) \bowtie S \tag{12}$$

2.8 Commutativity of Pi by Join (or Product)

$$\pi_L(R \bowtie_k S) = (\pi_{A_1,\ldots,A_n}(R)) \bowtie_k (\pi_{B_1,\ldots,B_m}(S)) \tag{13}$$

2.9 Associativity of Θ = Join, Product, Union, or Intersection

$$(R\Theta S)\Theta T = R\Theta(S\Theta T) \tag{14}$$

2.10 Commutation of Sigma with Set Operations $\Theta = \cap, \cup$,or $-$

$$\sigma_c(R\Theta S) = (\sigma_c(R))\Theta(\sigma_c(S)) \tag{15}$$

2.11 Pi Operation Can Commute with $\cup$

$$\pi_L(R \cup S) = (\pi_L(R)) \cup (\pi_L(S)) \tag{16}$$

2.12 Converting a sequence (Sigma,Product) in Join

$$(\sigma_c(RxS)) = (R \bowtie_k S) \tag{17}$$

2.13 Heuristic Optimization Algorithm Steps

(i) Using rule (6), decompose any operation of SELECTION with conjunctive conditions in a cascade using SELECTION operations, which allows a greater degree of freedom to move selection operations down the different branches of the tree.

(ii) Using rules (7), (9), (12), and (16) relating to the commutativity of SELECTION with other operations, move each SELECTION operation down the tree as far as the attributes included in the SELECTION condition allow.

(iii) Using rules (10), (11), and (14) concerning to commutativity and association of binary operations, reorder the relationships of the leaf nodes using the following criteria:

- First, position the relationships of the leaf nodes with the most restrictive SELECTION operations (which generate a relationship with the smallest number of tuples or with the smallest absolute size), in such a way that they are executed first in the query tree representation.
- Second, make sure that sorting the leaf nodes does not produce any Cartesian products.

(iv) Using rule (17), combine a Cartesian product operation with the following tree SELECTION operation to form a JOIN operation, in the case that the condition represents a JOIN condition.

(v) Using rules (8), (9), (13), and (16) relating to the sequence of PROJECTIONS and their switching with other operations. Decompose and move projection attribute lists to low through the tree as far as possible by creating new projection operations as needed.

(vi) Identify sub-trees that represent groups of operations that can be executed using a single algorithm.

2.14 Example of Optimization

If we have 3 relations: EMPLOYEE, WORK_IN, and PROJECT.

EMPLOYEE				
name	lastName	dni	birthdate	adress

WORK_IN		
dniEmployee	numProj	hours

PROJECT		
projectName	numProject	ubicationProject

We could reduce size of names to work easily. Then:

- `EMPLOYEE := E`
- `WORK_IN := W`
- `PROJECT := P`
- `lastName := ln`
- `projectName := pn`
- `numProj := np`
- `numProject := np`
- `dniEmp := dni`
- `dni := dni`
- `bornDate := bd`

If we have a query in relational algebra like this:

$$\pi_{\texttt{lastname}}(\sigma_{\texttt{projectName='Aquarious' and numProj=numProject and dniEmp=dni and bornDate>'1957-12-31'}})$$
$$(\texttt{EMPLOYEE}\ x\ \texttt{WORK\_IN}\ x\ \texttt{PROJECT}) \quad (18)$$

Using reduced name we have that equation (18) = (19):

$$\pi_{\texttt{E.ln}}(\sigma_{\texttt{P.pn='Aquarious' and P.np=W.np and E.dni=W.dni and E.bd>'1957-12-31'}})(\texttt{E}\ x\ \texttt{W}\ x\ \texttt{P}) \quad (19)$$

We could optimize firstly using the rule 6 and (.1 (i)) heuristic rule :

$$= \pi_{\texttt{E.ln}}(\sigma_{\texttt{P.pn='Aquarious'}}(\sigma_{\texttt{P.np=W.np}}(\sigma_{\texttt{E.dni=W.dni}}(\sigma_{\texttt{E.bd>'1957-12-31'}}(\texttt{E}\ x\ \texttt{W}\ x\ \texttt{P})))))$$

Second, using the rule (12) and .1 (ii) heuristic rule :

$$= \pi_{\texttt{E.ln}}(\sigma_{\texttt{P.pn='Aquarious'}}(\sigma_{\texttt{P.np=W.np}}(\sigma_{\texttt{E.dni=W.dni}}(\sigma_{\texttt{E.bd>'1957-12-31'}}(\texttt{E})\ x\ \texttt{W}\ x\ \texttt{P}))))$$

$$= \pi_{\texttt{E.ln}}(\sigma_{\texttt{P.np=W.np}}(\sigma_{\texttt{E.dni=W.dni}}(\sigma_{\texttt{E.bd>'1957-12-31'}}(\texttt{E})\ x\ \texttt{W}\ x\ \sigma_{\texttt{P.pn='Aquarious'}}(\texttt{P}))))$$

Third, using the rule (17) and .1 (iv) heuristic rule :

$$= \pi_{\texttt{E.ln}}(\sigma_{\texttt{P.np=W.np}}(\sigma_{\texttt{E.bd>'1957-12-31'}}(\texttt{E}) \bowtie_{E.dni=W.dni} \texttt{W})\ \texttt{x}\ \sigma_{\texttt{P.pn='Aquarious'}}(\texttt{P}))$$

$$= \pi_{\texttt{E.ln}}((\sigma_{\texttt{E.bd>'1957-12-31'}}(\texttt{E}) \bowtie_{E.dni=W.dni} \texttt{W}) \bowtie_{\texttt{P.np=W.np}} \sigma_{\texttt{P.pn='Aquarious'}}(\texttt{P}))$$

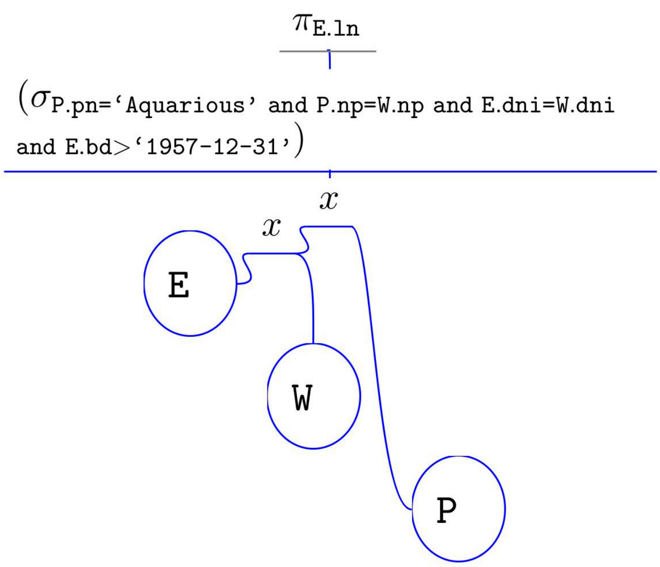
If we know that cost of operations is smaller changing the order we could apply the rule (14) and .1 (iii) heuristic rule :

$$= \pi_{\texttt{E.ln}}((\sigma_{\texttt{P.pn='Aquarious'}}(\texttt{P}) \bowtie_{P.np=W.np} \texttt{W}) \bowtie_{\texttt{E.dni=W.dni}} \sigma_{\texttt{E.bd>'1957-12-31'}}(\texttt{E}))$$

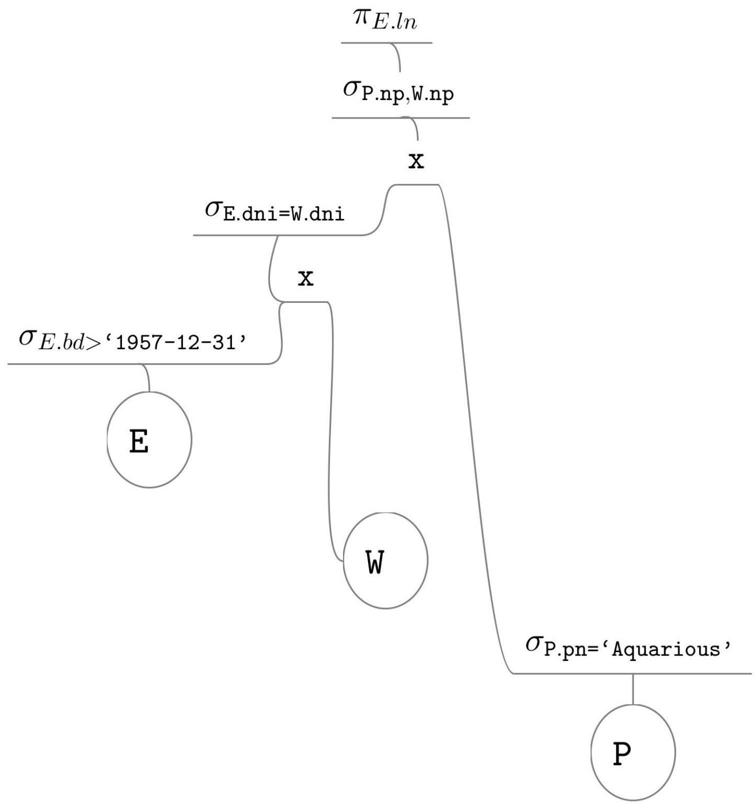
Finally, we use the .1 (v) heuristic rule

$$= \pi_{\texttt{E.ln}}((\pi_{\texttt{P.np}}(\sigma_{\texttt{P.pn='Aquarious'}}(\texttt{P})) \bowtie_{P.np=W.np} \pi_{W.dni,W.np}(\texttt{W})) \bowtie_{\texttt{E.dni=W.dni}} \pi_{E.ln,E.dni}(\sigma_{\texttt{E.bd>'1957-12-31'}}(\texttt{E})))$$

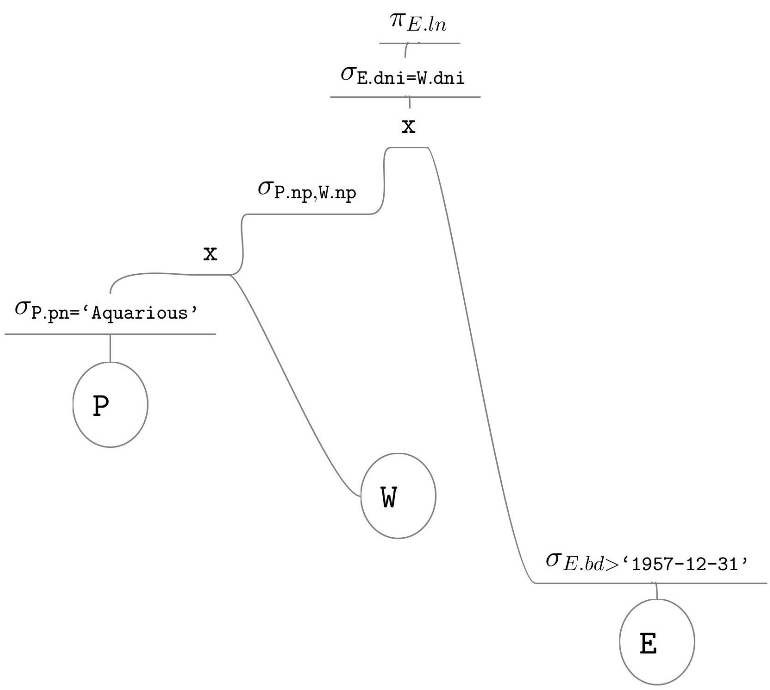
On the other hand, we could resolve this problem with help of graphics. Then, first we use the equation (19) to obtain the first graph:



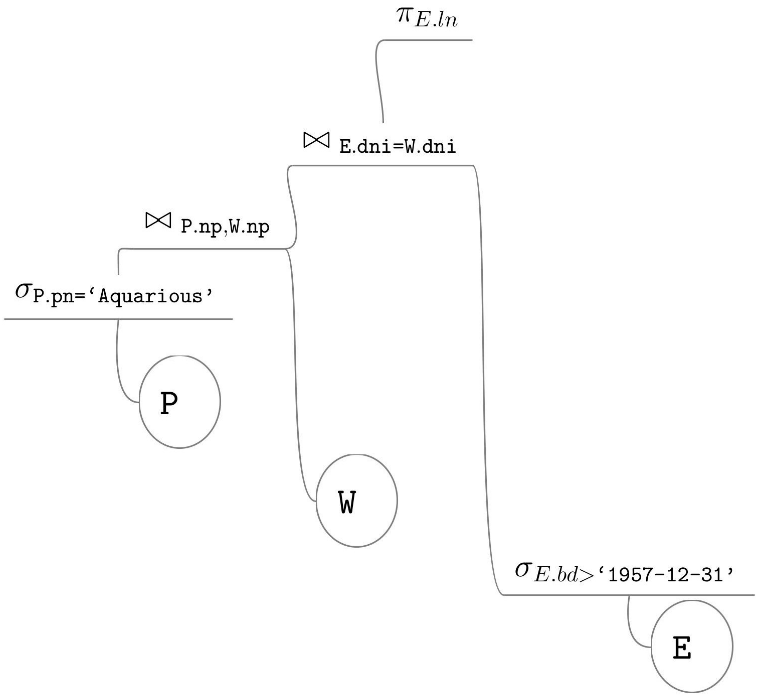
Using the heuristics rules .1 (i) and .1 (ii) we obtain:



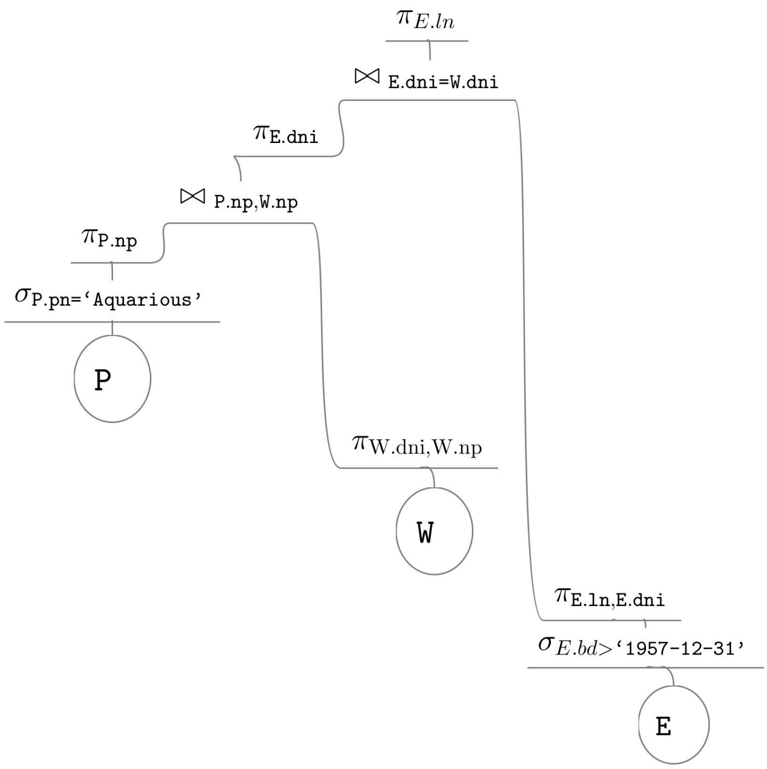
Using the heuristics rules .1 (iii)



Using the heuristics rules .1 (iv)



Using the heuristics rules .1 (v)



Then, our final result is:

$$\pi_{\texttt{E.ln}}((\pi_{\texttt{P.np}}(\sigma_{\texttt{P.pn='Aquarious'}}(\texttt{P})) \bowtie_{P.np=W.np} \pi_{W.dni,W.np}(\texttt{W})) \bowtie_{\texttt{E.dni=W.dni}} \pi_{E.ln,E.dni}(\sigma_{\texttt{E.bd>'1957-12-31'}}(\texttt{E})))$$