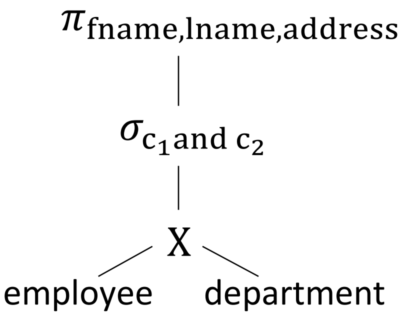
```
=========================================================

Name: Ernesto Diaz

Panther-ID: x x x - 0534

Course: COP 4722

Assignment#: 1

Due: Wed, Feb 7, 2018

I hereby certify that this work is my own and none of
it is the work of any other person.

                        Signature: ______________________
==========================================================
```

Q1: Retrieve the name and address of all employees who work for the 'Administration' department

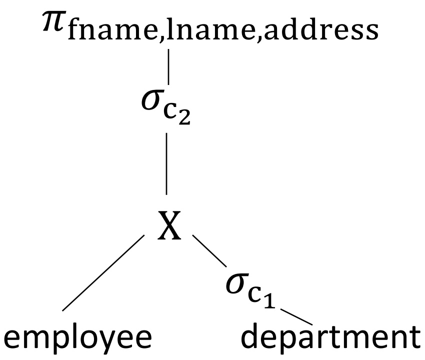
```
SELECT FName, LName, Address
FROM Employee, department
WHERE DName = 'Administration' and DNumber = DNO ;
```

C_1 → DName = 'Administration'
C_2 → DNumber = DNO

Initial Query Tree


$\pi_{\text{fname,lname,address}}$
$\sigma_{c_1 \text{ and } c_2}$
X
employee
department

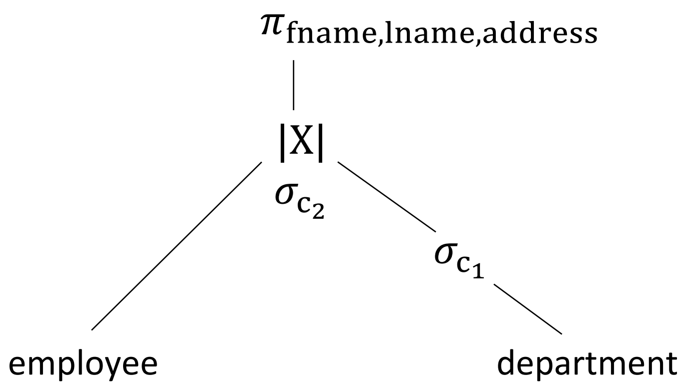
Step 1



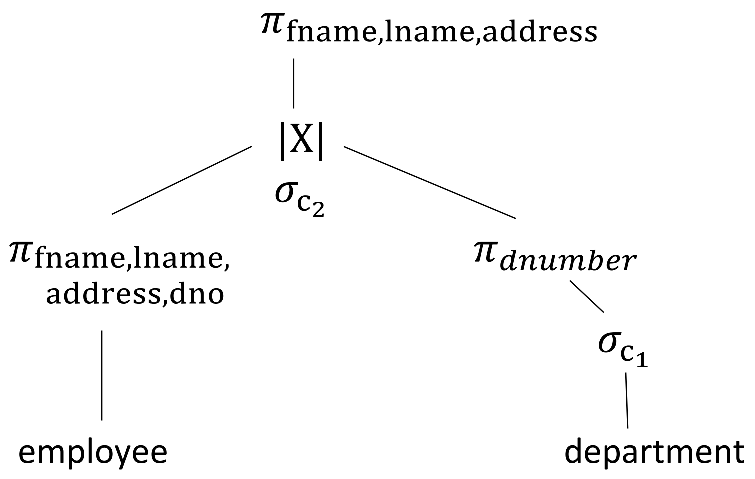
Step 2

No restrictive select conditions and associative joins to swap.

Step 3



Step 4


$\pi_{\text{fname,lname,address}}$
|X|
σ_{c_2}
$\pi_{\text{fname,lname, address,dno}}$
$\pi_{dnumber}$
σ_{c_1}
employee
department

Final Query

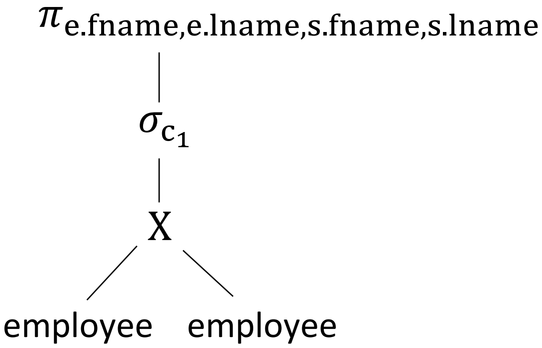
```
SELECT fname, lname, address
FROM (SELECT fname, lname, address, dno
      FROM employee) as L
JOIN
     (SELECT dnumber
      FROM department
      WHERE dname = 'Administration') as R
on L.dno = R.dnumber;
```

Q2: For each employee, retrieve the employee's first and last name and the first and last name of his/her immediate supervisor.

```
SELECT e.FName, e.LName, s.FName, s.LName
FROM Employee e, Employee s
WHERE e.SuperSSN = s.SSN ;
```

C_1 → e.SuperSSN = s.SSN

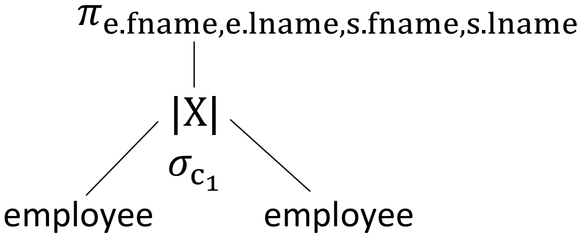
Initial Query Tree



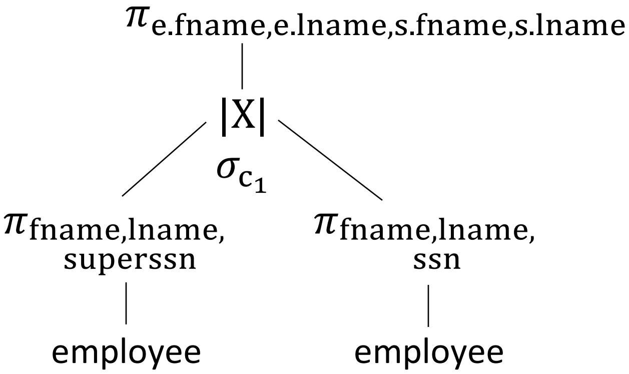
Step 1
No select conditions to move down the tree

Step2
No restrictive select conditions and associative joins to swap.

Step 3



Step 4


$\pi_{e.fname,e.lname,s.fname,s.lname}$
|X|
σ_{c_1}
$\pi_{fname,lname, superssn}$
$\pi_{fname,lname, ssn}$
employee
employee

Final Query

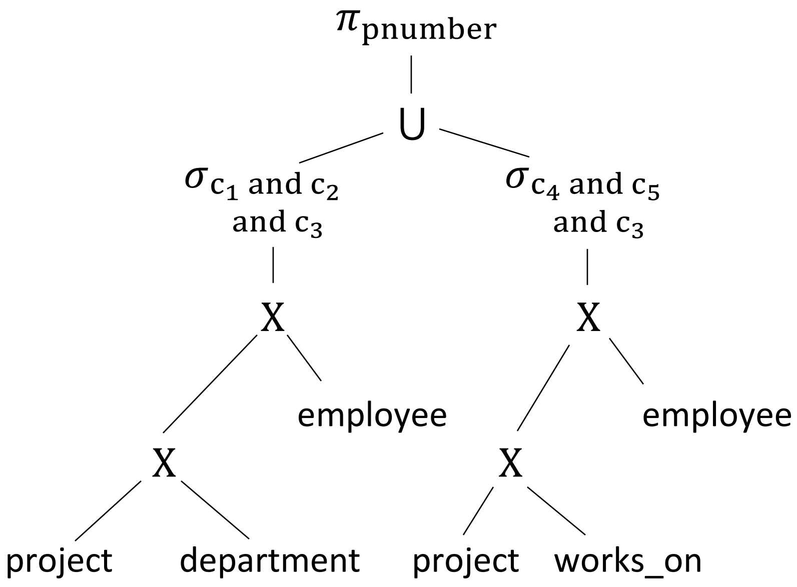
```
SELECT E.fname, E.lname, S.fname, S.lname
from (SELECT fname, lname, superssn
      FROM employee) as E
JOIN
     (SELECT fname, lname, ssn
      FROM employee) as S
on E.superssn = S.ssn;
```

Q3: Make a list of all project numbers for projects that involve an employee whose last name is 'Wong', either as a worker or as a manager of the department that controls the project.

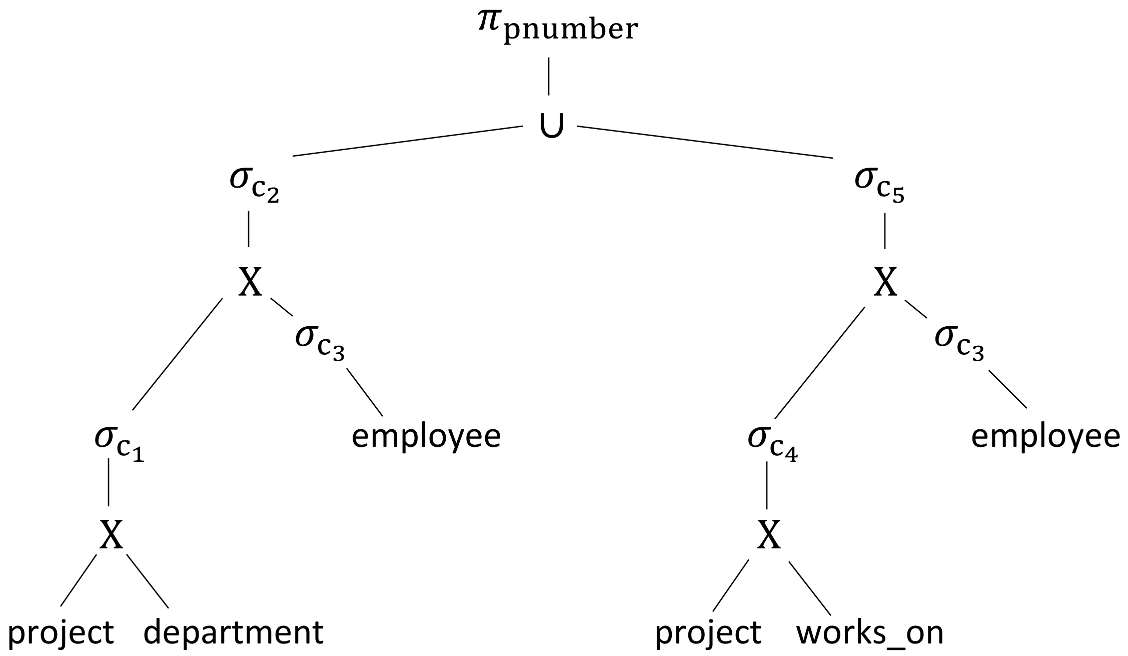
```
(SELECT Distinct PNumber
 FROM Project, Department, Employee
 WHERE DNum = DNumber and MgrSSN = SSN and LName = 'Wong')
UNION
(SELECT Distinct PNumber
 FROM project, Works_On, Employee
 WHERE PNumber = PNO and ESSN = SSN and LName = 'Wong' );
```

C_1 → `DNum = DNumber`
C_2 → `MgrSSN = SSN`
C_3 → `LName = 'Wong'`
C_4 → `PNumber = PNO`
C_5 → `ESSN = SSN`

Initial Query Tree

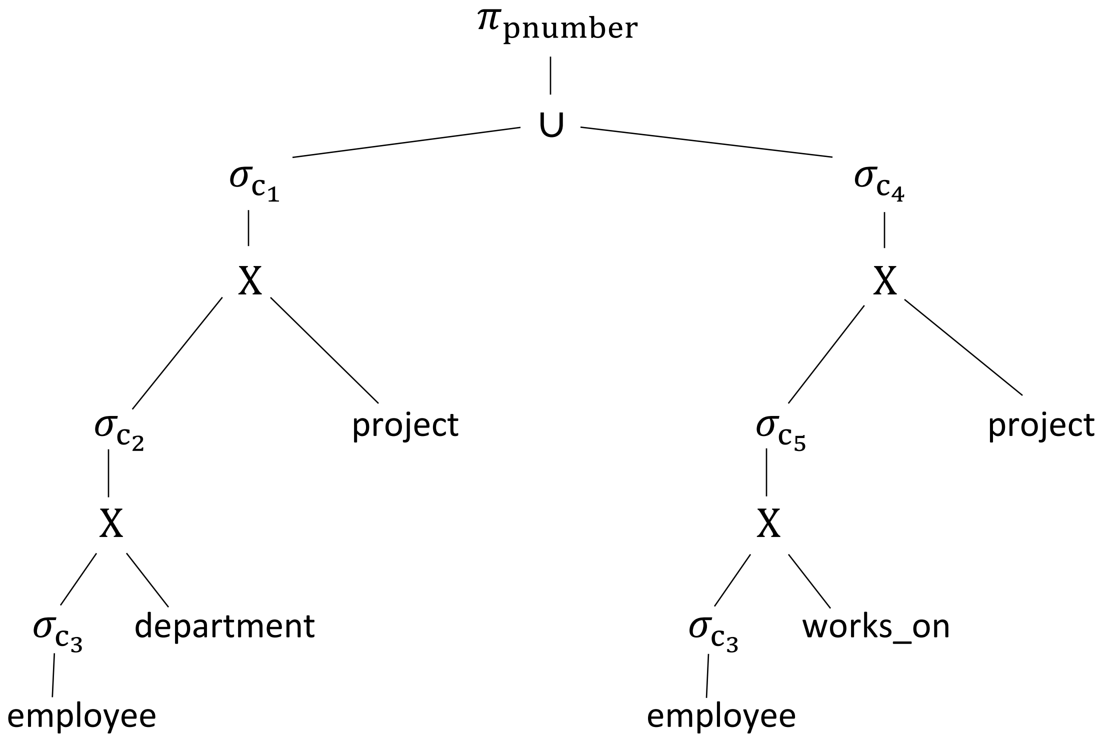

π_{pnumber}
U
$\sigma_{c_1 \text{ and } c_2 \text{ and } c_3}$
$\sigma_{c_4 \text{ and } c_5 \text{ and } c_3}$
X
X
employee
employee
X
X
project
department
project
works_on

Step 1



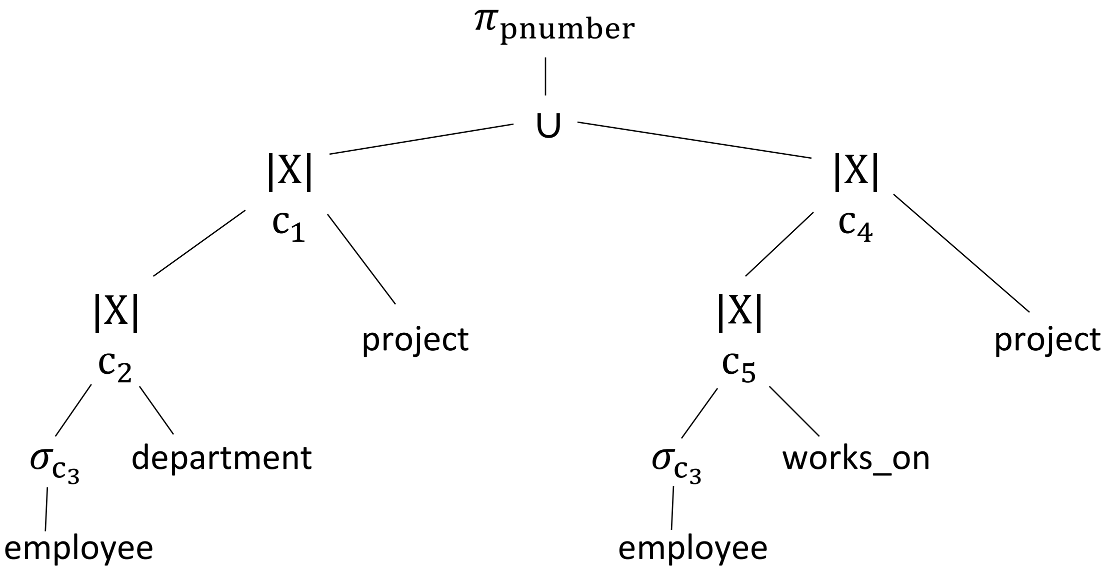
π_{pnumber}
∪
σ_{c_2}
σ_{c_5}
X
X
σ_{c_3}
σ_{c_3}
σ_{c_1}
employee
σ_{c_4}
employee
X
X
project
department
project
works_on

Step 2

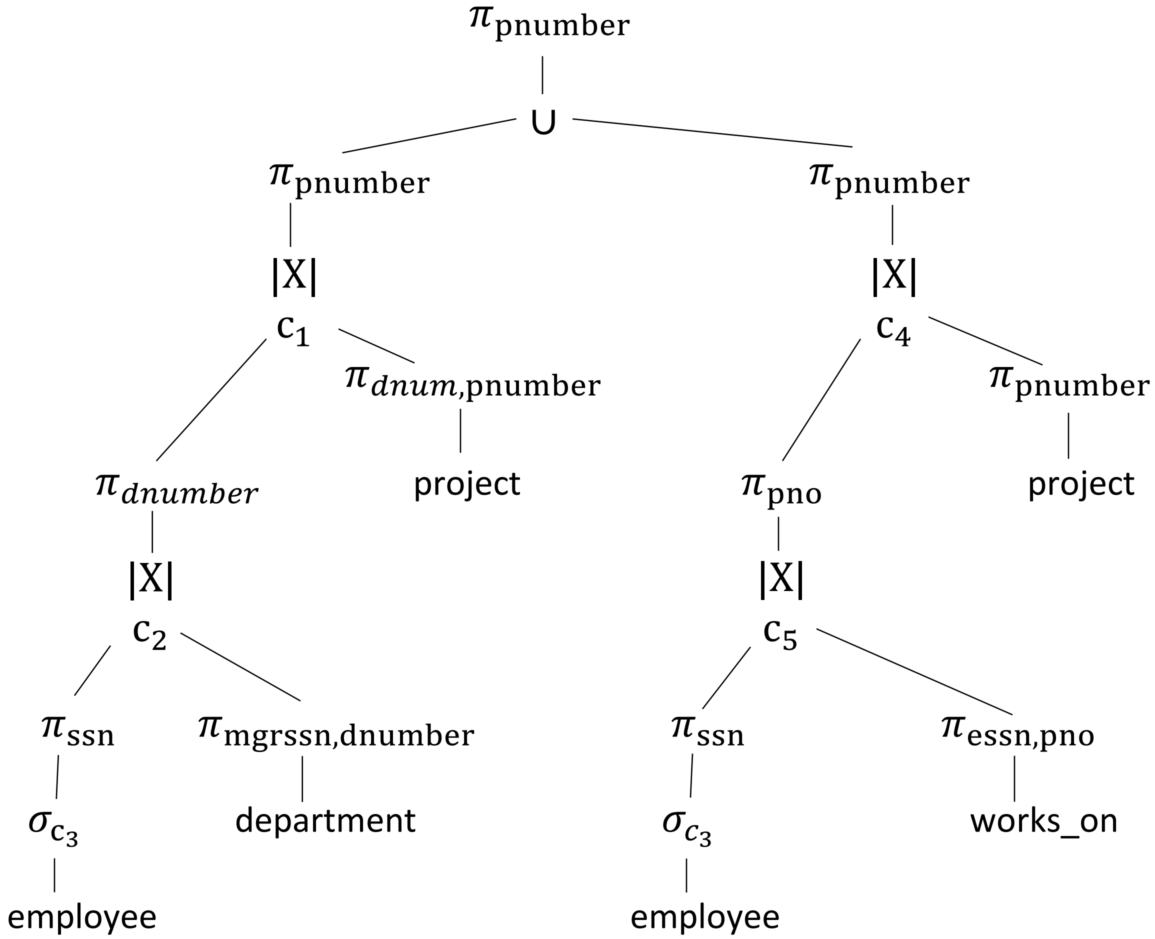


π_{pnumber}
∪
σ_{c_1}
σ_{c_4}
X
X
σ_{c_2}
project
σ_{c_5}
project
X
X
σ_{c_3}
department
σ_{c_3}
works_on
employee
employee

Step 3


π_{pnumber}
∪
|X| c_1
|X| c_4
|X| c_2
project
|X| c_5
project
σ_{c_3}
department
σ_{c_3}
works_on
employee
employee

Step 4


π_{pnumber}
∪
π_{pnumber}
π_{pnumber}
|X| c_1
|X| c_4
$\pi_{dnum,\text{pnumber}}$
π_{pnumber}
$\pi_{dnumber}$
project
π_{pno}
project
|X| c_2
|X| c_5
π_{ssn}
$\pi_{\text{mgrssn,dnumber}}$
π_{ssn}
$\pi_{\text{essn,pno}}$
σ_{c_3}
department
σ_{c_3}
works_on
employee
employee

Final Query

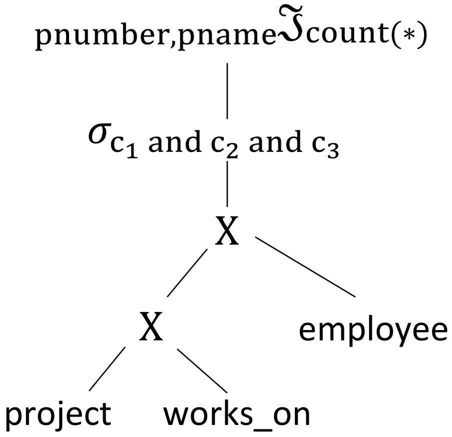
```
(SELECT pnumber
FROM ((SELECT dnum, pnumber
      FROM project) as L3
JOIN
     (SELECT dnumber
      FROM (SELECT ssn
            FROM employee
            WHERE lname = 'Wong') as L1
      JOIN
           (SELECT mgrssn, dnumber
            FROM department) as L2
      on L1.ssn = L2.mgrssn) as L1andL2
on L3.dnum = L1andL2.dnumber))

UNION

(SELECT pnumber
FROM ((SELECT pnumber
      FROM project) as R3
JOIN
     (SELECT pno
      FROM (SELECT ssn
            FROM employee
            WHERE lname = 'Wong') as R1
      JOIN
           (SELECT essn, pno
            FROM works_on) as R2
      on R1.ssn = R2.essn) as R1andR2
on R3.pnumber = R1andR2.pno));
```

Q4: For each project, retrieve the project number, the project name, and the number of employees from department 4 who work on the project.

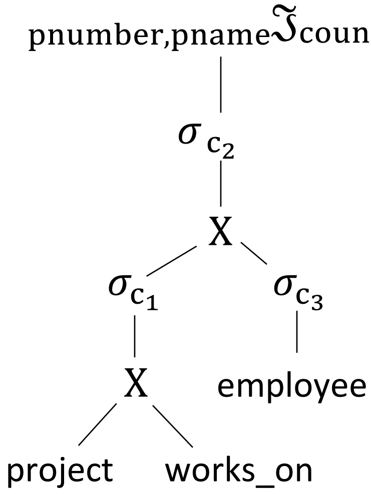
```
    SELECT PNumber, PName, COUNT(*)
    FROM Project, Works_On, Employee
    WHERE PNumber = PNO and SSN = ESSN and DNO = 4
    GROUP BY PNumber, PName;
```

C_1 → PNumber = PNO
C_2 → SSN = ESSN
C_3 → DNO = 4

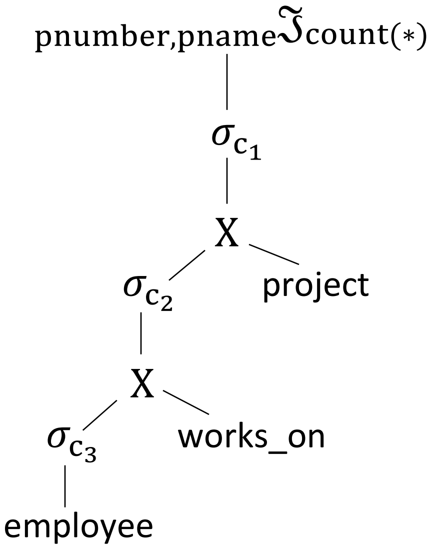
Initial Query Tree


$_{\text{pnumber,pname}}\mathfrak{I}_{\text{count}(*)}$
$\sigma_{\text{c}_1 \text{ and c}_2 \text{ and c}_3}$
X
X
employee
project
works_on

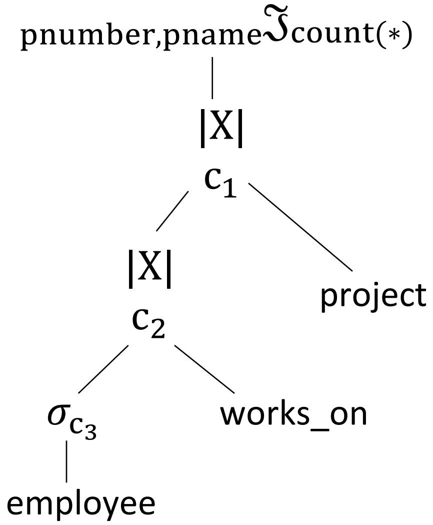
Step 1



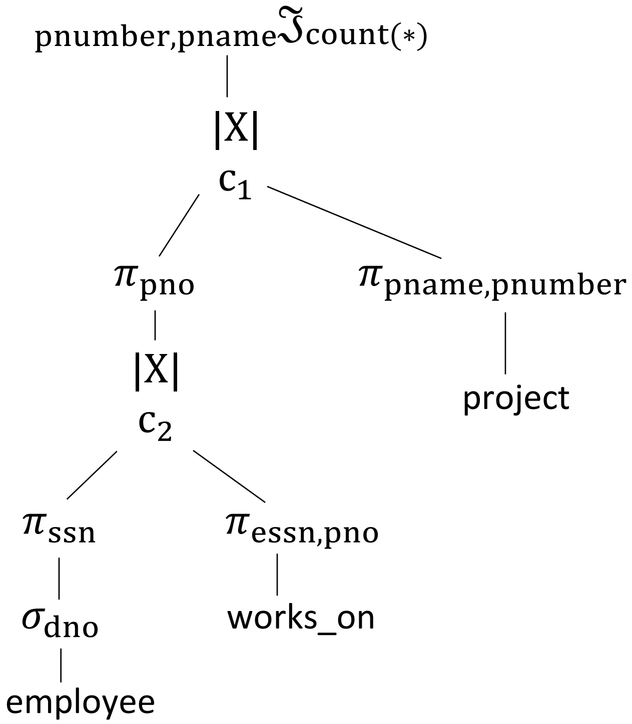
Step 2



Step 3



Step 4



Final Query

```
SELECT pnumber, pname, count(*)
FROM (SELECT pno
      FROM (SELECT ssn
            FROM employee
            WHERE dno = 4) as L1
      JOIN
          (SELECT essn, pno
           FROM works_on) as L2
      on L1.ssn = L2.essn) as L
JOIN
   (SELECT pname, pnumber
    FROM project) as R
on L.pno = R.pnumber
GROUP BY pnumber, pname
```

Q5: Extend the sort-merge join algorithm to implement the left outer join operation.

```
sort the tuples in R on attribute A; // assume R has n tuples (records)
sort the tuples in S on attribute B; // assume S has m tuples (records)

i = 1; // initialize the record pointer of table R
j = 1; // initialize the record pointer of table S

while ( (i <= n) && (j <= m) ) {
        if (R[i].A > S[j].B) {
                j++; // advance the record pointer of S;
        }
        else if (R[i].A < S[j].B ) {
                //output NULL for the attributes that find no match for the
                //left table
                output the combined tuple <R[p],NULL> to T;
                i++; // advance the record pointer of R
        }
        else { // R[i].A == S[j].B, so we output all matched pairs of tuples

                p = i; // p is the auxillary record pointer of table R
                while ( (p <= n) && (R[p].A == S[j].B) ) {

                        q = j; // q is the auxillary record pointer of table S
                        while ((q <= m) && (R[p].A == S[q].B)) {
                                output the combined tuple <R[p],S[q]> to T;
                                q++;
                        }

                        p++;
                }

                //add NULL's to the rest of the left table if the right
                //table's pointer has finished before the left
                if (q == m) {
                        while (p<=n) {
                                output the combined tuple <R[p],NULL> to T;
                                p++;
                        }

                i = p; // update the primary record pointer of table R
                j = q; // update the primary record pointer of table S
        }
}
```