**Optimización de Consultas Distribuidas en PostgreSQL**

Profesor Heider Sanchez

Dada la siguiente relación:

**Colegio**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Código** | **Nombre** | **Dirección** | **Ciudad** | **NivelAcad** | **TotalAlumnos** |

Poble la tabla con datos sintéticos y luego aplique fragmentación horizontal por rango en el atributo TotalAlumnos con el vector de partición [600, 1300].

Dada las siguientes consultas de usuario final:

1. Select \* From Colegio Order By TotalAlumnos.
2. Select \* From Colegio Order By Ciudad.
3. Select \* From Colegio Order By NivelAcad.
4. Select AVG(TotalAlumnos) From Colegio.
5. Select Ciudad, SUM(TotalAlumnos) From Colegio Group By Ciudad.

**P1. Algoritmos distribuidos localmente**

Asumiendo que cada fragmento se encuentra en SERVIDORES diferentes:

* Se le pide crear las sentencias SQL necesarias para implementar el algoritmo distribuido optimizado para cada consulta dada.
* Comente cada parte de su código indicando claramente las particiones intermedias.
* Si se usan tablas temporales, estas deben eliminarse una vez entregado el resultado.
* Mostrar el gráfico del plan de ejecución resultante.

**Solución**

Particionamos colegio:

CREATE TABLE colegio (

    codigo VARCHAR(10),

    nombre VARCHAR(100),

    direccion VARCHAR(200),

    ciudad VARCHAR(100),

    nivel\_acad VARCHAR(50),

    total\_alumnos INTEGER,

  PRIMARY KEY(codigo, total\_alumnos)

) PARTITION BY RANGE (total\_alumnos);

-- Partición para TotalAlumnos < 600

CREATE TABLE colegio\_p600 PARTITION OF colegio

    FOR VALUES FROM (MINVALUE) TO (600);

-- Partición para 600 <= TotalAlumnos <= 1300

CREATE TABLE colegio\_p600\_1300 PARTITION OF colegio

    FOR VALUES FROM (600) TO (1300);

-- Partición para TotalAlumnos > 1300

CREATE TABLE colegio\_p1300 PARTITION OF colegio

    FOR VALUES FROM (1300) TO (MAXVALUE);

SELECT \* FROM colegio\_p600;

SELECT \* FROM colegio\_p600\_1300;

SELECT \* FROM colegio\_p1300;

1. Consulta a

Select \* From Colegio Order By TotalAlumnos.

--- consulta a

SELECT \*

FROM (

  (SELECT \*

  FROM colegio\_p600

  ORDER BY total\_Alumnos)

  UNION ALL

  (SELECT \*

  FROM colegio\_p600\_1300

  ORDER BY total\_Alumnos)

  UNION ALL

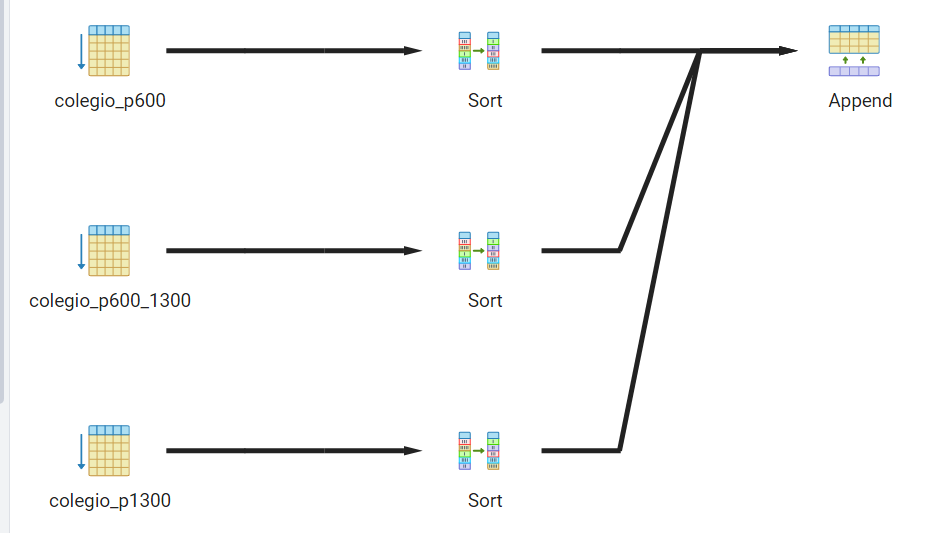
  (SELECT \*

  FROM colegio\_p1300

  ORDER BY total\_Alumnos)

) AS subquery;

Gráfico del plan de ejecución:



1. Consulta b

Select \* From Colegio Order By Ciudad.

--- consulta b

--- Partición intermedia 1: A-L

CREATE TABLE colegio\_frag\_intermedia1 AS

(

(SELECT \*

FROM colegio\_p600

WHERE ciudad >= 'A' AND ciudad <= 'L')

UNION ALL

(SELECT \*

FROM colegio\_p600\_1300

WHERE ciudad >= 'A' AND ciudad <= 'L')

UNION ALL

(SELECT \*

FROM colegio\_p1300

WHERE ciudad >= 'A' AND ciudad <= 'L')

);

--- Partición intermedia 2: M-Z

CREATE TABLE colegio\_frag\_intermedia2 AS

(

(SELECT \*

FROM colegio\_p600

WHERE ciudad >= 'M' AND ciudad <= 'Z')

UNION ALL

(SELECT \*

FROM colegio\_p600\_1300

WHERE ciudad >= 'M' AND ciudad <= 'Z')

UNION ALL

(SELECT \*

FROM colegio\_p1300

WHERE ciudad >= 'M' AND ciudad <= 'Z')

);

--- Query a partir de la fragmentación intermedia

SELECT \*

FROM (

  (SELECT \*

  FROM colegio\_frag\_intermedia1

  ORDER BY ciudad)

  UNION ALL

  (SELECT \*

  FROM colegio\_frag\_intermedia2

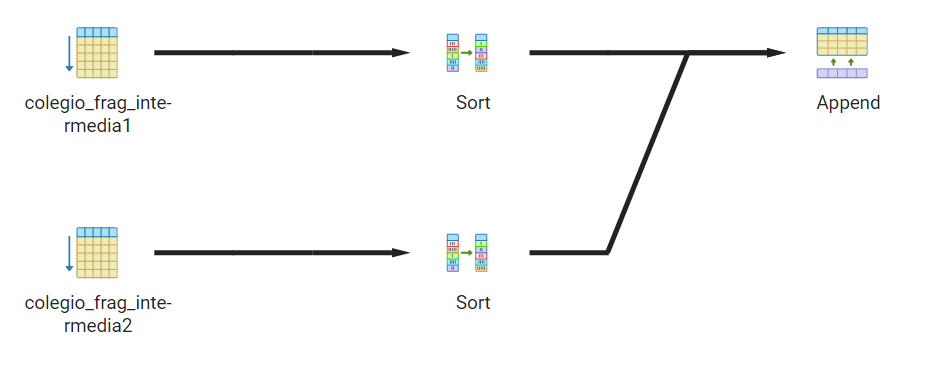
  ORDER BY ciudad)

) AS subquery

--- Borramos las tablas intermedias

DROP TABLE colegio\_frag\_intermedia1;

DROP TABLE colegio\_frag\_intermedia2;



1. Consulta c

Select \* From Colegio Order By NivelAcad.

-- c)

--- Partición intermedia 1: Nivel académico Primaria

CREATE TABLE colegio\_frag\_intermedia1 AS

((SELECT \*

FROM colegio\_p600

WHERE nivel\_acad IN ('Primaria'))

UNION ALL

(SELECT \*

FROM colegio\_p600\_1300

WHERE nivel\_acad IN ('Primaria'))

UNION ALL

(SELECT \*

FROM colegio\_p1300

WHERE nivel\_acad IN ('Primaria'))

);

--- Partición intermedia 2: Nivel académico Secundaria

CREATE TABLE colegio\_frag\_intermedia2 AS

((SELECT \*

FROM colegio\_p600

WHERE nivel\_acad IN ('Secundaria'))

UNION ALL

(SELECT \*

FROM colegio\_p600\_1300

WHERE nivel\_acad IN ('Secundaria'))

UNION ALL

(SELECT \*

FROM colegio\_p1300

WHERE nivel\_acad IN ('Secundaria'))

);

--- Query a partir de fragmentación intermedia

SELECT \*

FROM (

  (SELECT \*

  FROM colegio\_frag\_intermedia1

  ORDER BY nivel\_acad)

  UNION ALL

  (SELECT \*

  FROM colegio\_frag\_intermedia2

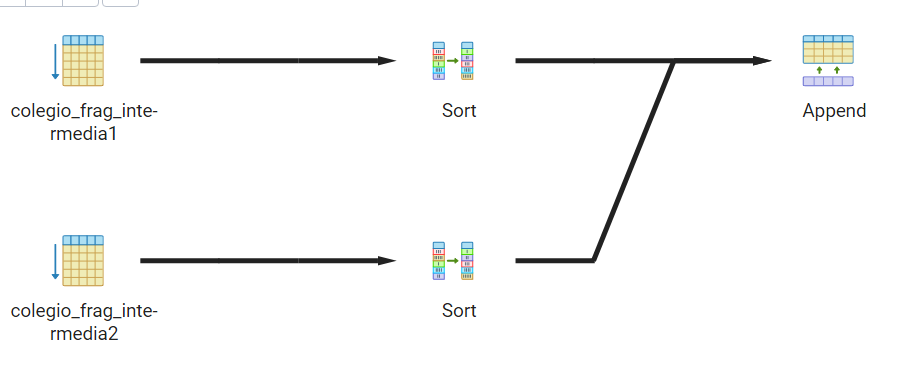
  ORDER BY nivel\_acad)

) AS subquery;

--- Eliminamos tablas intermedias

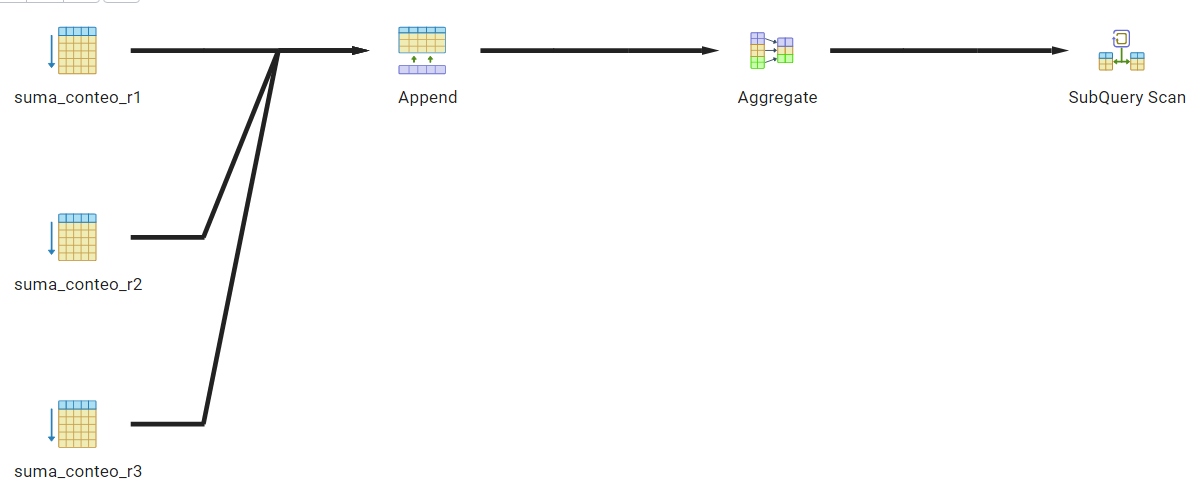
DROP TABLE colegio\_frag\_intermedia1;

DROP TABLE colegio\_frag\_intermedia2;

1. Consulta d

Select AVG(TotalAlumnos) From Colegio.

Para hallar el average, tuvimos que crear un fragmento intermedio por cada fragmento inicial que guardara cada suma y conteo correspondiente (así evitamos transmisión de tuplas). Luego, en cada fragmento intermediario “suma\_conteo\_r1”, “suma\_conteo\_r2” y “suma\_conteo\_r3” solo hay una tupla (conteo y suma de cada fragmento inicial), es decir, en total hay 3 tuplas solamente. Como eso ya no es costoso enviarlo al servidor central, hacemos UNION ALL y se lo enviamos al servidor central.



-- d

--Fragmentación intermedia 1

CREATE TABLE suma\_conteo\_R1 AS

(SELECT SUM(total\_alumnos) as suma, COUNT(total\_alumnos) as conteo

FROM colegio\_p600);

--Fragmentación intermedia 2

CREATE TABLE suma\_conteo\_R2 AS

(SELECT SUM(total\_alumnos) as suma, COUNT(total\_alumnos) as conteo

FROM colegio\_p600\_1300);

--Fragmentación intermedia 3

CREATE TABLE suma\_conteo\_R3 AS

(SELECT SUM(total\_alumnos) as suma, COUNT(total\_alumnos) as conteo

FROM colegio\_p1300);

-- Append de las 3 tuplas en el servidor central

SELECT sumaTotal/conteoTotal as promedio --transmisión de únicamente 3 tuplas al servidor central

FROM (

  SELECT SUM(suma) as sumaTotal, SUM(conteo) as conteoTotal

  FROM(

    SELECT \* FROM suma\_conteo\_R1

    UNION ALL

    SELECT \* FROM suma\_conteo\_R2

    UNION ALL

    SELECT \* FROM suma\_conteo\_R3

  ) as append\_sumas\_conteos

) as subconsulta;

--- Elimnación de fragmentos intermedios

DROP TABLE suma\_conteo\_r1;

DROP TABLE suma\_conteo\_r2;

DROP TABLE suma\_conteo\_r3;

1. Consulta e

Select Ciudad, SUM(TotalAlumnos) From Colegio Group By Ciudad.

--- Partición intermedia 1: A-L

CREATE TABLE colegio\_frag\_intermedia1 AS

(

(SELECT \*

FROM colegio\_p600

WHERE ciudad >= 'A' AND ciudad <= 'L')

UNION ALL

(SELECT \*

FROM colegio\_p600\_1300

WHERE ciudad >= 'A' AND ciudad <= 'L')

UNION ALL

(SELECT \*

FROM colegio\_p1300

WHERE ciudad >= 'A' AND ciudad <= 'L')

);

--- Partición intermedia 2: M-Z

CREATE TABLE colegio\_frag\_intermedia2 AS

(

(SELECT \*

FROM colegio\_p600

WHERE ciudad >= 'M' AND ciudad <= 'Z')

UNION ALL

(SELECT \*

FROM colegio\_p600\_1300

WHERE ciudad >= 'M' AND ciudad <= 'Z')

UNION ALL

(SELECT \*

FROM colegio\_p1300

WHERE ciudad >= 'M' AND ciudad <= 'Z')

);

--- Como están fragmentando por rango, puedo agrupar sin generar grupos repetidos en ambos fragmentos

SELECT ciudad, suma\_total --servidor central solo aplicaría suma

FROM (

  (SELECT ciudad, sum(total\_alumnos) as suma\_total

  FROM colegio\_frag\_intermedia1

  GROUP BY ciudad)

  UNION ALL

  (SELECT ciudad, sum(total\_alumnos) as suma\_total

  FROM colegio\_frag\_intermedia2

  GROUP BY ciudad)

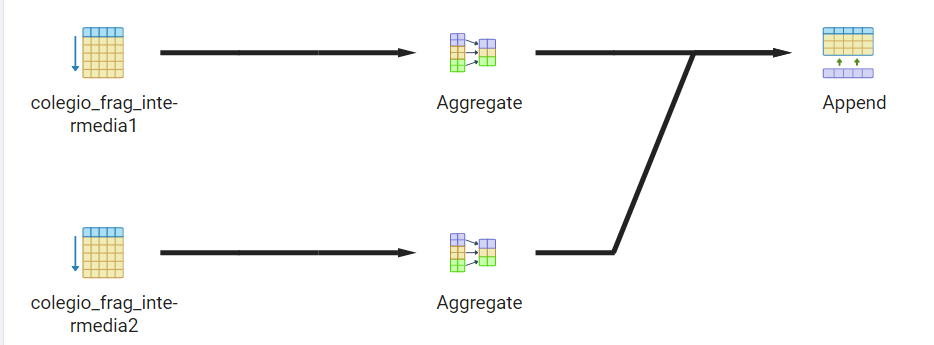
) as subconsulta;

--- Eliminamos fragmentaciones intermedias

DROP TABLE colegio\_frag\_intermedia1;

DROP TABLE colegio\_frag\_intermedia2;

Se observa un “Aggregate” en ambos fragmentos intermedios porque usamos el GROUP BY en ambos para luego unirlo con UNION ALL. Al servidor central se le devuelve el resultado, no necesita realizar una operación adicional.

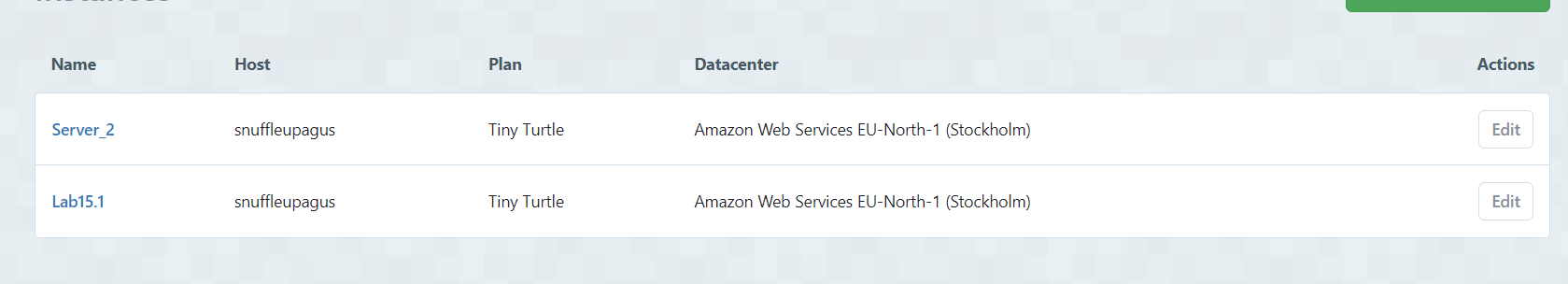


**P2. Algoritmos distribuidos en dos servidores**

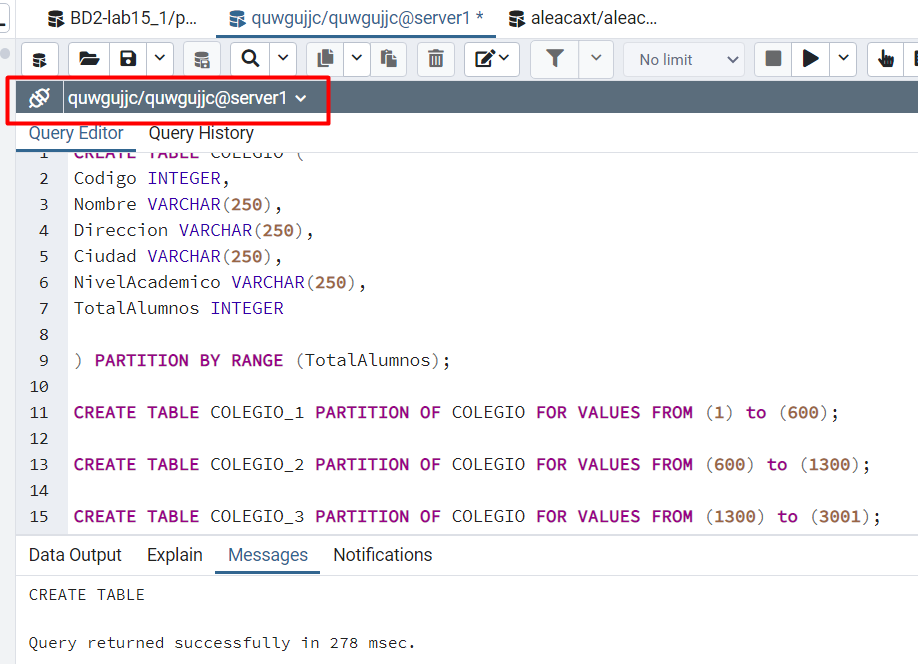
* Resolver los algoritmos distribuidos optimizados pero esta vez con al menos dos servidores PostgreSQL.
* Los algoritmos deben trabajar sobre los fragmentos asignados en los diferentes servidores.
  + Si le es necesario, puede prescindir del PARTITION BY y optar por una fragmentación manual.
* Mostrar el gráfico del plan de ejecución resultante.

**Nota:** El informe debe ser ordenado y convincente. Incluya los scripts legibles y los screenshots de evidencia.

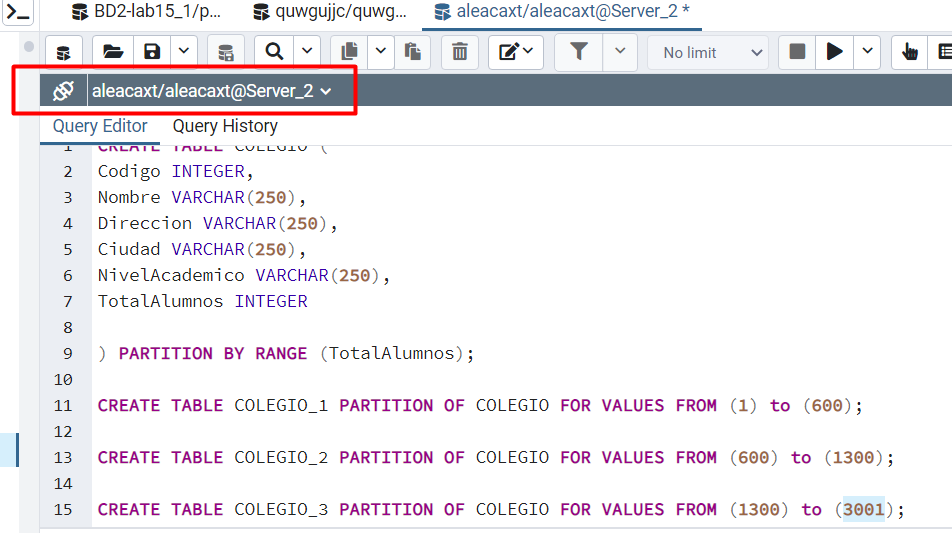
* + - 1. Crear servidores remotos

Para ello, creamos 2 instancias en ElephantSQL. Éstos serán nuestros 2 servidores remotos y nuestro servidor central será nuestro Postgres local.

* + - 1. Crear tablas en servidores remotos

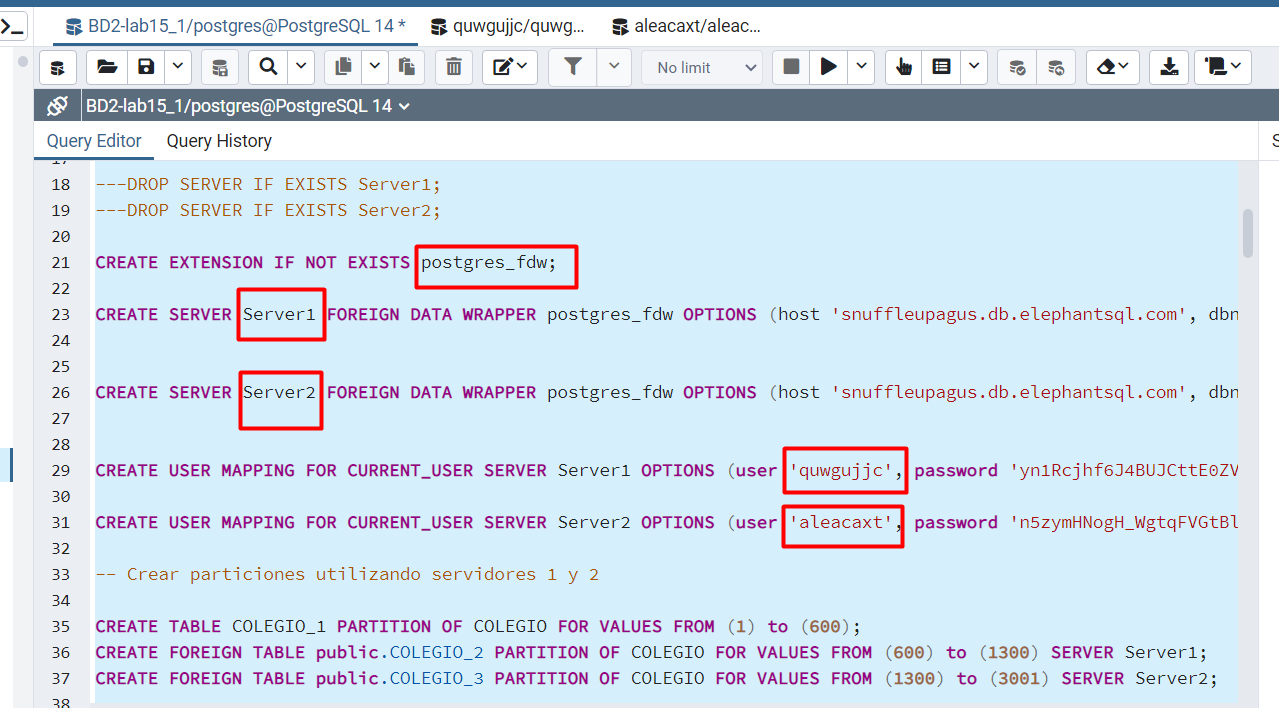
Creando en el servidor 1:

Creando en el servidor 2:



* + - 1. Conectar servidor local a servidores remotos

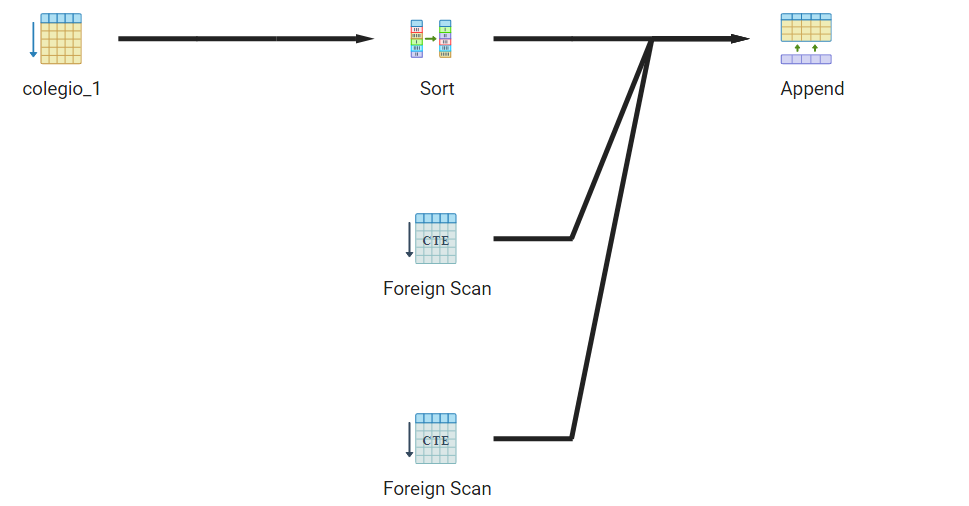
Para ello, hemos utilizado la extensión fdw de Postgres:



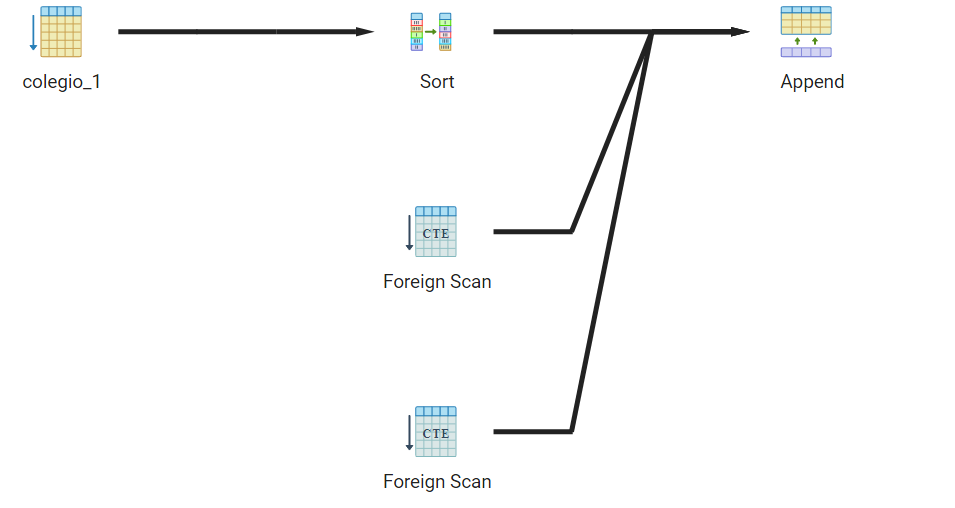
* + - 1. Realizar consultas optimizadas
         1. Consulta a

Select \* From Colegio Order By TotalAlumnos.

* + 1. No optimizada



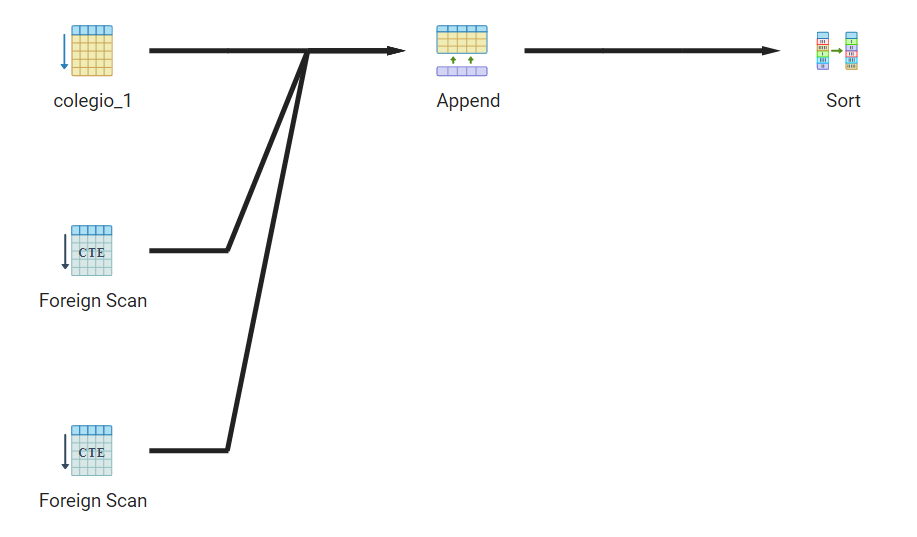
* + 1. Optimizada



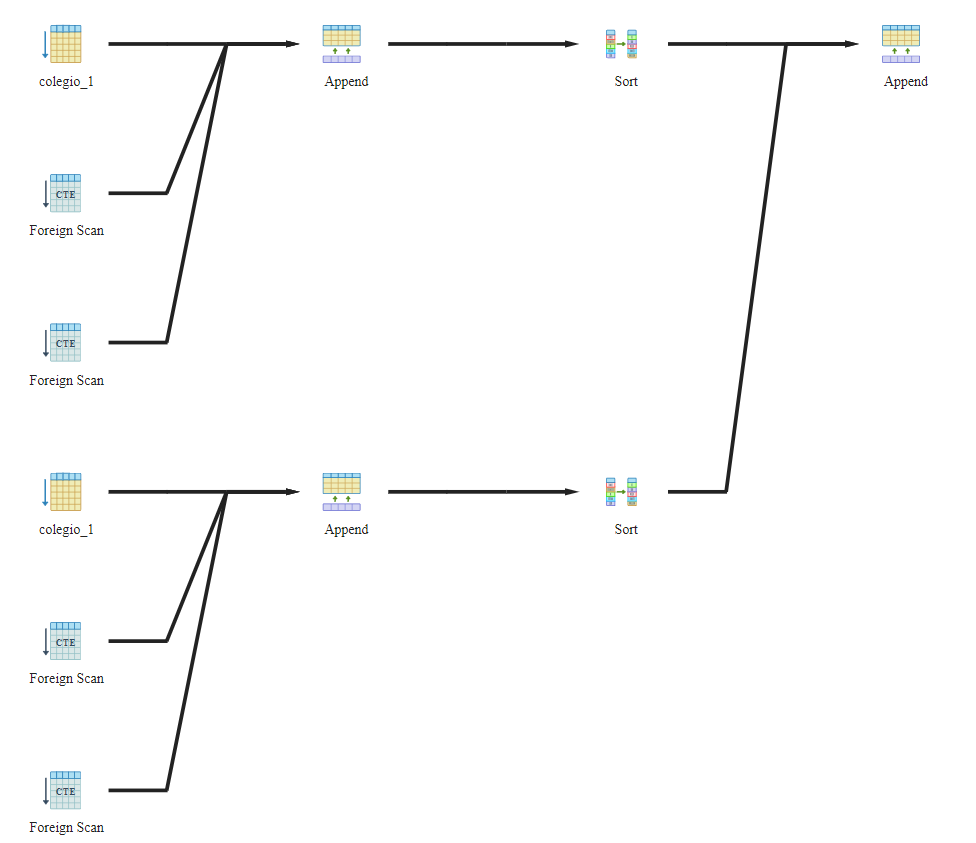
* 1. Consulta b

Select \* From Colegio Order By Ciudad

* + 1. No optimizada



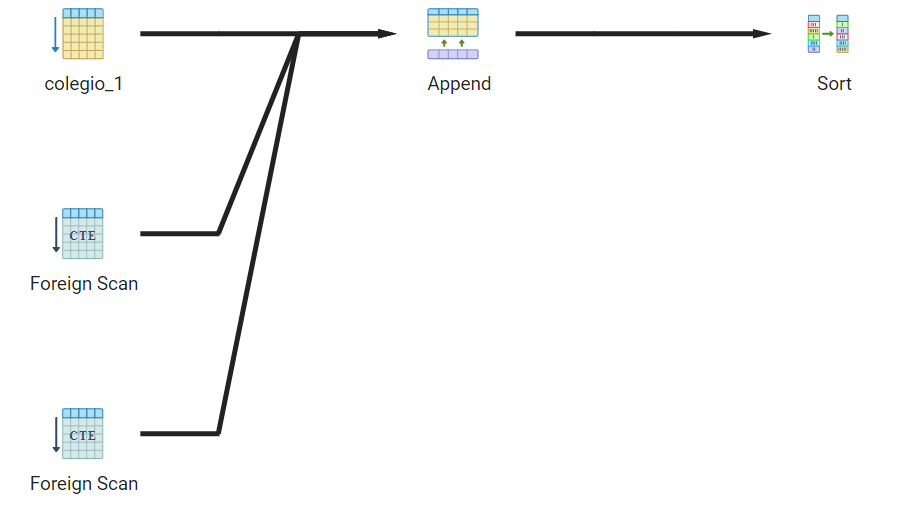
* + 1. Optimizada



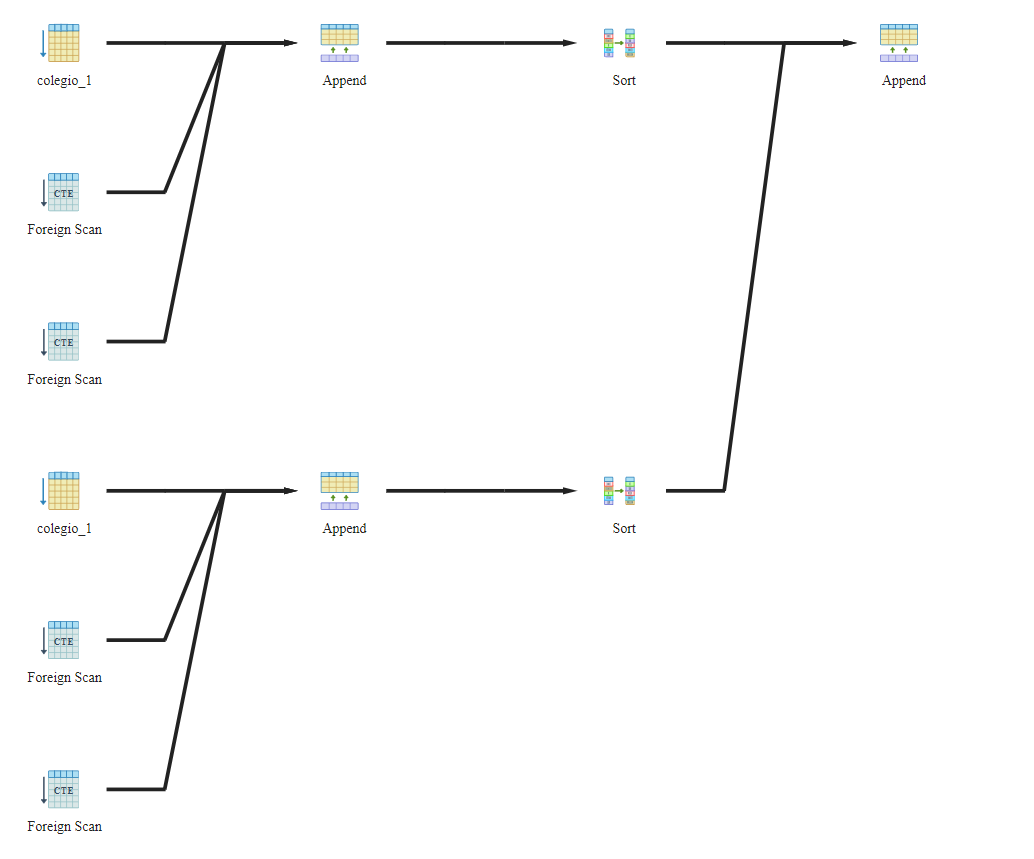
* 1. Consulta c

Select \* From Colegio Order By NivelAcad

* + 1. No optimizada

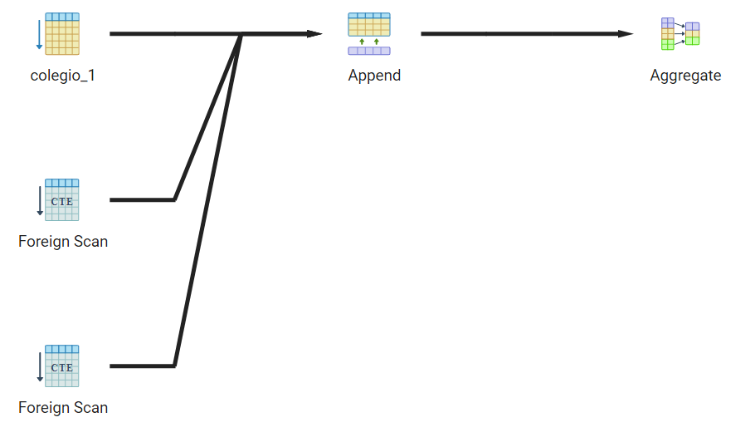


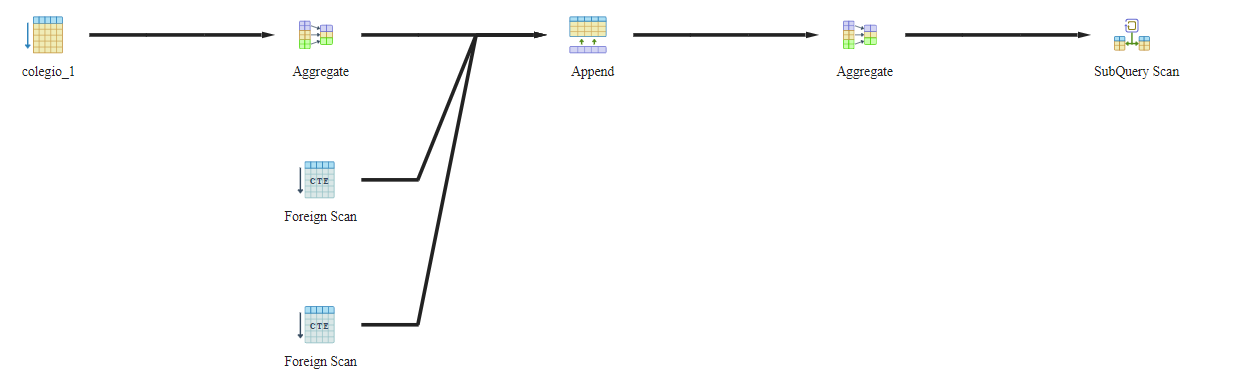
* + 1. Optimizada



* 1. Consulta c

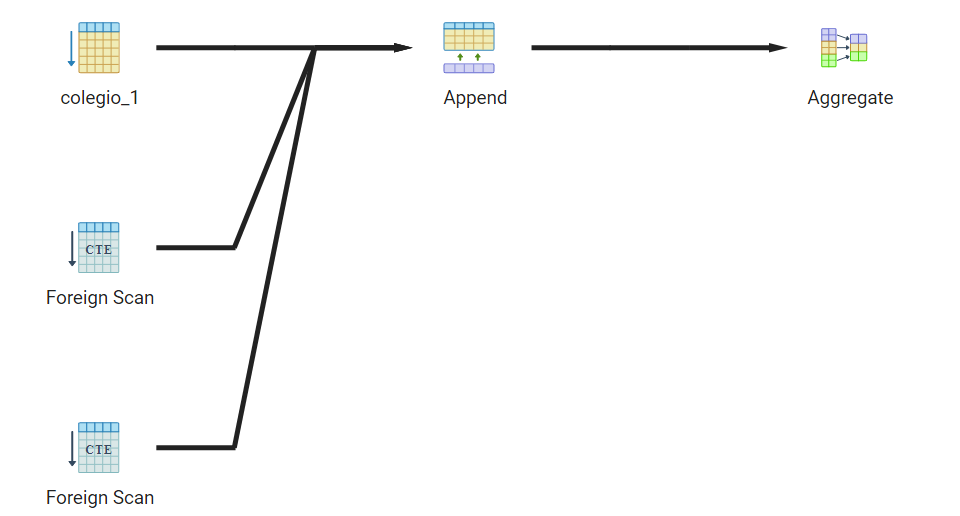
Select AVG(TotalAlumnos) From Colegio

* + 1. No optimizada
    2. Optimizada



* 1. Consulta e

Select Ciudad, SUM(TotalAlumnos) From Colegio Group By Ciudad.

* + 1. No optimizada
    2. Optimizada

