5.5. CARACTERÍSTICAS DE UN OPTIMIZADOR DISTRIBUIDO.

El optimizador en una BDD debe realizar todas las funciones de un optimizador centralizado vistas anteriormente más una serie de nuevas funcionalidades requeridas debido a la naturaleza de distribución de los datos.

- Decidir qué, cuándo y hacia donde será adecuado transferir relaciones o fragmentos de un sitio A a un sitio B.
- Decidir la mejor estrategia para realizar la transferencia de datos:
  - Transferir todos los datos en una sola petición
  - Transferir por demanda.
- En qué casos es adecuado reemplazar operaciones de JOIN por operaciones de SEMI – JOIN.

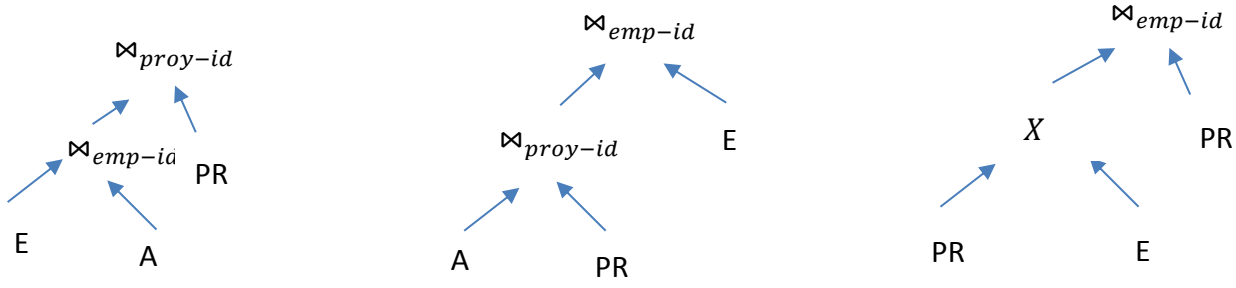
5.5.1. Espacio de búsqueda.

- Representa el conjunto de alternativas o planes de ejecución (conjunto de árboles de operadores) que pueden ser consideradas para ejecutar una consulta distribuida.
- El principal reto es la optimización de operaciones JOIN.
- Para una tabla con N tablas o relaciones, existen $O(N!)$ árboles de operadores equivalentes que pueden ser obtenidos al aplicar reglas de conmutatividad y asociatividad.

Ejemplo:

- Obtener posibles árboles de operadores para la siguiente sentencia:

```
Select nombre
From e, ae, pr
Where e.emp_id=ae.emp_id
And ae.proy_id = pr.proy_id
```



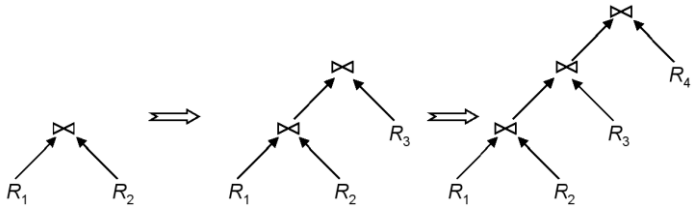
5.5.1.1. Reducción del espacio de búsqueda.

- Reducción aplicando heurísticas.
  - Ejecutar operaciones unarias primarias, después las binarias, ejecutar operaciones menos costosas primero como selección, proyección, etc.

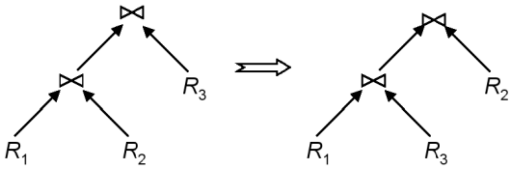
- Restringir por el tipo de árboles de operadores.
    - Arboles lineales Vs árboles combinados.

5.5.1.2. Estrategias para construir el espacio de búsqueda:

- Escaneo determinístico
    - Determinar todas las posibles opciones.
    - Inicia con las relaciones base, agregar una nueva relación en cada paso.
    - Garantiza que el resultado será la mejor opción
    - El costo de determinar la mejor opción puede ser elevado.



- Escaneo al azar.
    - Busca optimizaciones a partir de un escenario o solución particular.
    - No garantiza que el resultado será la mejor opción, pero el costo requerido para encontrar la solución es menor.
    - Útil para sentencias con 5 o más tablas.



5.5.2. Estadísticas en BDD

Considerar una relación $R(A_1, A_2, \ldots\ldots, A_k)$ que es fragmentada en $R_1, R_2, \ldots\ldots, R_r$ Las principales estadísticas empeladas en BDD son:

- Estadísticas de R:
    - Valor mínimo y máximo de cada atributo: $\min\{A_i\}, \max\{A_i\}$
    - Longitud de cada atributo: $length(A_i)$
    - Número de valores distintos de cada atributo en cada fragmento (cardinalidad): $card(A_i)$
    - Cardinalidad del dominio de un atributo: , $card(dom(A_i))$

- Estadísticas del fragmento:

Consideración importante: Las siguientes fórmulas consideran una distribución uniforme de los datos, por lo que como se revisó anteriormente, estos valores pueden ser erróneos a menos que se haga uso de un histograma.

5.5.2.1. Cardinalidad del fragmento

Representada por $card(R_i)$, total de registros del fragmento Ri

5.5.2.2. Cardinalidad de un atributo en un fragmento.

Se representa por $card(\pi_{A_i}(R_j))$. Representa el número de valores diferentes que contiene la columna Ai que pertenece a un fragmento Rj

5.5.2.3. Factor de selectividad.

- Representa una de las métricas más importantes para el optimizador. Su valor puede influir considerablemente para decidir o descartar planes de ejecución. Llamada también Factor de selectividad (SF)
- A nivel general, la selectividad es un valor numérico en el rango [0,1] que estima el porcentaje de registros que se obtendrían al ejecutar una operación como Selección, proyección, unión, intersección, join, semi-join, producto cartesiano principalmente.
- Un valor de selectividad **0** significa => Sin registros => ***Alta selectividad***
- Un valor de selectividad **1** significa => Se obtuvieron todos los registros de una fuente de datos => ***Baja selectividad***.
- Se representa por $SF_\sigma(f)$, donde ***f*** representa el predicado que se aplica a la operación de selección.

Los siguientes casos muestran fórmulas para estimar el factor de selectividad.

Selección en R con predicado de igualdad de un atributo A: $\boldsymbol{SF_\sigma(A = valor)}$

$$SF_\sigma(A = valor) = \frac{1}{card(\pi_A(R))}$$

Donde:
$card(\pi_A(R))$ Representa el número de valores distintos que existen para la columna A.
De lo anterior se puede ver que si $card(\pi_A(R)) = 1$, significa que la columna A contiene un solo valor. El factor de selectividad en este caso es 1.
Ejemplos:

A	*B*	*C*
3	1	5
4	1	5
5	1	5
6	1	5
7	1	6

- Obtener: $SF_\sigma(A = 3) = \frac{1}{card(\pi_A(R))} = \frac{1}{5} = 0.2$
- Obtener: $SF_\sigma(A = 7) = \frac{1}{card(\pi_A(R))} = \frac{1}{5} = 0.2$

Para estos 2 ejemplos, el factor es el mismo sin importar el valor se espera un 20% del total de los registros de la tabla que en este ejemplo sería 1 registro. Esto es correcto ya que la distribución es uniforme.

- Obtener: $SF_\sigma(B = 1) = \frac{1}{card(\pi_A(R))} = \frac{1}{1} = 1$

Para el este ejemplo, observar, se tiene una Baja selectividad, se obtuvieron el 100% de los registros.

- Obtener: $SF_\sigma(C = 6) = \frac{1}{card(\pi_A(R))} = \frac{1}{2} = 0.5$

En el ejemplo anterior, el factor indica que se obtendría el 50% de registros de la taba cuestión que es falso Solo existe un registro. Esto debido a la distribución heterogénea de los valores de C.

Selección en R con predicado de igualdad de un atributo A: $\boldsymbol{SF_\sigma(A > valor)}$

$$SF_s(A > valor) = \frac{\max(A) - valor}{\max(A) - \min(A)}$$

Selección en R con predicado de igualdad de un atributo A: $\boldsymbol{SF_\sigma(A < valor)}$

$$SF_s(A < valor) = \frac{valor - \min(A)}{\max(A) - \min(A)}$$

Ejemplos:

- Obtener: $SF_\sigma(A > 4) = \frac{7-4}{7-3} = \frac{3}{4} = 0.75$
- Obtener: $SF_\sigma(B > 1) = \frac{1-1}{1-1} = \propto \quad Sin\ registros$
- Obtener: $SF_\sigma(A < 4) = \frac{4-3}{7-3} = \frac{1}{4} = 0.25$

5.5.2.4. Cardinalidad para una operación de selección en R.

En los ejemplos anteriores se calculó el porcentaje de registros que se obtendrían. Si se desea obtener el número de registros o cardinalidad esperada, bastará multiplicar el factor de selectividad por la cardinalidad de la relación R, es decir:

$$card\left(\sigma_f(R)\right) = SF_\sigma(f) * card(R)$$

Ejemplo:

Obtener $card\left(\sigma_{A=3}(R)\right) = SF_\sigma(A = 3) * card(R) = \frac{1}{5} * 5 = 1$
Obtener $card\left(\sigma_{B=1}(R)\right) = SF_\sigma(B = 1) * card(R) = \frac{1}{1} * 5 = 5$
Obtener $card\left(\sigma_{c=6}(R)\right) = SF_\sigma(C = 6) * card(R) = \frac{1}{2} * 5 = 2.5 \sim 3$ Incorrecto por la distribución heterogénea.

Ejemplo:

Considerar nuevamente los datos del campo `nivel` en la tabla `puesto`.
Determinar el número de registros estimados que obtendría el estimador para la siguiente consulta:

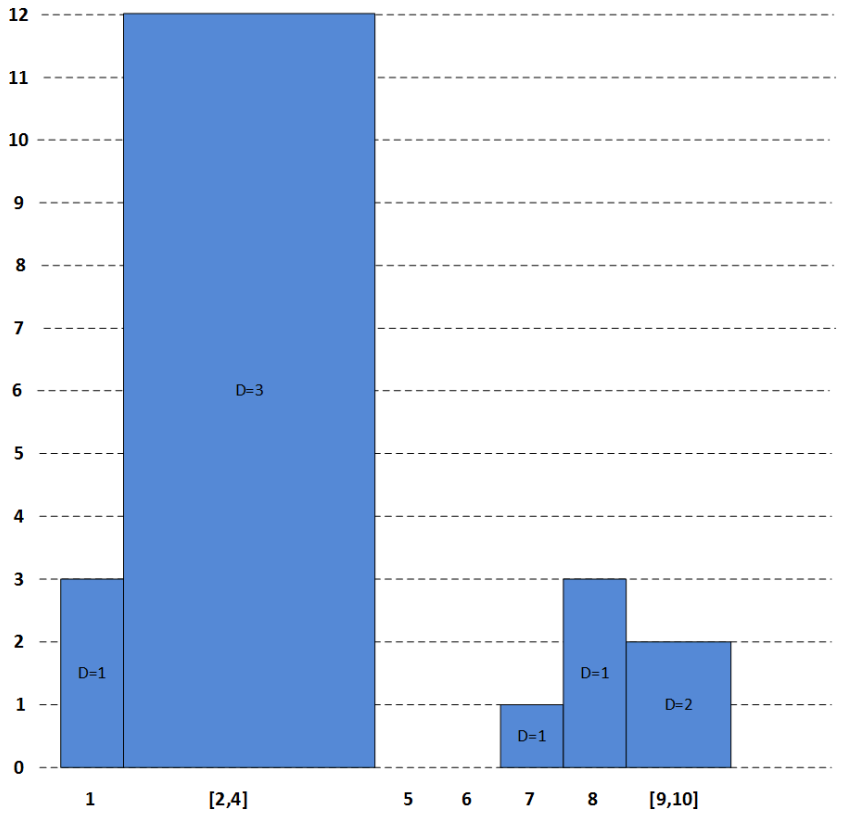
```
select * from puesto where nivel = 9
```

A. Considerando que no existe histograma
B. Considerando que existe histograma.

Solución:

Recordando los datos y el histograma del campo nivel:

Nivel	Num. registros
1	3
2	4
3	4
4	4
7	1
8	3
9	1
10	1



A. Considerar que no existe histograma.

Empleando la fórmula $card\left(\sigma_f(R)\right) = SF_\sigma(f) * card(R)$ donde $SF_\sigma(f) = \frac{1}{card(\pi_{nivel}(R))}$

$SF_\sigma(f) = \frac{1}{8}$

$card\left(\sigma_f(R)\right) = \frac{1}{8} * card(R) = \frac{1}{8} * 21 = 2.65 \sim \mathbf{3\ registros}.$

B. Considerar que existe histograma.

En este caso, el manejador ubica la barra o el rango de valores donde nivel = 9, para este caso corresponde con el último rango [9-10], D=2

Del histograma se obtiene que el número de valores distintos es D = 2, por lo tanto:

$$SF_\sigma(f) = \frac{1}{card(\pi_{nivel}(R))} = \frac{1}{2}$$

La cardinalidad en este caso, corresponde con el número de registros que se encuentran en el rango [9-10] es decir 2 registros.

$card\left(\sigma_f(R)\right) = \frac{1}{2} * card(R) = \frac{1}{2} * 2 = \mathbf{1\ registro}.$

Como se puede observar, el resultado correcto se obtiene al hacer uso del histograma.

Ejemplo:

Mismo procedimiento, pero ahora con la sentencia:

```
select *
from puesto
where nivel = 3
```

A. Considerar que no existe histograma.

$$card\left(\sigma_f(R)\right) = \frac{1}{8} * card(R) = \frac{1}{8} * 21 = 2.65 \sim \mathbf{3\ registros}.$$

B. Considerar que existe histograma.

En este caso, se considera el rango [2,4], D=3, $card\left(\pi_{nivel}(R)\right) = 12$

$$card\left(\sigma_f(R)\right) = \frac{1}{3} * card(R) = \frac{1}{3} * 12 = \mathbf{4\ registros}.$$

5.5.2.5. Cardinalidad para proyección

- Difícil de determinar si la distribución de los datos no es uniforme, requiere el uso de histograma.
- Para el caso de una llave primaria:

$$card\left(\pi_A(R)\right) = card(R)$$

5.5.2.6. Cardinalidad para un producto cartesiano

$$card(R\ X\ S) = card(R) * card(S)$$

5.5.2.7. Cardinalidad para una unión

$$card(R\ U\ S) = card(R) + card(S) => Limite\ superior$$
$$card(R\ U\ S) = \max\{card(R), card(S)\} => Limite\ inferior$$

5.5.2.8. Cardinalidad para diferencia

$$card(R - S) = card(R) => Limite\ superior$$
$$card(R - S) = 0 => Limite\ inferior$$

5.5.2.9. Selectividad para operaciones JOIN

- Porción de tuplas de una relación que participan en un JOIN:

$$SF_{\bowtie} = \frac{card(R \bowtie S)}{card(R) * card(S)}$$

5.5.2.10. Cardinalidad para operaciones JOIN.

- De la formula se puede observar que, si el número de registros obtenidos de un JOIN corresponde con el producto de ambas tablas, se obtendrá un factor igual a 1, es decir, se obtiene el máximo de registros posibles al efectuar un JOIN: El producto cartesiano R X S.

- Para calcular el número de registros (cardinalidad) que se obtendrían al ejecutar un JOIN, de la fórmula anterior:
    - Caso general:

$$card(R \bowtie S) = SF_{\bowtie} * card(R) * card(S)$$

Notar en la fórmula anterior, de las estadísticas se puede obtener de manera sencilla los valores de `card(R)` y `card(S)`. Sin embargo, los valores de los otros 2 términos no son tan directos. De aquí que se deben emplear técnicas adicionales para estimar el resultado de un JOIN como lo fueron las técnicas vistas anteriormente: NESTED LOOP, HASH JOIN y SORTED JOIN. Para el caso de BDD, existen técnicas llamadas ***técnicas de ordenamiento de JOINs*** que se discutirán en la siguiente sección.

Para facilitar la estimación de un JOIN, se pueden considerar los 2 siguientes casos especiales: Límite superior (peor de los casos) , y un JOIN entre 2 tablas con relación Uno a Uno :

- Límite superior:
  $card(R \bowtie S) \leq SF_{\bowtie} * card(R) * card(S)$

- Caso especial: R.A es llave primaria, y S.A es llave foránea, y la relación entre R y S es 1:1, es decir, cada registro de S tiene correspondencia con un solo registro de R:

$$card(R \bowtie S) = card(S)$$

5.5.2.11. Selectividad para Semi-Join

- Se puede estimar a través de la Aproximación de la selectividad de A en S:

$$SF_{\ltimes}(R \ltimes_A S) = SF_{\ltimes}(S.A) = \frac{card(\pi_A(S))}{card(dom[A])}$$

5.5.2.12. Cardinalidad de Semi -join

$$card(R \ltimes_A S) = SF_{\ltimes}(S.A) * card(R)$$

- Si R.A es una llave foránea en S (S.A es la PK), SF = 1 y el resultado corresponde con el tamaño de R

Ejemplo:

- ¿Cuántos registros de la tabla Empleado se obtendrán al ejecutar:

```
select e.*
from empleado e, puesto
p where e.puesto_id =e.puesto_id;
```

- Considerar que Empleado tiene 18 registros, la FK en empleado es NOT NULL.

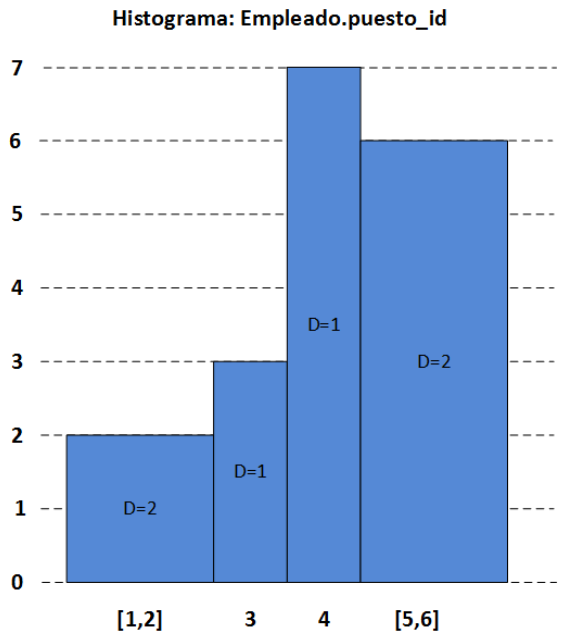
De acuerdo a lo anterior, el resultado sería la cardinalidad de R, en este caso, se obtendrían 18 registros. Demostración:

PUESTO

Empleado_id	Puesto_id
1	4
2	1
3	4
4	2
5	3
6	4
7	3
8	3
9	5
10	4
11	5
12	5
13	4
14	6
15	6
16	6
17	4
18	4

EMPLEADO

Puesto_id	nombre
1	Director
2	Subdirector
3	Lider
4	Programador
5	Analista
6	Administrador.



$SF_{\ltimes}(E \ltimes_{puesto-id} P) = SF_{\ltimes}(p.puesto - id) = \frac{card(\pi_{puesto-id}(P))}{card(dom[p.puesto\_id])} = \frac{6}{6} = 1$, dom = dominio de puesto_id

$card(E \ltimes_{puesto-id} S) = SF_{\ltimes}(p.puesto - id) * card(E) = 1 * 18 = \mathbf{18}$

Notar que, en este ejemplo, el histograma de Empleado no se usa.

5.5.3. Ordenamiento de operaciones JOIN distribuidos.

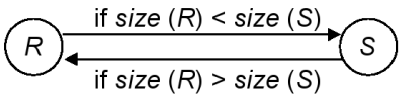
El ordenamiento de operaciones Join es de importancia particular en una BDD, principalmente por el hecho de que cada relación que participa en el JOIN puede estar en sitios diferentes. Un ordenamiento incorrecto puede generar un incremento en los tiempos de comunicación.

Existen varias técnicas:

- Optimizar el ordenamiento de forma directa sin Semi-Joins
- Algoritmo INGRES centralizado y distribuido.
- Algoritmo System R y System R*
- Algoritmos basados en el uso de Semi - Joins.
- Algoritmo Hill Climbing
- Algoritmo SDD-1

5.5.3.1. Ordenamiento de JOINs sin emplear Semi-Joins

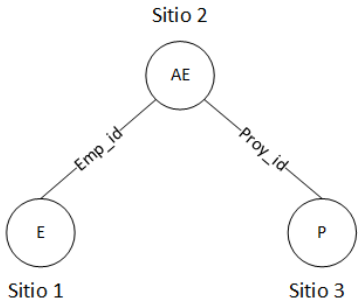
- Suponer que se desea estimar la cardinalidad de la operación $R \bowtie S$, donde R y S son fragmentos y se encuentran en sitios diferentes. Se considera que el costo de trasferir de cualquier sitio a otro es el mismo.
- La idea de esta estrategia es muy simple: Transferir la relación de menor cantidad.



- El problema se complica cuando se tienen más de 2 relaciones. Múltiples opciones pueden considerarse.

Ejemplo:

Suponer que se desea calcular $P \bowtie_{proy-id} AE \bowtie_{emp-id} E$ y suponer la siguiente distribución mostrada en la figura. Posibles opciones son:



Plan 1:
EMP -> Sitio 2
Sitio 2: $E' = E \bowtie_{emp-id} AE$
E'-> Sitio 3
Sitio 3: $E' \bowtie_{proy-id} P$

Plan 2:
AE-> Sitio 1
Sitio 1: $E' = E \bowtie_{emp-id} AE$
E'-> Sitio 3
Sitio 3: $E' \bowtie_{proy-id} P$

Plan 3:
AE -> Sitio 3
Sitio 3: A$E' = AE \bowtie_{proy-id} P$
AE'-> Sitio 1
Sitio 1: A$E' \bowtie_{emp-id} E$

Plan 4:
P -> Sitio 2
Sitio 2: $P' = P \bowtie_{proy-id} AE$
P'-> Sitio 1
Sitio 3: $P' \bowtie_{emp-id} E$

Plan 5:
EMP -> Sitio 2
P -> Sitio 2
Sitio 2: $P \bowtie_{proy-id} AE \bowtie_{emp-id} E$

- Para decidir el mejor PLAN, se debe contar con diversa información estadística: cardinalidades de fragmentos, cardinalidades de operaciones JOIN intermedias, etc.
- Otro aspecto es el paralelismo. Si los tiempos de transmisión son críticos, una opción podría ser el plan 5, 2 transmisiones se pueden efectúa al mismo tiempo.

- Otra opción es asumir el límite superior de la cardinalidad de un JOIN que considera como resultado, la cardinalidad del producto cartesiano. En este caso, los fragmentos se ordenan con respecto a su tamaño de menor a mayor y en ese orden se transmiten. Por ejemplo, el orden P-> AE->E emplearía el plan 4.

5.5.3.2. Algoritmo INGRES.

- Emplea una estrategia dinámica de optimización de consultas.
- Realiza una reducción recursiva de una sentencia basado en los siguientes conceptos:
  - Una consulta en la que participan N tablas puede ser descompuesta en N sub-consultas $q_1 \rightarrow q_2 \rightarrow . . . \rightarrow q_n$
  - La salida de q_i es consumida por q_{i+1}
  - Existen 2 técnicas para descomponer una sentencia:
    - Separación
    - Sustitución.
  - Al final del proceso, se obtienen 2 tipos de relaciones:
    - mono-relación (consulta en la que participa una sola relación).
    - Bi- relación Consulta en donde participan solo 2 relaciones, llamadas consultas irreducibles.
- La idea de hacer el procedimiento anterior, es la posibilidad de optimizar cada una de estas sub-consultas de forma separada, generando mejores planes de ejecución.

Ejemplo:
Descomponer la siguiente sentencia que muestra los nombres de los empleados que trabajan en proyectos de migración empleando la técnica de ***Separación*** de subconsultas.

```
q1: Select e.nombre
From e, ae,pr
Where e.emp_id = ae.emp_id
And ae.proy_id = pr.proy_id
And pr.nombre='migracion'
```

- Iteración 1: Descomponer q_1 $into$ q_{11} , separación de la relación 'pr'

```
q11: select pr.proy_id,  into v_pr
From pr
Where pr.nombre = 'migración'
```

Observar que q11 es una mono-relación, participa una sola relación: pr.
Continuando con el algoritmo, sustituyendo q_{11} en la consulta original

```
q1: Select e.nombre
From e, ae,v_pr
Where e.emp_id = ae.emp_id
And ae.proy_id = v_pr.proy_id
```

- Iteración 2: Descomponer q_1 $into$ q_{12} , separación de la relación 'ae'

q_{12}: select ae.emp_id into **v_ae**
From ae, v_pr
Where ae.proy_id =v_pr.proy_id;

Observar que q12 es una Bi-relación. Participan solo 2 relaciones.
Sustituyendo q_{12} en la tabla original

q_1: select e.nombre
From e, **v_ae**
Where e.emp_id = v_ae.emp_id

- q_1 ya no se puede reducir, es una relación bi-relación, por lo tanto, q_1 la llamaremos q_{13}, el algoritmo termina.

De lo anterior, q_1 fue descompuesta en $q_{11} \rightarrow q_{12} \rightarrow q_{13}$

La técnica de ***sustitución*** se aplica únicamente para las tablas bi-relación o irreducibles. Para este ejemplo, q_{12} y q_{13}. La idea es la siguiente:

- Suponer que se elige a q_{13}. Observar que participan 2 relaciones: e, v_ae .
- Suponer que la relación v_ae solo tiene pocos registros. Por simplicidad, suponer que solo tiene 2 registros con emp_id {4,5}
- Debido la condición del join e.emp_id = v_ae.emp_id, cada registro de v_ae tiene una sola correspondencia con la relación 'e' al ser llave foránea.
- Por lo anterior, la consulta q_{13} se puede sustituir por 2 consultas mono-relación que obtienen los nombres de los 2 empleados con id 4 y 5.
- Es decir: $q_{13} \rightarrow q_{131} \rightarrow q_{132}$

q_{131}: select e.nombre
From e
Where e.emp_id = 4

q_{132}: select e.nombre
From e
Where e.emp_id = 5

- En general, se tendrían N consultas mono-relación donde N es el número de registros de la relación v_ae.
- Finalmente, el algoritmo INGRES funciona de manera similar en BDD, con la diferencia que este se aplica a fragmentos en lugar de tablas, y solo se usa en fragmentaciones horizontales.

5.5.3.3. Algoritmo System R centralizado.

- Este algoritmo emplea las técnicas vistas en la primera parte de este capítulo: optimización estática a través del uso de planes de ejecución que consideran ordenamiento de operaciones JOIN, métodos para ejecutar un JOIN (Nested, Hash, Sorted), Access Paths.

- Esta técnica también hace uso de los llamados Planes Adaptativos los cuales hacen uso de los resultados parciales para mejorar los planes de ejecución y reducir opciones.

Ejemplo:

Determinar el plan de ejecución para la sentencia:

```
Select  e.nombre
From e, ae,pr
Where e.emp_id = ae.emp_id
And ae.proy_id = pr.proy_id
And pr.nombre='migracion'
```

Asumir:

- Existen índices en las PK de las tablas.
- Existe índice en el campo pr.nombre

Plan resultante:

Select statement					Fuente de datos
	NESTED LOOP				
		NESTED LOOP			
			TABLE ACCESS BY ROW_ID		PR
				UNIQUE INDEX SCAN	PR_NOMBRE_IDX
			FULL INDEX SCAN		AE_PK
		TABLE ACCESS BY ROW_ID			E

*5.5.3.4. Algoritmo System R * Distribuido.*

- Las relaciones completas son distribuidas, no considera esquema de fragmentación ni replicación.
- La compilación de la sentencia distribuida se coordina en el sitio donde se lanza llamado "***Sitio maestro***".
- El sitio maestro es el responsable de realizar las decisiones como ordenamiento de joins, método para transferir relaciones, selección de sitios para procesar JOINs, etc.
- Los sitios hacen optimizaciones locales.
- El principal reto en este algoritmo es el ordenamiento de JOINs y la transferencia de datos entre sitios.
- Existen 2 métodos de transferencia:
    - Ship – whole: Transferencia completa de la relación externa a un sitio donde se realizará el JOIN y almacenada en una tabla temporal.
        - Estrategia adecuada si las relaciones son pequeñas.
    - Fetch as Needed: Los registros son enviados al sitio donde se ejecutará el JOIN por demanda.
- Ambas estrategias tienen claras ventajas y desventajas:
    - Si las relaciones son pequeñas, Ship-Whole resulta mejor opción.
    - Si las relaciones son grandes, y el resultado del JOIN son pocos registros, Fetch as needed representa una mejor opción.

Con estos 2 métodos de transferencia se pueden considerar hasta 4 posibles estrategias para ejecutar un JOIN distribuido:

Estrategias de ejecución:

- Considerar a las relaciones R y S como operandos de la operación JOIN.
- LT => Tiempo de procesamiento local.
- CT => Tiempo de comunicación
- s => Promedio de tuplas de S que hacen match con una tupla de R

Estrategia 1: Enviar la relación outer al sitio donde se encuentra la relación Inner para ejecutar NESTED LOOP.
Costos asociados:

- Obtener la lista de tuplas de la relación outer
- Enviar la lista al sitio donde se encuentra la relación inner
- Ejecutar el Join conforme los registros de la relación outer van llegando.

$$costo = LT(obtener\ card(R)\ tuplas\ de\ R) + CT\big(size(R)\big) + LT(obtener\ s\ tuplas\ de\ S * card(R))$$

Estrategia 2: Enviar la relación Inner al sitio de la relación Outer para ejecutar NESTED LOOP.

- Notar que, en este caso, el JOIN no se puede ejecutar hasta recibir el 100% de las tuplas de la relación inner
- Lo anterior implica que la relación inner se debe almacenar en una tabla temporal Temp.

$$costo = LT(obtener\ card(S)\ tuplas\ de\ S) + CT\big(size(S)\big) + LT(guardar(S)\ en\ Temp) + LT(obtener\ card(R)\ tuplas\ de\ R) + LT(obtener\ \ s\ tuplas\ de\ Temp * \ \ card(R))$$

Estrategia 3: Obtener tuplas de la relación Inner por cada registro de la relación Outer conforme se necesitan.

- Para cada tupla de R, el atributo A que actúa como predicado del JOIN se envía al sitio donde se encuentra la relación S (inner).
- En el otro sitio se obtienen las "s" tuplas de S que hacen match con el atributo A recibido y se envían al sitio de la relación Outer.

$$costo = LT(obtener\ card(R)\ tuplas\ de\ R) + CT\big(length(A) * card(R)\big) + LT\big(obtener\ s\ tuplas\ de\ S * card(R)\big) + CT\big(size(s)\big) * card(R)$$

Estrategia 4: Mover ambas relaciones a un tercer sitio y ahí ejecutar el JOIN.

- La relación S (Inner) se mueve al tercer sitio y se guarda en una relación temporal.
- La relación R (Outer) se transmite al tercer sitio. El JOIN se ejecuta conforme van llegando las relaciones al sitio.

$$costo = LT(obtener\ \ card(S)\ tuplas\ de\ S) + CT\big(size(S)\big) + LT(guardar(S)\ en\ Temp) + LT(obtener\ card(R)\ tuplas\ de\ R) + CT\big(size(R)\big) + LT(obtener\ s\ tuplas\ de\ T\ \ * \ \ card(R))$$

- El optimizador calcula los costos de estas 4 estrategias y selecciona a la de menor costo.

- Para minimizar los costos locales, se puede hacer uso de índices, y en general aplicar las técnicas de un optimizador local vistas anteiormente.
-

5.5.3.5. Algoritmos basados en el uso de semi – joins

- Empleados para reducir el tiempo total de ejecución de un JOIN.
- La ventaja con respecto a la estrategia anterior es que no se tiene que transferir la relación completa. En este caso, el semi -join actúa como un reductor aplicado a una relación, de forma similar a una operación de selección.
- Un join entre 2 relaciones R y S puede ser calculado en términos de Semi -Joins empleando las siguientes reglas:

$R \bowtie_A S \Leftrightarrow (R \ltimes_A S) \bowtie_A S$ o de forma equivalente:
$\Leftrightarrow R \bowtie_A (S \ltimes_A R)$ o de forma equivalente:
$\Leftrightarrow (R \ltimes_A S) \bowtie_A (S \ltimes_A R)$

- El semi – join es adecuado si el costo de producir y enviar datos para su cálculo a un sitio es menor al costo de enviar la relación completa

Análisis de costos Join Vs Semi-Join:

- Sin semi -Join, se tienen que ejecutar los siguientes pasos, se asume que size(R) < size(S)

R -> Sitio 2
Sitio 2 : $R \bowtie_A S$

- Con semi-Join

$sitio\ 2: S' = \pi_A(S)$
$S' \rightarrow Sitio\ 1$
$sitio\ 1: R' = R \ltimes S'$
$R' \rightarrow Sitio\ 2$
$sitio\ 2: R' \bowtie_A S$

- Lo anterior implica que un semi-join es mejor si:

$size(\pi_A(S)) + size(R \ltimes S) < size(R)$

- El semi-join proporciona una mejor opción cuando pocos registros de R participan en el Join.
- Ejecutar el JOIN sin semi-Join es mejor si casi todos los registros de R participan en el Join.

5.5.3.6. Algoritmo Hill -Climbing

- Considerado como el primer algoritmo para realizar el procesamiento de consultas distribuidas.
- EL algoritmo inicia con el cálculo de una solución inicial. Esta solución corresponde a la que presente el menor costo de transferir todos los datos necesarios a un sitio candidato ES_0 en el que se realizará el procesamiento y la presentación de los resultados

$$costo = costo(ES_0)$$

- Posteriormente ES_0 es dividida en 2 estrategias: ES_1 y ES_2.

- ES_1 consiste en enviar una de las relaciones que participan en el JOIN al sitio de otra relación. Se aplica el JOIN localmente.
- ES_2 El resultado del JOIN anterior se envía al sitio elegido para mostrar resultados.

- Posteriormente, se calcula el costo considerando estos 2 pasos:

$$costo = costo(ES_1) + costo(join\ local) + costo(ES_2)$$

- Si el costo obtenido es < costo de ES_0 significa que esta división representa una mejor solución.
- Mientras este costo mejore, el algoritmo aplica esta división de forma recursiva. Por ejemplo, en la segunda iteración, ES_1 reemplazado por 2 nuevas estrategias $ES_3\ y\ ES_4$

$$costo = costo(ES_3) + costo(join\ local) + costo(ES_4) + costo(join\ local) + costo(ES_2)$$

- El algoritmo se detiene hasta no encontrar ganancia en el costo.

Ejemplo:

Mostrar los salarios de los empleados con puesto_id = 3 que trabajaron en el proyecto 'migración'.

```
Select pu.salario
From  pu, e, ae, pr
Where pu.puesto_id = e.puesto_id
Ando e.emp_id = ae.emp_id
And ae.proy_id = pr.proy_id
And pr.nombre ='migracion';
And pu.puesto_id = 3
```

Suponer las siguientes estadísticas:

Relación	Size	Sitio
E	8	1
PU	4	2
PR	1	3
AE	10	4

- Suponer que $size(EMP \bowtie \mathrm{PU}) = 8$, $size(PROY \bowtie \mathrm{AE}) = 2$, $size(AE \bowtie \mathrm{E}) = 10$
- Ignorar el procesamiento local.
- El costo de transmisión entre 2 sitio es 1 denotado por CT.

Alternativa 1:

Mostrar el resultado en el sitio 1.

$costo = costo(PU \rightarrow S_1) + costo(AE \rightarrow S_1) + costo(PR \rightarrow S_1)$

$costo = 4 * (CT) + 10 * (CT) + 1 * (CT) = 4 + 10 + 1 = 15$

Alternativa 2:

Mostrar el resultado en el sitio 2.

$costo = costo(E \rightarrow S_2) + costo(PR \rightarrow S_2) + costo(AE \rightarrow S_2)$

$costo = 8 * (CT) + 1 * (CT) + 10 * (CT) = 19$

Alternativa 3:

Mostrar el resultado en el sitio 3.

$costo = costo(E \rightarrow S_3) + costo(PU \rightarrow S_3) + costo(AE \rightarrow S_3)$
$costo = 8 * (CT) + 4 * (CT) + 10 * (CT) = 22$

Alternativa 4:
Mostrar el resultado en el sitio 4.
$costo = costo(E \rightarrow S_4) + costo(PU \rightarrow S_4) + costo(PR \rightarrow S_4)$
$costo = 8 * (CT) + 4 * (CT) + 1 * (CT) = 4 + 10 + 1 = \mathbf{13}$

- De lo anterior, la estrategia inicial con el menor costo obtenido es mover todas las relaciones al sitio 4.

$ES_0 = E \rightarrow S_4 \; ; \; PU \rightarrow S_4 \; ; \; PR \rightarrow S_4$

- El siguiente paso del algoritmo es hacer recursión moviendo relaciones a otros sitios diferentes al sitio 4.
- Se tienen 2 opciones para ES_0. Mover E al sitio de PU (sitio 2), o mover PU al sitio de E (P1) Es decir:

Iteración 1: intentar Enviar E a S2:
$ES_1: E \rightarrow S_2$
$ES_2: (E \bowtie PU) \rightarrow S_4$ Al enviar a E al sitio 2, se aplica un Join local con PU ya que este se encuentra en el sitio 2. El resultado se envía al sitio 4
$ES_3: \; PR \rightarrow S_4$
$costo = 8 + \; size(EMP \bowtie \text{PU}) + 1 = 8 + 8 + 1 = \mathbf{17}$
- Se observa que el costo no mejoró. Se intenta la segunda opción:

Iteración 2: intentar Enviar PU a S1:
$ES_1: PU \rightarrow S_1$
$ES_2: (E \bowtie PU) \rightarrow S_4$ Al enviar a E al sitio 2, se aplica un Join local con PU ya que este se encuentra en el sitio 2. El resultado se envía al sitio 4
$ES_3: \; PR \rightarrow S_4$
$costo = 8 + \; size(EMP \bowtie \text{PU}) + 1 = 8 + 4 + 1 = \mathbf{13}$

- Se observa que el costo no mejoró. Es el mismo. La recursividad se detiene, y por lo tanto la salida del algoritmo es conservar la solución ES_0 como la mejor opción.
- Observar que, a pesar de este algoritmo, la solución encontrada no es la mejor. Observar que pasa con la siguiente combinación en la que los resultados se muestran en el sitio 2.

$PR \rightarrow S_4$
$En \; S_4 : \; AE' = (PR \bowtie AE) \rightarrow S_1$
$En \; S_1: \; (AE' \bowtie E) \rightarrow S_2$
$costo = 1 + \; size(PR \bowtie AE) + size(AE' \bowtie E) = 1 + 2 + 2 \; = \mathbf{5}$

- Debido a que $size(AE') = 2$, al hacer JOIN con E, se obtendrían solo 2 registros ya que cada registro de AE solo puede hacer match con uno de E. Por lo tanto, $size(AE' \bowtie E)$ =2.
- El problema aquí es que AE' es una relación interna y se aplicó una optimización local con una relación intermedia. Este detalle no es visible para el algoritmo.

5.5.3.7. Algoritmo SDD-1

Mejora el algoritmo Hill Climbing en diversos aspectos:

- Este algoritmo hace uso de estadísticas llamadas "Database profiles" (perfiles).
- El algoritmo también selecciona una solución inicial que también es iterada de forma recursiva
- Se agrega un paso de post-optimización para mejorar el tiempo total de la solución seleccionada.
- El algoritmo también determina un orden adecuado, pero ahora de ***Semi -Joins***. El orden adecuado es aquel cuyo coto resulta ser menor al valor del costo que se ahorra (beneficio) al cambiar Relaciones por Semi-Joins.
- Dichos costos consideran ahora los tiempos de transmisión de un dato de un sitio a otro T_{TR}, y El tiempo que se requiere para cargar y recibir datos de un sitio a otro T_{MSG}
- De lo anterior, la fórmula para calcular el costo de transmisión de un Semi – Join es:

$$costo = (R \ltimes_A S) = T_{MSG} + T_{TR} * size(\pi_A(S))$$

- El costo del beneficio que ofrece el uso de un Semi- Join es:

$$costoBeneficio\ (R \ltimes_A S) = \left(1 - SF_{SJ}(S.A)\right) * size(R) * T_{TR}$$

- El algoritmo está formado de 4 etapas:
  - Inicialización (Identificación de Semi-Joins)
  - Selección de los semi-joins que ofrecen el mejor beneficio.
  - Determinar los sitios a donde se realizarán las transferencias de datos considerando costos menores (similar al algoritmo Hill Climbing).
  - Post-optimización. Identificar los Semi-Joins que solo afectan al sitio donde se van a presentar los resultados. Estos Semi-Joins son eliminados del cálculo final por ser locales.

El algoritmo asume que todas las relaciones pueden ser transmitidas de un sitio a otro excepto aquellos que se encuentran en el sitio.