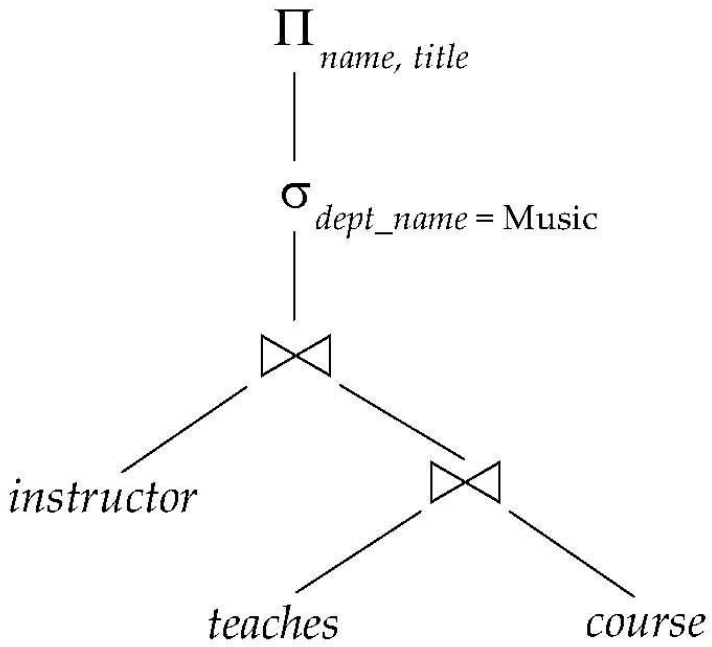
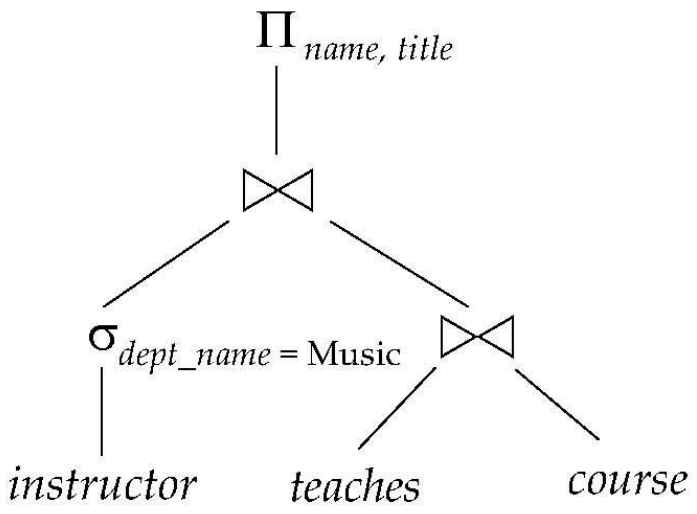
Capítulo 5

Optimización de consultas

Introducción

- Hay formas alternativas de evaluar una expresión de consulta:
  - Usando expresiones equivalentes
  - Usando diferentes algoritmos para cada operación





Introducción

- Un **plan de evaluación** define qué algoritmo es usado para cada operación y cómo se coordina la ejecución de las operaciones.
- En cada base de datos se puede ver el plan de evaluación de una consulta.
- La diferencia de costo entre planes de evaluación puede ser enorme.
  - Por ejemplo de segundos a días en algunos casos.

Introducción

- **Pasos en optimización de consultas basada en costo:**
  1. Generar expresiones lógicamente equivalentes usando **reglas de equivalencia.**
  2. Anotar expresiones resultantes para obtener planes de consulta alternativos.
  3. Elegir el plan más económico basado en el **costo estimado**.

Introducción

- **El costo estimado del plan se basa en:**
  - Información estadística acerca de tablas.
    - P.ej: número de tuplas, número de valores distintos de un atributo.
  - Estimación estadística para resultados intermedios.
    - Para computar el costo de expresiones complejas.
  - Fórmula de costo para algoritmos, computados usando estadísticas.
- Todas estas cosas las vimos en el capítulo anterior.
- Las estadísticas son computadas periódicamente porque
  - tienden a no cambiar radicalmente en un corto tiempo;
  - estadísticas algo imprecisas son útiles siempre que sean aplicadas consistentemente a todos los planes.

Introducción

- El número de transferencias de bloques es influenciado por:
  1. Los operadores lógicos elegidos para implementar la consulta.
  2. El tamaño de los resultados intermedios.
  3. Los operadores físicos usados para implementar los operadores lógicos.
  4. El método para pasar argumentos de un operador físico al siguiente.
  5. El orden de operaciones similares, especialmente reuniones y selecciones.
- El último ítem es menos conocido y va a quedar más claro durante este capítulo.

Transformación de expresiones de consulta

- Dos expresiones del álgebra de tablas son **equivalentes módulo ordenamiento de registros** si las dos expresiones generan el mismo multiconjunto de tuplas (i.e. las mismas tuplas en la misma cantidad para cada tupla).
  - O sea, dos expresiones equivalentes a lo más alteran el orden de las tuplas.
  - $r =_O s \Leftrightarrow O(r) = O(s)$, donde O es la operación de ordenación del álgebra de tablas en base a todas las columnas.
- Dos expresiones del álgebra de tablas son **equivalentes por igualdad** si las dos tablas son equivalentes en esquema e información.
- Una **regla de equivalencia** dice que las expresiones de dos formas son equivalentes usando uno de los tipos de equivalencia de arriba.
  - Podemos reemplazar una expresión de la primera forma por la segunda forma o vice-versa.

Reglas de equivalencia

- **Nomenclatura**:
- Usaremos Θ_r^N para indicar una lista de N columnas de la tabla r.
- Usaremos $\Theta_r^N = \Theta_s^N$ para indicar la igualación de cada columna de Θ_r^N con las columnas de Θ_s^N . Omitiremos la tabla y/o cantidad de columnas si no hay ambigüedad.
- Indicaremos con p_r a una proposición que sólo utilice columnas de r.

Reglas de equivalencia

1. La selección conjuntiva de operaciones puede ser deconstruida en una secuencia de selecciones individuales:

$$\sigma_{q_1 \wedge q_2}(E) = \sigma_{q_1}(\sigma_{q_2}(E))$$

2. Las operaciones de selección son commutativas:

$$\sigma_{q_1}(\sigma_{q_2}(E)) = \sigma_{q_2}(\sigma_{q_1}(E))$$

3. Solo la última en una secuencia de operaciones proyección es necesitada, las otras pueden ser omitidas.

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{Ln}(E))\ldots)) = \Pi_{L_1}(E)$$

Reglas de equivalencia

4. La reunión natural es asociativa:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

5. La siguiente regla es parecida a la definición de reunion selectiva:

$$\Pi_\theta(\sigma_{\theta_r^N = \theta_s^N}(r \times s)) = r \ {}_{\theta_r^N}\bowtie_{\theta_s^N} \ s \quad \text{con } \theta \text{ las columnas de } \bowtie$$

6. La siguiente regla muestra cómo podemos expresar una selección de una reunion selective por medio de una reunion selectiva.

$$\sigma_{\theta_r^N = \theta_s^N}\left(r \ {}_{\theta_r^M}\bowtie_{\theta_s^M} \ s\right) = r \ {}_{\theta_r^N, \theta_r^M}\bowtie_{\theta_s^N, \theta_s^M} \ s$$

Reglas de equivalencia

7. La siguiente regla nos permite aplicar selección antes de reunión selectiva:

$$\sigma_{p_r}(r \ {}_{\theta_r}\!\bowtie_{\theta_s} s) = \sigma_{p_r}(r) \ {}_{\theta_r}\!\bowtie_{\theta_s} s$$

8. La siguiente regla se deduce de reglas previas:

$$\sigma_{p_r \wedge p_s}(r \ {}_{\theta_r}\!\bowtie_{\theta_s} s) = \sigma_{p_r}(r) \ {}_{\theta_r}\!\bowtie_{\theta_s} \sigma_{p_s}(s)$$

9. La siguiente regla muestra cómo se comporta la proyección cuando se usa junto con la reunión selectiva.

$$\Pi_{\theta_r^N, \theta_s^O}(r \ {}_{\theta_r^M}\!\bowtie_{\theta_s^M} s) = \Pi_{\theta_r^N}(r) \ {}_{\theta_r^M}\!\bowtie_{\theta_s^M} \Pi_{\theta_s^O}(s) \quad \text{con } \theta_r^M \subseteq \theta_r^N \text{ y } \theta_s^M \subseteq \theta_s^N$$

Reglas de equivalencia

10. La concatenación “conmuta” alterando solo el orden de las tuplas:

    r ++ s $=_O$ s ++ r

11. La concatenación es asociativa.

12. Selección distribuye con concatenación, intersección y resta

$$\sigma_p(r \star s) = \sigma_p(r) \star \sigma_p(s) \quad \text{con} \star \in \{++, \cap, \setminus\}$$

13. La siguiente propiedad es más débil que la anterior.

$$\sigma_p(r \star s) = \sigma_p(r) \star s \quad \text{con} \star \in \{\cap, \setminus\}$$

14. La proyección distribuye con la concatenación:

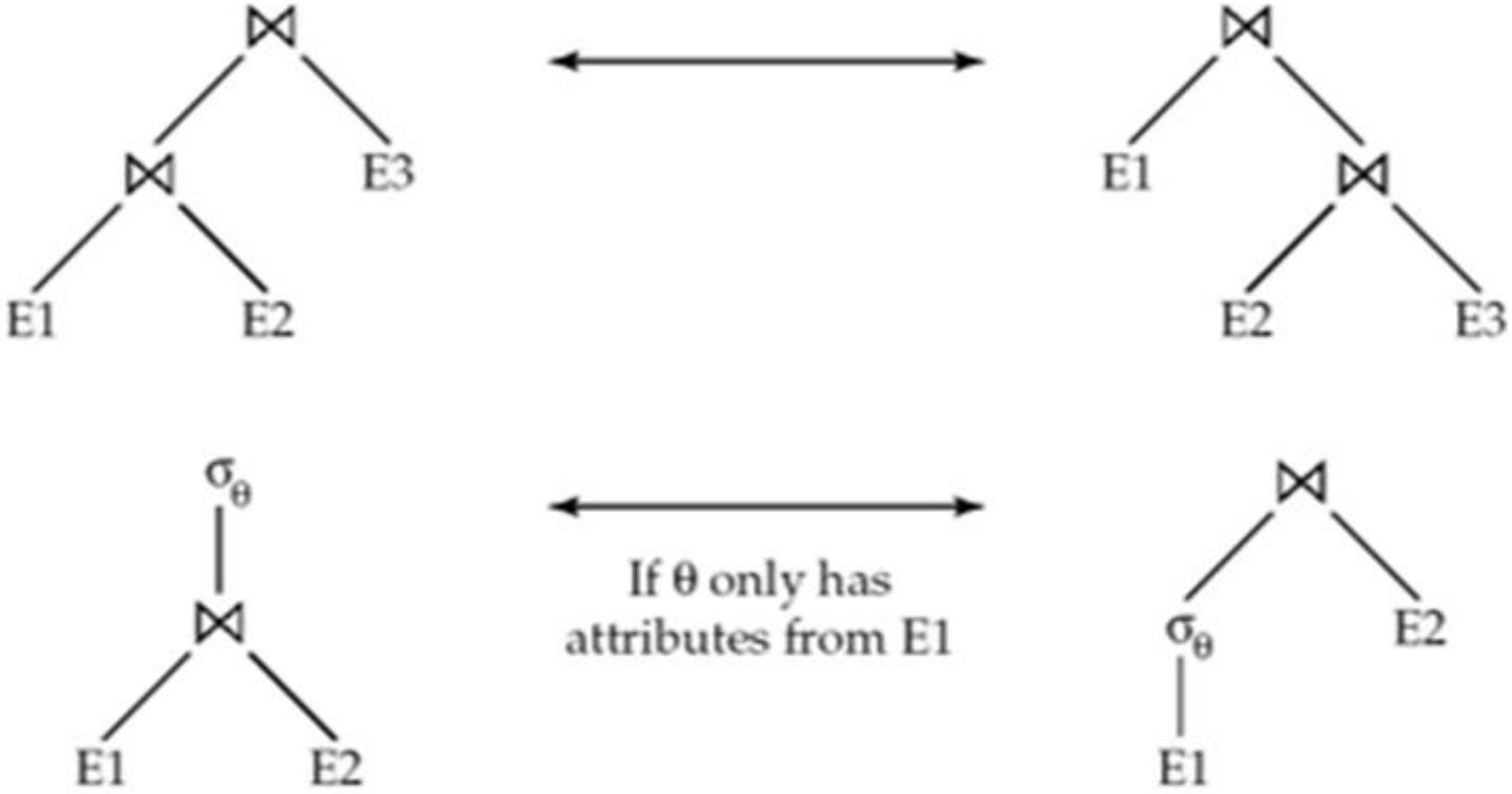
$$\Pi_\theta(r \mathbin{++} s) = \Pi_\theta(r) \mathbin{++} \Pi_\theta(s)$$

Reglas de equivalencia

15. Eliminación de duplicados distribuye con reunión natural:
    $v(r \bowtie s) = v(r) \bowtie v(s)$
16. Eliminación de duplicados conmuta con selección:
    $v(\sigma_P(r)) = \sigma_P(v(r))$

Reglas de equivalencia

- Las reglas de equivalencia se pueden representar gráficamente:



Empujando selecciones

- **Consulta**: encontrar los nombres de todos los instructores en el departamento de música, junto con los títulos de los cursos que enseñan.

$$\Pi_{name,\ title}(\sigma_{dept\_name=\ \text{“Music”}} (instructor \bowtie (teaches \bowtie \Pi_{course\_id,\ title}\ (course))))$$

- Transformación usando la regla 7:

$$\Pi_{name,\ title}((\sigma_{dept\_name=\ \text{“Music”}}\ (instructor)) \bowtie (teaches \bowtie \Pi_{course\_id,\ title}\ (course)))$$

- **Conclusión**: realizar la selección tan temprano como sea posible reduce el tamaño de la tabla a ser combinada.

Ejemplo con múltiples transformaciones

- Consulta: Encontrar los nombres de todos los instructores en el departamento de música que han enseñado un curso en 2009, junto con los títulos de los cursos que han enseñado.

$$\Pi_{name,\ title}(\sigma_{dept\_name=\ \text{“Music”} \wedge year\ =\ 2009} (instructor \bowtie (teaches \bowtie \Pi_{course\_id,\ title}\ (course))))$$

- Por asociatividad de la reunión natural (regla 4):

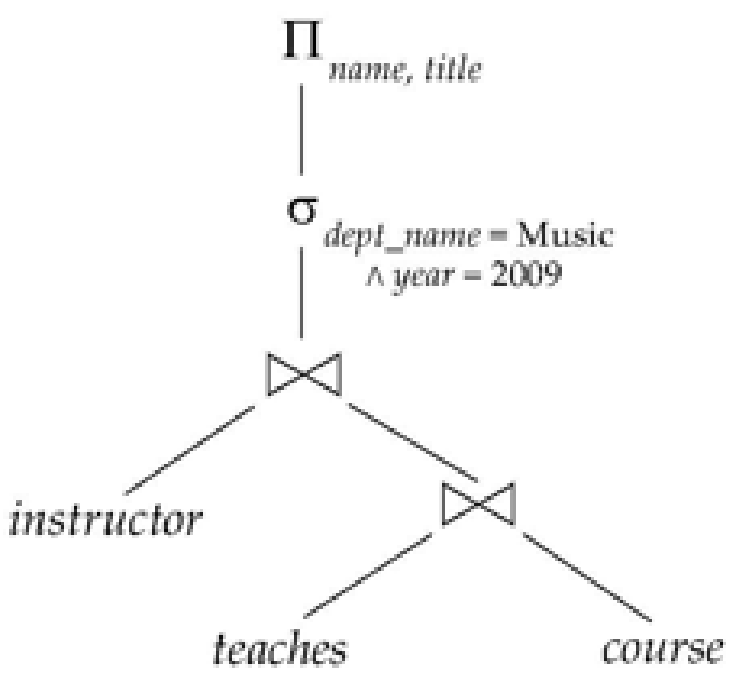
$$\Pi_{name,\ title}(\sigma_{dept\_name=\ \text{“Music”} \wedge gear\ =\ 2009} ((instructor \bowtie teaches) \bowtie \Pi_{course\_id,\ title}\ (course)))$$

- Usamos regla 7 de hacer selecciones temprano:

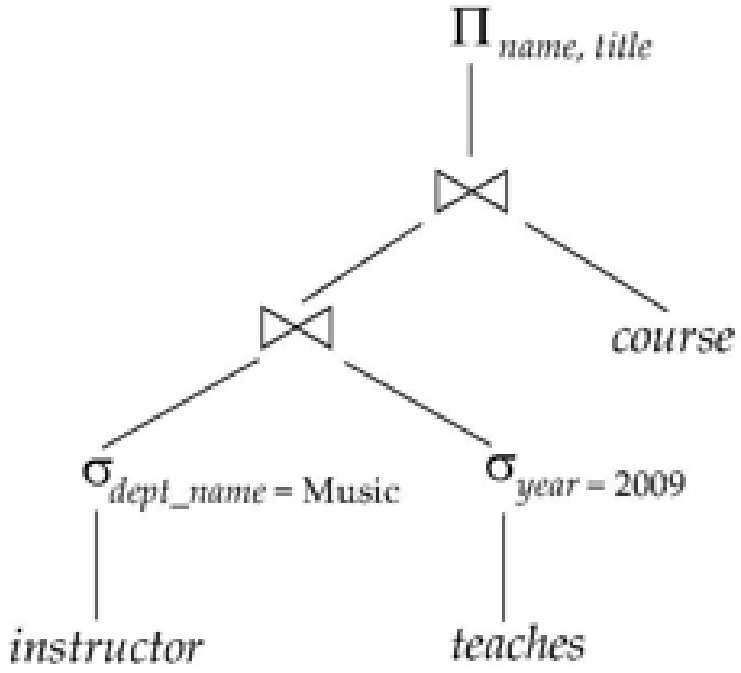
$$\sigma_{dept\_name\ =\ \text{“Music”}}\ (instructor) \bowtie \sigma_{\ year\ =\ 2009}\ (teaches)$$

Ejemplo con múltiples transformaciones

- La representación pictórica del resultado al que arribamos:



(a) Initial expression tree



(b) Tree after multiple transformations

Ordenamiento de reuniones naturales

- Para todas las tablas r_1, r_2, and r_3,

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

  (asociatividad de la reunion natural)

- Si $r_2 \bowtie r_3$ es bastante grande y $r_1 \bowtie r_2$ es pequeña, elegimos

$$(r_1 \bowtie r_2) \bowtie r_3$$

  de modo que computamos y almacenamos una tabla temporal más pequeña.

Ordenamiento de reuniones naturales

- Considere la expresión:

$$\Pi_{name,\ title}(\sigma_{dept\_name=\text{“Music”}}(instructor) \bowtie teaches) \bowtie \Pi_{course\_id,\ title}(course))))$$

- Podemos computar primero
  y luego combinamos el resultad
  pero el resultado de la primera reunión es
  grande.

$$teaches \bowtie \Pi_{course\_id,\ title}(course)$$

$$\sigma_{dept\_name=\text{“Music”}}(instructor)$$

- Solo una pequeña fracción de los instructores es probable que sean del departamento de música. Es mejor computar primero:

$$\sigma_{dept\_name=\text{“Music”}}(instructor) \bowtie teaches$$

Enumeración de expresiones equivalentes

- Optimizadores de consultas usan reglas de equivalencia para generar expresiones equivalentes a una expresión de consulta dada.
- Una idea es usar el siguiente procedimiento:

procedure genAllEquivalent(E)
$EQ = \{E\}$
repeat
    Match each expression E_i in EQ with each equivalence rule R_j
    if any subexpression e_i of E_i matches one side of R_j
        Create a new expression E' which is identical to E_i, except that
            e_i is transformed to match the other side of R_j
        Add E' to EQ if it is not already present in EQ
until no new expression can be added to EQ

Enumeración de expresiones equivalentes

- El procedimiento anterior es extremadamente costoso tanto en espacio como en tiempo.
- Los optimizadores pueden reducir enormemente el costo tanto en espacio como en tiempo usando algunas ideas.
- **Idea 1**: Requisito de espacio reducido por compartir sub-expresiones en común:
  - Cuando E1 es generado de E2 por una regla de equivalencia, usualmente solo el nivel más alto de ambos es diferente y subárboles debajo son los mismos y pueden ser compartidos usando punteros.
  - La misma sub-expresión puede ser generada múltiples veces. Detectar subexpresiones duplicadas y compartir una copia.

Enumeración de expresiones equivalentes

- **Idea 2**: requisitos de tiempo pueden ser reducidos por no generar todas las subexpresiones.
  - No es siempre necesario generar todas las expresiones que salen de aplicar reglas de equivalencia.
  - Si un optimizador toma estimación de costos de evaluación en consideración, puede evitar examinar algunas de las expresiones.
  - Para esto se pueden usar técnicas de **programación dinámica**.
    - Por ejemplo, se puede optimizar el orden de aplicación de reuniones naturales.

Optimización basada en costo

- Un **optimizador basado en costo** explora el espacio de todos los planes de evaluación de consultas que son equivalentes a la consulta dada y elige el que tiene el menor costo estimado.
  - reglas de equivalencia pueden ser usadas para generar planes equivalentes.
  - Explorar el espacio de todos los planes posibles puede ser demasiado costoso para consultas complejas.
- Un enfoque muy usado se llama **de abajo hacia arriba** (bottom-up):
  - Para cada subexpresión de un árbol de ejecución de consulta computamos los costos de todas las maneras de computar tal subexpresión.
  - Las posibilidades y costos de una subexpresión E son computados considerando las opciones para las subexpresiones de E, y combinándolas de todas las maneras posibles con implementaciones para el operador raíz de E.

Optimización basada en costo

- En el enfoque bottom-up consideramos solo el mejor plan para cada subexpresión cuando computamos los planes para una subexpresión más grande.
- Una variación de la estrategia bottom-up se llama **programación dinámica** donde mantenemos para cada subexpresión solo el plan de menor costo.
  - A medida que trabajamos hacia arriba en el árbol consideramos posibles implementaciones de cada nodo, asumiendo el mejor plan para cada subexpresión es usado.
  - Este enfoque no garantiza que se obtenga el mejor plan global, aunque a menudo lo hace.

Selección de orden de reunión basado en costo

- Cuando tenemos una reunión de dos tablas, necesitamos ordenar los argumentos. Podemos elegir aquel cuyo tamaño estimado es menor como el argumento izquierdo.
- Cuando la reunión involucra más de 2 tablas, el número de árboles posibles de reunión crece rápidamente.
- **Problema**: elegir el orden de reuniones naturales óptimo para una consulta

$$r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$$

  - Por asociatividad de reunión natural no pusimos paréntesis.

Selección de orden de reunión basado en costo

- Tenemos problemas similares para otras operaciones binarias como unión e intersección, pero estas operaciones son menos importantes en la práctica,
  - porque típicamente toman menos tiempo para ejecutar que las reuniones y
  - rara vez aparecen en grupos de 3 o más.
- Con n = 3 hay 12 diferentes órdenes de reunión natural.

$$
\begin{array}{llll}
r_1 \bowtie (r_2 \bowtie r_3) & r_1 \bowtie (r_3 \bowtie r_2) & (r_2 \bowtie r_3) \bowtie r_1 & (r_3 \bowtie r_2) \bowtie r_1 \\
r_2 \bowtie (r_1 \bowtie r_3) & r_2 \bowtie (r_3 \bowtie r_1) & (r_1 \bowtie r_3) \bowtie r_2 & (r_3 \bowtie r_1) \bowtie r_2 \\
r_3 \bowtie (r_1 \bowtie r_2) & r_3 \bowtie (r_2 \bowtie r_1) & (r_1 \bowtie r_2) \bowtie r_3 & (r_2 \bowtie r_1) \bowtie r_3
\end{array}
$$

- **Observación**: los distintos ordenes pueden cambiar también el orden de las columnas. Pero podemos proyectar según los atributos de la expresión original de la consulta para que eso no suceda.

Selección de orden de reunión basado en costo

- En general con *n* tablas hay (2(n-1))!/(n-1)! diferentes ordenaciones.
- Para n = 5 hay 1680 órdenes.
- Para n = 7 hay 665280 órdenes.
- Con n = 10 hay 17,6 mil millones de órdenes.
- **Solución**: No hace falta generar todos los órdenes de reuniones porque usando **programación dinámica** el orden de reunión de menor costo para cada subconjunto de $\{r_1,\ldots, r_n\}$ es computado solo una vez y almacenado para uso futuro.

Selección de orden de reunión basado en costo

- **Solución usando programación dinámica**:
  - Construimos una **tabla** con una entrada para cada subconjunto de una o más de las tablas de la reunión.
  - En la tabla ponemos:
    1. El tamaño estimado de la reunión de esas tablas. Ya vimos como se hace esto usando factor de selectividad.
    2. El menor costo de computar la reunión de esas tablas.
    3. La expresión que da lugar al menor costo.
  - La construcción de esa tabla se hace por **inducción** en el tamaño del subconjunto.

Selección de orden de reunión basado en costo

- **Solución usando programación dinámica (cont)**:
  - **Caso base:**
    - la entrada para una sola tabla r consiste del tamaño de r, un costo de 0 y la expresión que es r.
    - La entrada para un par de tablas $\{r_i, r_j\}$ tiene estimación de tamaño que es el producto de tamaños de r_i y r_j multiplicado por el factor de selectividad; tiene costo 0 porque no hay tablas intermedias involucradas, además tomamos la menor de r_i y r_j como el argumento izquierdo de la expresión de la reunión natural.

Selección de orden de reunión basado en costo

- **Solución usando programación dinámica (cont)**:
  - **Inducción**:
    - Consideramos todas las maneras de particionar el conjunto actual de tablas S en dos subconjuntos disjuntos S1 y S2. Para cada subconjunto consideramos la suma de:
      1. Los mejores costos de S1 y S2.
      2. Los tamaños para los resultados para S1 y S2.
    - Sea cual sea la partición que da el mejor costo, usamos esta suma como el costo de S y la expresión de S es la reunión natural de los mejores expresiones para S1 y S2.

Selección de orden de reunión basado en costo

- **Ejemplo**: considerar la reunión natural de 4 tablas R, S, T y U. Por simplicidad asumimos que cada una tiene 1000 tuplas. Además asumimos:

$R(a,b)$	$S(b,c)$	$T(c,d)$	$U(d,a)$
$V(R,a) = 100$			$V(U,a) = 50$
$V(R,b) = 200$	$V(S,b) = 100$		
	$V(S,c) = 500$	$V(T,c) = 20$	
		$V(T,d) = 50$	$V(U,d) = 1000$

- Para los conjuntos de una tabla los tamaños, costos y mejores planes son:

	$\{R\}$	$\{S\}$	$\{T\}$	$\{U\}$
Size	1000	1000	1000	1000
Cost	0	0	0	0
Best plan	R	S	T	U

Selección de orden de reunión basado en costo

- **Ejemplo (cont.):** Ahora consideramos pares de tablas.
- El costo para cada uno es 0 porque no hay tablas intermedias en la reunión de dos tablas (las 2 tablas son chicas y están en memoria principal).
- Hay dos posibles planes. Como todas las tablas tienen igual tamaño tomamos la primera en orden alfabético para ser el argumento izquierdo (independientemente del orden el tamaño es el mismo).
- Los tamaños de las tablas resultantes se calculan como dijimos. El resultado es:

	$\{R,S\}$	$\{R,T\}$	$\{R,U\}$	$\{S,T\}$	$\{S,U\}$	$\{T,U\}$
Size	5000	1,000,000	10,000	2000	1,000,000	1000
Cost	0	0	0	0	0	0
Best plan	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$

Selección de orden de reunión basado en costo

- **Ejemplo (cont.):** ahora consideramos la tabla para reuniones de 3 tablas. La única manera es tomar dos tablas para reunir primero.
- La estimación del tamaño del resultado es como siempre y omitimos los detalles de este cálculo.
- La estimación de costo para cada tripla de tablas es el tamaño de la tabla intermedia – la reunión de las 2 tablas elegidas. Como queremos que este costo sea tan pequeño como posible consideramos cada par de las 3 tablas y tomamos el par con el menor tamaño (usando la tabla previa).
- Si consideramos este par como el argumento del natural join obtenemos la tabla:

	$\{R,S,T\}$	$\{R,S,U\}$	$\{R,T,U\}$	$\{S,T,U\}$
Size	10,000	50,000	10,000	2,000
Cost	2,000	5,000	1,000	1,000
Best plan	$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(T \bowtie U) \bowtie S$

Selección de orden de reunión basado en costo

- **Ejemplo (cont.):**
  - Considerar {R, S, T}: consideramos los 3 pares posibles de tablas.
  - El costo de R ⋈ S es 5000, comenzar con R ⋈ T da un costo de 1000000, comenzar con S ⋈ T da un costo de 2000.
  - Como el último es el menor costo, elegimos el plan (S ⋈ T)⋈ R (en este caso la estimación del tamaño no depende del lugar donde ponemos R).

Selección de orden de reunión basado en costo

- **Ejemplo (cont.):**
  - Consideramos la situación de join de 4 tablas. Hay dos formas para hacerlo:
    - Tomar 3 tablas para combinar de la mejor manera posible y hacer la reunión natural con la cuarta.
    - Dividir las 4 tablas en pares de 2 y reunir los pares y luego combinar los resultados.
  - Hay 7 maneras de agrupar las reuniones naturales basados en los agrupamientos preferidos de las dos tablas previas:

Grouping	Cost
$((S \bowtie T) \bowtie R) \bowtie U$	12,000
$((R \bowtie S) \bowtie U) \bowtie T$	55,000
$((T \bowtie U) \bowtie R) \bowtie S$	11,000
$((T \bowtie U) \bowtie S) \bowtie R$	3,000
$(T \bowtie U) \bowtie (R \bowtie S)$	6,000
$(R \bowtie T) \bowtie (S \bowtie U)$	2,000,000
$(S \bowtie T) \bowtie (R \bowtie U)$	12,000

Selección de orden de reunión basado en costo

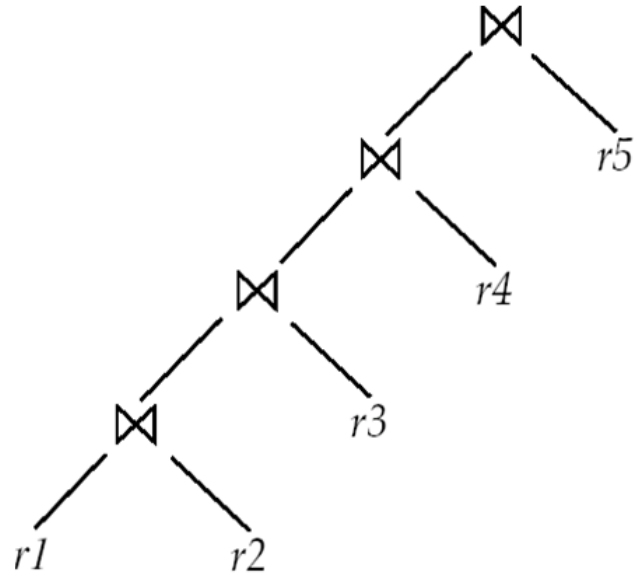
- **Ejemplo (cont.):**
  - El costo de (S ⋈ T)⋈ (R ⋈ U) es la suma de los tamaños y los costos de (S ⋈ T) y (R ⋈ U). Los costos de los pares son 0 y los tamaños son 2000 y 10000 respectivamente.
  - En la última línea de la tabla consideramos los pares (S ⋈ T) y (R ⋈ U). Debido a que elegimos la tabla más pequeña como argumento izquierdo, entonces la expresión es: (S ⋈ T)⋈ (R ⋈ U).
    - El hacer eso hace que los costos de los algoritmos vistos para reunión que tienen fórmula sea más conveniente.
  - En la tabla el menor de todos los costos es asociado con ((T ⋈ U) ⋈ S) ⋈R.
  - Esta expresión es la que se elige para computar la reunión natural; el costo es 3000.

Selección de orden de reunión basado en costo

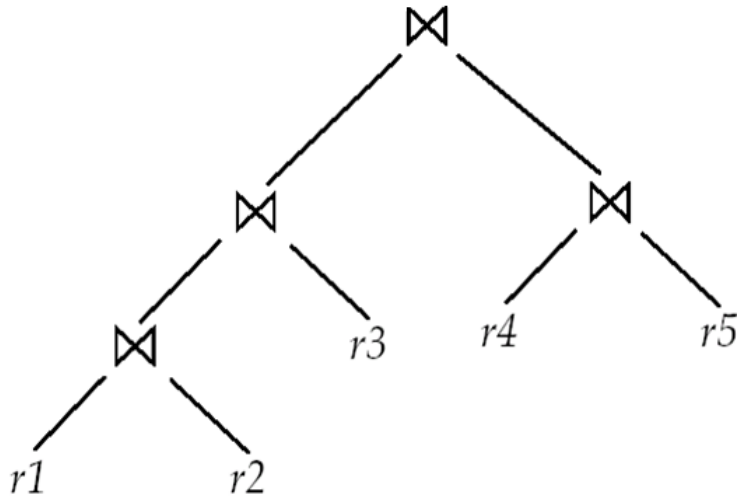
- **Evaluación del algoritmo anterior**:
  - usando tamaños de tablas como estimación de costo simplifica los cálculos en el algoritmo de programación dinámica.
  - Pero una desventaja de esta simplificación es que no involucra los costos actuales de las reuniones en el cálculo.
- **Solución**: modificar el algoritmo de programación dinámica anterior para tomar en cuenta algoritmos de reunión natural. Cuando se computa el costo de $S1 \bowtie S2$ sumamos el costo de S1, el costo de S2 y el menor costo de juntar los dos resultados usando el mejor algoritmo disponible.
  - La complejidad en tiempo del procedimiento puede probarse que es $O(3^n)$.
  - Con n = 10 este número es 59000 en lugar de 176 mil millones.
  - La complexidad en espacio es $O(2^n)$.

Árboles de reunión profunda a la izquierda

- En **árboles de reunión profunda a la izquierda**, el lado derecho de cada reunion natural es una table no el resultado de una reunion intermedia.



(a) Left-deep join tree



(b) Non-left-deep join tree

Árboles de reunión profunda a la izquierda

- **Para encontrar el mejor árbol de reunión profunda a la izquierda para un conjunto de n tablas**:
  - Modificar el algoritmo de programación dinámica anterior:
    - Si S tiene k tablas, para cada r en S, primero computamos la reunión de S – {r} y luego hacemos la reunión natural con r.
    - El costo de la reunión de S es el costo de S – {r} más el tamaño del resultado para S - {r}. Tomamos el r que da el menor costo.
    - La expresión de reunión para S tiene la mejor expresión reunión para S – {r} como argumento izquierdo de la reunión final y r como el argumento derecho.
    - El tamaño de S se calcula por la fórmula que usa factor de selectividad.
  - Si solo árboles de reunión profunda a la izquierda son considerados, el tiempo de complejidad para encontrar el mejor orden de reunión es: $O(n\ 2^n)$.
  - La complejidad en espacio permanece en $O(2^n)$.

Optimización basada en costo

- En los algoritmos anteriores, la optimización basada en costo es cara, pero vale la pena para consultas en conjuntos de datos grandes (las consultas típicas tienen un n pequeño, generalmente < 10).

Optimización basada en costo con reglas de equivalencia

- Bosquejamos cómo crear **un optimizador basado en costo de propósito general basado en reglas de equivalencia** que pueden manejar una amplia variedad de operadores de consulta.
  - El beneficio de usar reglas de equivalencia es que es fácil extender el optimizador con nuevas reglas que manejen diferentes construcciones de consultas.
- El procedimiento que genera todas las expresiones equivalentes a una consulta puede ser modificado para generar todos los planes de evaluación posibles.
- Una nueva clase de reglas de equivalencia llamada **reglas de equivalencia físicas** es agregada que permite que una operación lógica sea transformada en una operación física.

Optimización basada en costo con reglas de equivalencia

- Agregando tales reglas al conjunto original de reglas de equivalencia, el procedimiento puede generar todos los planes de evaluación posibles.
- Las técnicas de estimación de costo que vimos antes pueden ser usadas para elegir el plan óptimo.
- **Problema**: Pero el procedimiento es muy costoso.
- **Solución**: para hacer el enfoque trabajar eficientemente requiere de lo siguiente:
  1. Una representación eficiente en espacio de expresiones que permite hacer múltiples copias de las mismas subexpresiones cuando se aplican las reglas de equivalencia.
  2. Técnicas eficientes para detectar derivaciones duplicadas de la misma expresión.

Optimización basada en costo con reglas de equivalencia

3. Una forma de programación dinámica basada en memorización que almacena el plan de evaluación de consulta óptimo para una sebexpresión cuando es optimizado por la primera vez;
   - pedidos subsiguientes para optimizar la misma subexpresión son manejados retornando el plan ya memorizado.
4. Técnicas que evitan generar todos los posibles planes equivalentes.
   - Mantener la pista del plan más barato generado para toda subexpresión hasta aquí, y podando todo plan que es más caro que el plan más barato encontrado hasta aquí para una subexpresión.

- No entramos en más detalles debido a su complejidad.
- Este enfoque es adoptado por el optimizador de SQL server.

Optimización Heurística

- **Problema**: Optimización basada en costo es cara incluso con programación dinámica.
  - Aunque el costo de optimización de consultas puede ser reducido por algoritmos inteligentes, el número de diferentes planes de evaluación para una consulta puede ser muy grande y encontrar el plan óptimo para ese conjunto requiere mucho esfuerzo computacional.
- Una **heurística** es un enfoque para resolución de problemass que usa un método práctico o varios atajos con el fin de producer soluciones que pueden no ser óptimas pero son suficientes dado el tiempo o recursos limitados.
  - También se usa la frase: **estrategia de regla de pulgar**.

Optimización Heurística

- **Solución**: Los sistemas pueden usar heurísticas para reducir el número de elecciones que deben ser hechas cuando se trabaja con costos.
  - La **optimización heurística** transforma el árbol de consulta usando un conjunto de reglas que típicamente pero no siempre mejoran el desempeño de ejecución.
- La mayoría de los optimizadores incluye **heurísticas** para reducir el costo de la optimización de consultas,
  - con el riesgo potencial de no encontrar un plan óptimo.

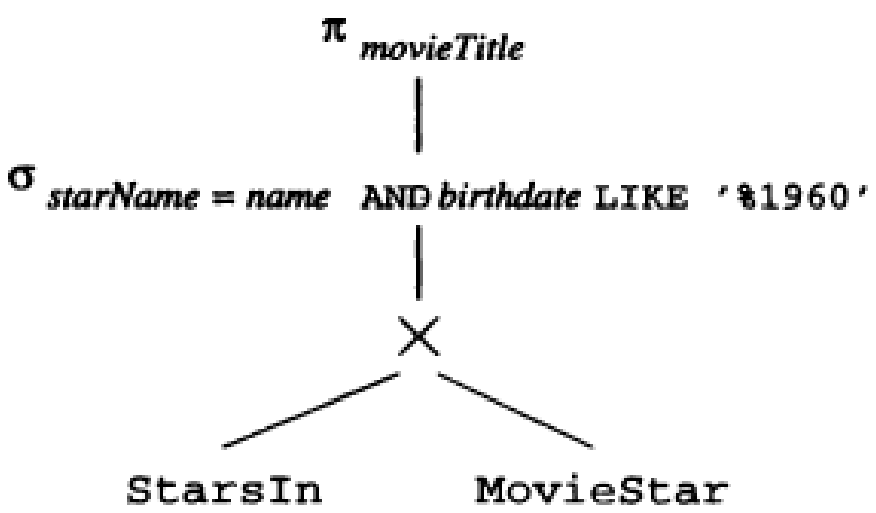
Optimización Heurística

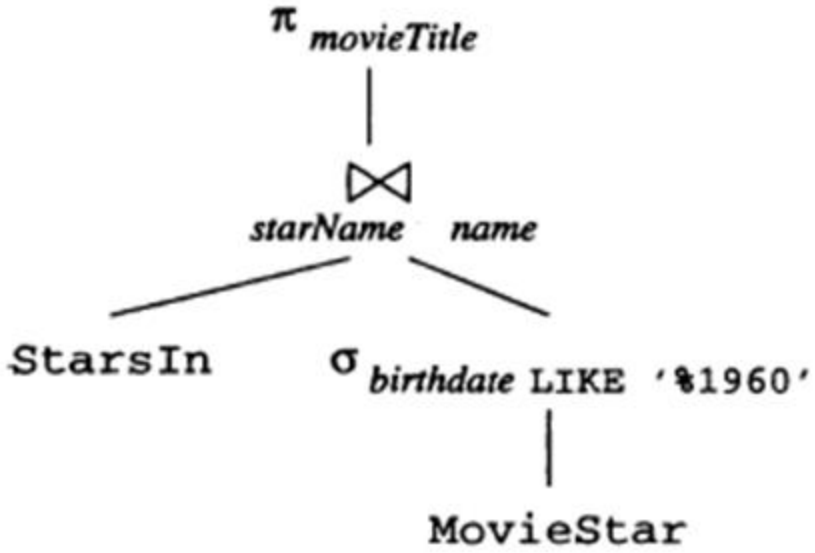
- **Ejemplo de optimización heurística:**
  - Realizar selección tempranamente (reduce el número de tuplas)
    - Empujar selección abajo en el árbol de ejecución tanto como sea posible.
    - Si una condición de selección es un AND de varias condiciones, podemos dividir la condición y empujar cada pieza abajo en el árbol de ejecución separadamente.
  - Realizar proyección temprano (reduce el número de atributos)
    - Proyecciones pueden ser empujadas abajo en el árbol.
  - Hacer la selección más restrictiva (i.e. con tamaño de resultado menor) antes de hacer otras selecciones.
    - P.ej. Si una condición de selección es un AND de varias condiciones, podemos aplicar selecciones sobre esas condiciones separadamente en algún orden buscando hacer la selección lo más restrictiva posible.

Optimización Heurística

- **Ejemplo de optimización heurística (cont):**
  - Hacer operaciones de reunión más restrictivas:
    - Restringir a tipos particulares de ordenes de reuniones (p.ej. Ordenes de reunión profunda a la izquierda)
  - Ciertas selecciones pueden ser combinadas con producto abajo para tornar las operaciones en una reunión (natural o selectiva),
    - la cual es generalmente más eficiente de evaluar esas operaciones separadamente.
    - Para esto suele ser conveniente introducir una proyección también.

Optimización Heurística





Ejemplo de uso de optimización heurística anterior: primero dividimos las dos partes de la selección en $\sigma_{\text{starName = name}}$ y $\sigma_{\text{birthdate LIKE '%1960'}}$. La última puede ser empujada abajo en el árbol debido a que el único atributo involucrado *birthdate* es de tabla *MovieStar*. La primera condición involucra atributos de ambos lados del producto y se puede introducir proyección de todos los atributos menos *name* pues la proyección de la consulta es mucho más restrictiva. Así, podemos aplicar definición de reunión selectiva según *starName* y *name*. Haciendo todo esto obtenemos el árbol de la derecha.

Optimización Heurística

- También una optimización heurística puede referirse a operadores físicos a usar.
- **Ejemplo de optimización heurística**:
  1. Para hacer reunión de varias tablas, comenzar juntando el par de tablas cuyo resultado tiene el tamaño estimado menor; luego repetir el proceso para el resultado de esa reunión y las otras tablas en el conjunto a ser combinado.
  2. Si el plan lógico usa una selección $\sigma_{A=C}$ (r) y r tiene un índice en el atributo A, realizar escaneo de ese índice en r (vimos 2 algoritmos para esto).

Optimización Heurística

- **Ejemplo de optimización heurística (cont)**:
  3. Si un argumento de una reunión tiene un índice en atributos de la reunión para la tabla de la derecha, usar reunión de loop anidado indexada.
  4. Si in argumento de la reunión está ordenado en los atributos de la reunión, es preferible usar sort-join a hash-join.
  5. Cuando se computa la unión o intersección de 3 o más relaciones, agrupar las más pequeñas primero

Enfoques híbridos

- **Enfoques híbridos**: Algunos sistemas gestores de BD combinan heurísticas con optimización parcial basada en costo.
  - Enfoques de optimización de consultas que aplican elecciones heurísticas de plan para algunas partes de la consulta con elección basada en costo basada en la generación de planes alternativos para otras partes de la consulta han sido adoptados en varios SGBD.
- **Ejemplo**: Muchos optimizadores siguen un enfoque basado en usar transformaciones heurísticas para manejar construcciones diferentes de reuniones y aplican el algoritmo de orden de reunión optimizado por costo para subexpresiones que involucran solo reuniones y selecciones.
  - Los detalles de tales heurísticas son específicos a optimizadores individuales.

Enfoques híbridos

- **Ejemplo**: Varios optimizadores consideran solo reunión profunda a la izquierda más heurísticas que empujan selecciones y proyecciones hacia abajo en el árbol de consulta.
  - Esto reduce la complejidad de la optimización y genera planes que permiten evaluación usando tubería.

Manejo de presupuesto de costo

- **Problema**: ¿Cómo evitar que el optimizador de consultas esté demasiado tiempo buscando el plan óptimo?
- **Solución**: La mayoría de los optimizadores permiten un **presupuesto de costo** ser especificado para optimización de consultas. La búsqueda del plan óptimo es terminado cuando el presupuesto de optimización de costo es excedido y el mejor plan encontrado hasta ese punto es retornado.

Manejo de presupuesto de costo

- **Problema**: el presupuesto puede ser demasiado alto o demasiado bajo.
- **Solución**: El presupuesto puede también ser **manejado dinámicamente**:
  - Si un plan barato es encontrado para una consulta, entonces el presupuesto de costo puede ser reducido.
  - Si el plan escogido hasta aquí es muy costoso, tiene sentido invertir más tiempo en optimización.
- Para explotar mejor esta idea, optimizadores usualmente aplican primero heurísticas baratas para encontrar un plan y luego comienzan optimización completa basada en costo con presupuesto basado en el plan escogido heurísticamente.

Conclusiones

- Incluso con el uso de heurísticas la optimización basada en costo impone una sobrecarga sustancial en el procesamiento de consultas.
  - Pero vale la pena para consultas complejas.
- El ahorro por optimización de consulta puede ser magnificado en aquellas aplicaciones que corren de manera regular donde una consulta puede optimizarse una vez y el plan de consulta seleccionado puede ser usado cada vez que la consulta es ejecutada.