Panorama, optimización de consultas a bases de datos

Temas

Porque optimización

Marco de optimización

- Operadores físicos
- Operadores lógicos
- Reglas de operadores

Necesidades: Costos y estadísticas

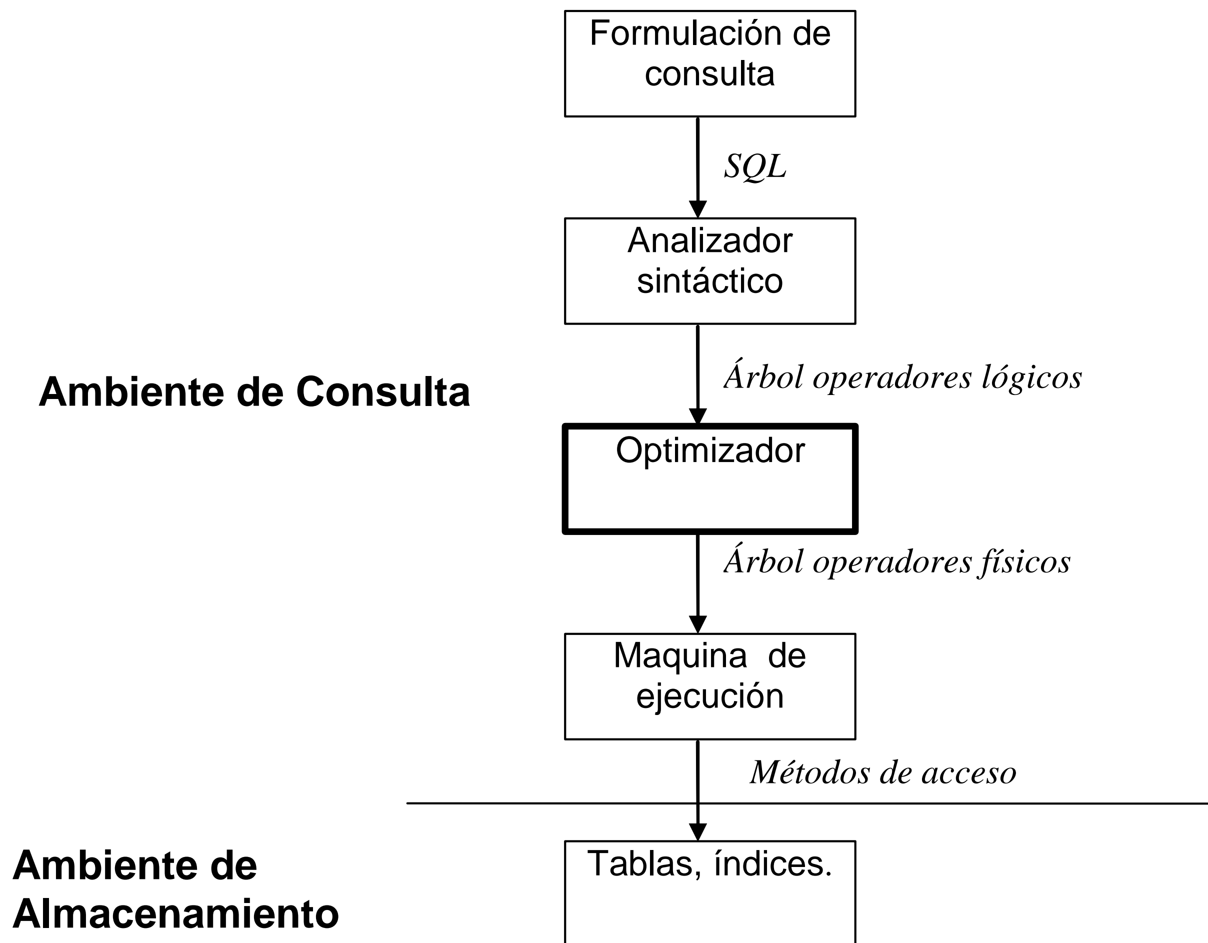
Estilos de optimización (Enumeración de planes)

Problemas relevantes

Cómo convivir con un optimizador

¿Porqué Optimización?

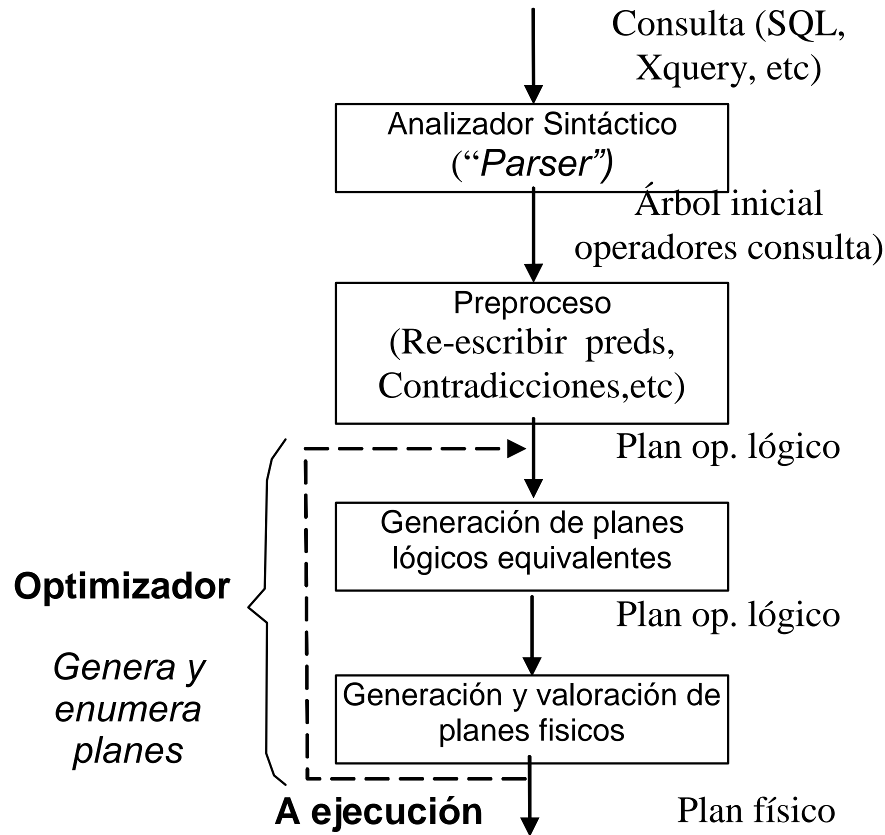
Posición del optimizador



¿Porqué Optimización?(cont)

- Lenguaje de consulta declarativo, sin importar estructura física de B.D.
- La ejecución procedural, toma en cuanta caract. físicas + operadores de la maquinaria de ejecución
- Estrategia de traducción a plan de ejecución <u>muy</u> importante para desempeño.
- Exploración de estrategias, generalmente combinatoria
- Pero…lo caro (y fino) es la maquinaria de ejecución.
- El optimizador no es tan caro de desarrollar

Marco de Optimización



Marco de Optimización (cont.)

Acceso a ambiente almacenamiento

Operadores físicos

- Asociados con fórmulas de costo.
- Encarnación de un operador lógico. Cada tipo proviene de uno (o dos, o unos pocos) tipos de operaciones lógicos
- Entradas: uno o dos data stream (aunque algunos pueden tener mas). Salida: un data-stream
- Árbol de ejecución. Costo asociado. Importa el estado del stream o streams de datos que le entran

Operadores lógicos

- Provienen ya sea de *parsing* o de reglas de transformación entre operadores lógicos.
- Entrada: una o mas relaciones; salida, una relación
- Árbol de operadores, ancestro de los físicos. No importa estado del stream, pero sí sus vecinos dentro del árbol .

Marco de Optimización (cont)

El ambiente de almacenamiento: Guarda tablas e índices (+ transacciones)

Tablas

- Particiones + replicaciones
- "Clustering"
- Operaciones: scan, GetTuples

Índices

- Implementación: Btree, Bitmap, etc.
- Tipos: Una columna, múltiples columnas, UDF's
- Operaciones: Scan, intersect, etc.

Marco de Optimización (cont)

Operadores físicos, Tablas

- *Scan*
- *Index Scan* (importante tener índice adecuado)
- *Select* (*Scan* + predicado)
- *Get*

Marco de Optimización (cont)

Operadores físicos, Joins

- Asociados con un cierto estado de salida:
  - ✓ *blocking* vs. *pipelining*
  - ✓ *sorted* vs. *non-sorted*
- Desempeño caracterizado por estado de salida de sus hijos.

Método	¿Blocking?	Sort	Requerimientos
NestedLoopJoin	No	Del lado izquierdo (si existe)	Nada. Lado Izq puede ser pipelined.
NLIndexJoin	No	Del lado izquierdo	Índice Lado Der, Lado Izq puede ser pipelined
SortJoin	Sí	Sort atributos	Sort en los streams de entrada
HashJoin	Sí	Ninguno	

Marco de Optimización (cont)

Operadores físicos, *GroupBy*, *Aggregates*

- Bloqueados
- Usan *Sort*, *Hash*.

Marco de Optimización

Operadores Lógicos

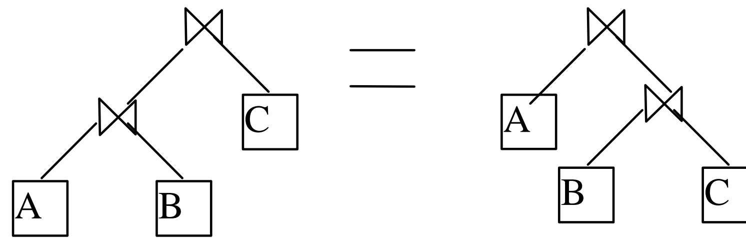
Operadores Lógicos

- Los mismos que los del Algebra relacional + agregados + algunos más (joins, *outer joins*, *semijoins*, *Subquery*, *CrossApply*, etc.)
- Forman subárboles, con uno o más hijos
- Tienen asociadas reglas de transformación
  - ✓ Entre operadores lógicos
  - ✓ De operadores lógicos a físicos

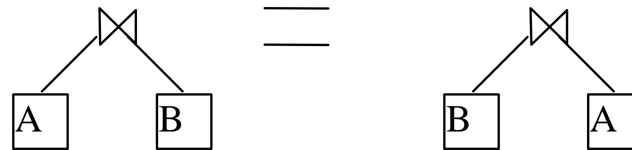
Marco de Optimización (cont)

Reglas de transformación de operadores lógicos.

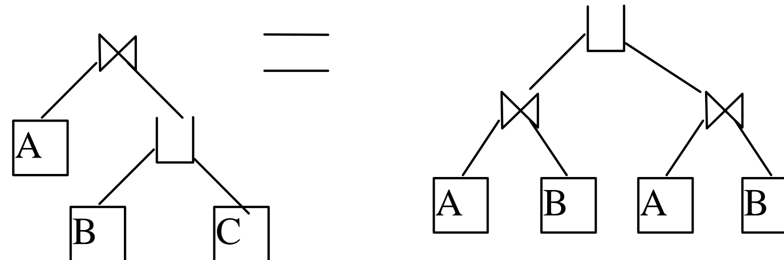
Asociativas



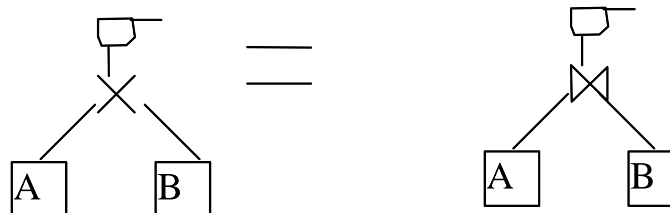
Conmutativas



Distributivas



Composición



Marco de Optimización

Reglas de transformación de operadores lógicos (cont)

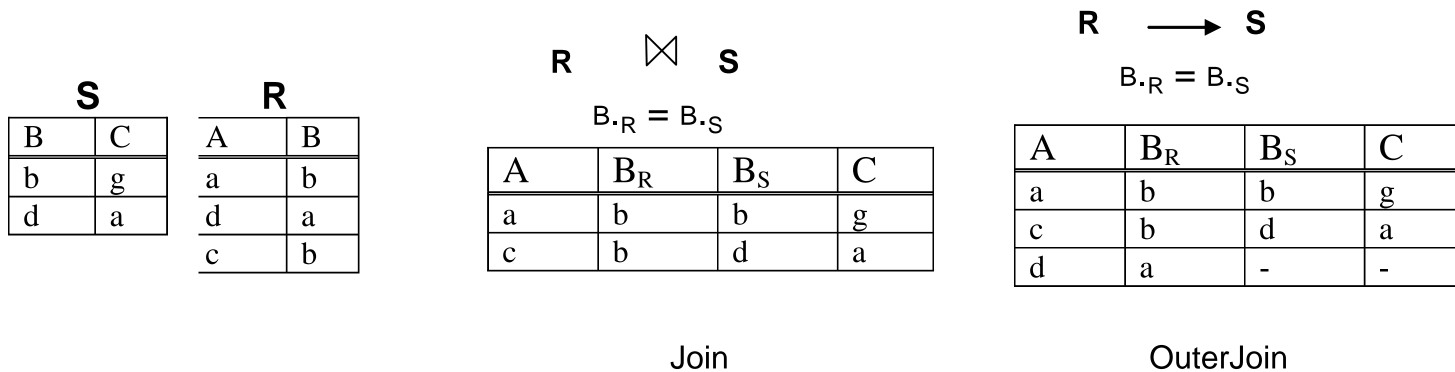
Algunas reglas especiales

- *Joins-OuterJoins*
- *GroupBy*
- *SubQueries*

Marco de Optimización

Reglas de transformación de operadores lógicos (cont)

Ordenamiento de OuterJoins, Joins.



S

B	C
b	g
d	a

R

A	B
a	b
d	a
c	b

$R \bowtie_{B.R = B.S} S$

A	B_R	B_S	C
a	b	b	g
c	b	d	a

Join

$R \rightarrow_{B.R = B.S} S$

A	B_R	B_S	C
a	b	b	g
c	b	d	a
d	a	-	-

OuterJoin

Secuencias de Joins-OuterJoins no pueden ordenarse (asociarse) en cualquier orden.

Marco de Optimización

Reglas de transformación de operadores lógicos (cont)

Regla para la libre asociación de Joins-outerjoins

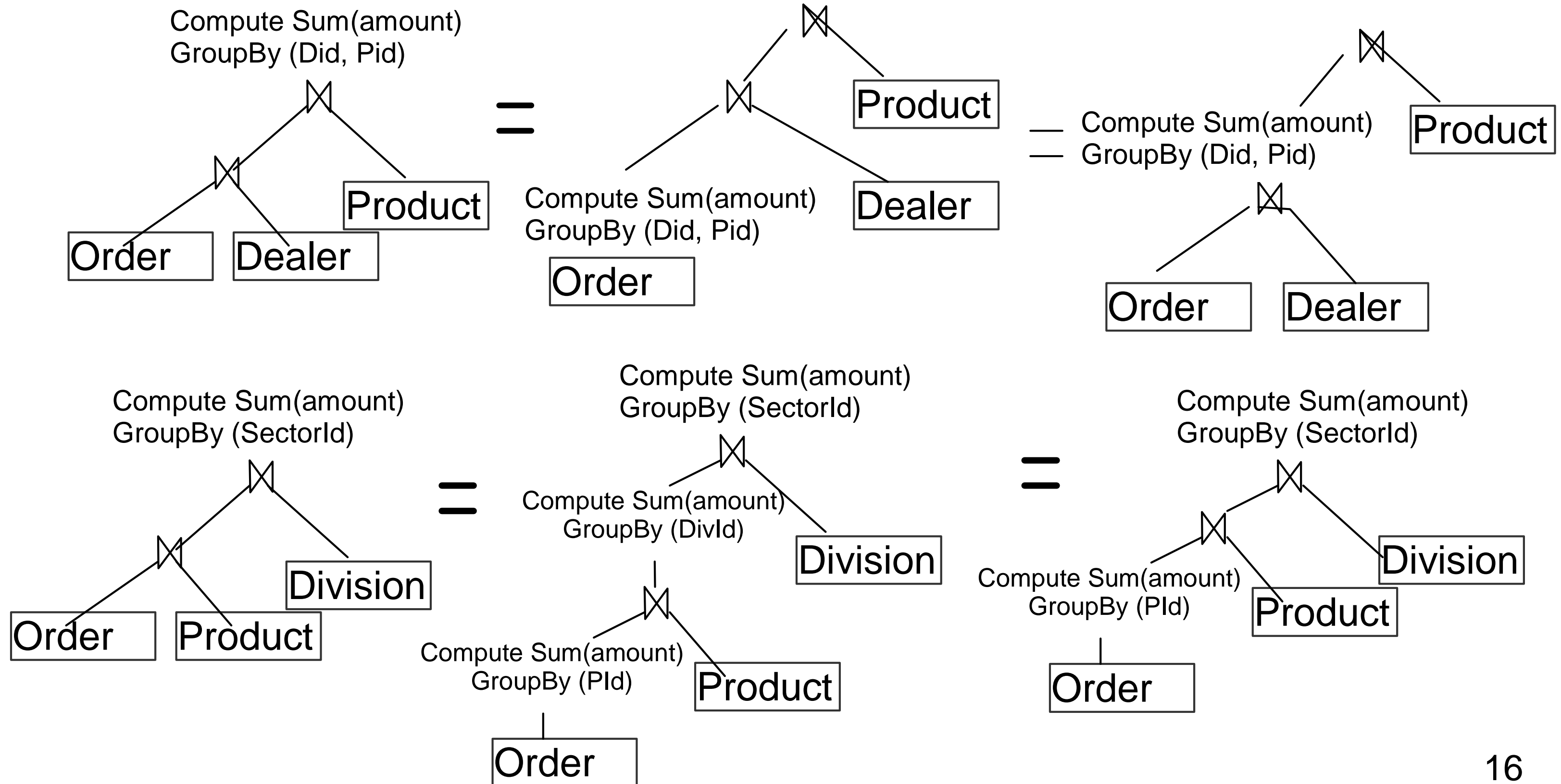
Una grafica de joins y OuterJoins que puede ejeturarse en cualquier orden

Marco de Optimización

Reglas de transformación de operadores lógicos (cont)

Empujar GroupBy, Aggregates en Joins.

Order(**Oid**, Pid, Amount, Division(**DivId**, ,descripcion, SectorId)
DealerId), Dealer(**DealerId**, Nombre, Dir), Product(**Pid**, DivId,descripcion)



Marco de Optimización

Reglas de transformación de operadores lógicos (cont)

Subqueries

OPCIONES

Subquery Correlacionada

```
SELECT C_CUSTKEY  FROM
CUSTOMER
WHERE 1000000<
   (SELECT SUM(O_TOTALPRICE)
   FROM ORDERS
   WHERE O_CUSTKEY=C_CUSTKEY)
```

a) Aggregate , después JOIN

```
SELECT C_CUSTKEY FROM CUSTOMER,
   (SELECT O_CUSTKEY FROM ORDERS
   GROUP BY O_CUSTKEY HAVING
   1000000<SUM(O_TOTALPRICE)) AS
   AGGRESULT
WHERE O_CUSTKEY=C_CUSTKEY)
```

b) OUTERJOIN, después Aggregate

```
SELECT C_CUSTKEY
FROM CUSTOMER LEFT OUTER JOIN
   ORDERS ON O_ CUSTKEY=C_CUSTKEY
GROUP BY C_KUSTKEY
HAVING 1000000 < SUM(O_TOTALPRICE)
```

Marco de Optimización

Reglas de transformación de op lógicos, *Subquery* (cont)

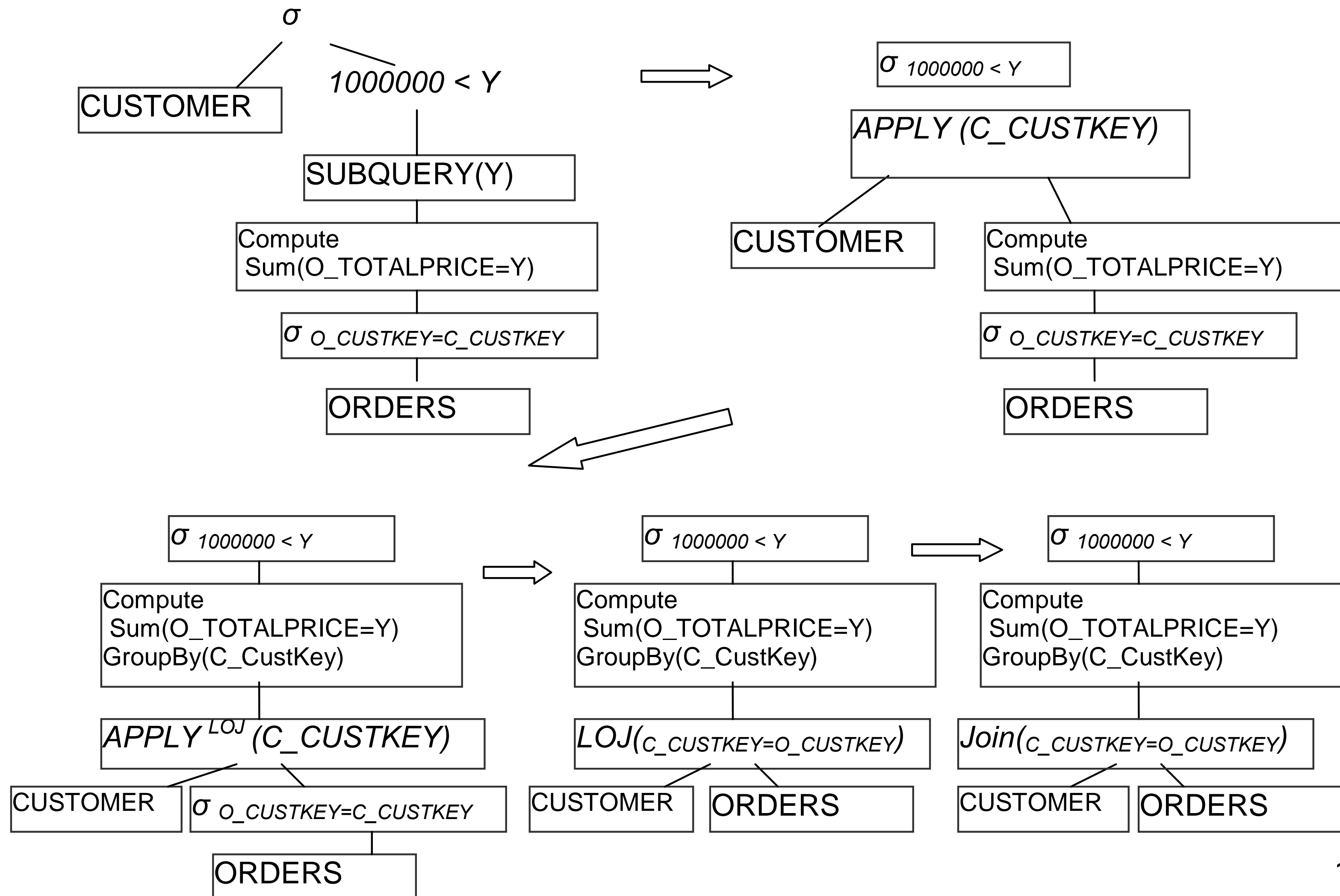
Operador *APPLY* ^{Op}

- Usado para sustituir a *Subquery*.
- Operador Lógico (tiene contraparte física):
- Tiene dos parámetros
  - Param. izq es una relación R,
  - Param, der, una expresión parametrizada E(R). Aplica la expresión a cada tuplo de R.
- Operador *Op* es uno de *CrossProduct*, *LeftOuterJoin*, *LeftSemijoin*, *LeftAntijoin* (default es CrossProduct)
- Reglas para empujarlo a través de Uniones, Diferencias, *CrossProducts*, Selecciones, *GroupBy.* Termina dando operadores estándar

Marco de Optimización

Reglas de transformación de op lógicos, *Subquery* (cont)

Introduciendo y moviendo un *Apply*



Necesidades para optimización

Para optimizar, necesario criterios de utilidad.

B. de Datos existentes, típicamente:

- Usan función de costo escalar
- Miden costo con formula que combina tiempo de acceso a disco +("factor $suizo_1$")*Tiempo CPU
- Otras posibilidades (e.g. múltiples objetivos, como el costo anterior **y** el tiempo del primer tuplo proporcionado por la consulta)

Necesidades para optimización (cont)

Para encontrar costos de acceso y de procesamiento, necesario tener estadísticas + información sobre clustering + información sobre dependencias funcionales

Estadísticas: Típicamente,

- # de registros de la tabla + histograma de cada columna + valores significativo
- Estadísticas compuestas se obtienen partiendo de suposición de independencia.
- No hay estadísticas de pares de columnas (en la misma relación o en diferentes)
- B. de datos las obtiene en “tiempos libres”, a partir de muestreo o de fuerza bruta.

Necesidades para optimización (fin)

Con datos estadísticos + esquema físico, se pueden calcular los costos.

."There are three kinds of lies:
lies, damned lies, and statistics.“
Benjamin Disraeli

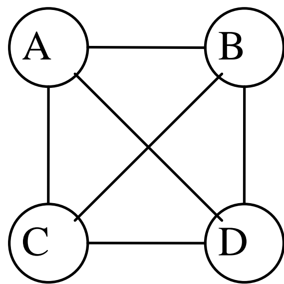
.“Yeah, it’s crooked, but it’s the
only game in town.“
Película de vaqueros

Estilos de optimización

- Con base en los métodos de generación de alternativas (planes lógicos, planes físicos + costos, necesario explorar (enumerar) alternativas de planes
- Tres estilos principales:
  1) De abajo hacia arriba: de sub-planes a plan. Se usa p.ej, para programación dinámica (Oracle, IBM)
  2) De arriba hacia abajo: De plan, se generan subplanes (exploración tipo SQLServer, Sybase, Postgres, Opt++)
  3) Muestreo: Se generan planes con (a veces) distribución uniforme

Estilos de optimización: Abajo hacia arriba

Programación Dinámica: Se generan soluciones óptimas de subproblemas, con ellas se generan soluciones óptimas de problemas mayores.



PROGRAMA DE JUNTAS

					ABCD	C_{min} Consulta (4Juntas)
ABC		ABD		ACD	BCD	C_{min} Tercetas Juntas
AB	AC	AD	BC	BD	CD	C_{min} Pares Juntas
	A	B	C		D	C_{min} Acceso Relacion

Principio de Optimalidad

$$C_{min}(ABC) = \min \begin{cases} C_{min}(AB) + C_{min}(C) + Costo(AB, C) \\ C_{min}(BC) + C_{min}(A) + Costo(BC, A) \\ C_{min}(AC) + C_{min}(B) + Costo(AC, B) \end{cases}$$

Estilos de optimización: Abajo hacia arriba (cont)

Programación dinámica:

- Reduce el numero de búsquedas de O(n!) a $O(n\ 2^n)$
- Subproblemas deben considerar estado de salida de solución (*blocking*, orden de *sort*)
- Ejemplo consideraba sólo *joins*: sistemas comerciales incluyen reglas de transformación entre operadores

Estilos de optimización: Abajo hacia arriba
(fin)

Aun así, $O(n\ 2^n)$ es muy grande. ¿Cómo bajar número de estados intermedios de programación dinámica?

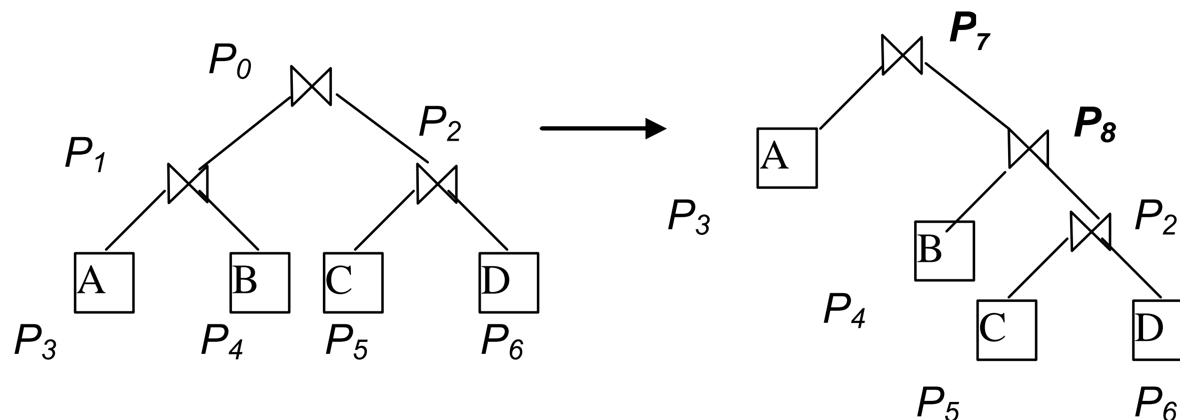
Algunas Opciones:

- Usar programación dinámica iterativa (si tienes "*n*" relaciones, usa P.D. para obtener los planes de "*k*" relaciones (*n*>*k*). Toma el mejor de esos "k" planes, hazlo fijo, y optimiza con el resto)
- Podar el árbol (*pruning*) - Antes de expandir obtén para cada nodo, cota superior de costo faltante. Expande los nodos mas baratos (costo+ estimado)
- Poda agresiva -como anterior + expande sólo un numero fijo de opciones
- Considerar sólo árboles zurdos para join *(left-deep)* – evita árboles frondosos *(bushy)*

Estilos de optimización: Arriba hacia abajo

- Se tiene un plan original, se modifican sus subplanes (recursivamente) según reglas
- Se tiene una estructura (*memo*) para evitar dar ciclos
- Las reglas pueden ser transformaciones lógicas o físicas.

Ejemplo: Al árbol original con raíz en P_0 se le aplica una transformación de asociatividad de juntas



Se generan dos (sub) árboles P_7 y P_8. P_7 es equivalente a P_0

Estilos de optimización: Arriba hacia abajo (cont)

Estructura memo

- Una para cada nodo (fisico, logico). Tienen Id.

Op Lógico:

- Operador +Id $_{hijoIzq}$+Id$_{hijoDer}$+ parámetros.
- Lista de reglas aplicadas.
- Clase de equivalencia

Op Fisico:

- Operador +Id $_{hijoIzq}$+Id$_{hijoDer}$+ parámetros.
- Estado de salida (*blocking*, orden de *sort*, etc.)
- Costo (minimo entre todas implementaciones físicas de las clase de eq. de hijos)

Estilos de optimización: Arriba hacia abajo (fin)

¿Qué falta describir?

Políticas para:

- Dirigir qué nodos hay que transformar
- Uso de “branch and bound”
- Cuándo parar la optimización

Comentarios

- Con las “reglas de podado”, los dos enfoques citados no difieren tanto.
- Hay sistemas (p.ej. Postgres) que los combinan: de abajo-arriba para preguntas con 5 tablas o menos, de arriba-abajo para preguntas mas complejas.

Estilos de optimización Generación aleatoria

- Usado generalmente para “orden de joins”
- Da resultados muy rápido resultados aceptables (no pierde el tiempo saliendo de mínimos locales)
- Métodos de generación uniforme: no mejores que los de generación con sesgo.

Problemas Interesantes

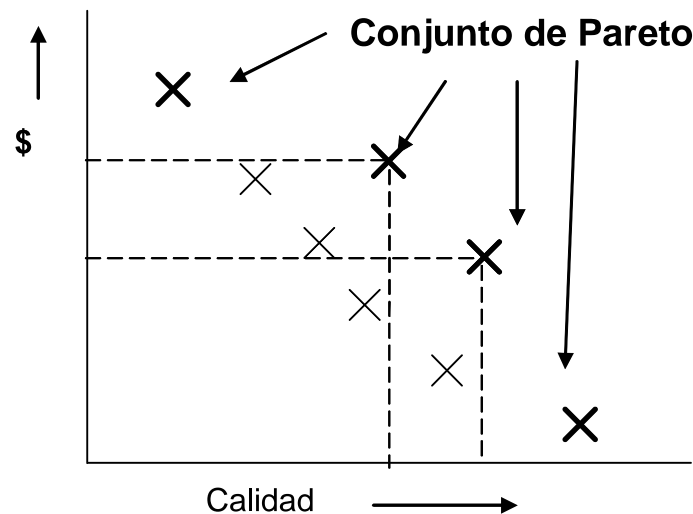
- Múltiples objetivos
- XML
- UDF's, uso de semántica
- Optimización robusta/adaptable

Advertencia

Opiniones personales, altamente sesgadas

Problemas Interesantes (múltiples objetivos)

- Optimización estándar usa un sólo criterio (e.g.,costo)
- ¿Qué pasa cuando se toman en cuenta 2 o mas criterios?



P.ej, usa “tiempo de procesamiento” y “tiempo de inicio de respuesta”

Problemas Interesantes (XML, XQuery)

- Las principales B. de Datos relacionales implementan XML sobre ellas
- Problema: bases de datos nativas.
- ¿Qué cambia?:
  - Nuevos operadores lógicos, nuevas reglas de transformación
  - Nuevos operadores físicos (muchos más tipos de índices, nuevos métodos de *join* estructural)
- Algo parecido está pasando con RDF y SPARQL

Probs. Interesantes (Opt. robusta/adaptable)

- No siempre las cosas salen como deseadas. Bueno poder cambiar a un mejor plan
- Dos enfoques:
  - Adaptación a nivel de plan de operadores (e.g. Leo)
  - Adaptación a nivel de tuplo (*eddies*)

Problemas Interesantes (UDFs)

- Una Table-UDF con tablas de entrada, equivalente a un nuevo operador lógico
- Una UDF escalar con tablas de entrada y escalar de salida es equivalente a un metodo de agregación o a una expresión de *subquery*
- Generalmente no tienen reglas de transformación, ni costos, etc.
- Si el usuario define sus reglas, puede equivocarse y meter contradicciones, falsedades.

Opt. robusta/adaptable (plan de operadores)

- Genera un plan con puntos de control (*checkpoints)*
- Al llegar a un "*checkpoint*", ¿Va todo razonablemente bien? Si no, genera otro plan

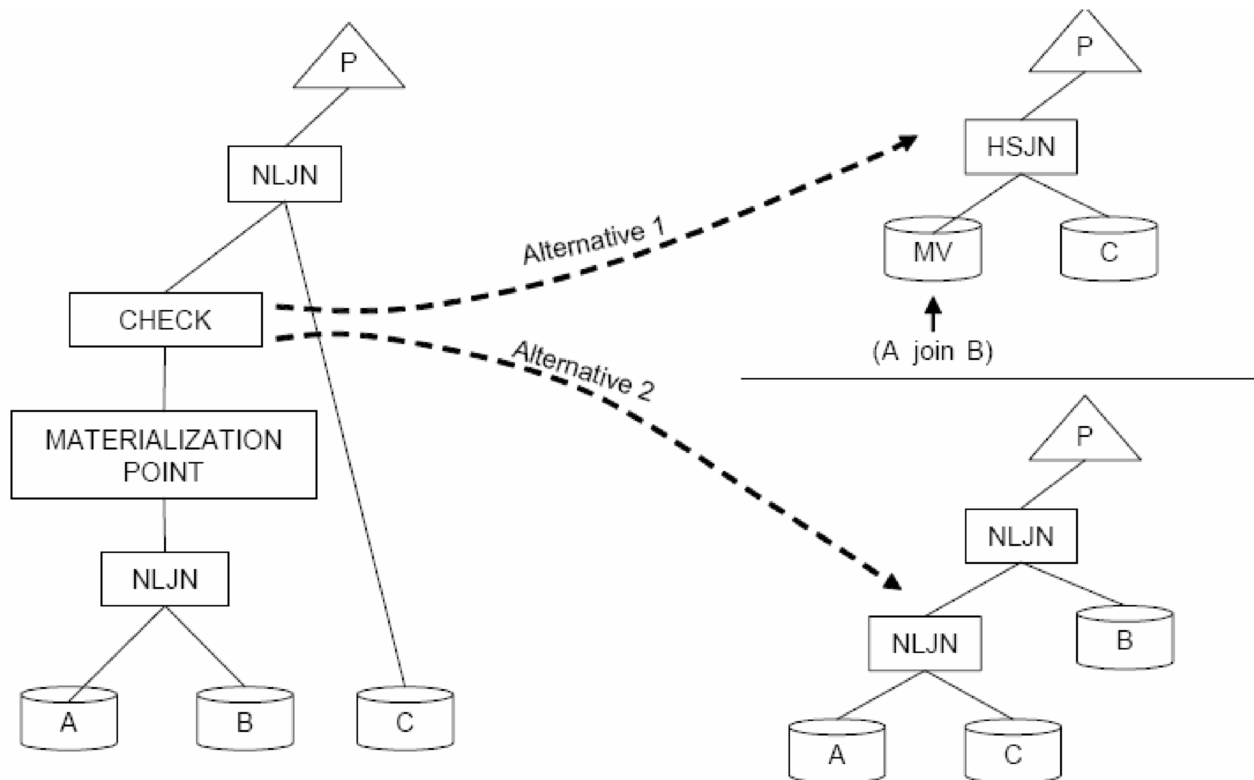
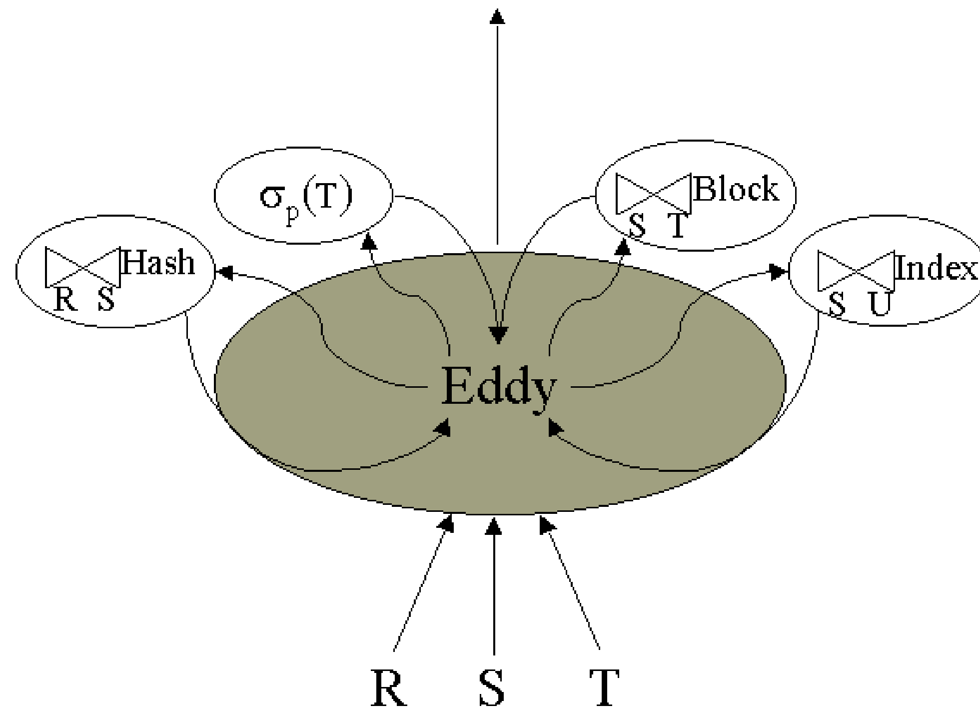


Figure 6: Two alternatives considered in re-optimization

Opt. robusta/adaptable (plan de tuplos)

- Eddy (o remolino, despachador de tuplos a operadores)
- Originalmente, Eddy toma tuplos de relaciones (R,S,T). Produce resultados intermedios.
- Rutea tuplos o resultados intermedios a operadores libres
- Un tuplo solo sale cuando pasa por todos los operadores



Cómo convivir con un optimizador

Si no se comporta como deseado:

- Consulte a un experto, si no:
- Dé "*showplan*" a consulta. Vea qué anda mal. Tras ello, vea si
  - Puede mejorarse con indices, con clustering, con particiones
  - Se puede mejorar la consulta con dando pistas (*hints)*
  - Se puede hacer trampa con estadísticas
  - Se puede reformular la consulta