PROCESAMIENTO DE CONSULTAS

OPTIMIZACIÓN DE CONSULTAS EN SISTEMAS RELACIONALES

consulta = recuperación y actualización de datos

- La **optimización** [automática] es...
  - Un reto: obligatorio si se debe lograr tiempo de ejecución de consultas aceptable
  - Una oportunidad: el alto nivel semántico de una expresión relacional permite su optimización antes de la ejecución

- En un **Sistema NO Relacional**:
  - Solicitudes de usuario expresadas a nivel semántico más bajo
  - El usuario realiza "optimizaciones" manuales:
    - Decide las operaciones necesarias y su orden de ejecución
    - Si se equivoca, el sistema no puede mejorar la situación
    - Debe tener conocimientos de programación (los que no tengan, no se beneficiarán de la posibilidad de consultas más óptimas)

OPTIMIZACIÓN DE CONSULTAS

- **VENTAJAS** DE LA OPTIMIZACIÓN
  1. El usuario no se preocupa de cómo formular la consulta
  2. El MÓDULO OPTIMIZADOR trabaja "mejor" que el programador, ya que:
     - **a.** Dispone de **información estadística** almacenada en el catálogo del sistema, y permite...
       - mayor precisión en la estimación de la eficiencia de c/estrategia, de forma que...
         - (con mayor probabilidad) elegirá el procedimiento más eficiente
     - **b.** Si cambian las estadísticas (es decir, si se hace una reorganización física de la BD)
       ⇒ **RE-OPTIMIZACIÓN** (pues quizás ahora convenga elegir una estrategia diferente)
       - Sist. Relacional: (trivial) El Optimizador re-procesa la consulta original
       - Sist. No Relacional: modificación del programa
     - **c.** Optimizador=programa ⇒ más paciente que un programador (considera más estrategias)
     - **d.** Optimizador = **compendio de aptitudes y servicios** de los mejores programadores
- **OBJETIVO** DEL Módulo OPTIMIZADOR:
  Elegir una estrategia eficiente para evaluar una expresión relacional dada
- Nota:
  - No hay garantía de que la estrategia elegida sea óptima, pero sí es mejor que la original

EL PROCESO DE OPTIMIZACIÓN

• Proceso con 4 etapas:

I. Traducir la expresión de la consulta a una representación interna

II. Convertirla a una forma canónica

III. Elegir procedimientos candidatos de bajo nivel

IV. Generar planes de consulta y Escoger el más económico

I. TRADUCCIÓN

I. TRADUCIR LA EXPRESIÓN A UNA REPRESENTACIÓN INTERNA

– que la máquina manipule mejor,

– eliminando errores y peculiaridades de sintaxis del lenguaje,

– preparando el camino para siguientes etapas

• Así que durante la TRADUCCIÓN, el sistema:
  - **Revisa la sintaxis** de la consulta
  - **Verifica** si existen los **nombres** de relaciones, atributos, índices, etc.
  - Sustituye nombres de cada **vista** por su expresión de cálculo o **definición**

• El FORMALISMO base de la REPRESENTACIÓN INTERNA debe ser:
  - RICO, para representar **toda** consulta posible
  - NEUTRAL, **sin predisponer** a **ciertas opciones** de optimización

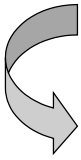
I. TRADUCCIÓN

- La mejor elección: Álgebra Relacional

Nombres de los empleados que trabajan en el proyecto nº 2

```
SELECT nombrep
FROM Empleado E, Trabaja_En T
WHERE E.nss = T.nsse AND T.nump=2 ;
```

$$\pi_{nombrep}\left(\sigma_{nump=2}\left(\text{Empleado} \bowtie_{nss=nsse} \text{Trabaja\_En}\right)\right)$$



resultado
↑
proyectar(E.nombrep)
↑
restringir(T.nump = 2)
↑
reunir(E.nss = T.nsse)
↗ ↖
Empleado(E) Trabaja_En(T)

Árbol Sintáctico Abstracto
(un árbol sintáctico o de consulta será la **representación** de una expresión algebraica)

II. CONVERSIÓN a Forma Canónica

II. CONVERTIRLA A FORMA CANÓNICA

- Optimizador, realiza optimizaciones...
  - de calidad garantizada,
  - sean cuales sean los valores reales de los datos y las rutas de acceso existentes

- Lenguajes de consulta (ej. SQL)
  - permiten **expresar** una **consulta** de muchas **formas diferentes**, pero
  - el **rendimiento no debe depender de cómo sea expresada** la consulta

» hay que pasarla a su **FORMA CANÓNICA EQUIVALENTE** (de idéntico resultado)
  - Elimina distinciones superficiales
  - Consigue representación más eficiente

- Aplicación de REGLAS DE TRANSFORMACIÓN, como esta:

$$\sigma_{Restricc\text{-}A}\left(A \bowtie_c B\right) = \left(\sigma_{Restricc\text{-}A}(A)\right) \bowtie_c B$$

III. ELECCIÓN DE PROCEDIMIENTOS

III. ELEGIR PROCEDIMIENTOS CANDIDATOS DE BAJO NIVEL

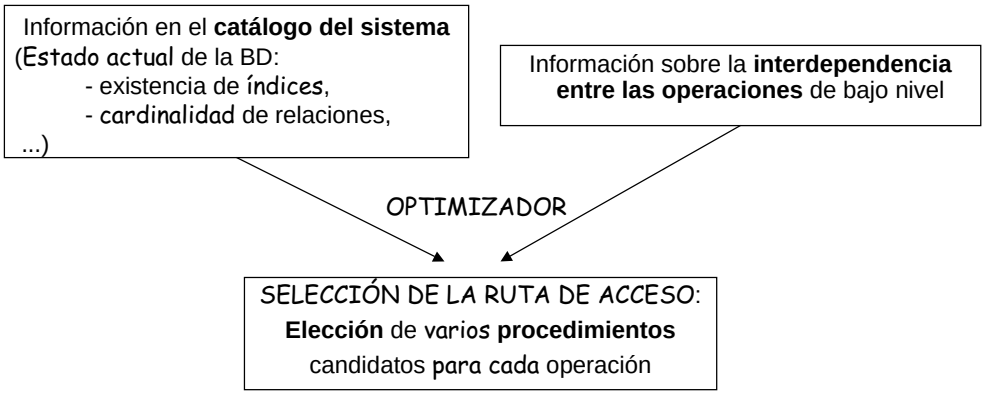
- Obtenida la Forma Canónica de la consulta, el Optimizador decide **cómo evaluarla**
- Debe considerar:
  - Existencia de índices u otros métodos de acceso
  - Distribución de valores de los datos almacenados
  - Agrupamiento físico (o no) de los registros
- Estrategia básica:

  Expresión ≡ serie de **operaciones de bajo nivel** (JOIN, Restricción...) **interdependientes**
- Optimizador tiene un conjunto de procedimientos para realizar cada operación

  Ejemplo: procedimientos para la operación de Restricción, según los casos...
  - restricción es una condición de igualdad según un campo clave candidata
  - campo de restricción indizado
  - campo de restricción no indizado, pero datos ordenados físicamente según ese campo

  etc.
- Cada procedimiento tiene asociada una medida del costo

III. ELECCIÓN DE PROCEDIMIENTOS



IV. PLANES DE CONSULTA

IV. GENERAR PLANES DE CONSULTA Y ELEGIR EL MÁS ECONÓMICO

- Plan de Consulta = **combinación de** varios **procedimientos candidatos** (uno por cada operación de bajo nivel de la consulta)

- En general existen muchos (¡demasiados!)
  - La tarea de ELEGIR EL PLAN MÁS ECONÓMICO, tendrá coste prohibitivo
  - Uso de **técnicas Heurísticas** para mantener el conjunto de planes de consulta generados dentro de unos límites razonables (**Reducción del *espacio de evaluación***)

- Necesario **MÉTODO DE ASIGNACIÓN DE COSTE** a cada plan de Consulta
  - $\text{Coste}(\text{Plan}) \equiv \Sigma_i \text{coste}(\text{procedimiento}_i)$
  - Cada procedimiento calcula su coste en base a la estimación del **nº de operaciones de E/S a disco requeridas,** del **uso de la CPU...**
  - **La estimación precisa de costes es difícil**, pues para estimar el nº E/S, es necesario estimar el tamaño de las tablas generadas como resultados intermedios, que depende de los valores actuales de los datos

TRANSFORMACIÓN DE EXPRESIONES (ETAPA II)

RESTRICCIÓN Y PROYECCIÓN

1. Una secuencia de restricciones sobre una relación A puede transformarse en una sola restricción

$$\sigma_{C1}(\sigma_{C2}(A)) \equiv \sigma_{C1 \text{ AND } C2}(A)$$

2. En una secuencia de proyecciones contra una relación A pueden ignorarse todas, salvo la última (si cada atributo mencionado en la última, también aparece en las demás)

$$\pi_{E1}(\pi_{E2}(A)) \equiv \pi_{E2}(A), \text{ sii } E1 \supseteq E2$$

3. Una restricción de una proyección puede transformarse en una proyección de una restricción

$$\sigma_R(\pi_P(A)) \equiv \pi_P(\sigma_R(A))$$

» Es una buena idea hacer RESTRICCIONES ANTES QUE PROYECCIONES, pues la restricción reduce el tamaño de la entrada para la proyección, e.d. la cantidad de datos que ordenar para eliminar duplicados

TRANSFORMACIÓN DE EXPRESIONES (ETAPA II)

DISTRIBUTIVIDAD

Sea f un operador unario y ⊗ un operador binario, **f es distributivo respecto de ⊗ si**

f(A ⊗ B)= f(A) ⊗ f(B)

1. σ es distributivo respecto de la UNIÓN, INTERSECCIÓN y DIFERENCIA

σ es distributivo respecto de JOIN si la condición de restricción consiste de condiciones separadas por un AND, una para c/relación-operando del JOIN

$$\sigma_{nump=2}(\text{Empleado} \bowtie_{nss=nsse} \text{Trabaja\_En}) \equiv \text{Empleado} \bowtie_{nss=nsse} (\sigma_{nump=2}(\text{Trabaja\_En}))$$

» Efecto: reduce el nº de tuplas examinadas en la siguiente operación en secuencia (por tanto, esa operación también producirá menos tuplas como resultado)

2. π es distributivo respecto de la UNIÓN e INTERSECCIÓN, pero NO la DIFERENCIA

π es distributivo respecto de JOIN si todos los atributos de reunión se incluyen en π

$$\pi_P(A \bowtie_J B) \equiv (\pi_{P\text{-}A}(A)) \bowtie_J (\pi_{P\text{-}B}(B)),$$

sii P=(P-A) UNION (P-B) y P incluye todo atributo de reunión que aparece en J

TRANSFORMACIÓN DE EXPRESIONES (ETAPA II)

CONMUTATIVIDAD

Sea ⊗ un operador binario, **⊗ es conmutativo si A ⊗ B = B ⊗ A , ∀ A,B**

En Álgebra Relacional, son conmutativas: UNIÓN, INTERSECCIÓN y JOIN
y no conmutativas: DIFERENCIA y DIVISIÓN

ASOCIATIVIDAD

Sea ⊗ un operador binario, **⊗ es asociativo si A ⊗ (B⊗C) = (A ⊗ B) ⊗ C , ∀ A,B,C**

En Álgebra Relacional, son asociativas: UNIÓN, INTERSECCIÓN y JOIN
y no asociativas: DIFERENCIA y DIVISIÓN

IDEMPOTENCIA

Sea ⊗ un operador binario, **⊗ es idempotente si A ⊗ A = A,, ∀ A**

En Álgebra relacional, son idempotentes : UNIÓN, INTERSECCIÓN y JOIN
y no idempotentes: DIFERENCIA y DIVISIÓN

TRANSFORMACIÓN DE EXPRESIONES (ETAPA II)

EXPRESIONES DE CÓMPUTO ESCALAR

- El Optimizador Relacional debe conocer reglas de transformación de expresiones aritméticas, pues aparecen en las consultas
- Reglas de transformación basadas en propiedades Conmutativa, Asociativa y Distributiva

EXPRESIONES CONDICIONALES (booleanas)

- El Optimizador Relacional debe saber aplicar reglas generales a operadores de **comparación** (>, <, ...) y **lógicos** (AND, OR, ...)

Ejemplos:

- Sean a y b dos atributos de dos relaciones distintas, entonces **a>b AND b>3** equivale a **b>3 AND a>3 AND a>b** ,, pues > es un operador transitivo
  La condición **a>3**, permite realizar una restricción antes del join necesario para evaluar a>b
- **a>b OR (c=d AND e<f)** equivale a **(a>b OR c=d) AND (a>b OR e<f)**,, OR distrib. resp. de AND

» Toda expresión condicional puede transformarse en su **Forma Normal Conjuntiva (FNC)**

TRANSFORMACIÓN DE EXPRESIONES (ETAPA II)

FORMA NORMAL CONJUNTIVA

Una expresión en FNC tiene la forma

$$\mathbf{C_1\ AND\ C_2\ AND\ \ldots\ C_n}$$

donde cada C_i no incluye ningún AND

- Ventajas de FNC
  - Es TRUE si todo C_i es TRUE, y es FALSE si algún C_i es FALSE
  - Ya que AND es conmutativo, el Optimizador puede **evaluar cada C_i en cualquier orden** (por ejemplo, en orden creciente en dificultad). En cuanto un C_i dé FALSE, el proceso puede acabar
  - En un **entorno de proceso paralelo**, es posible evaluar todos los C_i a la vez. Y en cuanto uno diera FALSE, el proceso acabaría

TRANSFORMACIÓN DE EXPRESIONES (ETAPA II)

TRANSFORMACIONES SEMÁNTICAS

Sea $\pi_{nsse}(\text{Empleado} \bowtie_{nss=nsse} \text{Trabaja\_En})$

- El JOIN hace corresponder una clave externa (en Trabaja_En, que además es NOT NULL, por se parte de la PK) con su correspondiente clave candidata (en Empleado),
- Cada tupla de Trabaja_En siempre tiene como contrapartida alguna tupla de Empleado
- Cada tupla de Trabaja_En contribuye con un valor de nsse al resultado global

» ¡¡ No se necesita el JOIN !!

Equivale a $\pi_{nsse}(\text{Trabaja\_En})$

Ojo: Esta **transformación es válida sólo por la semántica de la situación:**

Cada tupla de Trabaja_En corresponde a una tupla en Empleado, debido a la Restricción de Integridad Referencial y la R.I. de Entidad

TRANSFORMACIÓN DE EXPRESIONES (ETAPA II)

- *En general*, cada operando de un JOIN contendrá tuplas sin contrapartida en el otro operando y, por tanto, que no contribuyen al resultado; en estos casos, transformaciones como la del ejemplo NO son válidas
- **TRANSFORMACIÓN SEMÁNTICA**: la que sólo es válida debido a cierta Restricción de Integridad (de tipo cualquiera, no sólo R.I. Referencial)
- **OPTIMIZACIÓN SEMÁNTICA**: proceso de transformar una consulta en otra cualitativamente diferente pero que garantiza el mismo resultado, gracias a que los datos satisfacen cierta Restricción de Integridad

TRANSFORMACIÓN DE EXPRESIONES (ETAPA II)

- La posibilidad de escribir **EXPRESIONES ANIDADAS**, significa:

  1. Consultas representadas por una única expresión (no procedimiento de varias sentencias), lo cual evita realizar un análisis de flujo

  2. Expresiones anidadas definidas en términos de sub-expresiones, lo que permite al optimizador adoptar una variedad de tácticas de evaluación 'divide y vencerás'

  3. Sin expresiones anidadas no tendrían sentido las leyes generales de transformación

IMPLEMENTACIÓN de OPERADORES RELACIONALES ETAPA III

- Consideramos las operaciones
  PROYECCIÓN, JOIN (equirreunión o reunión natural) y AGREGACIÓN (SUM,COUNT,...)

- Similitudes desde el punto de vista de implementación: el sistema necesita **agrupar tuplas** según el valor de un atributo (o combinación de atributos)
  — PROYECCIÓN: para eliminar duplicados
  — JOIN: para encontrar las tuplas correspondientes
  — AGREGACIÓN: para calcular los valores agregados individuales (por grupo)

- El **agrupamiento** puede realizarse siguiendo varias **técnicas**:
  1. Fuerza Bruta
  2. Búsqueda por Índice
  3. Búsqueda Hash
  4. Mezcla
  5. Hash
  6. Combinaciones de las anteriores

IMPLEMENTACIÓN de OPERADORES RELACIONALES ETAPA III

1. FUERZA BRUTA (de ciclo anidado)

- Examinadas **todas** las **posibles combinaciones de tuplas**
- Para cada tupla t de R, obtener todas las s de S y probar si satisfacen la condición de reunión

```
for (i = 1 ; i ≤ m ; i++)
  for (j = 1 ; j ≤ n ; j++)
        if ( R[ i ].C == S[ j ].C )
              añadir la tupla reunida R[ i ] * S[ j ] al resultado;
```

- Cálculo del COSTE

1. Operaciones de **lectura** de tuplas = m * n
2. Operaciones de **escritura** de tuplas = Cardinalidad del *join* resultado
   - 2.a Caso de ***join uno-a-muchos*** (e.d. clave candidata / clave externa)
     cardinalidad del *join* resultado = cardinalidad de la relación con la clave externa (m ó n)
   - 2.b Caso de ***join muchos-a-muchos***
     Sea dCR = nº valores distintos del atributo de reunión C en la relación R y
     dCS = nº valores distintos del atributo de reunión C en la relación S
     (estimación suponiendo una **distribución uniforme** de los valores del atributo *C*)
     2 puntos de vista:

IMPLEMENTACIÓN de OPERADORES RELACIONALES ETAPA III

a. Para cada tupla de R habrá $\frac{n}{dCS}$ tuplas de S con el mismo valor C,
nº total de tuplas en el *join* = $\frac{n*m}{dCS}$ (a)

b. Para cada tupla de S habrá $\frac{m}{dCR}$ tuplas de R con el mismo valor C
nº total de tuplas en el *join*= $\frac{m*n}{dCR}$ (b)

- Si dCS ≠ dCR, las estimaciones (a) y (b) son diferentes
  - Existe algún valor de C que ocurra en R pero no en S, o viceversa
  - Cardinalidad del join resultado = menor estimador

- En la práctica interesa el **acceso** L/E **a bloques** (no a tuplas)
  - Sea bS (y bR) el nº tuplas de S (o R) en un bloque
  - R ocupa $\frac{m}{bR}$ bloques y S ocupa $\frac{n}{bS}$ bloques de disco
  - Lecturas de bloques: Ejemplo con m=100, n=10.000, bR=1 y bS=10
    - ***R exterior, S interior*** → $\frac{m}{bR} + \frac{m*n}{bS}$
    - ***S exterior, R interior*** → $\frac{n}{bS} + \frac{m*n}{bR}$

  » Conviene que la **relación del bucle exterior sea la menor** (la que ocupa menos bloques)

IMPLEMENTACIÓN de OPERADORES RELACIONALES ETAPA III

2. BÚSQUEDA POR ÍNDICE (*index lookup*)

- Existe un índice X sobre el atributo $S.C$ de la relación interior
- Ventaja sobre *Fuerza Bruta*:

  acceso directo (vía índice) a las **tuplas** de S **relacionadas** con cada tupla de R

  - Nº total de tuplas leídas de R y S = cardinalidad del resultado
  - *peor de los casos*: **cada tupla** leída de S está **en** un **bloque diferente** del disco
  - Nº total de bloques leídos = $\frac{m}{bR} + \frac{m * n}{dCS}$
  - Si m=100, n=10.000, bR=1, bS=10 y dCS=100, el total de bloques leídos es 10.100

- Si las tuplas de S se almacenan en secuencia ordenada según valor del atributo de reunión C, las lecturas de bloque se reducen a $\frac{m}{bR} + \frac{(m * n)/dCS}{bS} = 200$

» ventaja de mantener almacenadas las relaciones en una buena secuencia física

IMPLEMENTACIÓN de OPERADORES RELACIONALES ETAPA III

- **Sobrecarga por** el **acceso a índice** X:
  - Peor caso: cada tupla de R necesita una búsqueda completa en X para encontrar las tuplas correspondientes en S ⇒ lectura de 1 bloque en cada nivel de X
  - Si X tiene L niveles, son m*L lecturas extras de bloques
  - En la práctica L≤3 y el nivel superior de X reside en Memoria Principal (menos lecturas)

```
for (i = 1 ; i ≤ m ; i++)     /* bucle exterior */
        /* existen k entradas de índice X[1] .. X[k]
           con el valor del atributo indexado R[ i ].C */
  for (j = 1 ; j ≤ k ; j++)   /* bucle interior */
        /* sea S[ j ] la tupla de S indexada por X[ j ] */
        añadir la tupla reunida R[ i ] * S[ j ] al resultado;
```

IMPLEMENTACIÓN de OPERADORES RELACIONALES ETAPA III

3. BÚSQUEDA HASHING *(hashing lookup)*

- Similar *búsqueda por índice*, pero **camino rápido de acceso** a S según el atributo de reunión S.C **es una tabla hash** y no un índice

```
   /*supuesta una tabla hash H sobre S.C*/
for (i = 1 ; i ≤ m ; i++) {   /*bucle exterior*/
  k= hash( R[ i ].C );        /* existen h tuplas S[1] .. S[h] almacenadas en H[ k ] */
  for (j = 1 ; j ≤ h ; j++)   /*bucle interior*/
    if ( S[ j ].C == R[ i ].C )
         añadir la tupla reunida R[ i ] * S[ j ] al resultado;
}
```


IMPLEMENTACIÓN de OPERADORES RELACIONALES ETAPA III

4. MEZCLA *(merge ; fusión)*

- R y S almacenadas en secuencia ordenada según valor del atributo de reunión C:
  - Pueden examinarse según el orden físico y ambos procesos pueden ser sincronizados,
  - el join completo puede realizarse en una **única pasada** sobre los datos
- Técnica óptima
  - Cada bloque se 'toca' una sola vez (join uno-a-muchos)
  - Nº bloques leídos $\frac{m}{bR} + \frac{n}{bS}$

Factor crítico del rendimiento:

- ***Clustering* físico** de datos relacionados lógicamente
- En ausencia del *clustering*, **ordenar** una o ambas **relaciones** en tiempo de ejecución y mezclarlas (***clustering* dinámico**: técnica *sort/merge*)

```
/* supuesto un join muchos-a-muchos */
 r = s = 1;
 while (r≤m && s≤n) { /*bucle exterior*/
   v = R[ r ].C );
   for (j = s ; S[ j ].C < v ; j++) ;
   s = j;
   for (j = s ; S[ j ].C == v ; j++) /*bucle interior principal*/
     for (i=r ; R[ i ].C == v )  añadir la tupla reunida R[ i ] * S[ j ] al resultado;
   s = j;
   for (i = r ; R[ i ].C == v ; i++) ;
   r = i;
 }
```

IMPLEMENTACIÓN de OPERADORES RELACIONALES ETAPA III

5. HASH

- Necesita una **única pasada sobre los datos** de cada relación

1er Paso: **Construir tabla hash H** para S **sobre** los valores de **S.C,** c/entrada de H contiene:
- Valor de S.C y (opcional) valores de otros atributos de S
- Puntero a la tupla correspondiente

2º Paso: **Examinar R y aplicar la misma función Hash sobre R.C**
- Si una tupla de R colisiona en H con tuplas de S, entonces si S.C = R.C, se generan las tuplas reunidas adecuadas

- **Ventaja** sobre la técnica *merge*:
  - R y S no tienen por qué estar almacenadas en ningún orden,
  - tampoco es necesario ordenarlas dinámicamente

IMPLEMENTACIÓN de OPERADORES RELACIONALES ETAPA III

```
/* construir una tabla hash H sobre S.C */
for ( j = 1 ; j ≤ n ; j++ ) {
   k = hash ( S[ j ].C ) ;
   añadir S[ j ] a la entrada de tabla hash H[ k ] ;
}
/* búsqueda hash sobre R */
for (i = 1 ; i ≤ m ; i++) {   /* bucle exterior */
   k= hash( R[ i ].C );      /* existen h tuplas S[1] .. S[h] almacenadas en H[ k ] */
   for (j = 1 ; j ≤ h ; j++)  /* bucle interior */
        if ( S[ j ].C == R[ i ].C )
              añadir la tupla reunida R[ i ] * S[ j ] al resultado;
 }
```

• Inhibidores de Optimización

- Características incluidas por la mayoría de sistemas en el mercado
- Impiden al optimizador hacer un trabajo *de mejor forma*
- Estos inhibidores incluyen:
  - Tuplas duplicadas
  - Lógica de tres valores (TRUE, FALSE, NULL)