12. Procesamiento y Optimización de Consultas

Objetivos

- Comprender las tareas de procesamiento y optimización de consultas realizadas por un sistema gestor de bases de datos relacional.
- Conocer reglas heurísticas y de transformación de expresiones del álgebra relacional y cómo aplicarlas para mejorar la eficiencia de una consulta.
- Conocer diferentes estrategias de implementación de operaciones relacionales, en particular la de reunión (*join*), y cómo evaluar el coste estimado de cada estrategia.
- Identificar la información estadística de la base de datos necesaria para estimar el coste de ejecución de las operaciones del álgebra relacional.

12. Procesamiento y Optimización de Consultas

Contenidos

12. Procesamiento y optimización de consultas

Bibliografía

[EN 2002] Elmasri, R.; Navathe, S.B.: **Fundamentos de Sistemas de Bases de Datos**. 3ª Edición. Addison-Wesley. (Cap. 18)

[EN 1997] Elmasri, R.; Navathe, S.B.: **Sistemas de bases de datos. Conceptos fundamentales**. 2ª Ed. Addison-Wesley Iberoamericana. (Cap. 16)

[CBS 1998] Connolly et al.: **Database Systems: A Practical Approach to Design, Implementation and Management**. 2nd Ed. Addison-Wesley (Cap. 18)

12.1 Conceptos generales y objetivos

- **Crítica** a los primeros sistemas basados en el modelo relacional: **bajo rendimiento** de las consultas

⇨ Investigación y desarrollo de **algoritmos eficientes** para procesar consultas

- En un **sistema no relacional**...
  - Consultas expresadas en lenguaje procedural de bajo nivel (embebido, norm.)
  - El usuario selecciona la **estrategia de ejecución**: optimización "manual"
    - Decide las **operaciones** necesarias y su **orden** de ejecución
    - Si se equivoca, **el sistema no puede mejorar la situación**
    - Debe tener **conocimientos de programación** (si no los tienen, no se beneficiarán de la posibilidad de consultas más óptimas)

- En un **sistema relacional**...
  - Consultas expresadas con SQL: QUÉ datos y no CÓMO recuperarlos
  - El **SGBD selecciona la mejor estrategia de ejecución**
  - y tiene mayor **control sobre** el **rendimiento** del sistema

12.1 Conceptos generales y objetivos

Procesamiento de Consultas

Actividades involucradas en la recuperación de datos de la BD

Optimización de Consultas

Elección de una estrategia de ejecución eficaz para procesar cada consulta sobre la base de datos

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

12.1 Conceptos generales y objetivos

Objetivos del procesamiento de consultas

- Transformar una consulta SQL en una **estrategia de ejecución eficaz**, expresada en un lenguaje de bajo nivel
- Ejecutar dicha estrategia para recuperar los datos requeridos

– Existen muchas transformaciones equivalentes para una misma consulta

Objetivo de la optimización de consultas

- Elegir la **estrategia de ejecución** que **minimiza el uso de los recursos**

– En general, **no se garantiza que la estrategia elegida** por el SGBD **sea** la **óptima**, pero seguro que será una **estrategia razonablemente eficiente**

12.1 Conceptos generales y objetivos

- **Ventajas** de la optimización automática
  - El usuario **no** se **preocupa** de **cómo** formular la consulta
  - El Módulo **Optimizador trabaja** "**mejor**" que el programador, pues:
    - Dispone de **información estadística** en el diccionario de datos del SGBD
      - mayor precisión al estimar la eficiencia de cada posible estrategia...
      - y así (con mayor probabilidad) elegirá la más eficiente
    - Si **cambian las estadísticas** (tras reorganización física del esquema de BD,...)
      ⇨**re-optimización** (quizá ahora convenga elegir **otra** estrategia)
      - SGBD Relacional: (trivial) El Optimizador re-procesa la consulta original
      - SGBD No Relacional: modificación del programa!
    - El Optimizador es un programa
      ⇨ tiene más paciencia que un programador: **considera más estrategias**
    - El Optimizador es el **compendio de aptitudes y servicios** de los mejores programadores

12.2 Pasos del procesamiento de una consulta

1. **Análisis léxico, sintáctico y validación**
2. **Optimización**
3. **Generación de código**
4. **Ejecución**

- Estudiaremos con detalle los dos primeros pasos

12.2 Pasos del procesamiento de una consulta

1. Análisis léxico, sintáctico y validación

- **Análisis léxico**
  - **Identificar** los **componentes** (léxicos) en el texto de la consulta (SQL)
- **Análisis sintáctico**
  - **Revisar** la **sintaxis** de la consulta (corrección gramática)
- **Validación semántica**
  - **Verificar** validez de **nombres** de relaciones, atributos, y si tienen sentido
- **Traducción** de la consulta **a** una **representación interna**
  - que la **máquina manipula mejor**,
  - **eliminando peculiaridades** del lenguaje de alto nivel empleado (**SQL**),
  - El formalismo base de la representación interna debe ser…
    - **rico**, para representar toda consulta posible
    - **neutral**, sin predisponer a ciertas opciones de optimización
  - La mejor elección: **Álgebra Relacional**

12.2 Pasos del procesamiento de una consulta

1. Análisis léxico, sintáctico y validación (y 2)

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 ;
```

$$\pi_{nombrep}\left(\sigma_{nump=2}\left(EMPLEADO \bowtie_{nss=nsse} TRABAJA\_EN\right)\right)$$

resultado
–
proyectar(E.nombrep)
–
restringir(T.nump = 2)
–
reunir(E.nss = T.nsse)
↗ ↖
EMPLEADO(E) TRABAJA_EN(T)

Árbol de Consulta
(o árbol sintáctico abstracto)
es la **representación de una expresión algebraica**

12.2 Pasos del procesamiento de una consulta

2. Optimización

- El Optimizador de Consultas combina varias técnicas
- Las técnicas principales son las siguientes:

  – **Optimización heurística**
    - Ordenar las operaciones en una **estrategia de ejecución**

  – **Estimación de costes**
    - Estimar sistemáticamente el **costo de cada estrategia** de ejecución y
    - **Elegir** el plan (estrategia) con el **menor costo** estimado

12.2 Pasos del procesamiento de una consulta

2. Optimización (2)

- **Optimización Heurística**
  Aplicación de reglas heurísticas para modificar la representación interna de una consulta **(Álgebra Relacional o Árbol de consulta)** a fin de **mejorar su rendimiento**

- Varias expresiones del Álgebra Relacional pueden corresponder a la misma consulta
- Lenguajes de consulta, como SQL…
  – permiten expresar una misma consulta de muchas formas diferentes, pero
  – el rendimiento no debe depender de cómo sea expresada la consulta

12.2 Pasos del procesamiento de una consulta

2. Optimización (3)

- El **Analizador Sintáctico** genera **árbol de consulta inicial**
  - sin optimización ⇨ ejecución ineficiente
- El **Optimizador** de Consultas **transforma el árbol** de consulta inicial en **árbol de consulta final** equivalente y eficiente

  ⇦Aplicación de reglas de transformación guiadas por reglas heurísticas
- ➢ Conversión de la consulta en su **forma canónica equivalente**

12.2 Pasos del procesamiento de una consulta

2. Optimización (4)

- Obtenida la forma canónica de la consulta, el Optimizador decide **cómo evaluarla**

- **Estimación** sistemática **de costes**:
  Estimación y comparación de los costes de ejecutar una consulta con diferentes estrategias, y elegir la estrategia con menor coste estimado

- El punto de partida es considerar …

  consulta ● serie de **operaciones interdependientes**

  👁 Operaciones del Álgebra Relacional:

  JOIN, Proyección, Restricción, ∩, ∪…

12.2 Pasos del procesamiento de una consulta

2. Optimización (5)

- El Optimizador tiene un conjunto de **técnicas** para realizar cada operación

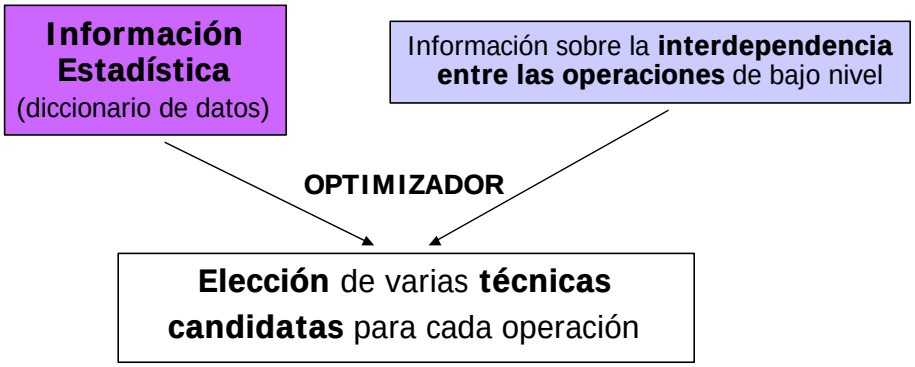
  Ejemplo: técnicas para implementar la operación de Restricción **s**
  - Búsqueda Lineal
  - Búsqueda Binaria
  - Empleo de Índice Primario o Clave de Dispersión
  - Empleo de Índice de Agrupamiento
  - Empleo de Índice Secundario

- Cada procedimiento tendrá asociada una **estimación del coste**

  COSTE = nº accesos a bloque de disco necesarios
  - **La estimación precisa de costes es difícil**, pues para estimar el nº de accesos a bloque, es necesario estimar el tamaño de las tablas (base o generadas como resultados intermedios), lo cual depende de los valores actuales de los datos

12.2 Pasos del procesamiento de una consulta

2. Optimización (6)



12.2 Pasos del procesamiento de una consulta

2. Optimización (7)

ⓘ Información estadística

– El éxito de la estimación del tamaño y coste de las operaciones incluidas en una consulta, depende de la cantidad y actualidad de la información estadística almacenada en el diccionario de datos del SGBD

- Para cada **relación**:
  - –**Cardinalidad** (nº de tuplas),
  - –**Factor de bloques** (nº de tuplas que caben en un bloque),
  - –Nº de **bloques ocupados**,
  - –**Método de acceso primario y otras** estructuras de acceso (índices, hash, etc),
  - –**Atributos indexados**, de **dispersión**, de **ordenamiento** (físico o no), etc.
- Para cada **atributo**:
  - –Nº de **valores distintos** almacenados,
  - –Valores **máximo y mínimo**, etc.
- Nº de niveles de cada **índice de múltiples niveles**

12.2 Pasos del procesamiento de una consulta

2. Optimización (y 8)

- El Optimizador genera **varios planes de ejecución** y **elige el plan más económico**
  - –**Plan de Ejecución** = combinación de técnicas candidatas
    (👁 una por cada operación de bajo nivel de la consulta)
  - –Formulación de una función de coste que minimizar

    Coste (plan) = S_i coste ($técnica_i$)
    - Medida en nº de accesos a bloque (transferencias de bloques memoria ↔ disco)
  - –En general, existen **muchos posibles planes de ejecución** (¡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**

12.3 Reglas de transformación de expresiones

Reglas generales de transformación

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)

$$\pi_{P2}(\pi_{P1}(A)) \equiv \pi_{P2}(A), \text{ sii } P2 \subseteq P1$$

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

$$\sigma_{C}(\pi_{P}(A)) \equiv \pi_{P}(\sigma_{C}(A))$$

Es una buena idea hacer **restricción antes que proyección**, pues la restricción reduce el tamaño de la entrada para la proyección (el número de filas que considerar)

12.3 Reglas de transformación de expresiones

Reglas generales de transformación (2)

ⓘ Distributividad

Sea f un operador unario y ⊗ un operador binario,

f es distributivo respecto de Ă si f(A Ă B) = f(A) Ă f(B)

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

$$\sigma_{C}(R \Theta S) \equiv (\sigma_{C}(R)) \Theta (\sigma_{C}(S))$$

donde $\Theta \in \{ \cup, \cap, - \}$

5. π es distributivo respecto de la **UNIÓN**

$$\pi_{P}(R \cup S) \equiv (\pi_{P}(R)) \cup (\pi_{P}(S))$$

12.3 Reglas de transformación de expresiones

Reglas generales de transformación (3)

6. σ es distributivo respecto de **JOIN** si la condición de selección C...

- contiene atributos que sólo pertenecen a una relación

$$\sigma_{nump=2}(EMPLEADO \bowtie_{nss=nsse} TRABAJA\_EN) \equiv EMPLEADO \bowtie_{nss=nsse} (\sigma_{nump=2}(TRABAJA\_EN))$$

- o puede escribirse como (c1 AND c2), y en c1 sólo intervienen atributos de R1 y en c2 sólo hay atributos de R2

$$\sigma_c(R1 \bowtie_J R2) \equiv (\sigma_{c1}(R1)) \bowtie_J (\sigma_{c2}(R2))$$

se **reduce el número de tuplas examinadas en la siguiente operación** en secuencia
(por tanto, esa operación también producirá menos tuplas)

12.3 Reglas de transformación de expresiones

Reglas generales de transformación (4)

7. π es distributivo respecto de **JOIN** si en la condición de reunión **J** sólo intervienen atributos incluidos en la lista de proyección P

$$\pi_P(R1 \bowtie_J R2) \equiv (\pi_{P\text{-}R1}(R1)) \bowtie_J (\pi_{P\text{-}R2}(R2))$$

sii P = (P1 UNION P2) y P incluye todo atributo de reunión que aparece en J

también se **reduce el número de tuplas examinadas en la siguiente operación** en secuencia
(por tanto, esa operación también producirá menos tuplas)

12.3 Reglas de transformación de expresiones

Reglas generales de transformación (5)

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

8. 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

9. 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

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

12.3 Reglas de transformación de expresiones

Reglas generales de transformación (6)

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

12.3 Reglas de transformación de expresiones

Reglas generales de transformación (7)

- Sean **a y b atributos de dos relaciones distintas**, R(...a...) y S(...b...)

- la condición **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** entre R y S necesario para evaluar a>b

- Sean **los atributos a, b, c, d, e, f**

- La condición **a>b OR (c=d AND e<f)** equivale a

  (a>b OR c=d) AND (a>b OR e<f)

  puesto que OR es distributivo respecto de AND

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

12.3 Reglas de transformación de expresiones

Reglas generales de transformación (y 8)

- **Forma normal conjuntiva**

  Una expresión en FNC tiene la forma

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

  donde cada C_i no incluye ningún AND

  - Es TRUE si todo Ci es TRUE, y es FALSE si algún Ci es FALSE

☺ Ventajas de la FNC

- Ya que AND es conmutativo, el Optimizador puede **evaluar cada** Ci **en cualquier orden** (por ejemplo, en orden creciente en dificultad). **En cuanto un Ci dé FALSE, el proceso puede acabar**
- En un **entorno de procesamiento paralelo**, es posible **evaluar todos los Ci a la vez**. Y en cuanto uno diera FALSE, el proceso acabaría

12.3 Reglas de transformación de expresiones

Reglas heurísticas

- Algunas **buenas heurísticas** que pueden ser aplicadas durante el procesamiento de consultas

1. Ejecutar **operaciones de restricción σ tan pronto como sea posible**
2. Ejecutar **primero las restricciones σ más restrictivas** (las que producen menor nº de tuplas)
3. **Combinar** un **producto cartesiano** × con una **restricción** σ subsiguiente cuya condición represente una condición de reunión, **convirtiéndolas en un join** ⋈
4. **Ejecutar** las operaciones de **proyección π tan pronto como sea posible**

12.4 Implementación de operac. relacionales

Implementación de la reunión (JOIN)

- Las **técnicas** para realizar una reunión pueden ser las siguientes:
  1. **Fuerza Bruta**
  2. **Búsqueda por Índice**
  3. **Búsqueda Hash**
  4. **Mezcla**
  5. **Hash**
  6. Combinaciones de las anteriores

- Daremos alguna indicación del cálculo del coste (función de coste), en términos de nº de accesos a bloque de disco

12.4 Implementación de operac. relacionales

Implementación de JOIN por **fuerza bruta**

- Examinar **todas** las **posibles combinaciones de tuplas de R y S**
- Para cada tupla de R, obtener todas las 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*** (es decir, 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)

Dos puntos de vista: (ver transparencia siguiente)

12.4 Implementación de operac. relacionales

Implementación de JOIN por **fuerza bruta** (y 2)

a. Para cada tupla de R habrá $\frac{n}{dCS}$ tuplas de S con el mismo valor de 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 de 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 q ocupa menos bloques)

12.4 Implementación de operac. relacionales

Implementación de JOIN por **búsqueda por índice**

- Existe un índice X sobre el atributo S.C de la relación interior S

```
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;
```

☺ Ventaja sobre la 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

12.4 Implementación de operac. relacionales

Implementación de JOIN por **búsqueda por índice** (y 2)

☺ Además, 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

☹ **Sobrecarga por** el **acceso al 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)

12.4 Implementación de operac. relacionales

Implementación de JOIN por **búsqueda hashing**

- Similar a la búsqueda por índice, pero el camino 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;
}
```


12.4 Implementación de operac. relacionales

Implementación de JOIN por **mezcla**

- Considera R y S almacenadas en orden según valores del atributo de reunión C
  - Ambas pueden examinarse según el orden físico y de forma sincronizada
  - 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 $\frac{m}{bR} + \frac{n}{bS}$

```
/* 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; }
```

Factor crítico del rendimiento:

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

12.4 Implementación de operac. relacionales

Implementación de JOIN por **hash**

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

1er Paso: **Construir tabla hash H** para S **sobre S.C**

Cada 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 de mezcla:

- Las relaciones R y S no tienen por qué estar almacenadas en ningún orden,
- tampoco es necesario ordenarlas dinámicamente

12.4 Implementación de operac. relacionales

Implementación de JOIN por **hash** (y 2)

```
/* 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;
}
```

Anexo. Otros enfoques de la optimización

Optimización semántica

- Enfoque diferente que puede combinarse con los que hemos visto
- **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 equivalente** (cualitativamente diferente pero que garantiza el mismo resultado) **y más eficiente**, gracias a que los datos satisfacen **restricciones de integridad** especificadas sobre el esquema de base de datos
- Con la aparición de las bases de datos de conocimiento y los sistemas expertos, es posible que esta técnica se incorpore a los SGBD futuros

Anexo. Otros enfoques de la optimización

Optimización semántica (y 2)

Sea $\pi_{nsse}(EMPLEADO \bowtie_{nss=nsse} TRABAJA\_EN)$

- 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 de EMPLEADO
- Cada tupla de TRABAJA_EN contribuye con un valor de nsse al resultado global

⇨ ¡¡ No se necesita el JOIN !!

Equivale a $\pi_{nsse}(TRABAJA\_EN)$

- **Transformación 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 a la **de entidad**
  - En general, cada operando de un JOIN contiene **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**