Pautas de controles pasados

Control 2, 2007-1

Control 2, 2007-2

Control 3, 2007-2

Materia para el examen

Modelado de datos

Modelo E/R, Relacional, E/R a Relacional, Normalización

Consultas

Álgebra relacional, SQL

Implementación

Optimización de consultas, Concurrencia

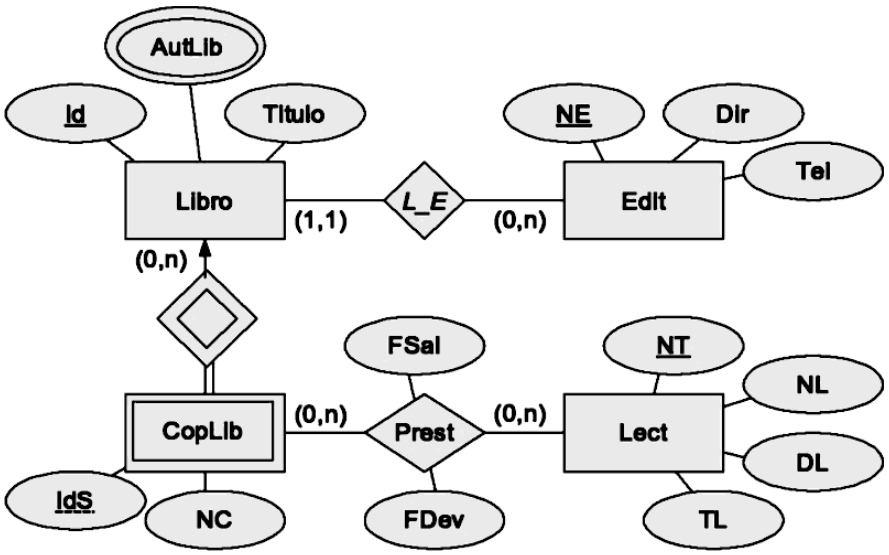
Pauta control 2 (2007-01)
cc42a – cc55a

Problema 1 Considere el siguiente esquema relacional:

```
Libro(Id, Titulo, NE)
 PRIMARY KEY (Id)
 FOREIGN KEY (NE) REF Edit

Edit(NE, Dir, Tel)
 PRIMARY KEY (NE)

AutLib(Id, NA)
 PRIMARY KEY (Id, NA)
 FOREIGN KEY (Id) REF Libro

CopLib(Id, IdS, NC)
 PRIMARY KEY (Id, IdS)
 FOREIGN KEY (Id) REF Libro

Prest(Id, IdS, NT, Fsal, Fdev)
 PRIMARY KEY (Id, IdS, NT)
 FOREIGN KEY (Id, IdS) REF CopLib
 FOREIGN KEY (NT) REF Lect

Lect(NT, NL, DL, TL)
 PRIMARY KEY (NT)
```

Haciendo ingeniería reversa, dibuje el diagrama Entidad-Relación del que proviene.

Solución. Una posible solución a este problema es:



Problema 2 Considere el esquema *R*(*A,B,C,D,E*) con un conjunto de dependencias funcionales $\{AB \rightarrow C,\ E \rightarrow B,\ B \rightarrow D\}$.

1. Indique violaciones a la FNBC (no olvide considerar las dependencias que no están explícitas).
2. Descomponga en FNBC. Indique llaves y dependencias funcionales funcionales heredadas en las tablas obtenidas. Indique si se pierde información sobre dependencias funcionales.

Solución. Para solucionar la pregunta es necesario conocer las llaves candidato. Como podemos ver, los atributos *A* y *E* no son determinados en ninguna dependencia funcional. Luego, toda llave debe contener *AE:*

$$AE \rightarrow ABE(pues\, E \rightarrow B), ABE \rightarrow ABCE(p.\, AB \rightarrow C), ABCE \rightarrow ABCDE(p.\, B \rightarrow D)$$

Luego, *AE* es llave.

Ahora es posible responder las preguntas:

1. Ninguna dependencia funcional de las presentadas respeta FNBC. Luego, toda dependencia funcional derivada de las anteriores, no trivial, y que no contenga *AE* en el lado izquierdo (determinante) viola FNBC.
2. Si la descomposición rompe dependencias funcionales depende del orden en que la descomposición ha sido realizada.
    1. Cortando según el orden en que han sido presentadas las dependencias funcionales:
        1. ABDE, ABC.
        2. ADE, ABC, EB.
        3. B determina D no puede ser aplicada; esta dependencia ha sido rota.
    2. Cortando en el orden inverso:
        1. ABCE, BD.
        2. ACE, BD, BE.
        3. AB determina C no puede ser aplicada; esta dependencia ha sido rota.
    3. Y otro orden más:
        1. ABCE, BD. (B determina D)
        2. ABE, ABC, BD. (AB determina C)
        3. AE, EB, ABC, BD. (E determina B)
        4. Este orden cuida de todo. Todas las dependencias han sido preservadas.

Problema 3 Considere los esquemas

```
Estudio(NombreEst, Direccion)
Pelicula(TituloP, Anho, Largo, NombreEst)
Director(Nombre, TituloP)
```

Escriba en SQL una consulta que responda a:

1. Todos los directores que han hecho películas con Estudios que tienen más de 100 películas.
2. El largo promedio de las películas de cada año.[1]

Solución.

1.
```
SELECT D.*
FROM Director D, Pelicula P
WHERE D.TituloP = P.TituloP
AND P.NombreEst IN (
 SELECT NombreEst
 FROM Pelicula
 HAVING Count(TituloP)>100
 GROUP BY NombreEst );
```

2.
```
SELECT Anho, AVG(Largo)
FROM Pelicula
GROUP BY Anho;
```

1 Ese fue un problema de redacción que persistió en la versión final de la prueba. De otra forma no habría aparecido una pregunta tan fácil. El problema original era mucho más complicado y debió requerir una consulta por lo menos dos veces más grande que la anterior (que también estaba fácil).

Pauta control 2 (2007-2)
cc42a – cc55a

Problema 1 (TranSanFelipe) Se tiene el siguiente esquema:

```
Bus(patente,numlinea,rutempr)
Empresa(rutempr,nombre)
Paradero(idparada,numlinea)
BusVisita(patente,idparada,hora,fecha,suben,bajan)
```

Se deben contestar las siguientes preguntas:

1. (SQL) Generar una lista que, para cada línea (numlinea), indique su número de buses, número de paraderos, promedio de usuarios que abordan por parada y número de empresas involucradas.
2. (Álgebra) ¿Hay líneas involucradas con todas las empresas? Cuáles son.
3. (SQL) Por paradero, indicar la varianza de los tiempos de visita al paradero, sin considerar el tiempo que pasa de un día para otro.

Solución.

1. La pregunta resulta muy sencilla, es sólo una agregación y una agrupación:

```
SELECT    Bus.numlinea, COUNT(DISTINCT Bus.patente),
          COUNT(DISTINCT Paradero.idparada), AVG(BusVisita.suben),
          COUNT(DISTINCT Bus.rutempr)
FROM      Bus, BusVisita, Paradero
WHERE     Bus.numlinea=Paradero.numlinea
          AND Paradero.patente=Bus.patente
GROUP BY  Bus.numlinea
```

2. La respuesta es una división sencilla: $\Pi_{numlinea,rutempr}(Bus) \div \Pi_{rutempr}(Empresa)$

3. La pregunta más difícil del control, probablemente:

```
SELECT    A.idparada, VAR(A.hora-B.hora)
FROM      BusVisita A, BusVisita B
WHERE     A.idparada=B.idparada AND A.fecha=B.fecha
          AND B.hora >= ALL(SELECT C.hora FROM BusVisita C
                            WHERE C.fecha=A.fecha AND
                                  C.idparada=A.idparada AND
                                  C.hora<A.hora);
GROUP BY  A.idparada
```

Problema 2 (Algoritmos) ¿Qué técnicas serían eficientes para la evaluación del *right outer join*?

Solución. El *right outer join* es como el *inner join* (join natural) sólo que permite NULLs en el lado derecho del join en los casos en que no se puede hacer match. Es evidente que:

$$A \text{ inner join } B \subseteq A \text{ right outer join } B$$

Si para una tupla de A existe una tupla de B que hace match, entonces su concatenación estará en el join externo, tal como en el natural. Pero si no, esta tupla de A se concatenará con una tupla nula. Como debemos saber cuándo no hay matchs para elementos de A en B, debemos hacer el recorrido por A. Hacerlo por B implica realizar operaciones adicionales sobre A (registrar qué tuplas no han tenido match), lo que es ineficiente. Luego la estrategia de evaluación debe contener:

```
1  A roj B := ∅
2  ∀ a∈A :
3     buscar b∈B , a∘b aceptable
4     if (∃b)
5           A roj B := {a∘b} ∪ A roj B
6     else
7           A roj B := {a∘NULL} ∪ A roj B
```

En el algoritmo, A roj B es el right outer join.
$a \circ b$ es la concatenación de las tuplas a y b.

Revisar cada elemento de A es una obligación, un costo hundido. No así con B. Discutamos algunas alternativas de evaluación:

1. **Iteración**. No parece buena idea. Véanse los siguientes principios de diseño.
2. **Ordenación** (o estructuras al vuelo). Algunas propuestas:
   1. Ordenar B. Luego ver buscar las tuplas de B que hacen match con la de A se vuelve una labor barata. Costo: $ordenar(B) + N(A) \times buscar(B) = O(b \log b + a \log b)$. *Convención*: a representa el tamaño de A y b el tamaño de B. Así, $N(A) = a$.
   2. Ordenar A y B. Luego de ordenar, se hace una pasada lineal en paralelo por A y B. Luego, el tiempo de búsqueda es constante. En este caso el costo queda en los ordenamientos: $O(a \log a + b \log b + a + b)$.
   3. Crear índice hash sobre B. El costo promedio sería: $O(b + a)$. Sin embargo, el costo de peor caso es: $O(b^2 + a\,b)$ (peor que los casos anteriores). Esto es debido a que buscar en hash es constante en caso promedio pero lineal en el peor caso.
   4. Discusión: las estrategias anteriores dependen de las constantes reales frente a los órdenes asintóticos, pero es mucho más influyente si hay una TDA de búsqueda que apoye los ordenamientos. Por ejemplo, si hay un árbol B+ sobre A (en los atributos convenientes al join), la opción 2 se vuelve significativamente barata pues el

costo de ordenar A se reduce. La opción 3, por otro lado, funciona muy mal si la tabla B estuviera ordenada o parcialmente ordenada. El orden cuadrático de peor caso se vuelve muy caro con tablas grandes. Si no es así, rinde mucho mejor que las alternativas 1 y 2 (que tienen costos estables).

3. **Índices**. A debe revisarse por completo, luego es beneficioso que los índices existan sobre B. Como hay búsqueda por igualdad, es particularmente conveniente un índice de hashing. Llevaría a un costo: $O(a)$. Si B sólo estuviera ordenado o tuviera índice B+, el orden sería: $O(a \log b)$. Notemos que en la práctica, B+ es mucho más rápido en búsquedas y más fácil de mantener que la tabla ordenada.

Problema 3 (Optimización de Consultas) Sea la siguiente consulta SQL:

```
SELECT A.A1, B.B1 FROM A, B, C
WHERE A.A2=C.C1 AND B.B1<=C.C2 AND C.C3 LIKE 'COCODRILO%';
```

Esta consulta se realiza sobre las tablas:

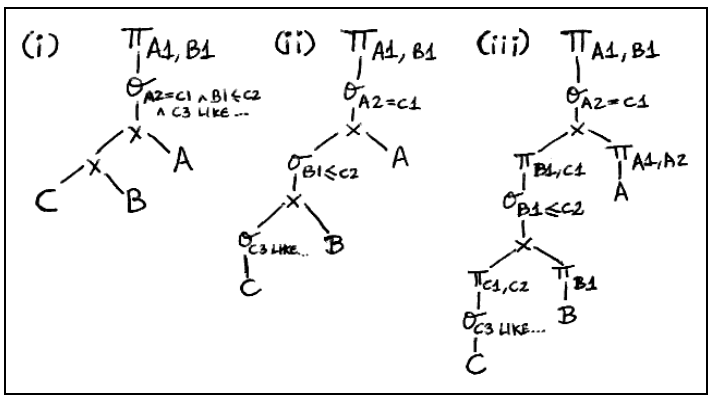
```
A(A1,A2,A3,A4) B(B1,B2) C(C1,C2,C3)
```

Conteste:

1. Dibuje el árbol de evaluación y optimícelo.
2. ¿Sobre qué atributos aplicaría índices?

Solución. La pregunta regalo del control. Cualquier orden de evaluación de $A \times B \times C$ se considera válido ya que no se considera que alumnos sean optimizadores de programas dinámicos, pero era fácil ver que $(A \times C) \times B$ o $(B \times C) \times A$ eran más convenientes que, por ejemplo, el orden original $(A \times B) \times C$ (por el asunto del *left-deep join* o cruza inclinada hacia la izquierda).

Sin mayor fundamento, se presenta la solución de (1) para las cruzas $C \times B \times A$:



En (i) se presenta el árbol original. En (ii) se presenta las selecciones bajadas. En (iii) se presentan las proyecciones bajadas.

Sobre (2), colocar dos índices bastaría: uno de hashing sobre A2 o C1, uno B+ sobre B1 o C2. Hay que notar que escoger dos veces la tabla C no es para nada bueno pues no es bueno sobrecargar una tabla de índices. C3 podría ser usado, dependiendo del optimizador, pero no se puede esperar que sea usado a diferencia de los casos anteriores.

Problema 4 (SQL o Álgebra) Sea el siguiente esquema:

`Alumno(RUT,curso,nota)`

Entregue el RUT de cada par de alumnos que haya cursado los mismos cursos y que hayan sacado la misma nota en cada uno.

Responda en álgebra relacional o en SQL si es que no es posible en álgebra.

Solución. Debido a problemas "logísticos", este problema no fue considerado en la nota. De todas formas, tiene solución en álgebra relacional:

Dos copias de Alumno:

$$A \leftarrow Alumno\,, \quad B \leftarrow Alumno$$

El número de cursos que tomó un alumno:

$$C \leftarrow {}_{RUT}\aleph_{(RUT\,,\,COUNT(curso)\,as\,N)}(Alumno)\,, \quad D \leftarrow C$$

Número de cursos comunes coincidentes entre alumnos:

$$E \leftarrow \rho_{(A.RUT\,as\,R1\,,\,B.RUT\,as\,R2)}\sigma_{(A.RUT \neq B.RUT\,,\,A.curso = B.curso\,,\,A.nota = B.nota)}(A \times B)$$

$$F \leftarrow {}_{R1\,,\,R2}\aleph_{(R1\,,\,R2\,,\,COUNT(*))}E$$

La solución:

$$G \leftarrow \sigma_{(C.RUT > D.RUT \wedge C.RUT = E.R1 \wedge D.RUT = E.R2 \wedge C.N = D.N \wedge D.N = F.N)}(C \times D \times F)$$

$$Resp \leftarrow \Pi_{(R1\,,\,R2)}G$$

Cualquier tupla de salida son dos alumnos que hicieron el mismo número de cursos y que los cursos que hicieron (que cumplen con el enunciado) cumplan la misma cantidad. Adicionalmente exigí Alumno1.RUT>Alumno2.RUT para evitar que $(s\,,t) \in Resp \Rightarrow (t\,,s) \in Resp$, que es una redundancia. Naturalmente, $(s\,,s) \notin Resp$.

Pauta control 3 (2007-2)
cc42a – cc55a

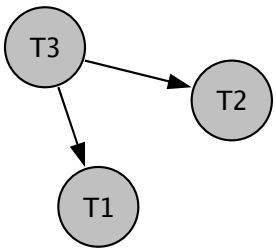
Problema 1 Este problema consistía en la evaluación de seriabilidad por resultado, vista y conflicto, del plan:

$$P: r_1(x), r_2(z), r_1(x), r_3(x), r_3(y), w_1(x), w_3(y), r_2(y), w_2(z), w_2(y)$$

Solución. Como sabemos, se cumple la siguientes relación:

$$P \text{ es seriable por conflicto} \Rightarrow P \text{ es seriable por resultado y por vista}$$

Luego, todo el problema se reduce a verificar seriabilidad por conflicto. Viendo el orden de las operaciones, tenemos que:



Como el grafo es acíclico, podemos concluir seriabilidad por conflictos, por vista y por resultado.

Problema 2 Este problema consiste en tomar el plan *P* anterior y contestar:

1. ¿Es posible intercalar los commit c_1, c_2, c_3 de forma de lograr un plan completo y estricto?
2. Generar un plan completo y no estricto intercalando $a_1, a_2, a_3, c_1, c_2, c_3$ en *P*.

Solución. Para (1) el siguiente plan *P'* cumple con ser estricto y completo:

$$P': r_1(x), r_2(z), r_1(x), r_3(x), r_3(y), w_1(x), c_1, w_3(y), c_3, r_2(y), w_2(z), w_2(y), c_2$$

Luego, la respuesta para (1) es *sí, es posible.*

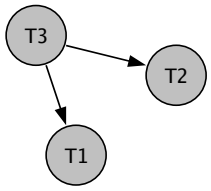
Para (2) basta dejar todos los *commit* o *abort* para el final.

$$P'': r_1(x), r_2(z), r_1(x), r_3(x), r_3(y), w_1(x), w_3(y), r_2(y), w_2(z), w_2(y), c_1, c_2, c_3$$

Problema 3 Generar un plan equivalente por conflictos a *P* y que cumpla con bloqueo de dos fases.

Solución. Una manera rápida de hacerlo es tomando un plan serial equivalente por conflictos a *P* y basarse en éste para generar un plan que cumpla con 2PL.

Retomando el grafo:



Vemos que sólo los planes seriales T3-T2-T1 y T3-T1-T2 pueden ser equivalentes por conflicto a *P*. Nótese que las flechas indican precedencia en los conflictos. Así, por ejemplo, es imposible que T1 ocurra antes que T3 en un plan equivalente por conflictos a *P*, pues $T3 \rightarrow T1$.

Tomemos T3-T1-T2 como *inspiración* para un plan equivalente por conflictos a *P*. Entonces sea el plan, con $l_n(\alpha)$ un bloqueo y $u_n(\alpha)$ un desbloqueo:

T1	*T2*	*T3*
		$l_3(x)$
		$l_3(y)$
		$r_3(x)$
		$u_3(x)$
$l_1(x)$		$r_3(y)$
$r_1(x)$	$l_2(z)$	$w_3(y)$
$r_1(x)$	$r_2(z)$	$u_3(y)$
$w_1(x)$	$l_2(y)$	
$u_1(x)$	$r_2(y)$	
	$w_2(z)$	
	$u_2(z)$	
	$w_2(y)$	
	$u_2(z)$	

Como vemos, el plan anterior cumple 2PL y es equivalente a *P*.