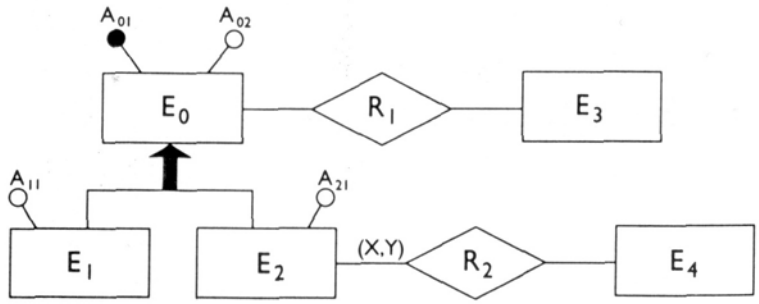
Examen de
Bases de datos y sistemas de información

I PARCIAL

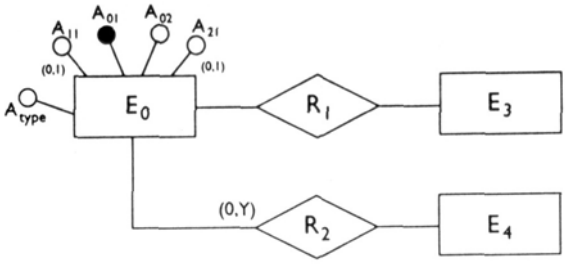
1) (1,25 puntos) ¿Qué efectos se pueden observar cuando se pliegan las entidades hijo sobre la entidad padre en una generalización?

R:

Este plegado es adecuado cuando las operaciones implican a los atributos de E0, E1 y E2 más o menos de la misma forma. Aunque mayor espacio de almacenamiento, requiere menos accesos.
Dado:



Se transforma en:



Efectos:

- Inconvenientes:
  - Aparecen valores nulos en las entidades E1 y E2 en los campos que sólo corresponden a la entidad E2 y E1, respectivamente.
  - Hay que añadir un atributo que especifique el tipo de entidad.
  - Por el mismo motivo: pérdida de espacio
  - Es posible que aparezcan inconsistencias si al añadir una relación a R2 se hace sobre una entidad E1
- Ventajas:
  - Sólo hay un esquema de relación y determinadas consultas que se refieran a entidades E1 y E2 se realizarán más rápidamente.

2) (1,875 puntos) Dados los esquemas de relación R(A,B,C) y S(D,E,F), se pide expresar las siguientes consultas en cálculo relacional de tuplas y cálculo relacional de dominios:

a) $\Pi_A(R)$

b) $\sigma_{B=5}(R)$

c) R×S

R:

a) $\Pi_A(R)$

CRT: {t |∃ q ∈ R (q[A] = t[A])}
CRD: {< t > | ∃ p, q (< t, p, q > ∈ R)}

b) $\sigma_{B=5}(R)$
CRT: {t | t ∈ R ∧ t[B] = 5}
CRD: {< a, b, c > | < a, b, c > ∈ R ∧ b = 5}

c) R×S
CRT: {t | ∃ p ∈ R ∃ q ∈ S (t[A] = p[A] ∧ t[B] = p[B] ∧ t[C] = p[C] ∧ t[D] = q[D]
∧ t[E] = q[E] ∧ t[F] = q[F])}
CRD: {<a,b,c,d,e,f> | (<a,b,c> ∈ R ∧ (<d,e,f> ∈ S))}

3) (1,875 puntos) Descomponer la relación r(A,B,C,D) en relaciones que cumplan 3FN dado el conjunto de dependencias funcionales {AB→C,C→D}.

R:
En primer lugar se comprueba si se encuentra ya en 3FN:
Para toda X → Y, o bien X es superclave o Y contiene algún atributo que pertenece a una clave candidata.

No hay claves candidatas de un atributo.
{AB}+ = {A,B,C,D} Clave candidata
{AC}+ = {A,C,D}
{AD}+ = {A,D}
{BC}+ = {B,C,D}
{BD}+ = {B,D}
{CD}+ = {C,D}

Para AB → C, AB es superclave
Para C → D, C no es superclave y D no pertenece a ninguna clave candidata. Por lo tanto, R NO ESTÁ EN 3FN.

Se aplica el algoritmo de descomposición en 3FN:

D= {R}
1. Encontrar un recubrimiento mínimo T de F
2. Para cada X → Y∈T, crear en D un esquema Ri con {X ∪{A1}∪...∪{Ak}} si Ri ⊄Rj ∈ D, dadas las D.F. X →{A1}, ..., X → {Ak}
3. Si ninguno de los esquemas contiene una clave de R, se crea uno nuevo.

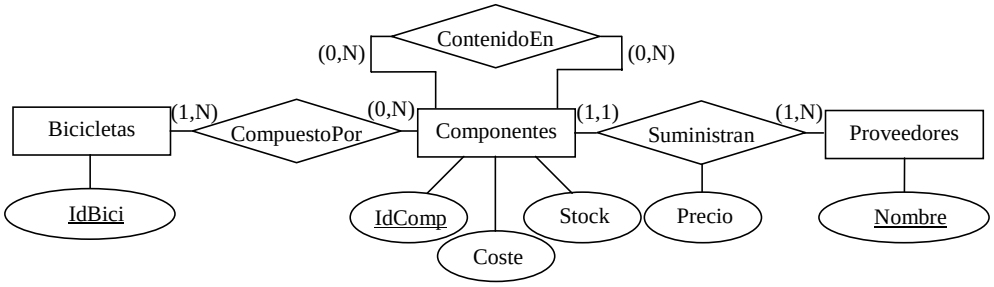
1. F es minimal porque C → D sólo tiene un atributo en el antecedente y consecuente, y porque de AB → C no se puede eliminar A o B sin eliminar su cierre, dado que ni A ni B son superclaves y AB sí.
2.AB → C: {A,B,C}
2.C → D: {C,D}
Dado que el primer conjunto contiene una clave candidata del esquema de R (la única que existe), no es necesario agregar nada más. Por lo tanto, la descomposición resulta en:
R1(A,B,C)
R2(C,D)
Estas dos relaciones se encuentran en 3FN porque el algoritmo anterior lo asegura.

4) (5 puntos) Un fabricante de bicicletas desea contar con una base de datos que describa los diferentes modelos que fabrica en términos de sus componentes. Desea determinar que, por ejemplo, el modelo B de bicicleta precisa del componente rueda, del componente cuadro y del componente manillar. Asimismo, estos componentes están descritos por otros subcomponentes. Por ejemplo, el componente rueda está compuesto por el componente llanta, radios, cámara, frenos y válvula. También desea contar con control de stock (existencias) para determinar el número de modelos que puede fabricar en un momento determinado, así como poder determinar cuándo y con qué método de contacto debe avisar al proveedor del componente correspondiente. Además, algunos proveedores pueden servir componentes como los que fabrica el fabricante de bicicletas, y desea tener la información para saber cuál es la opción más económica: fabricarlo él mismo o comprarlo al proveedor. Se pide:
   a) Construir el modelo entidad-relación imponiendo y explicando las restricciones de clave, de dominio y de cardinalidad mínimo-máximo que se estimen oportunas.

b) Traducir el modelo conceptual obtenido al modelo lógico sin considerar las operaciones y expresando las restricciones de integridad referencial y de participación total en notación algebraica.
c) Plantear una consulta en SQL para determinar los componentes de un modelo de bicicleta (sólo en un primer nivel, sin buscar componentes de componentes).
d) Plantear un procedimiento PL/SQL para determinar el coste de un componente en términos de todos sus subcomponentes (en todos los niveles).
e) Plantear un disparador que avise de que es necesario la compra de más unidades de un componente dado cuando se alcanza un nivel mínimo.

R:

a) Construir el modelo entidad-relación imponiendo y explicando las restricciones de clave, de dominio y de cardinalidad mínimo-máximo que se estimen oportunas.



a1) Significado de las entidades y relaciones

- Entidades:
  - Bicicletas: Modelos de bicicletas, identificados por un código IdBici
  - Componentes: Componentes con los que se fabrican las bicicletas, identificados por un código IdComp, con un coste de fabricación Coste y en cantidad indicada por Stock.
  - Proveedores: Proveedores que suministran determinados componentes, identificados por su nombre.
- Relaciones:
  - CompuestoPor: Bicicletas × Componentes
    Identifica los componentes que componen un determinado modelo de bicicleta.
  - ContenidoEn: Componentes × Componentes
    Identifica que algunos componentes están compuestos por otros.
    Los componentes más elaborados no son componentes de otros (por ejemplo, la rueda). Un componente puede ser subcomponente de varios.
  - Suministran: Componentes × Proveedores
    Indica los componentes que son suministrados por ciertos proveedores. Se asume que todos los proveedores suministran algún componente y que cada componente sólo tiene un proveedor.

a2) Restricciones de clave y de dominio

- Entidades:
  - Bicicletas(IdBici: String)
  - Componentes(IdComp: String, Coste: Number(7,2), Stock: Int)
  - Proveedores(Nombre: String)
- Relaciones:
  - Suministran(Precio: Int)

a2) Restricciones de cardinalidad mínimo-máximo

CompuestoPor:

- Bicicletas - (1,N) - CompuestoPor - Componentes: Cada bicicleta está formada por al menos un componente (mínimo 1). Una bicicleta puede estar compuesta por varios componentes (máximo N).
- Bicicletas - CompuestoPor - (0,N) - Componentes: No todos los componentes son componentes directos de una bicicleta (pueden ser componentes de otros componentes: el significado de CompuestoPor se entiende incluyendo el cierre transitivo de la relación ContenidoEn) (mínimo 0). El mismo componente puede formar parte de varias bicicletas (máximo N).

ContenidoEn:

- Subcomponente - (0,N) - ContenidoEn - Componente: Un componente puede no ser subcomponente de otro (por ejemplo, los más elaborados como la rueda) (mínimo 0). El mismo componente puede ser subcomponente de otros (por ejemplo, las zapatas son subcomponentes de los frenos delantero y trasero) (máximo N).

- Subcomponente - ContenidoEn - (0,N) - Componente: Un componente puede no tener subcomponentes (por ejemplo, los no elaborados, como una zapata) (mínimo 0). Un componente puede tener varios subcomponentes (máximo N).

Suministran:

- Proveedores - (1,N) - Suministran - Componentes: Cada proveedor suministra al menos un componente (mínimo 1). Un proveedor puede suministrar varios componentes (máximo N).
- Proveedores - Suministran - (1,1) - Componentes: Cada componente es suministrado por un proveedor en concreto (mínimo 1). El mismo componente no puede ser suministrado por más de un proveedor (máximo 1).

Nota: el modelo conceptual se podría simplificar aún más si se considera que una bicicleta puede ser un componente. Esto significaría la posibilidad de que los proveedores también pudiesen suministrar bicicletas.

b) Traducir el modelo conceptual obtenido al modelo lógico sin considerar las operaciones y expresando las restricciones de integridad referencial y de participación total en notación algebraica.
No es necesaria una fase de reestructuración (generalizaciones, mezclas o divisiones).

Modelo relacional:

- Entidades:
  - Bicicletas(<u>IdBici</u>)
  - Componentes(<u>IdComp</u>, Coste, Stock)
  - Proveedores(<u>Nombre</u>)
- Relaciones:
  - CompuestoPor(<u>IdBici</u>, <u>IdComp</u>)
  - ContenidoEn(<u>IdSubcomp</u>, <u>IdComp</u>)
  - Suministran(<u>IdComp</u>, NombreProv, Precio)

Restricciones de integridad referencial:

Con respecto a CompuestoPor:

- $\Pi_{IdBici}(CompuestoPor) \subseteq \Pi_{IdBici}(Bicicletas)$
- $\Pi_{IdComp}(CompuestoPor) \subseteq \Pi_{IdComp}(Componentes)$

Con respecto a ContenidoEn:

- $\Pi_{IdComp}(ContenidoEn) \subseteq \Pi_{IdComp}(Componentes)$
- $\Pi_{IdSubcomp}(ContenidoEn) \subseteq \Pi_{IdComp}(Componentes)$

Con respecto a Suministran:

- $\Pi_{IdComp}(Suministran) \subseteq \Pi_{IdComp}(Componentes)$
- $\Pi_{Nombre\,\mathrm{Pr}\,ov}(Suministran) \subseteq \Pi_{Nombre}(\mathrm{Pr}\,oveedores)$

Restricciones de participación total:

Con respecto a CompuestoPor:

- $\Pi_{IdBici}(CompuestoPor) = \Pi_{IdBici}(Bicicletas)$

Con respecto a Suministran:

- $\Pi_{IdComp}(Suministran) = \Pi_{IdComp}(Componentes)$
- $\Pi_{Nombre\,\mathrm{Pr}\,ov}(Suministran) = \Pi_{Nombre}(\mathrm{Pr}\,oveedores)$

c) Plantear una consulta en SQL para determinar los componentes de un modelo de bicicleta (sólo en un primer nivel, sin buscar componentes de componentes).

```
SELECT IdComp
FROM CompuestoPor
WHERE IdBici = 'Modelo'
```

d) Plantear un procedimiento PL/SQL para determinar el coste de un componente en términos de todos sus subcomponentes (en todos los niveles).

```
CREATE TABLE Componentes(IdComp VARCHAR(15) PRIMARY KEY, Coste
NUMBER(7,2), Stock INTEGER);
```

```
INSERT INTO Componentes VALUES ('rueda',50,200);
INSERT INTO Componentes VALUES ('llanta',50,200);
INSERT INTO Componentes VALUES ('radio',1,200);
INSERT INTO Componentes VALUES ('cámara',10,200);
INSERT INTO Componentes VALUES ('cubierta',20,200);
INSERT INTO Componentes VALUES ('válvula',6,200);
INSERT INTO Componentes VALUES ('tapón',1,200);
INSERT INTO Componentes VALUES ('manillar',30,200);
INSERT INTO Componentes VALUES ('cuadro',25,200);

CREATE TABLE ContenidoEn(IdSubcomp VARCHAR(15) REFERENCES
Componentes(IdComp), IdComp VARCHAR(15) REFERENCES
Componentes(IdComp), PRIMARY KEY(IdSubcomp,IdComp));
INSERT INTO ContenidoEn VALUES ('llanta','rueda');
INSERT INTO ContenidoEn VALUES ('radio','rueda');
INSERT INTO ContenidoEn VALUES ('cámara','rueda');
INSERT INTO ContenidoEn VALUES ('cubierta','rueda');
INSERT INTO ContenidoEn VALUES ('válvula','rueda');
INSERT INTO ContenidoEn VALUES ('tapón','válvula');

CREATE OR REPLACE FUNCTION Coste(p_Componente
Componentes.IdComp%TYPE) RETURN Integer IS
  CURSOR c_Componentes IS
    SELECT IdSubcomp
    FROM ContenidoEn
    WHERE IdComp=p_Componente;
  v_Coste NUMBER(8,2);
  v_CosteComponente NUMBER(7,2);
  v_IdSubcomp VARCHAR(15);
BEGIN
  SELECT Coste INTO v_Coste FROM Componentes WHERE
Componentes.IdComp=p_Componente;
  OPEN c_Componentes;
  LOOP
    FETCH c_Componentes INTO v_IdSubcomp;
    EXIT WHEN c_Componentes %NOTFOUND;
    -- El siguiente es un cursor implícito:
    SELECT Coste INTO v_CosteComponente FROM Componentes WHERE
Componentes.IdComp=v_IdSubcomp;
    v_Coste := v_Coste +
               v_CosteComponente +
               Coste(v_IdSubcomp);
  END LOOP;
  CLOSE c_Componentes;
  RETURN v_Coste;
END Coste;

SQL> begin
  2  dbms_output.put_line(coste('válvula'));
  3  end;
  4  /
7

Procedimiento PL/SQL terminado correctamente.

SQL> begin
```

```
  2  dbms_output.put_line(coste('rueda'));
  3  end;
  4  /
138
```

e) Plantear un disparador que avise de que es necesario la compra de más unidades de un componente dado cuando se alcanza un nivel mínimo.

El siguiente disparador se activa cada vez que se modifica o se inserta una nueva fila en la tabla Componentes, mostrando un mensaje relativo a todos los componentes que tienen un stock menor de 10 unidades.

```
CREATE OR REPLACE TRIGGER disparadorMinimos
AFTER UPDATE OR INSERT ON Componentes
DECLARE
  CURSOR c_Minimos IS
    SELECT IdComp, Stock FROM Componentes WHERE Stock < 10;
  v_IdComp Componentes.IdComp%TYPE;
  v_Stock  Componentes.Stock%TYPE;
BEGIN
  OPEN c_Minimos;
  LOOP
    FETCH c_Minimos INTO v_IdComp, v_Stock;
    EXIT WHEN c_Minimos%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_IdComp||' por debajo del stock
mínimo.');
  END LOOP;
  CLOSE c_Minimos;
END;
```

II PARCIAL

Nota: (X; Y puntos) significa que X es la puntuación del problema sobre 10 para los alumnos que no deban realizar el primer parcial, mientras que Y es la puntuación para quienes deban examinarse de ambos. Si Y no aparece, los alumnos con toda la asignatura no deben resolver el problema.

1) (1,3; 2,6 puntos) Considérese un fichero de registros de longitud variable con campos A:string(10), B:string(100), C:string(15), donde el campo C sólo aparece para ciertos valores de A. Se mantienen dos índices secuenciales para el fichero, respectivamente para los campos A y C. El índice sobre A es primario y escaso (sólo aparece la menor entrada en orden alfabético) y el índice sobre C denso. Los bloques tienen un tamaño de 256 bytes y se desea que contengan bits de existencia. Se asume una agrupación de bloques de longitud variable sin tramos. Se mantiene una cabecera que indica el inicio de la cadena de bloques libres del fichero de datos. Las direcciones de los bloques ocupan 4 bytes. Se pide:
   a) ¿Cuánto espacio se desperdicia en cada estructura?
   b) Hacer un esquema del fichero de datos y de los dos índices suponiendo los siguientes registros: r1(a,l,i), r2(a,m,j), r3(b,n), r4(bb,o,k), r5(cc,p).
   c) Describir las operaciones búsqueda, inserción, borrado y modificación.
   d) Describir la nueva situación al insertar los registros: r6(c,q,l), r7(b,r,i), r8(a,r,m).
   e) Crear mapas de bits sobre el campo A.

R:

   a) ¿Cuánto espacio se desperdicia en cada estructura?

Fichero de datos:

Tamaño del registro: 10 (campo A) + 100 (campo B) + 15 (campo C) = 125 bytes
Tamaño del bloque: 256 bytes
En el bloque es necesario almacenar la dirección del siguiente bloque de datos (4 bytes) y la dirección del siguiente bloque libre (4 bytes) y bits de existencia (1 bit * número de registros).
También se podría pensar en un indicador del tipo de registro almacenado, pero no es necesario dado que el valor de A define implícitamente el tipo de registro.

Primer caso: registros con campos A, B y C
Sólo se puede almacenar un registro con los tres campos (A+B+C): 125 bytes + 4 bytes + 4 bytes + 1 bit = 133 bytes + 1 bit
Con dos registros sería necesario: 125*2 bytes + 4 bytes + 4 bytes + 2 bits = 258 bytes + 2 bits
Espacio desperdiciado: 256 bytes-(133 bytes+1bit) = 122 bytes + 7 bits
Segundo caso: registros con campos A, B
Caben 2 registros: 110*2+4+4+2bits=228 bytes + 2 bits, se desperdician 27 bytes y 6 bits
Tercer caso: 1 registro con campos A, B y C, y otro con campos A y B
Caben 2 registros: 125+110+4+4+2bits=243 bytes + 2 bits, se desperdician 12 bytes y 6 bits
Fichero de índice A:
Cada entrada del índice: 10 bytes (clave A) + 4 bytes (dirección del bloque) = 14 bytes
Tamaño del bloque: 256 bytes
En el bloque es necesario almacenar la dirección del siguiente bloque del índice (4 bytes) y bits de existencia (1 bit * número de entradas).
$14*n*8+4*8+n \leq 256*8$, expresado en bits
$n(14*8+1) \leq 256*8-4*8$, $n \leq 17{,}8$, $n=17$
Espacio desperdiciado: 256-17*14 - 4 - 17/8 = 12 bytes - 1 bit
Fichero de índice C:
Cada entrada del índice: 15 bytes (clave C) + 4 bytes (dirección del bloque) = 19 bytes
Tamaño del bloque: 256 bytes
En el bloque es necesario almacenar la dirección del siguiente bloque del índice (4 bytes) y bits de existencia (1 bit * número de entradas).
$19*n*8+4*8+n \leq 256*8$, expresado en bits
$n(19*8+1) \leq 256*8-4*8$, $n \leq 13{,}2$, $n=13$
Espacio desperdiciado: 256-13*19 - 19/8 = 7 bytes - 3 bits

b) Hacer un esquema del fichero de datos y de los dos índices suponiendo los siguientes registros: r1(a,l,i), r2(a,m,j), r3(b,n), r4(bb,o,k), r5(cc,p).

Datos:
Un posible escenario es el siguiente:
b1: 10 | a,l,i | -> b2: 11|a,m,j|b,n| -> b3: 10|bb,o,k|-> b4: 10|cc,p| -> nil
Aunque también es posible:
b1: 10 | a,l,i | -> b2: 11|a,m,j|b,n| -> b3: 11|bb,o,k|cc,p| -> nil
Suponemos el primero en lo que sigue.

Índice A:
1110...0 | a,b1 | b,b2 | cc,b4 |

Índice C:
1110...0 | i,b1 | j,b2 | k,b3 |

c) Describir las operaciones búsqueda, inserción, borrado y modificación.

- Búsqueda:
  - Según el índice A. Se busca en el índice escaso de forma secuencial (desde el primer registro) hasta encontrar la mayor clave menor o igual que el valor de búsqueda. Si es igual se va a la dirección de bloque y se exploran consecutivamente los registros de ese bloque y consecutivos hasta recuperar todas las entradas con el mismo valor de búsqueda. Para determinar el siguiente bloque a buscar se utiliza el puntero al siguiente bloque.
  - Según el índice C. Se busca en el índice denso de forma secuencial (desde el primer registro) hasta encontrar la clave. Si no se encuentra, no hay ningún registro que cumpla la condición de búsqueda. En caso contrario se procede a buscar en el bloque apuntado por el puntero del índice como en el caso anterior.
  - Según el campo A. Es necesario explorar secuencialmente (desde el primer registro) el fichero de datos.

  Las búsquedas que impliquen condiciones compuestas sobre los campos A y C pueden usar los índices sobre estos campos.
- Inserción:
  Se realiza una búsqueda para determinar el lugar en donde se debe insertar el registro. Si hay espacio libre en el bloque se copia el registro en el bloque, se marca el bit de existencia, se actualiza la cadena de

bloques libres (para eliminar el bloque de la cadena, dado que no habrá posibilidad de insertar otro, ya que el bloque ya contiene otro registro) y se actualizan los índices.
Si no hay espacio libre se solicita un nuevo bloque al sistema operativo y se inserta en la cadena del archivo de datos, se copia el registro en el bloque, se marca el bit de existencia, se actualiza la cadena de bloques libres (si se ha insertado un registro con campos A y B) y se actualizan los índices.
Para actualizar un índice en inserción es necesario buscar el lugar en el índice en el que se debe insertar la clave y el puntero al bloque. Es necesario en general desplazar las entradas posteriores. Si el bloque del índice está lleno, se solicita un nuevo bloque al sistema operativo y se concatena al final de la cadena de bloques del índice. Este nuevo bloque contendrá sólo la última entrada del índice. Hay que actualizar también los mapas de bits de existencia de los bloques del índice.

- Borrado:
  Se asume que se desea borrar sólo un registro (se podrían borrar todos los registros con el mismo valor de la clave).
  Se busca el registro a borrar. Si no se encuentra, no se hace nada.
  Si se encuentra, se borra el registro del bloque. Si queda otro registro, se desmarca el bit de existencia correspondiente, se actualiza la cadena de bloques libres y se actualizan los índices. Si era el último registro, se devuelve el bloque al sistema de ficheros, se actualiza la cadena de bloques de datos del fichero de datos, se actualiza la cadena de bloques libres (sólo si el registro sólo tenía campos A y B) y se actualizan los índices
  Para actualizar un índice en borrado es necesario buscar el lugar en el índice en el que se encuentra la clave y el puntero al bloque. En el índice escaso es posible que no se encuentre, por lo que no habría que hacer nada. En el índice denso se encontrará con seguridad y será necesario realizar una búsqueda de otros registros con el mismo valor de la clave para determinar si hay que eliminar esta entrada del archivo de índices. En caso afirmativo hay que desplazar las entradas posteriores un lugar a la izquierda. Es posible que el último bloque de la cadena quede sin uso, por lo que se devuelve al sistema de ficheros y se actualiza la cadena de bloques del archivo de índices.
- Modificación:
  El método más sencillo es realizar un borrado seguido de una inserción. Otra alternativa es combinar las operaciones en una única (más eficiente).

d) Describir la nueva situación al insertar los registros: r6(c,q,l), r7(b,r,i), r8(a,r,m).
r7 no es legítimo: debería tener sólo dos campos, ya que el valor 'b' del campo A aparece previamente en el registro r3, que tiene sólo dos campos.

Datos:
b1: 10 | a,l,i | -> b11: |a,r,m| -> b2: 11|a,m,j|b,n| -> b3: 10|bb,o,k|-> b4: 11|c,q,l|cc,p| -> nil

Índice A:
1110...0 | a,b1 | b,b3 | c,b41 |

Índice C:
111110...0 | i,b1 | j,b2 | k,b4 | l,b41 | m,b21 |

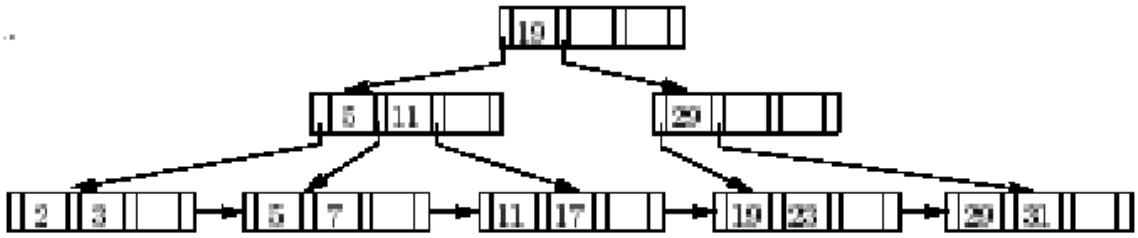
e) Crear mapas de bits sobre el campo A.

Valor a	1110000
Valor b	0001000
Valor bb	0000100
Valor c	0000010
Valor cc	0000001

2) (1,5 puntos)
   a) Constrúyase un árbol B+ con nodos de cuatro punteros con el conjunto de valores de la clave (2, 3, 5, 7, 11, 17, 19, 23, 29, 31). Supóngase que el árbol está inicialmente vacío y que se añaden los valores en orden ascendente.
   b) Muéstrese el árbol después de insertar 9, 10, 8 y de borrar 23 y 19.

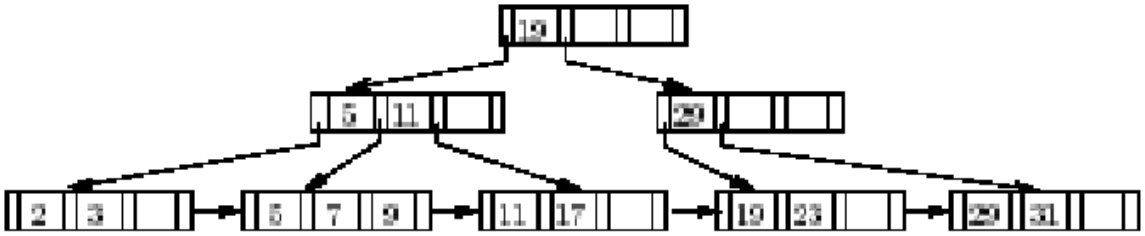
R: Ej.

a)

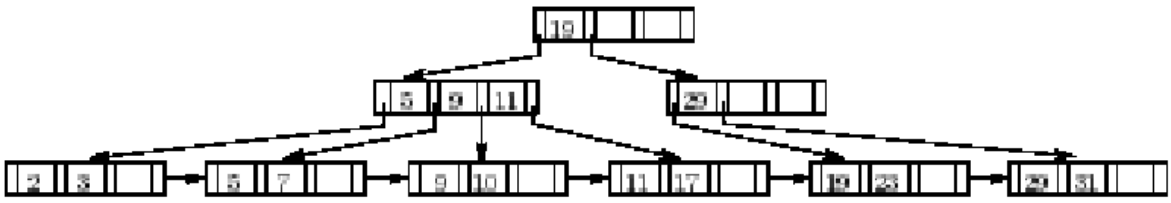


b)

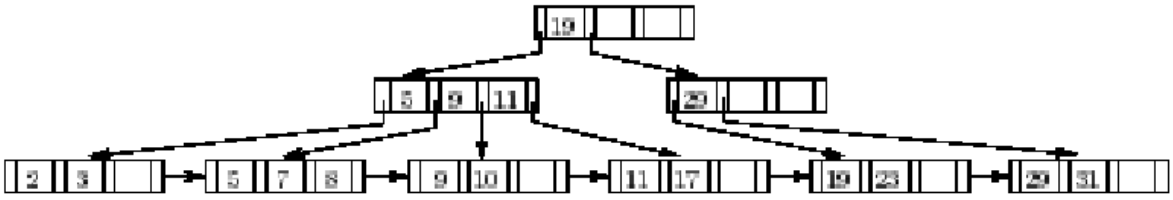
Insertar 9:



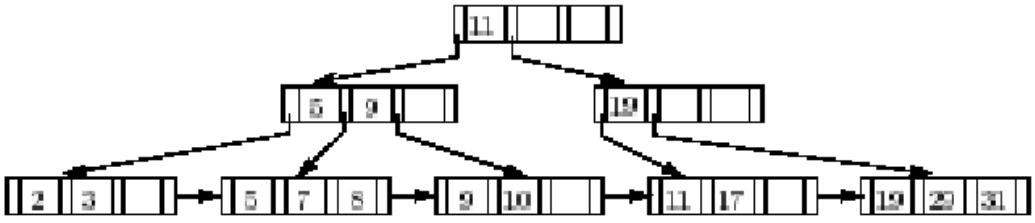
Insertar 10:



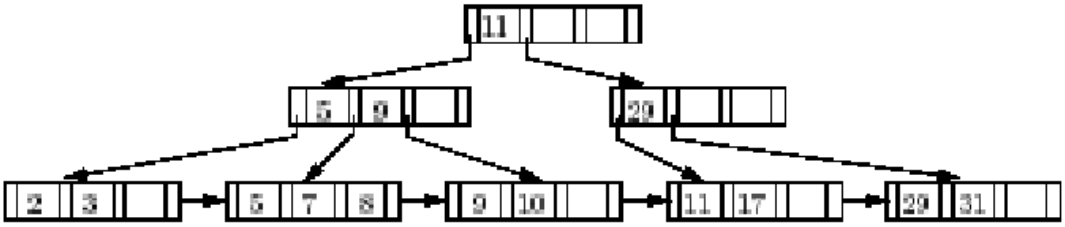
Insertar 8:



Borrar 23:



Borrar 19:



3) (0,6 puntos) Descríbase por qué no es adecuada la estrategia de sustitución LRU para el siguiente algoritmo de reunión y cuál sería la mejor estrategia.

```
for each p in P do
  for each c in C do
    if p[atributo]=c[atributo] then
      incluir en el resultado un nuevo registro como combinación de p y c.
```

R:
Política de extracción inmediata: se saca el bloque que corresponda a p para el que se haya realizado el ciclo completo de C.
Para cada ciclo de C el usado menos recientemente es el que se necesitará más tarde: inverso de la política LRU. Es mejor usar MRU (más recientemente utilizado, Most Recently Used).

4) (0,9 puntos) Explíquense las analogías y diferencias entre los ficheros invertidos y los mapas de bits.
R:
Analogías:
Los ficheros invertidos guardan una **lista** de direcciones de registros por cada valor de la clave, mientras que los mapas de bits guardan un **array** binario. Los ficheros invertidos guardan además para cada valor de la clave el número de registros con ese valor. Con los mapas de bits es posible realizar condiciones de búsqueda binaria complejas mediante simples operaciones lógicas binarias.

5) (0,9; 1,85 puntos) Demuéstrese que se cumple la equivalencia $\sigma_\theta(E_1 =\!|\times|\, E_2) = \sigma_\theta(E_1) =\!|\times|\, E_2$, donde θ sólo utiliza atributos de E1.
R:
θ utiliza sólo atributos de E_1. Por lo tanto, si cualquier tupla t en la salida de (E_1]x| E_2) se filtra mediante la selección del lado izquierdo, todas las tuplas en E_1 cuyos valores sean iguales a $t[E_1]$ se filtran por la selección del lado derecho. Por lo tanto:

$$\forall t, t \notin \sigma_\theta(E_1 =\!|\times|\, E_2) \Rightarrow t \notin \sigma_\theta(E_1) =\!|\times|\, E_2$$

Empleando un razonamiento similar, se puede concluir que

$$\forall t, t \notin \sigma_\theta(E_1) =\!|\times|\, E_2 \Rightarrow t \notin \sigma_\theta(E_1 =\!|\times|\, E_2)$$

Las dos ecuaciones anteriores implican la equivalencia dada. Esta equivalencia es útil porque la evaluación del lado derecho evita que se produzcan muchas tuplas en la operación (E_1]x| E_2), que, de todas formas, se eliminarán del resultado. Así, la expresión del lado derecho se puede evaluar más eficientemente que la del lado izquierdo.

6) (0,9; 1,85 puntos) Considérense las relaciones r1(A,B,C), r2(C,D,E) y r3(E,F). Supóngase que no hay claves principales, excepto el esquema completo. Sean V(C,r1)=900, V(C,r2)=1.100, V(E,r2)=50 y V(E,r3)=100. Supóngase que r1 tiene 1.000 tuplas, r2 tiene 1.500 tuplas y r3 tiene 750 tuplas. Estímese el tamaño de r1 |x| r2 |x| r3.
R:
El tamaño estimado de la relación se puede determinar calculando el número medio de tuplas que se reunirían con cada tupla de la segunda relación. En este caso, por cada tupla de r_1, 1.500/V (C, r_2) = 15/11 tuplas (por término medio) de r_2 se reunirían con ella. La relación intermedia tendría 15.000/11 tuplas. Esta relación se reúne con r_3 para conseguir un resultado de aproximadamente 10.227 tuplas (15.000/11 × 750/100 = 10.227). Una buena estrategia reuniría primero r_1 y r_2, dado que la relación intermedia es de tamaño parecido a r_1 o r_2. Después se reúne r_3 con este resultado.

7) (1,2 puntos) Considérense las dos transacciones siguientes:

```
T1:  leer(A);
     leer(B);
     si A = 0 entonces B := B + 1;
     escribir(B).
T2:  leer(B);
     leer(A);
     si B = 0 entonces A := A + 1;
     escribir(A).
```

Sea el requisito de consistencia A = 0 ∨ B = 0, siendo los valores iniciales A = B = 0.

a) Demuéstrese que toda ejecución secuencial en la que aparezcan estas transacciones conserva la consistencia de la base de datos.
b) Muéstrese una ejecución concurrente de T1 y T2 que produzca una planificación no secuenciable.
c) ¿Existe una ejecución concurrente de T1 y T2 que produzca una planificación secuenciable?

R:

a) Demuéstrese que toda ejecución secuencial en la que aparezcan estas transacciones conserva la
Hay dos ejecuciones posibles: $T_1\ T_2$ y $T_2\ T_1$.
Caso 1:

	A	B
inicialmente	0	0
después de T_1	0	1
después de T_2	0	1

Consistencia encontrada: $A = 0 \lor B = 0 \equiv T \lor F = T$
Caso 2:

	A	B
inicialmente	0	0
después de T_1	1	0
después de T_2	1	0

Consistencia encontrada: $A = 0 \lor B = 0 \equiv F \lor T = T$

b) Muéstrese una ejecución concurrente de T1 y T2 que produzca una planificación no secuenciable.
Cualquier entrelazamiento de T_1 y T_2 resulta en una planificación no secuenciable, como se verá en el apartado c). Por ejemplo:

T_1	T_2
leer(A)	
	leer(B)
	leer(A)
leer(B)	
si $A = 0$ **entonces** $B = B$	
+ 1	
	si $B = 0$ **entonces** $A = A$
	+ 1
	escribir(A)
escribir(B)	

c) ¿Existe una ejecución concurrente de T1 y T2 que produzca una planificación secuenciable?
No hay ninguna ejecución paralela resultante en una planificación secuenciable. Del apartado a se sabe que una planificación secuenciable resulta en $A = 0$ _ $B = 0$. Supóngase que se empieza con T_1 **leer**(A). Entonces, cuando la planificación termine, no importa cuando se ejecutan los pasos de T_2, $B = 1$. Supóngase ahora que se empieza ejecutando T_2 antes de completar T_1 . Entonces T_2 **leer**(B) dará a B un valor de 0 . Así, cuando se complete T_2, $A = 1$. Así, $B = 1 \land A = 1 \rightarrow \neg (A = 0 \lor B = 0)$. Análogamente empezando con T_2 **leer**(B).

8) (0,9; 1,85 puntos) Dadas las transacciones T1: LOCK A; LOCK B; UNLOCK A; UNLOCK B; y T2: LOCK B; UNLOCK B;LOCK A; UNLOCK A; se pide:
   a) Determinar el valor semántico de los elementos A y B en la planificación secuencial en la que T2 sigue a T1.
   b) Determinar si es posible encontrar una planificación secuenciable equivalente a la anterior.

R:

a) Determinar el valor semántico de los elementos A y B en la planificación secuencial en la que T2 sigue a T1.

T1		T2	
LOCK A		LOCK B	

LOCK B		UNLOCK B	$f_3(B)$
UNLOCK A	$f_1(A,B)$	LOCK A	
UNLOCK B	$f_2(A,B)$	UNLOCK A	$f_4(A,B)$

Tiempo		A	B
1	T1: LOCK A	A_0	B_0
2	T1: LOCK B	A_0	B_0
3	T1: UNLOCK A	$f_1(A_0,B_0)$	B_0
4	T1: UNLOCK B	"	$f_2(A_0,B_0)$
5	T2: LOCK B	"	$f_2(A_0,B_0)$
6	T2: UNLOCK B	"	$f_3(f_2(A_0,B_0))$
7	T2: LOCK A	"	"
8	T2: UNLOCK A	$f_4(f_1(A_0,B_0), f_2(A_0,B_0))$	"

b) Determinar si es posible encontrar una planificación secuenciable equivalente a la anterior.
Se puede hacer desde dos puntos de vista conocidos:
1) Construcción del grafo de precedencias:
Si T1 comienza antes que T2 hay que añadir un arco de T1 a T2 porque T2 debe bloquear A después del desbloqueo en la tercera instrucción de T1. La única posibilidad de entrelazamiento entre ambas es que T2 ejecute LOCK B después de LOCK A de T1. Por lo tanto habría que añadir un arco de T2 a T1, con lo que se formaría un ciclo.
Si T2 comienza antes que T1 hay que añadir un arco de T2 a T1 debido a UNLOCK B. La única posibilidad de entrelazamiento es que después de esta instrucción entre la primera de T1, pero entonces hay que añadir otro arco de T1 a T2 a causa de la primera instrucción de T1 (LOCK A). Por lo tanto, también se formaría un ciclo.
En conclusión, cualquier planificación entrelazada forma un ciclo en el grafo de precedencias, por lo que no es posible encontrar ninguna planificación secuenciable equivalente a la secuencial.
2) Determinación de la equivalencia semántica de los valores de los elementos.
Si T1 desbloquea A después de T2, se tendrá un valor f1(X1,Y1), mientras que si sucede al contrario se tendrá un valor f4(X4,Y4). Ocurre algo análogo con B: si T1 desbloquea B después de T2, se tendrá un valor f2(X2,Y2), mientras que si sucede al contrario se tendrá un valor f3(X3). Por lo tanto, las funciones semánticas serán diferentes en cualquier planificación que no sea la secuencial y, por tanto, no se podrá encontrar ninguna planificación secuenciable equivalente a la secuencial.

9) (0,9 puntos) Dada la tabla gastos(fecha: date, importe: number(7,2)), se pide escribir una consulta SQL en Oracle que genere un informe que muestre los gastos mensuales encolumnados con el siguiente formato:

Mes	Enero	Febrero	...	Diciembre
Importe			...	

R:

```
CREATE TABLE gastos(fecha DATE, importe NUMBER(7,2));
INSERT INTO gastos VALUES (TO_DATE('1/1/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/2/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/3/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/4/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/5/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/6/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/7/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/8/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/9/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/10/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/11/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/12/2003','DD/MM/YYYY'),1);
INSERT INTO gastos VALUES (TO_DATE('1/1/2003','DD/MM/YYYY'),2);
INSERT INTO gastos VALUES (TO_DATE('1/2/2003','DD/MM/YYYY'),3);
INSERT INTO gastos VALUES (TO_DATE('1/3/2003','DD/MM/YYYY'),4);
INSERT INTO gastos VALUES (TO_DATE('1/4/2003','DD/MM/YYYY'),5);
INSERT INTO gastos VALUES (TO_DATE('1/5/2003','DD/MM/YYYY'),6);
```

```
INSERT INTO gastos VALUES (TO_DATE('1/6/2003','DD/MM/YYYY'),7);
INSERT INTO gastos VALUES (TO_DATE('1/7/2003','DD/MM/YYYY'),8);
INSERT INTO gastos VALUES (TO_DATE('1/8/2003','DD/MM/YYYY'),9);
INSERT INTO gastos VALUES (TO_DATE('1/9/2003','DD/MM/YYYY'),10);
INSERT INTO gastos VALUES (TO_DATE('1/10/2003','DD/MM/YYYY'),11);
INSERT INTO gastos VALUES (TO_DATE('1/11/2003','DD/MM/YYYY'),12);
INSERT INTO gastos VALUES (TO_DATE('1/12/2003','DD/MM/YYYY'),13);
INSERT INTO gastos VALUES (TO_DATE('1/1/2002','DD/MM/YYYY'),1000);
INSERT INTO gastos VALUES (TO_DATE('1/1/2004','DD/MM/YYYY'),2000);

SELECT 'Importe' "Mes", SUM("Enero") "Enero", ..., SUM("Diciembre")
"Diciembre"
FROM
((SELECT SUM(importe) "Enero", ..., 0 "Diciembre"
  FROM gastos
  WHERE TO_CHAR(fecha,'MM')='01' AND TO_CHAR(fecha,'YY')='03')
 UNION
 (SELECT 0 "Enero", ..., SUM(importe) "Diciembre"
  FROM gastos
  WHERE TO_CHAR(fecha,'MM')='12' AND TO_CHAR(fecha,'YY')='03'))
;

Mes          Enero ...  Diciembre
------- ---------- ... ----------
Importe          3             14

1 fila seleccionada.
```

10) (0,9; 1,85 puntos) Dadas las tablas reservas(vuelo: varchar(10), plaza: varchar(4)) y plazas(vuelo: varchar(10), plaza: varchar(4)), se pide escribir una transacción en Oracle que informe de las plazas libres de un vuelo, permita hacer una reserva mientras se atiende a un cliente y permita descartarla si el cliente no efectúa finalmente la compra del billete. Explicar las consecuencias de la transacción en la operación concurrente con respecto a su funcionalidad como sistema de reserva de plazas.

R:

```
CREATE TABLE reservas(vuelo varchar(10), plaza varchar(4), PRIMARY KEY
(vuelo,plaza));
CREATE TABLE plazas(vuelo varchar(10), plaza varchar(4), PRIMARY
KEY(vuelo,plaza));
INSERT INTO plazas VALUES ('v1','p1');
INSERT INTO plazas VALUES ('v1','p2');
SELECT * FROM reservas;

SET AUTOCOMMIT OFF;
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE Reserva(p_Vuelo VARCHAR) AS
  CURSOR c_PlazasLibres IS
    SELECT plaza
    FROM plazas
    WHERE vuelo=p_Vuelo
    MINUS
    SELECT plaza
    FROM reservas
    WHERE vuelo=p_Vuelo
    ;
  v_Plaza VARCHAR(10);
  v_PlazasLibres BOOLEAN;
  v_Confirmar VARCHAR(1);
BEGIN
  v_PlazasLibres := FALSE;
```

```
  OPEN c_PlazasLibres;
  LOOP
    DBMS_OUTPUT.PUT_LINE('Plazas libres del vuelo '||p_Vuelo);
    FETCH c_PlazasLibres INTO v_Plaza;
    EXIT WHEN c_PlazasLibres %NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_plaza);
    v_PlazasLibres := TRUE;
  END LOOP;
  CLOSE c_PlazasLibres;
  IF v_PlazasLibres THEN
    DBMS_OUTPUT.PUT_LINE('Plaza a reservar:');
    READ(v_Plaza);
    INSERT INTO reservas VALUES (p_Vuelo,v_Plaza);
    DBMS_OUTPUT.PUT_LINE('¿Desea confirmar la reserva (S/N)?');
    READ(v_Confirmar);
    IF v_Confirmar = 'S' THEN
      DBMS_OUTPUT.PUT_LINE('Plaza confirmada.');
      COMMIT;
    ELSE
      ROLLBACK;
    END IF;
  ELSE
    DBMS_OUTPUT.PUT_LINE('No hay plazas libres.');
  END IF;
END Reserva;
```

Situaciones que pueden suceder al emitir dos transacciones T1 y T2 correspondientes a dos ejemplares de la ejecución del procedimiento anterior (y no hay ningún otro en ejecución):

1) T1 termina antes de que empiece T2 o viceversa (ejecución secuencial):
   Las dos transacciones operan con los datos actualizados de las reservas y no se interfieren entre sí.
2) T2 comienza antes de que T1 alcance su punto de compromiso o retroceso:
   Según el modelo de bloqueos de Oracle, T1 bloquea en escritura la tabla reservas hasta que se alcance su compromiso o retroceso porque incluye una instrucción de actualización sobre esta tabla. T2 puede mostrar las plazas libres porque la tabla reservas está bloqueada sólo para escritura. Sin embargo, la vista corresponde a una situación posiblemente desfasada (no se sabrá si corresponde con la realidad hasta que finalice T1). T2 se quedaría bloqueada después de solicitar la plaza a reservar. Sólo cuando T1 alcance su punto de compromiso, la instrucción de inserción de la plaza a reservar tendrá éxito o fallará. Si tiene éxito es porque T1 ha retrocedido o ha reservado una plaza diferente de T2. Si falla es porque T1 ha reservado la misma plaza que T2, y la restricción de clave impide que T2 actualice reservas.

Por lo tanto, esta solución no es la más adecuada porque un usuario que haya elegido una plaza mantendrá al resto de transacciones en espera. El intervalo de tiempo entre la decisión de la compra de la plaza y su actualización debe ser mínimo para aumentar la concurrencia. Esto se consigue en este ejemplo trasladando la instrucción de inserción antes de la instrucción que emite el mensaje de plaza confirmada.

Nota: El esquema de la solución que se presenta aquí es válido salvo en un detalle: desde PL/SQL en la lado del servidor (server-side) no es posible leer datos introducidos por el usuario, ya que los procedimientos almacenados en el servidor se ejecutan sin interacción con el cliente (desde donde se leerían los datos). Sería necesario usar Developer/Forms para conseguir la entrada por teclado del usuario o bien usar una secuencia de comandos (batch) en la consola SQL*Plus usando la orden ACCEPT.