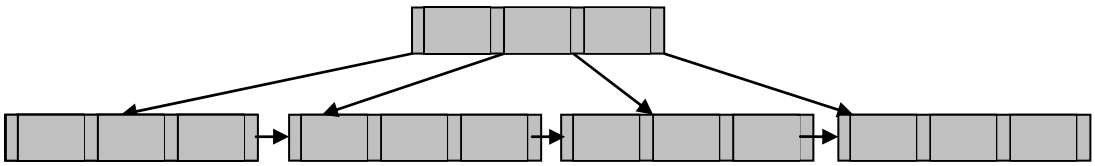
Examen de Ficheros y bases de datos (cód. 520)
Ingeniería Técnica en Informática de Gestión
Convocatoria de septiembre
II Parcial

1) (1,4 puntos) Considérese un árbol B+ con *n*=4.
   a) (0,7 puntos) ¿Cuál es el mínimo número de nodos necesarios para alojar 12 valores diferentes de la clave? Dibújese el árbol resultante.

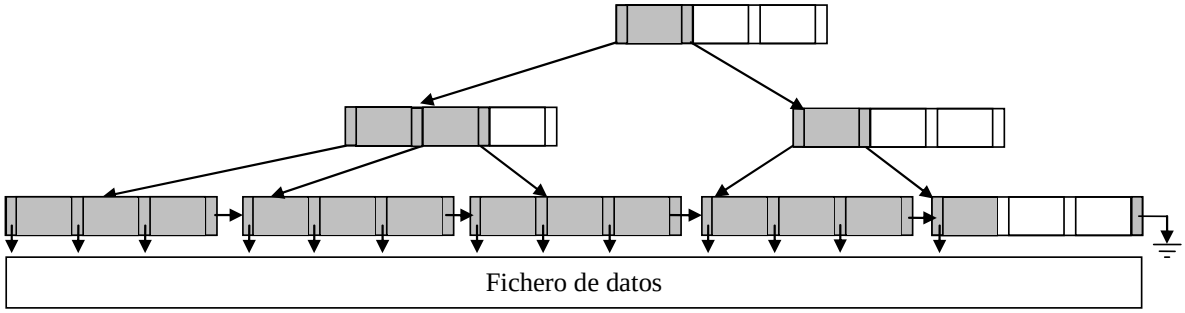
Solución:
Con n=4 se pueden alojar n-1=3 valores de la clave por nodo. Como todos los valores se deben representar en los nodos hoja, son necesarios $\left\lceil \frac{12}{3} \right\rceil = 4$ nodos hoja. Como restricciones que debe cumplir este árbol se tiene:
El nodo raíz tiene entre 1 y 4 hijos.
Cada nodo interno tiene entre $\left\lceil \frac{4}{2} \right\rceil = 2$ hijos y 4.
Los nodos hoja contienen entre $\left\lceil \frac{4-1}{2} \right\rceil = 2$ y 4-1=3 valores.

En este caso, el nodo raíz puede apuntar a todos los nodos hoja porque n (número de punteros por nodo) es 4. No es posible estructuralmente usar un número menor de nodos. La estructura es (donde los espacios ocupados para valores de la clave se sombrean en gris):



   b) (0,7 puntos) Respóndase el apartado a) para 13 valores y determínese el número de valores y punteros sin usar. Calcúlese el porcentaje de ocupación suponiendo que un valor de la clave ocupa igual que un puntero.

Solución:
En este caso son necesarios $\left\lceil \frac{13}{3} \right\rceil = 5$ nodos hoja. Como el nodo raíz no puede apuntar a todos ellos como en el apartado a), es necesario añadir un nivel adicional, que necesariamente contendrá un nodo raíz, dos nodos internos y 5 nodos hojas, el menor número de nodos posible.



Cada nodo tiene 3 valores y 4 punteros, hay 8 nodos, en total (3+4)*8=56 valores y punteros, de los cuales no se usan 7 valores y 7 punteros (14). Por lo tanto, el porcentaje de ocupación es $\frac{56-14}{56} = 75\%$

2) (2,1 puntos) Dado el esquema Empleados(Nombre, DNI, Sueldo), Teléfonos(DNI, Teléfono), Domicilios(DNI, Calle, Código postal), Códigos postales(Código postal, Población, Provincia), se pide formular consultas en Oracle para:
   a) (1,2 puntos) Consulta SQL para listar *todos* los empleados (independientemente de que tenga o no domicilios o teléfonos asociados) que muestre Nombre, DNI, Calle, Población, Provincia, Código postal, Teléfono con las dos sintaxis (clásica de Oracle y estándar ANSI) para las reuniones (join).

Solución:

- Con la sintaxis estándar para las reuniones.

```
SELECT Nombre, DNI, Calle, Población, Provincia, "Código postal", Teléfono
FROM (((Empleados NATURAL LEFT OUTER JOIN Teléfonos) NATURAL LEFT OUTER
JOIN (Domicilios NATURAL INNER JOIN "Códigos postales")));
```

- Con la sintaxis clásica de Oracle para las reuniones.

```
SELECT Nombre, Empleados.DNI, Calle, Población, Provincia,
Domicilios."Código postal", Teléfono
FROM Empleados, Domicilios, "Códigos postales", Teléfonos
WHERE Empleados.DNI=Teléfonos.DNI(+) AND Empleados.DNI=Domicilios.DNI(+)
AND Domicilios."Código postal"="Códigos postales"."Código postal"(+);
```

En ambos casos, el resultado es el mismo:

```
NOMBRE          DNI       CALLE               POBLACIÓN   PROVINCIA  Códig TELÉFONO
--------------- --------- ------------------- ----------- ---------- ----- ---------
Antonio Arjona  12345678A Avda. Complutense   Madrid      Madrid     28040
Antonio Arjona  12345678A Cántaro                                    28004
Carlota Cerezo  12345678C                                                  611111111
Carlota Cerezo  12345678C                                                  931111111
Laura López     12345678L Diamante            Peñarroya   Córdoba    14200 913333333
Pedro Pérez     12345678P Carbón              Lucena      Córdoba    14900 644444444

6 filas seleccionadas.
```

   b) (0,9 puntos) Consulta SQL para incrementar en un 10% el sueldo de todos los empleados de forma que no se supere en ningún caso 2.100 €, pero asegurando que se pueda deshacer el cambio (sin usar retrocesos de transacciones). Escríbase también la solución que permita deshacer el cambio.

Solución:

Es necesario anotar las tuplas cambiadas. Para ello recurrimos a una tabla, que se define como:

```
CREATE TABLE Cambios(DNI CHAR(9) PRIMARY KEY, Sueldo NUMBER(6,2));
```

A continuación se anota el cambio y se realiza la modificación:

```
DELETE FROM Cambios;
INSERT INTO Cambios SELECT DNI, Sueldo FROM Empleados WHERE Sueldo*1.1 <=
2100;
UPDATE Empleados SET Sueldo = Sueldo*1.1
WHERE Sueldo*1.1 <= 2100;
```

Para recuperar el cambio realizado:

```
UPDATE Empleados SET Sueldo = (SELECT Cambios.Sueldo FROM Cambios WHERE
Cambios.DNI=Empleados.DNI);
```

3) (2,9 puntos) Programar un disparador que actualice la tabla Estadísticas(Provincia, Empleados, SueldoMedio, SueldoMáximo, SueldoMínimo) cada vez que se inserte una tupla en la tabla Empleados del problema anterior. En esta tabla se tiene una fila por cada provincia con el número de empleados de esa provincia, el sueldo medio, mínimo y máximo. Se considera que existe previamente creada una fila en la tabla Estadísticas por cada provincia. Supóngase en este problema que cada empleado sólo tiene un domicilio asociado.

Solución:

La definición de la tabla Estadísticas puede ser:

```
CREATE TABLE Estadísticas(Provincia CHAR(50) PRIMARY KEY,
                          Empleados INTEGER,
                          SueldoMedio NUMBER(10,2),
                          SueldoMáximo NUMBER(6,2),
                          SueldoMínimo NUMBER(6,2));
```

La actualización de la tabla Estadísticas presenta el problema de la tabla mutante. En este caso, si el disparador se activa por cada fila, no se puede consultar Empleados para calcular los agregados sobre sueldo, ya que esta tabla se está actualizando. Sin embargo, se puede actualizar con respecto a la nueva tupla introducida en Empleados y la información que existe en Estadísticas, que no está mutando. Así, al insertar un nuevo empleado, se calcularía:
Estadísticas.Empleados= Estadísticas.Empleados+1
Estadísticas.SueldoMáximo=máximo(Empleados.Sueldo, Estadísticas.SueldoMáximo)
Estadísticas.SueldoMínimo=mínimo(Empleados.Sueldo, Estadísticas.SueldoMínimo)
El sueldo medio al añadir uno empleado más se puede obtener en función del sueldo medio y el número de empleados anteriores. Se puede ver fácilmente que la media de n elementos es:

$$\overline{x_n} = \frac{x_1 + ... + x_n}{n}$$

Al añadir un nuevo elemento, la media es:

$$\overline{x_{n+1}} = \frac{x_1 + ... + x_n + x_{n+1}}{n+1}$$

Esta expresión se puede reescribir en términos de $\overline{x_n}$, n y x_{n+1}:

$$\overline{x_{n+1}} = \frac{x_1 + ... + x_n + x_{n+1}}{n+1} = \frac{x_1 + ... + x_n}{n+1} + \frac{x_{n+1}}{n+1} = \frac{1}{\frac{n+1}{x_1 + ... + x_n}} + \frac{x_{n+1}}{n+1}$$

$$\frac{1}{\frac{n+1}{x_1 + ... + x_n}} = \frac{1}{\frac{n}{x_1 + ... + x_n} + \frac{1}{x_1 + ... + x_n} \cdot \frac{n}{n}} = \frac{1}{\overline{x_n}^{-1} + \frac{\overline{x_n}^{-1}}{n}} = \frac{n\overline{x_n}}{n+1}$$

$$\overline{x_{n+1}} = \frac{n\overline{x_n}}{n+1} + \frac{x_{n+1}}{n+1} = \frac{n\overline{x_n} + x_{n+1}}{n+1}$$

Por lo tanto:
Estadísticas.SueldoMedio=(Estadísticas.Empleados*
Estadísticas.SueldoMedio+:NEW.Sueldo)/(Estadísticas.Empleados+1)

```
CREATE OR REPLACE TRIGGER ActualizaEstadísticas
  AFTER INSERT ON Empleados
  FOR EACH ROW
BEGIN
  UPDATE Estadísticas SET
    Empleados=Empleados+1,
    SueldoMedio=(Empleados*SueldoMedio + :NEW.Sueldo)
              /(Empleados+1),
    SueldoMáximo=GREATEST(SueldoMáximo,:NEW.Sueldo),
    SueldoMínimo=LEAST(SueldoMínimo,:NEW.Sueldo)
  WHERE Provincia=
    (SELECT "Códigos postales".Provincia
     FROM Domicilios, "Códigos postales"
     WHERE :NEW.DNI=Domicilios.DNI AND
     Domicilios."Código postal"="Códigos postales"."Código postal");
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error inesperado.');
END;
/
```

Otra opción es retrasar la actualización de la tabla Estadísticas hasta que se haya completado la orden de inserción completamente, es decir, usar el disparador en el nivel de orden, no de fila. El problema es el rendimiento, porque habría que calcular los agregados sobre toda la tabla Empleados en cada inserción y, lo que es peor, para todas las provincias.

```
CREATE OR REPLACE TRIGGER ActualizaEstadísticas
  AFTER INSERT ON Empleados
BEGIN
  DELETE FROM Estadísticas;
  INSERT INTO Estadísticas
```

```
    SELECT Provincia,COUNT(*),AVG(Sueldo),MAX(Sueldo),MIN(Sueldo)
    FROM Empleados, Domicilios, "Códigos postales"
    WHERE
      Empleados.DNI=Domicilios.DNI AND
      Domicilios."Código postal"="Códigos postales"."Código postal"
    GROUP BY Provincia;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error inesperado.');
END;
/
```

4) (3,6 puntos) Se desea indexar un archivo r(A: String(20), B: String(250), C: String(350)), con representación ASCII y 100.000 filas, bajo la clave A (con repeticiones) mediante una estructura de índice secundario denso y asignación enlazada. El archivo secundario contiene 20.000 filas. Cada cajón se implementa en un bloque. Los bloques tienen un tamaño de tan sólo 50 bytes, sus direcciones ocupan 4 bytes y tienen mapas de bits de existencia para los registros. Se pide:
   a) (0,6 puntos) Describir la estructura y tamaño de los ficheros de índice, de datos y los cajones.

Solución:

En media, el fichero de datos tendrá 5 entradas repetidas (100.000/20.000) por cada entrada del fichero de índice (que es denso y representa a todos los valores de la clave que aparecen en el fichero de datos). Por lo tanto, cada entrada del fichero de índice tiene asociado un cajón con 5 entradas que apuntan al fichero de datos.

- Fichero de índice:
  Registros de pares (Valor de la clave, Dirección de bloque)
  Tamaño: 20.000*(20+4) = 480.000 bytes
  Bloque: Mapa de bits de existencia (N bits), N registros y dirección de enlace (4 bytes).
- Fichero de datos:
  Registros de tuplas (A, B, C)
  Tamaño: 100.000*(20+250+350)=62.000.000 bytes
  Bloque: Mapa de bits de existencia (N bits), N registros y dirección de enlace (4 bytes).
- Cajones:
  Registros de direcciones de bloque
  Tamaño: 20.000*5*4=400.000 bytes
  Bloque: Mapa de bits de existencia (N bits), N registros y dirección de enlace (4 bytes).

   b) (2,0 puntos) Calcular el factor de bloqueo de los ficheros de datos y de índice y de los cajones.

Solución:

Para el fichero de datos:
$N*(20+250+350)*8+4*8+N \leq 50*8$, donde:

- N*(20+250+350)*8=N*4.960 es el tamaño en bits ocupado por un registro (20 bytes para el campo A, 250 para el campo B y 350 para el campo C).
- 4*8=32 es el número de bits ocupado por la dirección del siguiente bloque para la asignación enlazada
- N son los bits necesarios para el mapa de bits de existencia
- 50*8=400 es el número de bits en los 50 bytes de un bloque

$N*(4960+1) \leq 400-32$

$N \leq 0{,}0742$

Por tanto, el factor de bloqueo N del fichero de datos es de 0,0742 registros por bloque. Calculando su inversa, se obtiene que son necesarios 13,5 bloques para alojar un registro. Para bloques fijos (la situación más habitual), se usan 14 bloques y se desperdicia medio bloque por cada registro.

Para el fichero de índice:

$N*(20+4)*8+4*8+N \leq 50*8$, donde:

- N*(20+4)*8=N*192 es el tamaño en bits ocupado por un registro (20 bytes para el campo clave A y 4 bytes para la dirección de bloque).
- 4*8=32 es el número de bits ocupado por la dirección del siguiente bloque para la asignación enlazada
- N son los bits necesarios para el mapa de bits de existencia
- 50*8=400 es el número de bits en los 50 bytes de un bloque

$N*(192+1) \leq 400-32$

$N \leq 1{,}9$

Por tanto, el factor de bloqueo N del fichero de índice es de 1 registro por bloque.

Para los cajones:

$N*(4)*8+4*8+N \leq 50*8$, donde:

- N*(4)*8=N*192 es el tamaño en bits ocupado por un registro (4 bytes para la dirección de bloque).
- 4*8=32 es el número de bits ocupado por la dirección del siguiente bloque para la asignación enlazada
- N son los bits necesarios para el mapa de bits de existencia
- 50*8=400 es el número de bits en los 50 bytes de un bloque

$N*(32+1) \leq 400-32$

$N \leq 11{,}15$

Por tanto, el factor de bloqueo N del fichero de índice es de 11 registros por bloque. En media, sólo será necesario un cajón por cada entrada del índice, pero también tienen dirección de enlace para contemplar el caso en que hay más de 11 entradas por un valor de la clave.

c) (1,0 puntos) Calcular el tiempo medio de acceso a una tupla en concreto bajo su valor de la clave en dos supuestos: sin usar el fichero de índice y usándolo. Determinar la ganancia de velocidad en este último caso. Considérese un tiempo de lectura de bloque de 12 ms.

Solución:

Sin usar el fichero de índice:

En media hay que recorrer 100.000/2=50.000 registros del fichero de datos secuencialmente, es decir, 50.000*14 = 700.000 bloques. A 12ms por bloque se invierten: 700.000*12ms=8.400 s = 140 m = 2 horas y 20 minutos

Usando el fichero de índice:

Se recorren 20.000/2=10.000 registros del fichero de índice secuencialmente, es decir, 10.000 bloques. Despreciando el acceso al bloque del cajón (1 bloque) y el de datos (14 bloques), a 12ms por bloque se invierten: 10.000*12ms=120 s = 10 min.
Haciendo las cuentas de forma precisa: (10.000+1+14)*12ms=120,18 s=10 min y 0,18*60=10,8 s = 10 min 10 s y 8 décimas

Una conclusión inmediata es reconocer que el tamaño de bloque es muy pequeño y, aunque se mejora el acceso con el índice, se debería aumentar.