Examen de Bases de datos y sistemas de información
Convocatoria de septiembre
II PARCIAL

1) (2,9 puntos) Considérese un fichero secuencial indexado denso con asignación enlazada con los campos A:Byte(2), B:String(10), C:String(20), cuya clave, que no admite duplicados, es el campo A y con mapas de bits de existencia en el fichero de datos y de índice. El formato de los caracteres es ASCII y las cadenas de caracteres son de longitud fija. En la unidad de disco se tiene que las direcciones de bloque son de 4 bytes, la frecuencia de rotación del disco es de 7.200 rpm, el tiempo de búsqueda es de 10 ms, hay 512 sectores por pista, 5.000 pistas por cilindro, dos sectores por bloque, 1.024 bytes por sector, 16 platos de doble cara y las escrituras necesitan el doble de tiempo que las lecturas. Se pide:
   a) (0,7 puntos) Calcular el tamaño de bloque y de la unidad de disco en bytes, y los tiempos de lectura y escritura de bloque.

Solución:

Tamaño de bloque: 1 bloque * 2 sectores/bloque * 1.024 bytes/sector = 2.048 bytes
Tamaño de la unidad de disco: 16 platos * 2 caras/plato * 5.000 pistas/cara * 512 sectores/pista * 1.024 bytes/sector = 83.886.080.000 bytes, aprox. 80 Gb
Suponiendo una distribución aleatoria de los bloques en disco, el tiempo medio de lectura de un bloque es:
Tiempo medio operación E/S=Tiempo de búsqueda + Tiempo de latencia + Tiempo de transmisión
Tiempo de búsqueda=10ms

Tiempo de latencia=1/2*Inversa de la frecuencia = $\frac{1}{2}\frac{1}{7.200\frac{1}{\min}\frac{1\min}{60.000ms}} = 4{,}17ms$

Tiempo de transmisión=2* $\frac{1}{512}$ *Tiempo de rotación (el doble del tiempo de latencia) = 0,032ms

Tiempo medio operación lectura=10+4,17+0,032 = 14,2 ms
Tiempo medio operación escritura=2*14,2 ms = 28,4 ms

   b) (0,7 puntos) Calcular el factor de bloqueo de los ficheros de datos y de índice.

Solución:

Para el fichero de datos:
N*(2+10+20)*8+4*8+N ≤ 2.048*8, donde:
- N*(2+10+20)*8=N*256 es el tamaño en bits ocupado por un registro (2 bytes para el campo A, 10 para el campo B y 20 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
- 2.048*8=16.384 es el número de bits en los 2.048 bytes de un bloque

N*(256+1)≤ 16.384 -32
N≤63,62
Por tanto, el factor de bloqueo N del fichero de datos es de 63 registros por bloque.

Para el fichero de índice:
N*(2+4)*8+4*8+N ≤ 2.048*8, donde:
- N*(2+4)*8=N*48 es el tamaño en bits ocupado por un registro (2 bytes para el campo 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
- 2.048*8=16.384 es el número de bits en los 2.048 bytes de un bloque

N*(48+1)≤ 16.384 -32
N≤333,71
Por tanto, el factor de bloqueo N del fichero de índice es de 333 registros por bloque.

c) (1,5 puntos) Calcular el tiempo necesario para reorganizar el archivo de datos y de índices asumiendo que se tienen 10.000 registros de datos, 300 registros en el archivo de desbordamiento cuyos valores de la clave se encuentran entre dos consecutivos del fichero de índice en dos casos: cuando los valores están comprendidos entre el primero y el segundo registros del fichero de índice, y cuando los valores están comprendidos entre el penúltimo y el último registros.

Solución:

Para reorganizar el fichero hay que leer el archivo de desbordamiento secuencialmente e insertar los registros en el fichero de datos (maestro). La técnica más apropiada en este caso será leer el fichero de desbordamiento, ordenarlo en memoria e insertar de forma ordenada. Si no hay memoria suficiente, hay que usar una ordenación externa con coste O(log n).
Para insertar hay que leer el fichero de índice para localizar el bloque de datos adecuado, reorganizarlo con el nuevo registro, reclamar y escribir los nuevos bloques de datos que provengan del fichero de desbordamiento y escribir el resto de bloques de datos (hay que tener en cuenta que no se podrá simplemente reconectar la cadena del resto de bloques, para compactar el fichero de datos).

El primer caso es cuando los valores están comprendidos entre el primero y el segundo registros del fichero de índice.
1 lectura del fichero de índice para localizar el bloque de datos donde insertar.
1 lectura del fichero de datos para leer el bloque de datos donde insertar.
$\left\lceil \frac{300}{63} \right\rceil = 5$ lecturas de bloque del archivo de desbordamiento

$\left\lceil \frac{10.000}{63} \right\rceil = 159$ lecturas de bloque del archivo de datos (para escribir posteriormente)
$\left\lceil \frac{10.000+300}{63} \right\rceil = 164$ escrituras de bloque en el archivo de datos para insertar los registros del fichero de desbordamiento
$\left\lceil \frac{10.000+300}{334} \right\rceil = 31$ escrituras de bloque en el archivo de índices (no es necesario leer sus contenidos anteriores) para apuntar a los nuevos registros del fichero de datos

Número total de lecturas: 1+1+5+159=166 bloques
Número total de escrituras: 164+31=195 bloques
Tiempo total = 166*14,19+195*2*14,19=7.887,58 ms, es decir, casi 8 segundos.

El segundo caso es cuando los valores están comprendidos entre el penúltimo y el último registros del fichero de índice.
$\left\lceil \frac{10.000}{334} \right\rceil = 30$ lecturas de bloque en el archivo de índices para localizar el bloque de datos donde insertar.
1 lectura del fichero de datos para leer el bloque de datos donde insertar.
$\left\lceil \frac{300}{63} \right\rceil = 5$ lecturas de bloque del archivo de desbordamiento
$\left\lceil \frac{10.000+300}{63} \right\rceil = 164$ escrituras de bloque en el archivo de datos para insertar los registros del fichero de desbordamiento
$1+\left\lceil \frac{300}{334} \right\rceil = 2$ escrituras de bloque en el archivo de índices para apuntar a los nuevos registros del fichero de datos (reescribiendo adecuadamente el último que ya existía)

Número total de lecturas: 30+1+5=36 bloques
Número total de escrituras: 164+2=166 bloques

Tiempo total = 36*14,19+166*2*14,19=5.221,92 ms, es decir, un poco más de 5 segundos.

2) (2 puntos) Dadas las tablas t(A INTEGER, B INTEGER, Borrado CHAR(2)) y s(A INTEGER, B INTEGER, Hora CHAR(8), Fecha CHAR(10)), se pide:
   a) (0,7 puntos) Crear las tablas con restricciones de clave, dominio para t.Borrado (sólo se admiten los valores 'Sí' y 'No'), e integridad referencial $\pi_A(s) \subseteq \pi_A(t)$, $\pi_B(s) \subseteq \pi_B(t)$.
   b) (1 punto) Programar un disparador sobre la tabla t que se active en los borrados de fila, de forma que las filas borradas se copien a la tabla s, junto con información de la hora (hh:mm:ss) y la fecha (dd/mm/aaaa) en la que se produjo el borrado. Además, en lugar de borrar físicamente la fila, se debe anotar en la tabla t como borrada lógicamente, es decir, estableciendo el indicador de Borrado a TRUE.
   c) (0,3) ¿Qué problema aparece con las restricciones de integridad referencial?

Solución:

Se crean las tablas:

```
CREATE TABLE t(A INTEGER PRIMARY KEY, B INTEGER, Borrado CHAR(2), CHECK (Borrado IN
('Sí','No')));
CREATE TABLE s(A INTEGER REFERENCES t DEFERRABLE, B INTEGER REFERENCES t DEFERRABLE, Hora
CHAR(8), Fecha CHAR(10));
```

No se define clave primaria para la tabla s porque puede haber tuplas repetidas (por ejemplo, al borrar una tupla r, insertarla y volverla a borrar en el mismo segundo).

Se insertan datos en t:

```
INSERT INTO t VALUES (0,0,'No');
INSERT INTO t VALUES (1,1,'No');
INSERT INTO t VALUES (2,2,'No');
INSERT INTO t VALUES (3,3,'No');

SELECT * FROM t;

         A          B BORRADO
---------- ---------- -------
         0          0 No
         1          1 No
         2          2 No
         3          3 No
```

La primera solución que se nos podría ocurrir sería la siguiente:

```
CREATE OR REPLACE TRIGGER Borrado1T
  AFTER DELETE ON t
  FOR EACH ROW
BEGIN
  INSERT INTO s
    VALUES(:OLD.A,
           :OLD.B,
           TO_CHAR(SYSDATE,'hh:mi:ss'),
           TO_CHAR(SYSDATE,'dd/mm/yyyy'));
  INSERT INTO t
        VALUES(:OLD.A,:OLD.B,'Sí');
END;
/
```

Sin embargo, no es posible usar un disparador de este estilo porque la segunda instrucción INSERT modifica la tabla sobre la que se activa el disparador bajo una actualización (un borrado de filas en este caso). Oracle (ni otros SGBD como, por ejemplo, DB2) no admite modificaciones de estas tablas para disparadores en el nivel de filas (row triggers) porque podrían provocar ciclos. Las tablas que tienen definidos disparadores en el nivel de filas se denominan mutantes cuando se está en la fase de activación de estos disparadores. Sólo se permite modificar estas tablas cuando se sabe que no se van a producir más disparos en el nivel de filas, es decir, sólo para disparadores en el nivel de instrucciones (statement triggers). Estos disparadores se definen en Oracle omitiendo la declaración FOR EACH ROW. El inconveniente que

aparece es que, dado que el disparador sólo se activa al final de la ejecución de la instrucción de borrado, no se sabe cuáles son las filas borradas (i.e., no se tiene acceso a los valores :OLD y :NEW).
La siguiente solución que se nos puede ocurrir es usar dos disparadores y una tabla intermedia x con la estructura de t. El objetivo es almacenar las filas borradas en x con un disparador en el nivel de filas y, una vez borradas todas las filas, activar un disparador a nivel de instrucciones que traslade las filas a t, como se muestra a continuación.

```
CREATE TABLE x(A INTEGER PRIMARY KEY, B INTEGER, Borrado CHAR(2));

CREATE OR REPLACE TRIGGER Borrado1T
  AFTER DELETE ON t
  FOR EACH ROW
BEGIN
  INSERT INTO s
    VALUES(:OLD.A,
           :OLD.B,
           TO_CHAR(SYSDATE,'hh:mi:ss'),
           TO_CHAR(SYSDATE,'dd/mm/yyyy'));
  INSERT INTO x
        VALUES(:OLD.A,:OLD.B,'Sí');
END;
/

CREATE OR REPLACE TRIGGER Borrado2T
  AFTER DELETE ON t
BEGIN
  INSERT INTO t
    SELECT * FROM x;
  DELETE FROM x;
END;
/
```

Recuérdese que hay que evitar siempre que sea posible el uso de cursores para conseguir el máximo rendimiento, y en este caso es claramente posible.

Con esta solución aparece otro problema: la tabla t puede ser actualizada desde diferentes transacciones, por lo que no se asegura la corrección del resultado. La solución es usar tablas privadas a cada transacción, lo cual puede conseguirse mediante los paquetes, como se muestra a continuación.

```
CREATE OR REPLACE PACKAGE BorradoT AS
  TYPE t_t IS TABLE OF t%ROWTYPE;
  v_t t_t := t_t();
  v_índice INTEGER := 0;
END BorradoT;
/

CREATE OR REPLACE TRIGGER Borrado1T
  AFTER DELETE ON t
  FOR EACH ROW
BEGIN
  INSERT INTO s
    VALUES(:OLD.A,
           :OLD.B,
           TO_CHAR(SYSDATE,'hh:mi:ss'),
           TO_CHAR(SYSDATE,'dd/mm/yyyy'));
  BorradoT.v_índice := BorradoT.v_índice + 1;
  BorradoT.v_t.EXTEND;
  BorradoT.v_t(BorradoT.v_índice).A := :OLD.A;
  BorradoT.v_t(BorradoT.v_índice).B := :OLD.B;
  BorradoT.v_t(BorradoT.v_índice).Borrado := 'Sí';
END;
/

CREATE OR REPLACE TRIGGER Borrado2T
  AFTER DELETE ON t
DECLARE
```

```
  v_i INTEGER;
BEGIN
  v_i := BorradoT.v_t.FIRST;
  WHILE v_i <= BorradoT.v_t.LAST LOOP
    INSERT INTO t
      VALUES(BorradoT.v_t(v_i).A, BorradoT.v_t(v_i).B, BorradoT.v_t(v_i).Borrado);
    v_i := BorradoT.v_t.NEXT(v_i);
  END LOOP;
  BorradoT.v_t.DELETE;
  BorradoT.v_índice := 0;
END;
/
```

La última solución que se propone es usar disparadores INSTEAD OF (en lugar de), que permiten impedir la acción de disparo (en nuestro caso el borrado) y efectuar otra acción en su lugar. El inconveniente es que no pueden funcionar directamente sobre las tablas, sino sobre vistas. Por otra parte, esto presenta la ventaja de poder definir la vista como todas las columnas de t salvo Borrado, que permanecería oculta, mostrando sólo las filas no borradas "lógicamente".

```
CREATE OR REPLACE VIEW v AS SELECT A, B FROM t WHERE Borrado='No';

CREATE OR REPLACE TRIGGER Borrado1T
  INSTEAD OF DELETE ON v
  FOR EACH ROW
BEGIN
  UPDATE t SET Borrado = 'Sí' WHERE A=:OLD.A;
  INSERT INTO s
    VALUES(:OLD.A,
           :OLD.B,
           TO_CHAR(SYSDATE,'hh:mi:ss'),
           TO_CHAR(SYSDATE,'dd/mm/yyyy'));
END;
/
```

En las dos primeras soluciones, y para permitir que se borre primero de t y después se inserte sin que se viole la restricción de integridad referencial de s, hay que desactivar el modo de autocompromiso y diferir la comprobación de las restricciones hasta el punto de compromiso.

```
SET AUTOCOMMIT OFF;
SET CONSTRAINTS ALL DEFERRED;
```

Al borrar una tupla de t:

```
DELETE FROM t WHERE A=0;

COMMIT;

SELECT * FROM t;

         A          B Borrado
---------- ---------- -------
         0          0 Sí
         1          1 No
         2          2 No
         3          3 No

SELECT * FROM s;

         A          B HORA     FECHA
---------- ---------- -------- ----------
         0          0 04:42:37 17/06/2005
```

En la última solución, al borrar una tupla de v:

```
DELETE FROM v WHERE A=0;
```

```
SELECT * FROM v;

         A          B
---------- ----------
         1          1
         2          2
         3          3

SELECT * FROM t;

         A          B BO
---------- ---------- --
         0          0 Sí
         1          1 No
         2          2 No
         3          3 No

SELECT * FROM s;

         A          B HORA     FECHA
---------- ---------- -------- ----------
         0          0 04:55:51 21/06/2005
```

En cualquiera de las tres alternativas surge el problema de la inserción de filas que ya estén en t, pero borradas. Para solucionarlo se crea un disparador que actúe consecuentemente en las inserciones, de forma que ponga a 'No' el campo Borrado si la fila existe. Por ejemplo, para la última solución:

```
CREATE OR REPLACE TRIGGER InserciónV
  INSTEAD OF INSERT ON v
  FOR EACH ROW
DECLARE
  v_A t.A%TYPE;
  v_Borrado t.Borrado%TYPE;
BEGIN
  SELECT A, Borrado INTO v_A, v_Borrado FROM t WHERE A=:NEW.A;
  IF v_A = :NEW.A AND v_Borrado='No' THEN
    -- Se intenta insertar una tupla existente: dará error de clave primaria
    INSERT INTO t VALUES(:NEW.A,:NEW.B,'No');
  ELSE
    -- Se inserta una tupla que había sido borrada
    UPDATE t SET Borrado='No' WHERE A=:NEW.A;
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    -- Se inserta una tupla que no existe
    INSERT INTO t VALUES(:NEW.A,:NEW.B,'No');
END;
/
```

Partiendo de las tablas vacías:

```
INSERT INTO v VALUES (0,0);
INSERT INTO v VALUES (1,1);
INSERT INTO v VALUES (2,2);
INSERT INTO v VALUES (3,3);

SELECT * FROM t;

         A          B BO
---------- ---------- --
         0          0 No
         1          1 No
         2          2 No
         3          3 No

INSERT INTO v VALUES (0,0);
```

```
*
ERROR en línea 1:
ORA-00001: restricción única (INF00.SYS_C0015889) violada
ORA-06512: en "INF00.INSERCIÓNV", línea 8
ORA-04088: error durante la ejecución del disparador 'INF00.INSERCIÓNV'

DELETE FROM v WHERE A=0;

1 fila suprimida.

INSERT INTO v VALUES (0,0);

1 fila creada.
```

Para volver al estado inicial:

```
DROP TRIGGER Borrado1T;
DROP TRIGGER Borrado2T;
DROP PACKAGE BorradoT;

DROP VIEW v;
DROP TABLE x;
DROP TABLE s;
DROP TABLE t;
```

3) (2,4 puntos) Considérense las dos transacciones siguientes.

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

Sea el requerimiento de consistencia $A = 0 \vee B = 0$, siendo los valores iniciales A = B = 0. Se pide:

a) (0,7 puntos) Demuéstrese que toda ejecución secuencial en la que aparezcan estas transacciones conserva la consistencia de la base de datos.

Solución:

Hay dos ejecuciones posibles: $T1\ T2$ y $T2\ T1$.
Caso 1: $T1\ T2$

	A	B
inicialmente	0	0
después de T1	0	1
después de T2	0	1

Consistencia encontrada: $A = 0 \vee B = 0 \equiv T \vee F = T$
Caso 2: $T2\ T1$

	A	B
inicialmente	0	0
después de T2	1	0
después de T1	1	0

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

b) (0,7 puntos) Muéstrese una ejecución concurrente de T1 y T2 que produzca una planificación no secuenciable.

Solución:

Cualquier entrelazamiento de $T1$ y $T2$ resulta en una planificación no secuenciable.

T1	*T2*	*A en T1*	*B en T1*	*A en T2*	*B en T2*	*A en BD*	*B en BD*

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

c) (1 punto) ¿Existe alguna ejecución concurrente de T1 y T2 que produzca una planificación secuenciable?

Solución:

No hay ninguna ejecución paralela que sea una planificación secuenciable. Del apartado (a) se sabe que una planificación secuenciable resulta en $A = 0 \vee B = 0$. Supóngase que se empieza con $T1$ **leer**(A). Entonces, cuando la planificación termine, independientemente de cómo se ejecuten los pasos de $T2$, $B = 1$. Supóngase ahora que se empieza ejecutando $T2$ antes de completar $T1$. Entonces $T2$ **leer**(B) dará a B un valor de 0. Así, cuando se complete $T2$, $A = 1$. Por tanto, $B = 1 \wedge A = 1 \rightarrow \neg (A = 0 \vee B = 0)$.
Análogamente empezando con $T2$ **leer**(B).

4) (1 punto) ¿Se pueden crear tablas en Oracle que no cumplan la primera forma normal? Dése un ejemplo de creación de una de estas tablas y una consulta de selección sobre ella.

Solución:

```
CREATE TYPE typTeléfono AS OBJECT (
  Teléfono VARCHAR2(9),
  Tipo     CHAR(1)
);
/

CREATE TYPE typTeléfonos AS TABLE OF typTeléfono;
/

CREATE OR REPLACE TYPE ClsEmpleados AS OBJECT (
  ID        NUMBER(5),
  Nombre    VARCHAR2(20),
  Apellidos VARCHAR2(20),
  Teléfonos typTeléfonos,
);
/

CREATE TABLE Empleados OF ClsEmpleados NESTED TABLE Teléfonos STORE AS tabTeléfonos;

SELECT Teléfono
FROM THE(SELECT Teléfonos FROM Estudiantes WHERE Nombre='Pedro');
```

5) (1,7 puntos) Considérese la siguiente instancia de la relación Cuentas:

Cuenta	***Sucursal***	***Saldo***
C-217	Galapagar	750
C-101	Centro	500
C-110	Centro	600
C-215	Becerril	700
C-102	Navacerrada	400
C-201	Navacerrada	900
C-218	Navacerrada	700
C-222	Moralzarzal	700
C-305	ColladoMediano	350

a) (0,7 puntos) Constrúyase un índice escaso de mapa de bits sobre los atributos Sucursal y Saldo, separando los valores de saldo en cuatro rangos: menores que 250, entre 250 y menores que 500 entre 500 y menores que 750, y 750 o mayores.

Solución:

Mapa de bits para Sucursal:

Galapagar	1 0 0 0 0 0 0 0 0
Centro	0 1 1 0 0 0 0 0 0
Becerril	0 0 0 1 0 0 0 0 0
Navacerrada	0 0 0 0 1 1 1 0 0
Moralzarzal	0 0 0 0 0 0 0 1 0
Collado Mediano	0 0 0 0 0 0 0 0 1

Mapa de bits para Saldo:

L1	0 0 0 0 0 0 0 0 0
L2	0 0 0 0 1 0 0 0 1
L3	0 1 1 1 0 0 1 1 0
L4	1 0 0 0 0 1 0 0 0

donde L1 está por debajo de 250, L2 desde 250 hasta debajo de 500, L3 desde 500 hasta antes que 750 y nivel L4 desde 750.

b) (1 punto) Considérese una consulta que solicite todas las cuentas en Centro con un saldo de 500 o más. Muéstrense los pasos implicados en la consulta así como los mapas de bits intermedios y finales construidos para responder la consulta.

Solución:

Para encontrar todas las cuentas de Centro con saldo de 500 o más, se calcula la unión de mapas de bits para los niveles L3 y L4 y, después, se intersecta con el mapa de bits para Centro.

Centro	0 1 1 0 0 0 0 0 0
L3	0 1 1 1 0 0 1 1 0
L4	1 0 0 0 0 1 0 0 0
L3 $\cup$ L4	1 1 1 1 0 1 1 1 0
Centro $\cap$ (L3 $\cup$ L4)	0 1 1 0 0 0 0 0 0

De este modo, las tuplas requeridas son C-101 y C-110.