Examen de Bases de datos y sistemas de información
Convocatoria de junio
I PARCIAL

Nota: Es obligatoria la realización del primer parcial para los estudiantes que no hayan superado un 4 en la convocatoria de Febrero.

1) (2.25 puntos) Dados los esquemas de relación
Empleados(NumEmpl, Nombre, Apellido, Salario, Teléfono, DNI, IdDirección)
Direcciones(IdDir, Calle, Número, Piso, Código postal, Ciudad)
Se pide:
   a) (0.5 puntos) Listar los nombres y apellidos de los empleados de la calle Alcalá de Madrid que tengan salario nulo.
   b) (0.5 puntos) Listar todos los empleados que compartan dirección con otro empleado
   c) (0.5 puntos) Listar los códigos postales con más de tres empleados
   d) (0.75 puntos) Subir un 10% el salario de los empleados que ganen menos que la media de salarios de alguna ciudad que empiece por M y tenga 5 letras.

Solución:

a)

```
SELECT Nombre, Apellido
FROM Direcciones, Empleados
WHERE IdDirección=IdDir
 AND Calle = 'Alcala'
 AND Ciudad='Madrid'
 AND Salario IS NULL
```

Otra opción:

```
SELECT Nombre, Apellido
FROM Empleados E
WHERE Salario IS NULL
  AND EXISTS (SELECT '1'
              FROM Direcciones
              WHERE IdDir= E.IdDirección
                AND Calle = 'Alcala'
                AND Ciudad='Madrid')
```

b)

```
SELECT Nombre,Apellido
FROM Empleados E
WHERE IdDirección in (SELECT  idDirección
                      FROM Empleados
                      GROUP BY idDirección
                      HAVING COUNT(*)>1)
```

Otra Opción

```
SELECT Nombre,Apellido
FROM Empleados E
WHERE EXISTS (SELECT '1'
              FROM Direcciones,Empleados
              WHERE Iddirección=IdDir
                AND NumEmpl<>E.NumEmpl
                AND IdDirección=E.IdDirección)
```

c)

```
SELECT "Código postal", COUNT(*)
FROM Direcciones, Empleados
WHERE Iddirección=IdDir
```

```
GROUP BY "Código postal"
HAVING COUNT(*)>3
```

d)

```
UPDATE Empleados E
SET Salario = Salario* 1.1
  WHERE Salario  < ANY (SELECT AVG(Salario)
                        FROM Direcciones,Empleados
                        WHERE Iddirección=IdDir
                          AND Ciudad LIKE 'M____'
                        GROUP BY Ciudad)
```

2) (1.75 puntos) Dar el DDL para crear ambas tablas del ejercicio 1, definiendo:
- (0.25 puntos) Las claves primarias
- (0.25 puntos) Una regla de integridad referencial
- (0.25 puntos) Una restricción de dominio que imponga que el salario sea positivo
- (0.25 puntos) Sólo se permite valores nulos en los campos salario, teléfono, número y piso.
- (0.25 puntos) No pueden haber dos direcciones con la misma calle, número, piso, código postal y ciudad.

(0.5 puntos) Insertar en el orden adecuado una fila en cada tabla

Solución:

Creación de las tablas con las restricciones pedidas

```
CREATE TABLE Direcciones(
  IdDir INTEGER PRIMARY KEY,
  Calle CHAR(30) NOT NULL,
  Número CHAR(4),
  Piso CHAR(4),
  "Código postal" CHAR(5) NOT NULL,
  Ciudad CHAR(30),
  UNIQUE (Calle, Número, Piso, "Código postal", Ciudad)
) ;

CREATE TABLE Empleados(
  NumEmpl INTEGER PRIMARY KEY,
  Nombre CHAR(20) NOT NULL,
  Apellido CHAR(30) NOT NULL,
  Salario NUMBER(7,2),
  Teléfono CHAR(9),
  DNI CHAR(9) NOT NULL,
  IdDirección INTEGER NOT NULL REFERENCES Direcciones,
  CHECK (Salario >0)
)
```

Hay que insertar filas primero en Direcciones y luego en Empleados, pues Empleados es dependiente de Direcciones por la restricción de integridad referencial definida.
INSERT INTO Direcciones VALUES (10, 'Alcalá', 20, 2, 28030, 'Madrid')
INSERT INTO Empleados VALUES (2, 'Luis', 'García', NULL, NULL, 234567, 10)

3) (2 puntos) Mostar la consulta a) del ejercicio 1) con consultas del álgebra relacional, calculo relacional de tuplas, cálculo relacional de dominios y QBE

Solución:

Álgebra relacional:

EmplSalNulo ← πNombre, IdDirección(σ(Salario IS NULL)(Empleados))

Salida ← πNombre (EmplSalNulo $\bowtie$ Direcciones)

Iddirección = IdDir
AND Calle = 'Alcalá'
AND Ciudad = 'Madrid'

Cálculo relacional de tuplas

{t | ∃u(
((u ∈ Empleados)
∧
(t[Nombre] = u[Nombre])
∧
(t[IdDirección] = u[IdDirección])
∧
(t[Salario] IS NULL)
∧
(∃v∈Direcciones (v[IdDir] = u[IdDirección)
∧ v[Calle] = 'Alcalá'
∧ v[Ciudad] = Madrid'
}

Cálculo relacional de dominios:

{<Nombre> |
∃ Salario (<Nombre, Salario, IDdir> ∈ Empleados
∧ <Salario> = NULL)
∧
(∃ f (<IDdir, Calle, Ciudad>∈Direcciones
∧ <Calle> = 'Alcalá'
∧ <Ciudad> = 'Madrid')
}

QBE:

Direcciones	*IdDir*	*Calle*	*Ciudad*
	_x	Alcala	Madrid

Empleados	*Nombre*	*IdDirección*	*Salario*
	P._y	*_x*	NULL

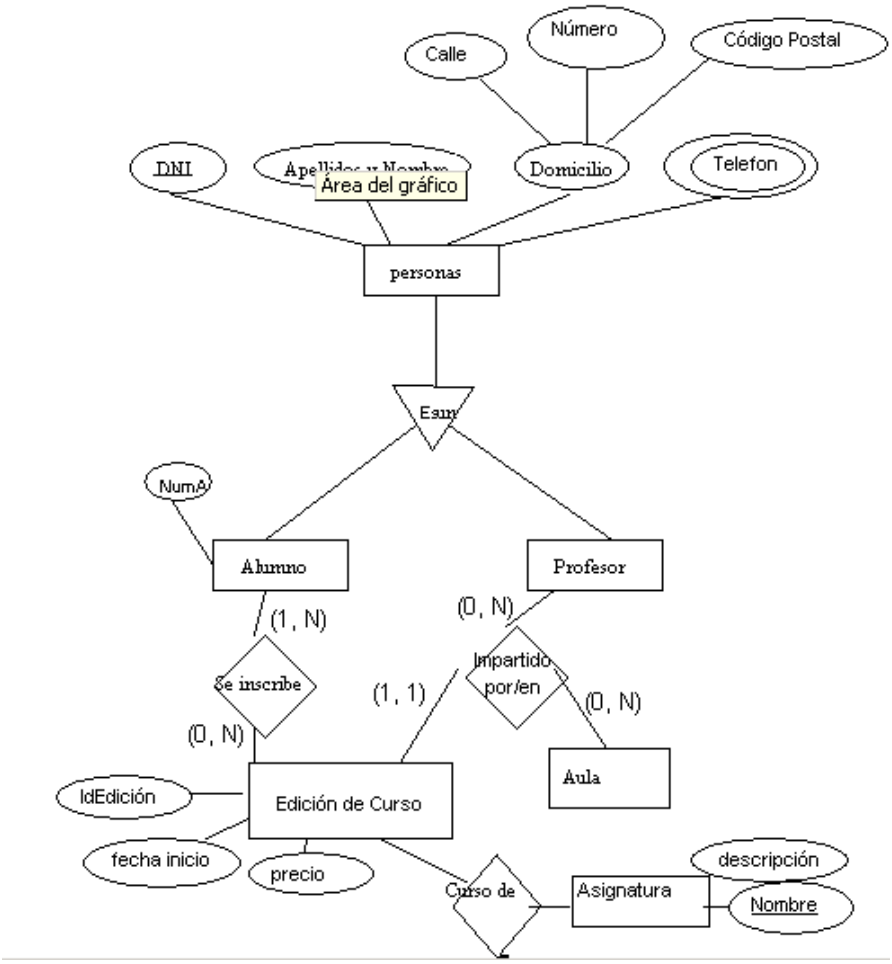
4) (2 puntos) Una academia de formación oferta cursos de asignaturas de distintas carreras de varias universidades. Necesita un sistema de información para conocer el DNI, apellidos y nombre, teléfonos y domicilio de sus alumnos y profesores. Debe saberse las asignaturas en las que se inscribe cada alumno, y el profesor y aula de cada curso. Evitar entidades débiles y usar generalizaciones donde sea posible.

a) Construir un modelo EER para la modelar los datos de la academia, incluyendo las relaciones de cardinalidad mínimo-máximo.

b) Traducir a modelo relacional.

Solución:

a)



b)

Personas(DNI, TipoA/P, ApellidosYNombre)
Teléfonos(DNI, Número)
Domicilio(Calle, Número, CódigoPostal, DNI)
Ediciones(IdEdición, FechaInicio, Precio, NombreAsignatura, DNIProfesor, Aula)
Asignaturas(Nombre, Descripción)
SeInscribe(DNIAlumno, IdEdición)

5) (1 punto) En la academia del ejercicio 4 se ha propuesto incluir un total de alumnos en la relación de Ediciones. Dar una opinión razonada sobre la redundancia sabiendo que en media:
   a. Hay 20 alumnos en cada curso
   b. Se realizan 30 inscripciones o cancelaciones a cursos cada día
   c. Se consulta el número total de alumnos de una edición una vez al día.

Solución:

Se asume que no se cuentan las comprobaciones de integridad referencial y que los alumnos y ediciones ya están dados de alta cuando se formaliza la inscripción.

Sin redundancia

30 inscripciones: 30 escrituras en SeInscribe

Consulta: Una lectura en Ediciones y 20 en SeInscribe

Coste total: 30*2 + 1+ 20 = 81

Con redundancia:

30 inscripciones: 30 escrituras en SeInscribe + 30 lecturas en Ediciones + 30 escrituras en Ediciones (para sumar o restar en contador de alumnos en ediciones, se lee y modifica)

Consulta: Una lectura en Ediciones

Total Coste: 30*2 + 30 + 30*2 + 1 = 151

Preferimos un diseño sin redundancia

6) (1 punto) Determínese si los conjuntos F y G de dependencias funcionales son equivalentes
F = { E $\rightarrow$ C, EC $\rightarrow$ D, A $\rightarrow$ ED, A $\rightarrow$ B}
G = { E $\rightarrow$ CD, A $\rightarrow$ EB}

Solución:
Para demostrar que son equivalentes, se puede ver que toda X$\rightarrow$Y $\in$ F verifica X$\rightarrow$Y $\in$ G+ y que toda X$\rightarrow$Y $\in$ G verifica X$\rightarrow$Y $\in$ F+

Primero con las de F en G+
- E $\rightarrow$ C $\in$ G+, pues en G , {E}+ = {E, C, D}
- EC $\rightarrow$ D $\in$ G+, pues en G , {E, C}+ = {E, C, D}
- A $\rightarrow$ ED $\in$ G+, pues en G , {A}+ = {A, E, B, C, D}
- A $\rightarrow$ B $\in$ G+, pues en G , {A}+ = {A, E, B, C, D}

Ahora al revés:
- E $\rightarrow$ CD $\in$ F+, pues en F, {E}+ = {E, C, D}
- A $\rightarrow$ EB $\in$ F+, pues en F, {A}+ = {A, D, B, E, C}

Por lo tanto sí son equivalentes.

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

1) (2 puntos) Dada la siguiente relación, que muestra los directores de cada departamento (cada departamento tiene un solo director y no hay directores diferentes con el mismo nombre) y su dirección, y que están identificados por su nombre y apellidos:

Departamento	Apellidos	Nombre	Dirección
Ventas	García	Paco	c/ Mayor, 1
Compras	García	Paco	c/ Mayor, 1
Contabilidad	Sánchez	Pedro	Avda. España, 4
Personal	García	Juan	Avda. España, 4

Se pide:

a. Determinar todas las dependencias funcionales.

Solución:

La clave primaria es Departamento dado que sólo hay un director por departamento y cada director tiene una sola dirección. No hay claves candidatas. Además de las DF deducidas de la clave primaria, se verifica: Apellidos, Nombre $\rightarrow$ Dirección. El resto de las DF son las triviales.

Determinar cuáles de las siguientes descomposiciones son sin pérdida (lossless) en la instancia dada y en general, así como su forma normal.

b. r1(Departamento, Apellidos), r2(Apellidos, Nombre, Dirección)

Solución:

No es sin pérdida. Apellidos no identifica a una persona y, por tanto, la reunión asociará a un departamento todas las personas con los mismos apellidos. Con esta instancia se obtendría:

Departamento	Apellidos
Ventas	García
Compras	García
Contabilidad	Sánchez
Personal	García

Apellidos	Nombre	Dirección
García	Paco	c/ Mayor, 1
Sánchez	Pedro	Avda. España, 4
García	Juan	Avda. España, 4

Departamento	Apellidos	Nombre	Dirección
Ventas	García	Paco	c/ Mayor, 1
Ventas	García	Juan	Avda. España, 4
Compras	García	Paco	c/ Mayor, 1
Compras	García	Juan	Avda. España, 4
Contabilidad	Sánchez	Pedro	Avda. España, 4
Personal	García	Paco	c/ Mayor, 1
Personal	García	Juan	Avda. España, 4

En el caso general se comprueba fácilmente que es con pérdida porque la dependencia funcional Departamento -> Nombre deducida de la clave primaria no se conserva.
r1 se encuentra en FNBC porque sólo tiene dos atributos.

r2 se encuentra en FNBC porque en la única dependencia funcional que se conserva en la relación (Apellidos, Nombre -> Dirección) el antecedente es clave primaria.

c. r1(Departamento, Apellidos, Nombre), r2(Nombre, Dirección)

Solución:

Esta descomposición es sin pérdida en esta instancia particular de la base de datos porque no hay dos personas con el mismo nombre, así que la reunión de las dos relaciones da de nuevo la relación original. Sin embargo, Nombre no identifica en general a una persona y, por lo tanto, la reunión podría dar lugar a información incorrecta. La dependencia funcional Nombre, Apellidos -> Dirección no se conserva.
r1 se encuentra en FNBC porque en la única dependencia funcional que se conserva en la relación (Departamento -> Nombre, Apellidos) el antecedente es clave primaria.
r2 se encuentra en FNBC porque sólo tiene dos atributos.

Departamento	Apellidos	Nombre
Ventas	García	Paco
Compras	García	Paco
Contabilidad	Sánchez	Pedro
Personal	García	Juan

Nombre	Dirección
Paco	c/ Mayor, 1
Pedro	Avda. España, 4
Juan	Avda. España, 4

d. r1(Departamento, Apellidos, Nombre), r2(Apellidos, Nombre, Dirección)

Solución:

Esta reunión es siempre sin pérdida porque los atributos Apellidos y Nombre se encuentran en ambas relaciones, y en la segunda relación aparecen como clave. Esta descomposición está en FNBC.

Departamento	Apellidos	Nombre
Ventas	García	Paco
Compras	García	Paco
Contabilidad	Sánchez	Pedro
Personal	García	Juan

Apellidos	Nombre	Dirección
García	Paco	c/ Mayor, 1
Sánchez	Pedro	Avda. España, 4
García	Juan	Avda. España, 4

2) (1,5 puntos) Dado el siguiente esquema relacional en Oracle.
Empleados(DNI, Sueldo base, Retención IRPF, Retención SS, Sueldo neto)
PorcentajeIRPF(ID, Porcentaje)
Corresponde(DNI, ID).
Se pide programar un disparador tal que al insertar una nueva tupla en PorcentajeIRPF o se modifique el porcentaje de IRPF se calculen automáticamente "Retención IRPF" (igual a "Sueldo base" * "Porcentaje" / 100) y "Sueldo neto" (igual a "Sueldo base" - "Retención IRPF" – "Retención SS").

Solución:

Se crean las tablas con las restricciones de clave e integridad referencial:

```
CREATE TABLE PorcentajeIRPF(ID INTEGER PRIMARY KEY, Porcentaje NUMBER(5,2));
CREATE TABLE Empleados(DNI VARCHAR2(9) PRIMARY KEY, "Sueldo base" NUMBER(8,2),
"Retención IRPF" NUMBER(8,2), "Retención SS" NUMBER(8,2), "Sueldo neto" NUMBER(8,2));
CREATE TABLE Corresponde(DNI VARCHAR2(9) PRIMARY KEY REFERENCES Empleados DEFERRABLE, ID
INTEGER REFERENCES PorcentajeIRPF DEFERRABLE);
```

Las restricciones de integridad referencial de Corresponde se definen diferibles por el motivo que se explicará cuando se inserten los primeros datos.
Se crea el disparador para actualizar los campos "Retención IRPF" y "Sueldo neto" que se activa después de la actualización o inserción de tuplas en la tabla PorcentajeIRPF:

```
CREATE OR REPLACE TRIGGER CálculoSueldo
AFTER UPDATE OR INSERT ON PorcentajeIRPF
BEGIN
    UPDATE Empleados E
    SET "Retención IRPF"= "Sueldo base" *
      (SELECT PorcentajeIRPF.Porcentaje
       FROM PorcentajeIRPF, Corresponde
       WHERE PorcentajeIRPF.ID = Corresponde.ID AND
             Corresponde.DNI=E.DNI)
      / 100;
    UPDATE Empleados
    SET "Sueldo neto"= "Sueldo base" - "Retención IRPF" - "Retención SS";
END;
/
```

Este disparador no es precisamente eficaz porque recorre toda la tabla Empleados sin centrarse sólo en los cambios necesarios.
Con otra alternativa, más sencilla y que recorre sólo las filas afectadas, el que el disparador se activa por cada fila (FOR EACH ROW) y, por tanto, podemos acceder a :NEW y :OLD. Además, sólo se emite una instrucción UPDATE:

```
CREATE OR REPLACE TRIGGER CálculoSueldo
AFTER UPDATE OR INSERT ON PorcentajeIRPF
FOR EACH ROW
BEGIN
    UPDATE Empleados E
    SET "Retención IRPF"= "Sueldo base" * :NEW.Porcentaje / 100,
        "Sueldo neto"= "Sueldo base" * (1 - :NEW.Porcentaje / 100) - "Retención SS"
    WHERE :NEW.ID=(SELECT ID FROM Corresponde WHERE Corresponde.DNI=E.DNI);
END;
/
```

A continuación se insertan los datos iniciales con objetivo de que se actualicen automática y correctamente los campos "Retención IRPF" y "Sueldo neto". La tabla PorcentajeIRPF se debe rellenar al final con objeto de que cuando se active el disparador encuentre datos tanto en Empleados como en Corresponde. Esto obliga a que las restricciones de integridad referencial no se puedan comprobar en el momento de las inserciones en la tabla Corresponde, puesto que no hay registros asociados aún en la tabla PorcentajeIRPF, con la que mantiene una restricción de integridad referencial en el campo ID. Por lo tanto es necesario diferir su comprobación. El resultado:

```
SET AUTOCOMMIT OFF;
SET CONSTRAINTS ALL DEFERRED;
DECLARE
BEGIN
  INSERT INTO Empleados VALUES ('1', 30000.00, 0.00, 3000.00, 0.00);
  INSERT INTO Empleados VALUES ('2', 40000.00, 0.00, 4000.00, 0.00);
```

```
  INSERT INTO Empleados VALUES ('3', 25000.00, 0.00, 2500.00, 0.00);

  INSERT INTO Corresponde VALUES ('1',1);
  INSERT INTO Corresponde VALUES ('2',2);
  INSERT INTO Corresponde VALUES ('3',1);

  INSERT INTO PorcentajeIRPF VALUES(1,20.00);
  INSERT INTO PorcentajeIRPF VALUES(2,30.00);

  COMMIT;
END;
/

Procedimiento PL/SQL terminado correctamente.

SELECT * FROM Empleados;

DNI       Sueldo base Retención IRPF Retención SS Sueldo neto
--------- ----------- -------------- ------------ -----------
1               30000           6000         3000       21000
2               40000          12000         4000       24000
3               25000           5000         2500       17500

SET AUTOCOMMIT ON;

UPDATE PorcentajeIRPF SET Porcentaje=10.00 WHERE ID=1;

SELECT * FROM Empleados;

DNI       Sueldo base Retención IRPF Retención SS Sueldo neto
--------- ----------- -------------- ------------ -----------
1               30000           3000         3000       24000
2               40000          12000         4000       24000
3               25000           2500         2500       20000
```

Para volver al estado inicial:

```
DELETE FROM Corresponde;
DELETE FROM PorcentajeIRPF;
DELETE FROM Empleados;

DROP TRIGGER CálculoSueldo;

DROP TABLE Corresponde;
DROP TABLE PorcentajeIRPF;
DROP TABLE Empleados;
```

3) (1 punto) Constrúyase un árbol B+ insertando valores en el orden de la secuencia (2, 3, 5, 7, 11, 17, 19, 23, 29, 31, 35, 20) cuando caben 6 punteros en un nodo. Supóngase que el árbol está inicialmente vacío. Muéstrese al menos cada paso en la construcción del árbol en el que se añadan o eliminen nodos.

Solución:

Restricciones:
Nodo raíz: entre 1 y 6 hijos (1 y n).
Nodo hoja: entre 3 y 5 valores ($\lceil (n-1)/2 \rceil$ y n-1).
Nodos internos: entre 3 y 6 hijos ($\lceil n/2 \rceil$ y n).

4) (1,5 puntos) Dadas las relaciones r1(A ,B, C) y r2(C, D, E) con las siguientes propiedades: r1 tiene 20.000 tuplas, r2 tiene 45.000 tuplas, 25 tuplas de r1 caben en un bloque y 30 tuplas de r2 caben en un bloque. Estímese el número de accesos a bloques requeridos, utilizando cada una de las siguientes estrategias para la reunión de r1 y r2:
a. (0,75 puntos) Reunión en bucle anidado

Solución:

r1 necesita $\left\lceil \frac{20.000}{25} \right\rceil = 800$ bloques y r2 $\left\lceil \frac{45.000}{30} \right\rceil = 1.500$. Supónganse M páginas de memoria. Si M >800, la reunión puede hacerse fácilmente en 1.500 + 800 accesos a disco, empleando incluso reunión en bucle anidado sencilla. Así, consideraremos sólo el caso en que M ≤ 800 páginas.
Empleando r1 como la relación externa se necesitan 20.000 * 1.500+800 = 30.000.800 accesos a disco, si r2 es la relación externa son necesarios 45.000 * 800 + 1.500 = 36.001.500 accesos a disco.

b. (0,75 puntos) Reunión en bucle anidado por bloques

Solución:

Si r1 es la relación externa se necesitan ⌈800 / (M-1)⌉ * 1.500 + 800 accesos a disco, si r2 es la relación externa son necesarios ⌈1.500 / (M-1) ⌉ * 800 + 1.500 accesos a disco.

5) (2 puntos) Considérese un fichero secuencial indexado denso con asignación enlazada con los campos A:Byte(2), B:String(10), C:String(20), cuya clave es el campo A y con mapas de bits en el fichero de datos y de índice. El formato de los caracteres es UNICODE y las cadenas de caracteres son de longitud variable y con terminador. El tamaño de bloque es de 1.024 bytes, 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 128 sectores por pista y las escrituras necesitan el mismo tiempo que las lecturas. Se pide:
a. (1 punto) Dibujar la estructura del fichero de datos y calcular el espacio desperdiciado en un bloque de datos en el mejor y peor de los casos.

Solución:

- El mejor caso es con ocupación máxima del bloque. Para calcular el factor de bloqueo N (número de registros por bloque):

N*(2+10*2+2+20*2+2)*8+4*8+N ≤ 1.024*8, donde:
N*(2+10*2+2+20*2+2)*8=N*528 es el tamaño en bits ocupado por un registro (2 bytes para el campo A, 10*2+2 para el campo B, dado que los caracteres ocupan dos bytes por su representación UNICODE y la cadena necesita un terminador (también 2 bytes), y 20*2+2 para el campo C por las mismas razones.
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
1.024*8=8.192 es el número de bits en los 1.024 bytes de un bloque
N*(528+1)≤8.192-32
N≤15,4
Se toma N=15 registros por bloque. Por lo tanto, el espacio desperdiciado es:
8192-15*(2+10*2+2+20*2+2)*8-32-15=225 bits, es decir, un poco más de 28 bytes (28,125 exactamente, 28 bytes y 1 bit)

- El peor caso es con ocupación mínima del bloque, es decir, con sólo un registro:

8192-(2+10*2+2+20*2+2)*8-32-15=7617 bits, es decir, un poco más de 952 bytes (952,125 exactamente, 952 bytes y 1 bit)

b. (1 punto) Dibujar la estructura del fichero de índice y, considerando sólo el tiempo debido a operaciones de E/S de bloques de disco, calcular el tiempo necesario para insertar un registro con un valor nuevo de la clave que ocupe la última posición en el archivo de datos según la clave cuando el fichero de datos contiene 10.005 registros con valores de la clave de 0 a 64K-2 y no ha habido desbordamiento. Se asume una asignación enlazada de bloques libres.

Solución:

Estructura del fichero de índices: registros con dos campos, uno para el valor de la clave (2 bytes) y otro para la dirección de bloque (4 bytes).
N*(2+4)*8+4*8+N ≤ 1.024*8
N*(48+1)≤8.192-32
N≤166,53
Se toma N=166 registros por bloque.
En cada bloque caben 166 registros. Por lo tanto, el fichero de índices contiene $\lceil 10.005/166 \rceil$ bloques, es decir, $\lceil 60{,}27 \rceil$=61 (y quedan 45 bytes libres en el último bloque, para 7 registros más, por lo que no es necesario reclamar un nuevo bloque para el fichero de índices).

Es necesaria una búsqueda secuencial en el fichero índice para determinar el lugar que corresponde al nuevo registro. En total son necesarias 61 lecturas en el fichero de índices y 1 escritura para actualizar el último bloque.

En el fichero de datos se escribe un registro en la última posición, que hay que ver si corresponde al último bloque. Como caben 15 registros por bloque, hay ocupados $\lceil 10.005/15 \rceil$ bloques, es decir, exactamente 667 bloques. Por lo tanto, hay que reclamar un nuevo bloque, lo que supone una lectura de bloque en la lista de bloques libres para actualizar el puntero inicial (en memoria) de la cadena de bloques libres. Para enlazar el nuevo bloque reclamado es necesaria 1 lectura del último bloque y su escritura, más la escritura del nuevo bloque. En total son necesarias 2 lecturas y 2 escrituras.

Suponiendo una distribución aleatoria de los bloques en disco, el tiempo medio de lectura o de escritura 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= $\frac{1}{128}$ *Tiempo de rotación(el doble del tiempo de latencia)=0,07ms
Tiempo medio operación E/S=10+4,17+0,07=14,24 ms

El total del tiempo para la inserción del nuevo registro = (61+1+2+2)*14,24=939,84 ms, es decir, casi un segundo.

6) (2 puntos) Dadas las tablas t(A,B) y s(B,C), se pide:
   a. (1,25 puntos) Programar un procedimiento almacenado en PL/SQL que borre las filas de t que no cumplan la restricción de integridad referencial $\pi_B(t) \subseteq \pi_B(s)$, y las inserte en la tabla u(A,B).
   b. (0,75 puntos) ¿Qué podría ocurrir si el procedimiento se aborta durante su ejecución? Considérense los posibles modos de autocompromiso.

Solución:

Se crean las tres tablas:

```
CREATE TABLE t(A INTEGER PRIMARY KEY, B INTEGER);
CREATE TABLE s(B INTEGER PRIMARY KEY, C INTEGER);
CREATE TABLE u(A INTEGER, B INTEGER);
```

Se insertan datos en ellas con algunas filas que no cumplan la restricción de integridad referencial:

```
INSERT INTO s VALUES (0,0);
INSERT INTO s VALUES (3,0);

INSERT INTO t VALUES (0,0);
INSERT INTO t VALUES (1,1);
INSERT INTO t VALUES (2,2);
INSERT INTO t VALUES (3,3);

SELECT * FROM t;

         A          B
---------- ----------
         0          0
         1          1
         2          2
         3          3

SELECT * FROM s;

         B          C
---------- ----------
         0          0
         3          0
```

Se escribe el procedimiento:

```
CREATE OR REPLACE PROCEDURE IRst AS
BEGIN
  INSERT INTO u
   SELECT *
    FROM t
    WHERE B NOT IN
     (SELECT B
      FROM s);
  DELETE
  FROM t
  WHERE B NOT IN
   (SELECT B
    FROM s);
END;
/
```

Con la instrucción INSERT se seleccionan *todas* (por ello no se usa DISTINCT en la SELECT exterior) las tuplas de t que no encuentren correspondencia en s. Como B es clave en S, tampoco se usa DISTINCT en la SELECT interior.
Hay que evitar siempre el uso de cursores para conseguir el máximo rendimiento, y en este caso es claramente posible.
Al ejecutar el procedimiento:

```
EXEC IRst;

SELECT * FROM u;
```

```
         A          B
---------- ----------
         1          1
         2          2

SELECT * FROM t;

         A          B
---------- ----------
         0          0
         3          3
```

Si el modo de autocompromiso está activado, cada instrucción SQL se valida inmediatamente después de su ejecución, por lo que si se aborta el procedimiento entre el INSERT y el DELETE, las tablas quedan inconsistentes, puesto que u contendría tuplas que no se han borrado de t.
Si el modo de autocompromiso está desactivado, es necesario comprometer los cambios del procedimiento al final con la instrucción COMMIT. Esto asegura que las modificaciones del procedimiento se realizarán de forma atómica y si ocurre un fallo entre el INSERT y el DELETE o en el propio DELETE, los cambios del INSERT no se validarán, y las tablas permanecerán consistentes.