Base de Datos (75.15 / 75.28 / 95.05)

Evaluación Integradora - 20 de febrero de 2019

TEMA 20182C4					
CRT		**CyT**		**DR**	
Proc.		**Rec.**		**NoSQL**	
Corrigió: **Nota:**					

Padrón: _____

Apellido: _____

Nombre: _____

Cantidad de hojas: _____

☐ Aprobado ☐ Insuficiente

Criterio de aprobación: El examen está compuesto por 6 ítems, cada uno de los cuales se corrige como B/B-/Reg/Reg-/M. Se aprueba con nota mayor o igual a 4(cuatro), equivalente a desarrollar el 60 % del examen correctamente.

1. *(Cálculo Relacional de Tuplas)* Buenos Aires es un horno. Esta noche Samuel invitó a algunos amigos a su casa para disfrutar de la piscina y refrescarse con algunos *cocktails*. Para prepararse descargó de Internet unas tablas que detallan la composición de 1000 *cocktails* distintos, indicando el nombre de cada *cocktail* y los ingredientes que lo componen. A su vez, agregó a la tabla de ingredientes un atributo *booleano* que indica si el ingrediente se encuentra en su alacena ó no:

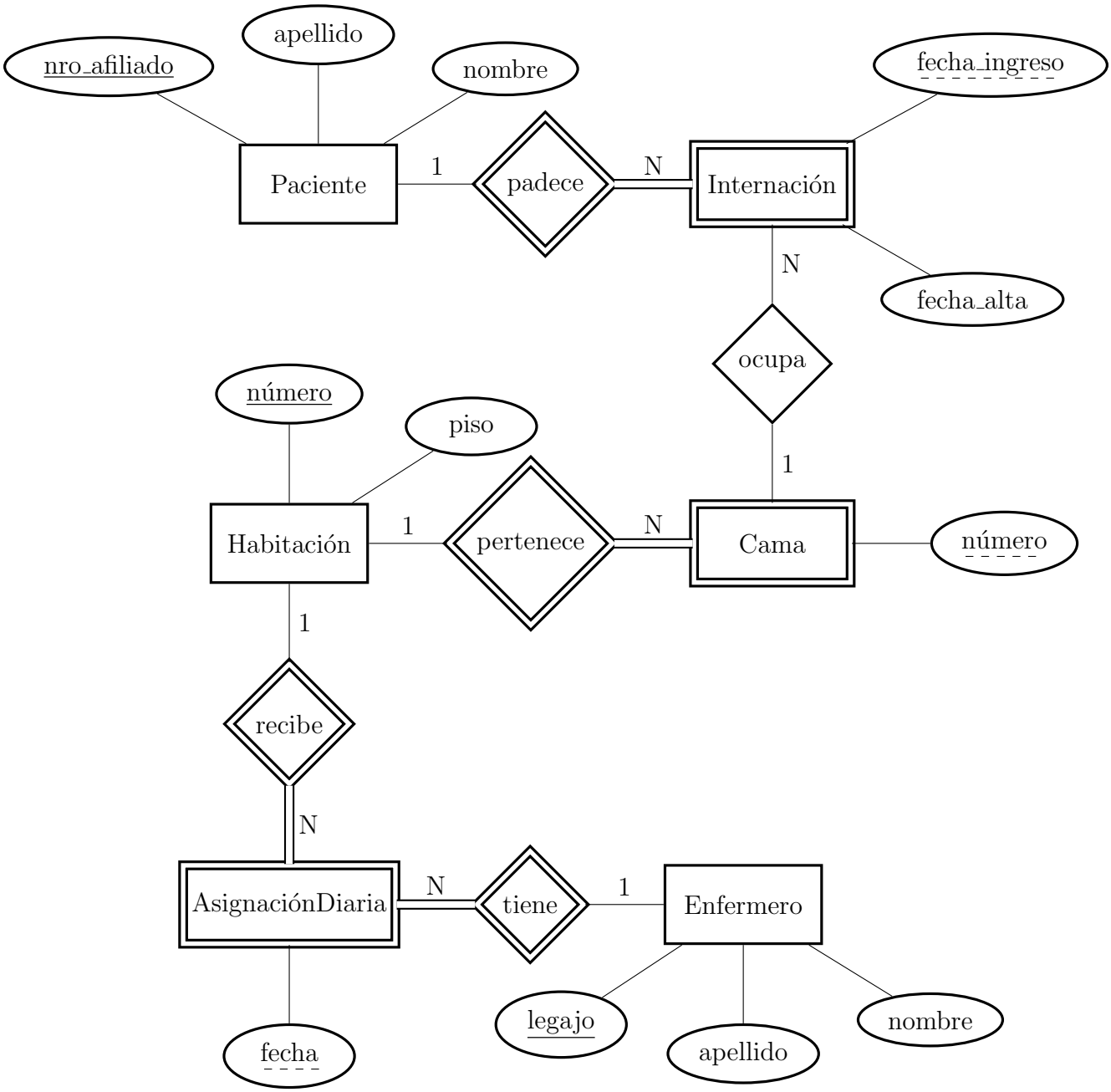
   - `Cocktails(nombre_cocktail, tamaño_ml, instructivo)`
   - `Ingredientes(nombre_ingrediente, descripción, en_alacena)`
   - `Composiciones(nombre_cocktail, nombre_ingrediente, cantidad)`

   Escriba una consulta en *Cálculo Relacional de Tuplas* que devuelva los nombres de aquellos *cocktails* que pueden ser preparados con los ingredientes que Samuel tiene en la alacena.

   Nota: Para los ingredientes que sí se encuentran en la alacena, suponga que Samuel tiene cantidad suficiente como para preparar cualquier cantidad de *cocktails*.

2. *(Concurrencia y Transacciones)* Enuncie el protocolo de *lock* de dos fases (2PL, *Two-phase lock*). Indique si un SGBD que lo utilice garantizará la serializabilidad y/o la recuperabilidad de las transacciones que ejecuta.

3. *(Diseño relacional)* El siguiente diagrama Entidad-Interrelación captura las distintas internaciones de los pacientes de una clínica, las camas en que son ubicados, las habitaciones existentes y la asignación diaria de personal de enfermería a las habitaciones:



Se pide:

a) Proponga un conjunto de dependencias funcionales F sobre la relación universal `R(nroAfiliado, apellAfiliado, nombreAfiliado, fechaIngreso, fechaAlta, numeroCama, numeroHabitacion, piso, fecha, legajoEnf, apellEnf, nombreEnf)`, que capture todas las dependencias funcionales que puedan deducirse del diagrama.

b) Traduzca el diagrama anterior a un *esquema de base de datos relacional*. Señale la clave primaria de cada relación y las claves foráneas existentes.

4. *(Procesamiento de Consultas)* Un popular servicio de streaming de películas bajo demanda está construyendo su sistema de recomendaciones, y quiere calcular para cada par de películas (p_1, p_2) la cantidad de usuarios que han visto ambas. Para calcular estas co-ocurrencias utilizará la siguiente tabla que indica qué usuarios vieron cada película:

   - `VisionesUsuarios(nro_usuario, cod_película)`

   El objetivo inicial es calcular la siguiente junta:

   $$VisionesUsuarios \bowtie_{1.nro\_usuario=2.nro\_usuario} VisionesUsuarios$$

   La tabla *VisionesUsuarios* tiene un tamaño de `B(VisionesUsuarios)=10.000.000`, y la cantidad de memoria disponible para realizar la operación es de `M=1001` bloques. Por último, la cantidad de usuarios distintos en la tabla es de `V(nro_usuario, VisionesUsuarios) = 10.000.000`.

   La compañía ha probado diversas técnicas para realizar la junta de forma eficiente por el método de *junta hash GRACE*, pero no ha logrado resolverla. A continuación le mostramos como referencia un típico pseudocódigo de una *junta hash GRACE* convencional:

Algoritmo 1: `HashearTabla(T, a, h, MAX)`

Entrada : Una tabla T en disco, un atributo de junta a, una función de hash h y un valor MAX que es el máximo valor devuelto por la función de hash.
Salida : Una partición de T en subtablas $T_1, T_2, ..., T_{MAX}$ en disco.

```
Alocar MAX + 1 bloques M_0, ..., M_MAX vacíos en memoria
Crear MAX subtablas vacías T_1, ..., T_MAX en disco
Para (cada bloque de T) {
    Cargar bloque en M_0
    Para (cada tupla t en M_0) {
        H = h(t.a)
        Agregar t al bloque de memoria M_H
        Si (M_H está lleno) {
            Enviar M_H a la subtabla T_H en disco y vaciar M_H
        }
    }
}
Liberar los MAX + 1 bloques de memoria
Devolver direcciones de T_1, T_2, ..., T_MAX en disco
```

Algoritmo 2: `JuntaHashGRACE(T1, T2, a, h, MAX)`

Entrada : Dos tablas T_1 y T_2 en disco, un atributo de junta a, una función de hash h y un valor MAX que es el máximo valor devuelto por la función de hash.
Salida : El resultado de la junta de T_1 y T_2 por el atributo a en disco.

```
Crear tabla resultado R vacía en disco
T1_1, ..., T1_MAX = HashearTabla(T1, a, h, MAX)
T2_1, ..., T2_MAX = HashearTabla(T2, a, h, MAX)
Para (cada i ∈ {1..MAX}) {
    Cargar T1_i entera en memoria        /* Requiere B(T1_i) bloques disponibles */
    Cargar T2_i entera en memoria        /* Requiere B(T2_i) bloques disponibles */
    Agregar T1_i ⋈_{1.a=2.a} T2_i a R    /* Requiere 1 bloque para armar resultado */
    Liberar memoria ocupada por T1_i y T2_i
}
Devolver dirección de R en disco
```

a) En primer lugar alguien propuso utilizar una función de hash $h(nro\_usuario)$ que devolviera un valor entre 1 y 10000, pero ésto no funcionó. Explique cuál fue el problema de este enfoque.

b) Luego alguien propuso utilizar una función de hash $h(nro\_usuario)$ que devolviera un valor entre 1 y 1000, pero tampoco funcionó. Explique cuál fue el problema ahora.

c) Finalmente alguien propuso modificar la *junta hash GRACE* utilizando dos funciones de hash *distintas*, $h_1(nro\_usuario)$ y $h_2(nro\_usuario)$, cada una devolviendo un valor entre 1 y 300, y de esta manera pudo resolver la junta de forma eficiente.

Para hacerlo, modificó la función `JuntaHashGRACE()` de la siguiente manera:

Algoritmo 3: `JuntaHashGRACE(`$T_1, T_2, a, h_1, h_2, MAX$`)`

Entrada : Dos tablas T_1 y T_2 en disco, un atributo de junta a, dos funciones de hash h_1 y h_2, y un valor MAX que es el máximo valor devuelto por cada función de hash.

Salida : El resultado de la junta de T_1 y T_2 por el atributo a en disco.

```
Crear tabla resultado R vacía en disco
T1_1, ..., T1_MAX = HashearTabla(T1, a, h1, MAX)
T2_1, ..., T2_MAX = HashearTabla(T2, a, h1, MAX)
Para (cada i ∈ {1..MAX}) {
    T1_i,1, ..., T1_i,MAX = HashearTabla(T1_i, a, h2, MAX)
    T2_i,1, ..., T2_i,MAX = HashearTabla(T2_i, a, h2, MAX)
}
Para (cada i ∈ {1..MAX}) {
    Para (cada j ∈ {1..MAX}) {
        Cargar T1_i,j entera en memoria  /* Requiere B(T1_i,j) bloques disponibles */
        Cargar T2_i,j entera en memoria  /* Requiere B(T2_i,j) bloques disponibles */
        Agregar T1_i,j ⋈_1.a=2.a T2_i,j a R   /* Requiere 1 bloque para result. */
        Liberar memoria ocupada por T1_i,j y T2_i,j
    }
}
Devolver dirección de R en disco
```

Estime el costo resultante de esta variante de la *junta hash GRACE* (sin considerar el almacenamiento en disco del resultado) y explique por qué finalmente funciona.

5. *(Recuperación)* Un SGBD implementa el algoritmo de recuperación UNDO con checkpoint activo. Luego de una falla, el sistema encuentra el siguiente archivo de log:

```
01 (BEGIN, T1);
02 (BEGIN, T2);
03 (WRITE T1, X, 18);
04 (WRITE T2, Y, 4);
05 (COMMIT, T1);
06 (BEGIN CKPT, T2);
07 (BEGIN, T3);
08 (WRITE T3, X, 0);
09 (WRITE T2, Z, 6);
10 (COMMIT, T2);
11 (END CKPT);
12 (WRITE T3, Z, 8);
13 (BEGIN, T4);
14 (WRITE T4, Y, 0);
```

Explique cómo se llevará a cabo el procedimiento de recuperación, indicando hasta qué punto del archivo de log se deberá retroceder, y qué cambios deberán ser realizados en disco y en el archivo de log.

6. *(NoSQL)* Explique la diferencia entre *clave de partición (partition key)* y *clave de clustering (clustering key)* de una *column family* en *Cassandra*, indicando qué identifica cada una.