Base de Datos (75.15 / 75.28 / 95.05)

Evaluación Integradora - 12 de julio de 2017

TEMA 20171C3						**Padrón:** _____
MR		**DR**		**Proc.**		**Apellido:** _____
Rec.		**CyT**		**NoSQL**		**Nombre:** _____
Corrigió:						**Cantidad de hojas:** _____
Nota:						☐ 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. *(Modelo relacional)* Dadas dos relaciones $R(\underline{A_1}, A_2, ..., A_n)$ y $S(\underline{B_1}, \underline{B_2}, ..., B_m)$, en donde B_2 hace referencia a R, indique si las siguientes afirmaciones son verdaderas ó falsas, justificando cada una de sus respuestas.

   a) La regla de integridad referencial exije que $\forall s \in S : s[B_2] \neq$ `NULL`.

   b) La regla de integridad de entidad exije que $\forall r \in R : r[A_1] \neq$ `NULL`.

   c) La regla de integridad referencial exije que $\forall r \in R : \exists s \in S / s[B_2] = r[A_1]$.

   d) La regla de integridad de entidad exije que $\forall s \in S : s[B_2] \neq$ `NULL`.

   e) La regla de integridad referencial exije que $\forall s \in S : \exists r \in R / s[B_2] = r[A_1]$.

   f) La regla de unicidad exije que $\forall s_1, s_2 \in S, s_1 \neq s_2 : s_1[B_2] \neq s_2[B_2]$.

2. *(Diseño relacional)* Dada una relación $R(A, B, C, D, E, G, H, I)$ con un conjunto de dependencias funcionales $F = \{B \rightarrow D, AC \rightarrow E, I \rightarrow C, E \rightarrow AB\}$, indique si las siguientes afirmaciones son verdaderas ó falsas, justificando cada una de sus respuestas.

   a) $AEGHI$ es clave candidata de R.

   b) $H^+ = \emptyset$.

   c) $ABCGHI$ es superclave de R.

   d) $(CE \rightarrow D) \in F^+$

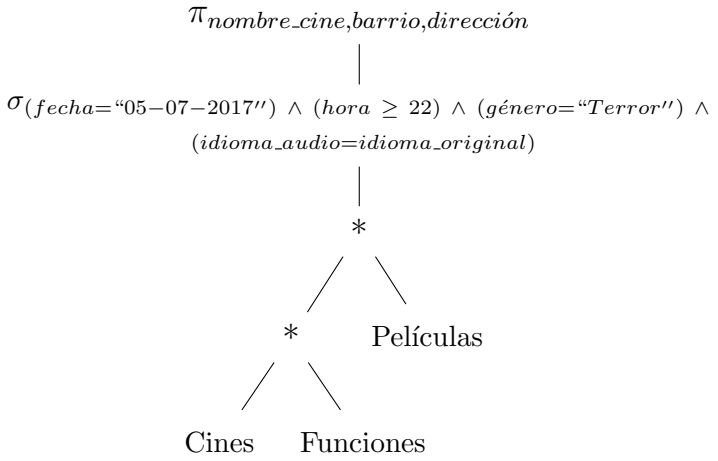
   e) F es un cubrimiento minimal de sí mismo.

   Nota: La notación X^+ denota la clausura de un conjunto de atributos ó del conjunto de dependencias, según el caso.

3. *(Procesamiento de consultas)* Hoy es miércoles y vamos al cine. Las siguientes relaciones nos indican las funciones disponibles en los distintos cines esta semana:

   - `Cines(nombre_cine, barrio, dirección)`
   - `Funciones(nombre_cine, fecha, hora, sala, nombre_película, idioma_audio)`
   - `Películas(nombre_película, género, año, idioma_original)`

   Nos interesa encontrar el listado de cines (columnas *nombre_cine*, *barrio* y *dirección*) en que proyectarán alguna película de terror en idioma original (`idioma_audio = idioma_original`) esta noche después de las 22:00hs. Para resolver esta consulta construímos el siguiente plan de consulta preliminar, representado en forma de árbol:



   Construya un plan de consulta optimizado, aplicando todas las reglas de optimización heurística para los operadores. Considere la siguiente información de catálogo:

CINES	**FUNCIONES**	**PELÍCULAS**
n(Cines) = 200	n(Funciones) = 8000	n(Películas)=100
		V(género, Películas)=10

4. *(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 (BEGIN, T3);
06 (WRITE T1, Z, 20);
07 (COMMIT, T1);
08 (WRITE T2, Z, 6);
09 (BEGIN CKPT, T2, T3);
10 (WRITE T3, X, 0);
11 (COMMIT, T2);
12 (WRITE T3, Z, 8);
13 (BEGIN, T4);
14 (WRITE T4, Y, 0);
```

   Explique cómo se llevará a cabo el procedimiento de recuperación, indicando qué cambios deben ser realizados en disco y en el archivo de log.

5. *(Concurrencia y Transacciones)* Enuncie el protocolo de *lock* de dos fases estricto (S2PL, *Strict two-phase lock*). Indique si un SGBD que lo utilice garantizará la serializabilidad y/o la recuperabilidad de las transacciones que ejecuta.

6. *(NoSQL)* Una red social almacena datos sobre sus usuarios y las páginas que les gustan en una base de datos Neo4J. Los usuarios *(User)* y las páginas *(Page)* tienen la siguiente estructura:

```
CREATE (p:User {name: 'Pablo', place: 'Mar del Plata'})
       (l:User {name: 'Luisa', place: 'Escobar'})
        ...
       (teg:Page {name: 'TEG', type: 'Juegos'})
       (aldosivi:Page {name: 'Club Atletico Aldosivi', type: 'Clubes'})
       ...
```

Las relaciones de amistad entre los usuarios se almancenan a través de la interrelación `IS_FRIEND`:

```
MATCH (p:User {name: 'Pablo'}),
      (l:User {name: 'Luisa'})
CREATE (p)-[:IS_FRIEND]->(l)
MATCH ... CREATE ...
```

Mientras que la información sobre qué páginas le gustan a cada usuario se almacena a través de la interrelación `LIKES`:

```
MATCH (p:User {name: 'Pablo'}),
      (teg: Page {name: 'TEG'})
CREATE (p)-[:LIKES]->(teg)
MATCH (p:User {name: 'Pablo'}),
      (a:Page {name: 'Club Atletico Aldosivi'})
CREATE (p)-[:LIKES]->(a)
MATCH ... CREATE ...
```

La siguiente consulta en Cypher obtiene la cantidad de páginas que agradan en común a Pablo con cada uno de sus amigos:

```
MATCH (amigo:User)-[:IS_FRIEND]-(p:User),
      likes_comun_amigo = (amigo:User)-[:LIKES]->(pag1:Page)<-[:LIKES]-(p:User)
WHERE p.name='Pablo'
RETURN amigo.name, count(likes_comun_amigo)
```

Nos interesa recomendar a Pablo nuevos amigos. Modifique la consulta anterior para obtener la cantidad de páginas que agradan en común a Pablo con los amigos de sus amigos, pero que no son amigos de Pablo. El formato del resultado debería ser *(nombre_amigo_amigo, páginas_común)*.