UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Politècnica Superior d'Enginyeria
de Vilanova i la Geltrú

DISSENY I ADMINISTRACIÓ DE BASES DE DADES
CONTROL 2

Data: 3 de juny de 2019 **Temps: 2 hores**

Problema 1 [2,5 punts]

La Generalitat de Catalunya vol gestionar el sistema universitari català que consisteix en diferents universitats formades per campus els quals imparteixen docència de diferents graus (títols universitaris) en diferents municipis.

Cada campus pertany a una universitat, està situat en un municipi i ofereix diferents graus (la impartició d'un grau en un campus s'anomena docència). Per evitar un excés d'oferta i ajustar els recursos docents disponibles, cada grau s'imparteix com a molt una vegada a cada municipi. Dit d'una altra manera, en un municipi no pot haver-hi docència duplicada del mateix grau.

Considerem la relació de la docència a les universitats catalanes Rel(D, G, C, M, U) on cada tupla correspon a la docència D del grau G realitzada en el campus C situat en el municipi M i que pertany a la universitat U.

(a) Analitza les dependències funcionals.

(b) Justifica per què la relació de la docència Rel(D, G, C, M, U) no està normalitzada ni en tercera forma normal (3NF) ni en Boyce-Codd (BCNF).

(c) Normalitza fins a 3NF sense arribar a BCNF, justificant la resposta.

(d) Normalitza fins a BCNF, justificant la resposta.

Problema 2 [2,5 punts]

Considerem les dues transaccions següents sobre la taula F, que inclou una clau primària INTEGER K i un valor addicional REAL V, existint, per tant, les columnes F.K i F.V i a on K determina V:

T1: Recorre totes les tuples (K,V) de F, sumant tots els valors trobats pel camp V i retornant la suma obtinguda.

T2: Recorre totes les tuples (K,V) de F i localitza, al llarg del recorregut, la clau Kmax més gran de la taula; fa un segon recorregut i localitza la clau Kmin més petita de la taula; divideix per 2 el valor del camp V en la tupla que correspon a Kmax i finalment multiplica per 2 el valor del camp V en la tupla que correspon a Kmin.

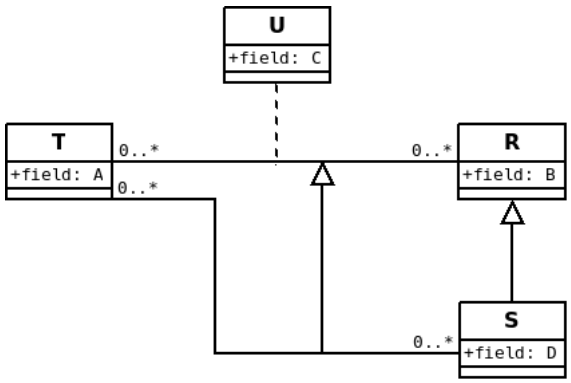
(a) Mostra, mitjançant un exemple concret inventat per tu, que si no es té en compte un aïllament adequat, al fer concurrentment les transaccions T1 i T2 poden obtenir-se resultats diferents de qualsevol execució serial.

(b) Identifica dels 4 nivells d'aïllament estàndard quin és el menor nivell d'aïllament que evita la interferència entre T1 i T2, explicant perquè s'evita i justificant que és el nivell d'aïllament mínim adequat.

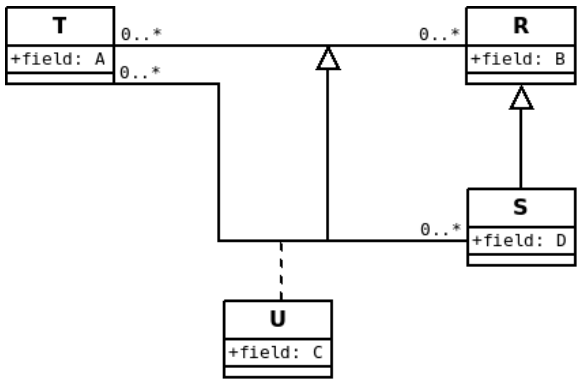
Problema 3 [2 punts]

(a) Explica breument les opcions que disposem per implementar casos d'especialització/generalització d'UML en esquemes relacionals, indicant els seus respectius avantatges i inconvenients.

(b) Considerem el cas concret d'aquest diagrama UML, on es suposa que A és l'únic atribut de T, B l'únic de R, C l'únic d'U i D l'únic atribut aportat per l'especialització S. Suposem també que A, B i C són camps voluminosos i que D no ho és. Decideix, justificant la resposta amb l'anàlisi dels avantatges i inconvenients, quina estratègia és preferible per implementar cadascuna de les dues especialitzacions.



(c) Repeteix l'apartat anterior per aquest altre diagrama UML, que difereix en la classe associativa i suposant a més a més que ara l'atribut D és voluminós i B no ho és.



Problema 4 [1,5 punts]

(a) Explica la tàctica que segueix el planificador per executar aquesta query:

```
EXPLAIN SELECT * FROM accounts a JOIN owners o ON a.acc_id=o.acc_id
        WHERE o.own_name='John' AND a.type='L';
                                        QUERY PLAN
-------------------------------------------------------------------------------
 Nested Loop  (cost=0.29..501.55 rows=6 width=36)
   ->  Seq Scan on owners o  (cost=0.00..380.36 rows=17 width=22)
         Filter: (own_name = 'John'::text)
   ->  Index Scan using accounts_pkey on accounts a  (cost=0.29..7.13 rows=1)
         Index Cond: (acc_id = o.acc_id)
         Filter: (type = 'L'::bpchar)
```

(b) Podem fer algun canvi que permeti al planificador canviar l'estratègia per tal de millorar els temps previstos? Quin?

(c) Explica la tàctica que segueix el planificador per executar aquesta segona query i perquè difereix tant de la primera sent les dues queries molt similars:

```
EXPLAIN SELECT * FROM accounts a JOIN owners o ON a.acc_id=o.acc_id
        WHERE o.own_name='John' OR a.type='L';
                                QUERY PLAN
-------------------------------------------------------------------------
 Hash Join  (cost=280.00..662.89 rows=6832 width=36)
   Hash Cond: (o.acc_id = a.acc_id)
   Join Filter: ((o.own_name = 'John'::text) OR (a.type = 'L'::bpchar))
   ->  Seq Scan on owners o  (cost=0.00..330.09 rows=20109 width=22)
   ->  Hash  (cost=155.00..155.00 rows=10000 width=14)
         ->  Seq Scan on accounts a  (cost=0.00..155.00 rows=10000 width=14)
```

(d) Podem fer algun canvi que permeti al planificador canviar l'estratègia per tal de millorar els temps previstos? Quin?

Problema 5 [1,5 punts]

(a) Indica dos avantatges i dos inconvenients dels sistemes de bases de dades no relacionals respecte als sistemes relacionals.

(b) Quina seria la crida al mètode *insert*() de MongoDB equivalent a aquest INSERT de SQL?

```
INSERT INTO users (username, password) VALUES ('John', '1234');
```

(c) Quina seria la crida al mètode $find()$ de MongoDB equivalent a aquest SELECT de SQL que fa una selecció i una projecció?

```
SELECT username FROM users WHERE username='John' AND password='1234';
```

Solució Problema 1

a) Les dependències funcionals són:

- C → U
- C → M
- D → G
- D → C
- M G → D

b) La relació R (D, G, C, M, U) no està ni en 3NF ni en BCNF, doncs la primera dependència funcional no es podria posar com a clau de la relació (ni tampoc U pertany a una clau).

c) En 3NF (la dependència funcional C → M no es pot posar com a clau de la relació R2, però 3NF ho permet ja que M pertany a una altra clau):

R1 (**C**, U)

R2 (**D**, **M, G**, C)

d) En BCNF i per tant en 3NF (totes les dependències funcionals estan representades com a claus de les relacions R1, R2 o del JOIN de R1 i R2):

R1 (**C**, U, M)

R2 (**D**, G, C)

Solució Problema 2

a) Suposem que la taula F contingui aquestes dos tuples (K, V): (1, 3) i (2, 4).

Una execució serial podria ser T1 + T2, llavors la suma obtinguda de T1 seria 7.

Una altra execució serial podria ser T2 + T1, llavors primer es modificarien les tuples quedant (1, 6) i (2, 2) i la suma obtinguda de T1 seria 8.

En cas d'una execució concurrent de T1 i T2 sense l'aïllament necessari, podria passar que es calcules la suma de T1 entremig de les dues modificacions que fa T2, per exemple quan només s'ha modificat la tupla que correspon a Kmax dividint el valor de V per 2 (o sigui en el moment que les tuples són (1, 3) i (2, 2) es faci l'acció de T1 donant com a resultat 5 que és diferent de les dues execucions serials).

b) Amb un aïllament de READ COMMITTED seria suficient ja que en l'aïllament READ COMMITTED una transacció que consulti una dada modificada per altres transaccions obtindrà un valor escrit per una transacció que ja hagi arribat a COMMIT. Per tant les dades consultades per la transacció T1 seran les que hi han quan encara no s'ha fet cap modificació per la T2 o quan la T2 ja ha arribat a COMMIT havent modificat les dues dades.

Vist d'una altra manera, l'aïllament de READ COMMITTED (i els aïllaments més estrictes com REPETEABLE READ i SERIALIZABLE) posa panys d'escriptura quan es modifica una dada que són exclusius, i per tant no permetrien fer la lectura d'aquesta dada modificada fins que aquests panys no s'alliberessin a l'arribar al COMMIT.

Solució Problema 3

a) Disposem de dues opcions per implementar una classe especialitzada en un esquema relacional:

(a) Fer una taula per l'especialització que té com a clau primària la mateixa de la relació base i que és a més clau forana de la relació base. Aquesta nova taula conté els atributs exclusius de l'especialització.

(b) Posar els camps de l'especialització en la mateixa taula de la relació base i un booleà que indiqui si la tupla està especialitzada o no (o un camp de tipus enter o enumerat si tenim vàries especialitzacions que hem de diferenciar).

Les mateixes dues opcions podríem usar per implementar l'especialització de les relacions entre dues classes.

b) Millor la segona opció doncs l'atribut D que hauríem d'afegir a la taula base pesa poc i el concepte associatiu U ja està definit entre la relació base R i T. A més a més, si féssim la primera opció, l'atribut B que seria la clau primària i forana de la taula especialitzada és voluminós.

c) Millor la primera opció doncs l'atribut D pesa molt i el concepte associatiu U només està entre l'especialització S i T. Així la relació base R, que sol contenir moltes més tuples que la seva especialització S, és molt més lleugera.

Solució Problema 4

a) Per realitzar el JOIN fa un bucle dins d'un altre (Nested Loop):

El bucle exterior fa un recorregut seqüencial (Seq Scan) per la taula *owners* filtrant les tuples que $own\_name =' John'$. Per cadascuna de les tuples obtingudes fa una cerca usant un índex (Index Scan) pel camp *acc_id* amb la condició $acc\_id = o.acc\_id$ i filtrant les tuples que $type =' L'$. El planificador calcula que usant aquest índex obtindrà una sola tupla (segurament perquè *acc_id* és una clau primària o alternativa de la taula); per tant el bucle interior només tindrà una iteració i el Nested Loop serà ràpid.

b) Podríem afegir un índex pel camp *own_name* de la taula *owners*, així el planificador canviaria el primer Seq Scan per un Index Scan millorant els temps d'execució:

```
CREATE INDEX ON owners(own_name);
```

c) Per realitzar el JOIN farà un Hash Join amb la condició $o.acc\_id = a.acc\_id$ i filtrant només els resultats que compleixin la condició $(o.own\_name =' John')OR(a.type =' L')$ seguint aquests passos:

Primer fa un recorregut seqüencial (Seq Scan) per la taula *accounts* i calcula un valor de dispersió per cada tupla segons el valors del camp *acc_id*.

Segon fa un recorregut seqüencial (Seq Scan) per la taula *owner* i usant la funció de dispersió calculada en el primer pas troba amb cost constant les tuples de la taula *accounts* que compleixen la condició $o.acc\_id = a.acc\_id$, filtrant només els resultats que compleixin la condició $(o.own\_name =' John')OR(a.type =' L')$.

Remarcar que aquests dos recorreguts no són un Nested Loop, sinó que es fan només una vegada cadascun, primer el de la taula accounts i després el de la taula owners,

aconseguint un cost lineal enlloc de quadràtic.

d) És impossible millorar l'estratègia del planificador degut al OR que hi ha en la condició del WHERE que provoca que el resultat del JOIN no depengui exclusivament que el own_name dels $owners$ sigui $'John'$ o que el $type$ dels $accounts$ sigui $'L'$. En aquest cas la millor estratègia és la que ja s'ha usat, fer un Hash Join amb un cost lineal i filtrar els resultats.

Solució Problema 5

a) Avantatges:

- Descentralitzats: Distribuïts en varis servidors/centres de dades.
- Escalen linealment amb màquines "normals".
- Tolerància a errors elevada.

Inconvenients:

- Dificultats pel processat transaccional tradicional.
- Incompatibilitat amb els sistemes relacionals (no hi ha transaccions, no hi ha restriccions, no hi ha vistes, ...).
- No hi ha modelat de dades.

b) Inserim un document a la col·lecció users:

```
db.users.insert({'username': 'John', 'password': '1234'})
```

c) Cerquem documents a la col·lecció users. En el primer document indiquem les condicions de cerca (selecció), en el segon document indiquem els atributs visibles o invisibles (projecció):

```
db.users.find({'username': 'John', 'password': '1234'}, {'username': 1})
```

o bé

```
db.users.find({'username': 'John', 'password': '1234'}, {'password': 0})
```