Tentamen Databases

16 april 2012

13:30 - 16:30, Educatorium-Gamma

- Scheur de antwoordvellen doormidden.
- Maak elke vraag op een ander vel.
- Vermeld op elk vel je naam en studentnummer. Indien één van deze zaken ontbreekt, wordt het vel niet nagekeken.
- Toon bij het inleveren je collegekaart.
- Schrijf en formuleer duidelijk.
- Je mag een A4 met aantekeningen raadplegen.
- Het tentamen duurt 3 uur en bestaat uit 7 opgaven.
- Gebruikte afko's:
  2PL = Two-phase locking
  2PC = Two-phase commit
  BCNF = Boyce-Codd normaalvorm
  COORD = Coordinator
  DP = Dependency preserving
  FD = functional dependency
  RA = Relationele algebra
  SQL = SQL (Structured Query Language)
- Vergeet niet de vakevaluatie in te vullen. Zie de education pagina.
- **Succes!**

Puntentelling: (totaal = 104 punten; 100 punten geeft een 10)

1: 16 punten
2: 16 punten
3: 16 punten
4: 16 punten
5: 12 punten
6: 12 punten
7: 16 punten

1

1 Algemeen

Geef van de volgende beweringen aan of deze correct zijn of niet. Een simpel JA of NEE volstaat. Er hoeft geen toelichting gegeven te worden, maar het mag wel, mits je het kort houdt.

1. Dat elke FD $X \rightarrow Y$ uit de oorspronkelijke FD-set F in één van de relatieschema's past, is een voldoende voorwaarde voor de DP-eigenschap.
2. Als een schema in BCNF is, is dit schema ook in 3NF.
3. Is de volgende herschrijfregel geldig?
   $X \rightarrow Z, Y \rightarrow Z \Rightarrow XY \rightarrow Z$
4. Is de volgende herschrijfregel geldig?
   $XY \rightarrow Z \Rightarrow X \rightarrow Z$
5. Voor elk relatieschema is een verliesvrije DP 3NF decompositie mogelijk.
6. Als de transactiemanager werkt volgens het UNDO-mechanisme, dan mag deze database-data naar disk schrijven nadat commitment heeft plaatsgevonden.
7. Als de transactiemanager werkt volgens het REDO-mechanisme, dan mag deze database-data naar disk schrijven nadat commitment heeft plaatsgevonden.
8. Het 2PC-protocol kan geblokkeerd raken als één van de participants een `<do not commit>`-stem heeft uitgebracht.
9. Elke serializeerbare schedule wordt geaccepteerd door een 2PL-scheduler.
10. Een B-tree-index op attribuut `A` biedt potentieel goede ondersteuning voor de verwerking van een selectiecriterium van het type `A < constante`.

2

2 Functional dependencies

Ten behoeve van de registratie van boeken in onze informaticabibliotheek hebben we (onder andere) de volgende attributen:
bknr, isbn, rubriek, volgnr, exnr, titel, auteur.
Elk exemplaar van elk boek heeft een uniek boeknummer (*bknr*) dat gebruikt wordt voor de registratie van leningen. Dit nummer is met een magneetcode aangebracht. Het begrip isbn wordt bekend verondersteld. Elk boek wordt in een rubriek ondergebracht. Meerdere boeken binnen dezelfde rubriek worden met een volgnummer (*volgnr*) onderscheiden en meerdere exemplaren onder dit volgnummer met een exemplaarnummer (*exnr*).
Voorbeeld: we hebben een rubriek *G2.0* Daarbinnen hebben we verschillende boeken: *[G2.0 1], [G2.0 2]* enzovoort. Het feit dat we twee exemplaren van *[G2.0 17]* hebben wordt aangegeven met exemplaarnummers: *[G2.0 17 (1)], [G2.0 17 (2)].* De aanduiding tussen rechte haken vind je op het kaft van elk boek terug.
Geef van de volgende FD's aan of deze wel of niet geldig zijn. Geef desgewenst een korte toelichting.

1. $bknr \rightarrow isbn$
2. $bknr \rightarrow rubriek$
3. $bknr \rightarrow volgnr$
4. $bknr \rightarrow exnr$
5. $isbn \rightarrow bknr$
6. $isbn \rightarrow exnr$
7. $exnr \rightarrow bknr$
8. $exnr \rightarrow isbn$
9. $auteur, titel \rightarrow bknr$
10. $bknr \rightarrow auteur, titel$
11. $isbn \rightarrow auteur, titel$
12. $volgnr, exnr \rightarrow auteur, titel$

3 Normalisatie

$R = (ABCDEG)$
$F = \{C \rightarrow E, CD \rightarrow B, BC \rightarrow D, E \rightarrow DG\}$

(i) Bereken een minimal cover voor F. Licht de stappen die u maakt kort toe.

(ii) Geef een verliesvrije, 3NF, DP decompositie van R met behulp van het daartoe geschikte algoritme.

4 Queries

Een bioscoop regelt zijn kaartverkoop online en heeft daartoe de volgende database.

```
Film (fid, titel, jaar, reg)
Klant (kid, naam, adres, woonplaats, gebdat)
Ticket (fid, kid, datum, tijd)
```

Een film wordt geïdentificeerd door een `fid`, heeft een titel, een jaar van verschijning en één regisseur.
Een klant wordt geïdentificeerd door een `kid`, heeft naw-gegevens en een geboortedatum.
Een ticket wordt gekocht door een klant voor een filmvoorstelling op een bepaald moment.
In dit schema zijn enkele simplificaties verwerkt (bijvoorbeeld één ticket per klant) die niet zo realistisch zijn, maar daar maken we ons niet druk om. De frase "heeft X gezien" wordt hieronder geïnterpreteerd als "heeft een ticket gekocht in ons theater voor X".
We hebben de volgende queries.

Q1: Welke klanten (naam) hebben wel eens een film van Kubrick gezien.
Q2: Welke klanten (naam) hebben geen enkele film van Kubrick gezien.
Q3: Welke klanten (naam) hebben uitsluitend films van Kubrick gezien.
Q4: Geef de tweetallen klanten (kidA, kidB) zodat klant A elke film heeft gezien die klant B gezien heeft.

Hieronder volgen expressies in de RA of in SQL. Geef voor elke query aan welke expressie(s) met de query corresponderen. Houd voor de snelheid van het nakijken de volgende lay out aan in je antwoord (dit is een voorbeeld).

```
Q1:  E1 E2 E3
Q2:  E4 E5
Q3:  E4 E6
Q4:  geen
```

De relatie tussen queries en expressies is many-to-many en optioneel. Ga *niet* uit van de specifieke beperkingen van SQLite.

E1: $\pi_{naam}(Klant \bowtie Ticket \bowtie \sigma_{reg="Kubrick"}(Film))$

E2: $\pi_{naam}(Klant \bowtie Ticket \bowtie \sigma_{reg<>"Kubrick"}(Film))$

E3: $\pi_{naam}(Klant \bowtie (\pi_{kid}(Ticket) \cup \pi_{kid}(Ticket \bowtie \sigma_{reg="Kubrick"}(Film))))$

E4: $\pi_{naam}(Klant \bowtie (\pi_{kid}(Ticket) \cap \pi_{kid}(Ticket \bowtie \sigma_{reg="Kubrick"}(Film))))$

E5: $\pi_{naam}(Klant \bowtie (\pi_{kid}(Ticket) \bowtie \pi_{kid}(Ticket \bowtie \sigma_{reg<>"Kubrick"}(Film))))$

E6: $\pi_{naam}(Klant \bowtie (\pi_{kid}(Ticket) - (\pi_{kid}(Ticket \bowtie \sigma_{reg="Kubrick"}(Film)))))$

E7: $\pi_{naam}(Klant \bowtie (\pi_{kid}(Ticket) - (\pi_{kid}(Ticket \bowtie \sigma_{reg<>"Kubrick"}(Film)))))$

E8:

```
SELECT naam FROM Klant
WHERE kid IN (
    SELECT kid FROM Ticket T, Film F
    AND T.fid = F.fid AND reg = "Kubrick"
)
```

E9:

```
SELECT naam FROM Klant K
WHERE kid IN (
    SELECT kid FROM Ticket T, Film F
    WHERE T.fid = F.fid AND reg <> "Kubrick"
)
```

E10:

```
SELECT naam FROM Klant
WHERE EXISTS (
    SELECT * FROM Ticket T, Film F
    WHERE T.fid = F.fid AND reg = "Kubrick"
)
```

E11:

```
SELECT naam FROM Klant
WHERE EXISTS (
    SELECT * FROM Klant K, Ticket T, Film F
    WHERE K.kid = T.kid AND T.fid = F.fid AND reg = "Kubrick"
)
```

E12:

```
SELECT naam FROM Klant K
WHERE EXISTS (
    SELECT * FROM Ticket T, Film F
    WHERE K.kid = T.kid AND T.fid = F.fid AND reg <> "Kubrick"
)
```

E13:

```
SELECT naam FROM Klant K
WHERE NOT EXISTS (
    SELECT * FROM Ticket T, Film F
    AND K.kid = F.kid AND T.fid = F.fid AND reg <> "Kubrick"
)
```

E14:

```
SELECT naam FROM Klant K
WHERE NOT EXISTS (
    SELECT * FROM Ticket T, Film F
    AND K.kid = F.kid AND T.fid = F.fid AND reg = "Kubrick"
)
```

E15:

```
SELECT A.kid AS kidA, B.kid AS kidB
FROM Klant A, Klant B
WHERE NOT EXISTS (
    SELECT * FROM Film F
    WHERE NOT EXISTS (
        SELECT * FROM Ticket T
        WHERE T.fid = F.fid AND T.kid = kidB AND T.kid <> kidA
) )
```

E16:

```
SELECT A.kid AS kidA, B.kid AS kidB
FROM Klant A, Klant B
WHERE NOT EXISTS (
    SELECT * FROM Ticket T
    WHERE T.kid = kidB
    AND NOT EXISTS (
        SELECT * FROM Ticket T2
        WHERE T2.kid = kidA AND T.fid = T2.fid
) )
```

E17:

```
SELECT A.kid AS kidA, B.kid AS kidB
FROM Klant A, Klant B
WHERE NOT EXISTS (
    SELECT * FROM Ticket T
    WHERE T.kid = kidB
    AND NOT EXISTS (
        SELECT * FROM Ticket T2
        WHERE T2.kid = kidA AND T.fid <> T2.fid
) )
```

5 Concurrency

Hieronder zijn twee schedules gegeven. Geef aan of deze schedules serializeerbaar zijn of niet. Zo ja, geef dan de equivalente serieële schedules. Zo nee, geef dan de/een cycle in de precedentiegraaf.

S1

T1	T2	T3	T4	T5
			w(z)	
r(x)				
	w(x)			
				r(y)
w(z)				
	r(y)			
				r(x)
		r(z)		
			w(y)	

S2

T1	T2	T3	T4	T5
			r(z)	
r(x)				
	w(x)			
				r(y)
r(z)				
	r(y)			
				r(x)
		r(z)		
			w(y)	

6 recovery

We beschouwen nonquiescent recovery met **REDO** logging. Hieronder vind je een log met after images.

```
<START T1>
<T1, A, 5>
<START T2>
<T2, B, 10>
<COMMIT T1>
<START T3>
<T3, C, 10>
<START CKPT (T2, T3)>
<T2, D, 15>
<START T4>
<START T5>
<T4, F, 25>
<T5, H, 12>
<COMMIT T2>
<COMMIT T5>
<END CKPT>
<T4, G, 30>
```

Stel dat een crash optreedt direct na `<T4, G, 30>` .

(i) Welk gedeelte van de log file wordt gescand?

(ii) Welke transacties worden undone?

(iii) Welke transacties worden redone?

7 Query processing

We gaan uit van twee relatieschema's R en S die één attribuut A gemeenschappelijk hebben. We definiëren een algebraïsche operator $\blacktriangleright$ (anti-join) als volgt:

> $R \blacktriangleright S$ bevat de tuples r in R waarvoor geldt dat er geen tuple s in S bestaat met $r.A = s.A$.

Met andere woorden, de anti-join selecteert uit R de tuples die de natural join met S niet zullen "overleven".
Je hoeft bij deze opgave geen formele bewijzen te leveren.

(i) Geef van de volgende algebraïsche regels aan of ze geldig zijn. Het antwoord kan luiden *nee*, *ja* (= onvoorwaardelijk) of *onder voorwaarden*. Geef in het laatste geval aan welke de voorwaarden zijn.

(a) $\sigma_p(R \blacktriangleright S) \equiv \sigma_p(R) \blacktriangleright S$
(b) $\sigma_p(R \blacktriangleright S) \equiv \sigma_p(R) \blacktriangleright \sigma_p(S)$
(c) $R \blacktriangleright S \equiv R \blacktriangleright (\pi_A S)$
(d) $R \bowtie S \equiv (R - (R \blacktriangleright (\pi_A S))) \bowtie S$

(ii) Druk één van de queries van opgave 4 uit in de RA, waarbij je gebruik maakt van de anti-join. Vermeld om welke query het gaat.

Einde