Tentamen Databases voor ica

17 april 2014

16:30 - 19:30, Educatorium-Gamma

- Beantwoord vragen 1, 2, 4 en 5 door het invullen van de bijbehorende pagina's van dit tentamen en deze pagina's afzonderlijk in te leveren.
- Beantwoord de overige vragen op een standaard tentamenvel; scheur hiertoe het vel verticaal 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)
- **Succes!**

Puntentelling: (totaal = 100 punten)
1: 16 punten
2: 16 punten
3: 16 punten
4: 16 punten
5: 12 punten
6: 12 punten
7: 12 punten

Studentnr:	**Achternaam:**

1 Algemeen

Geef van de volgende beweringen aan of deze correct zijn of niet. Een simpel JA of NEE volstaat. Maak het hokje met het foute antwoord zwart.
Er hoeft geen toelichting gegeven te worden.

1. **ja** **nee:** 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. **ja** **nee:** Als een schema in 4NF is, is dit schema ook in BCNF.

3. **ja** **nee:** Is de volgende herschrijfregel geldig?
   $X \rightarrow U, Y \rightarrow V \Rightarrow XU \rightarrow YV$

4. **ja** **nee:** Voor elk relatieschema is een verliesvrije DP 3NF decompositie mogelijk.

5. **ja** **nee:** Als de transactiemanager werkt volgens het UNDO+REDO-mechanisme, dan mag deze database-data naar disk schrijven nadat commitment heeft plaatsgevonden.

6. **ja** **nee:** Als de transactiemanager werkt volgens het UNDO+REDO-mechanisme, dan mag deze database-data naar disk schrijven voordat commitment heeft plaatsgevonden.

7. **ja** **nee:** Het 2PC-protocol kan geblokkeerd raken als één van de participants een `[ready]`-stem heeft uitgebracht.

8. **ja** **nee:** Elke schedule die geaccepteerd wordt door een 2PL-scheduler is serializeerbaar.

9. **ja** **nee:** Een hash index op attribuut `A` biedt goede ondersteuning voor de verwerking van een selectiecriterium van het type `A = c1 AND B = c2`.

10. **ja** **nee:** Een SQL-query zal door één en hetzelfde DBMS altijd naar dezelfde executiemethode vertaald worden.

Studentnr:	**Achternaam:**

2 Functionele afhankelijkheden

We hebben de volgende attributen ten behoeve van de landelijke registratie van personenauto's, hun eigenaars (altijd één persoon) en de bijbehorende, verplichte, aansprakelijkheidsverzekering(en). Het gaat hier slechts om particuliere autobezitters; voor elke auto en verzekeringsperiode wordt een aparte polis opgesteld. Gedurende het eigenaarschap kunnen verschillende verzekeringen gelden. Er worden zowel actuele als historische gegevens bijgehouden.

- *kenteken:* we kijken alleen naar Nederlandse auto's
- *eigenaar:* sofinummer
- *merk* van de auto
- *type* van de auto
- *bouwjaar* van de auto
- *kleur* van de auto
- *polisnr* van de verzekering
- *maatschappij:* de verzekeraar
- *datumi:* de ingangsdatum van het eigenaarschap
- *datume:* de einddatum van het eigenaarschap
- *vdatumi:* de ingangsdatum van de verzekering
- *vdatume:* de einddatum van de verzekering

(i) Geef alle attributen die functioneel afhankelijk zijn van *kenteken*.

(ii) Geef alle attributen die functioneel afhankelijk zijn van *eigenaar*.

(iii) Geef alle attributen die functioneel afhankelijk zijn van *polisnr*.

(iv) Geef alle attributen die functioneel afhankelijk zijn van *kenteken, datumi*.

(v) Geef alle attributen die functioneel afhankelijk zijn van *eigenaar, datumi*.

Maak voor je antwoorden gebruik van de ruimte hieronder.

(i):

(ii):

(iii):

(iv):

(v):

3 Normaalvormen

Stel we hebben een relatieschema $R(ABCDEFG)$ en een set FDs $F_R = \{C \to AG, A \to DF, CD \to FB, B \to G, F \to G\}$.

(i) Geef een verliesvrije, dependency preserving 3NF-decompositie van R. Laat zien welke methode je gebruikt.

Stel we hebben een relatieschema $S(ABCD)$ en een verzameling FD's
$F_S = \{A \to B, B \to C, C \to D\}$.

(ii) Geef twee verschillende verliesvrije BCNF decomposities. Laat zien welke methode je gebruikt.

(iii) Geef aan welke van de twee BCNF decomposities uit onderdeel (ii) DP zijn. Licht toe.

4 Queries

Een bedrijf dat gegevens bijhoudt over de woningmarkt heeft een database met daarin een tabel waarin gegevens over woningen worden bijgehouden, een tabel waarin het aanbod wordt bijgehouden en een tabel waarin de verkooptransacties worden bijgehouden. Elke woning heeft een uniek objectnummer waaronder het bij het bedrijf bekend staat, een adres (inclusief huisnummer, toevoeging huisnummer en plaats), een type (vrijstaande woning, hoekwoning, tussenwoning, appartement), een woonoppervlakte, een grondoppervlakte en een bouwjaar. Aangeboden woningen hebben een aanbodnummer, een vraagprijs, de datum waarop het object in de verkoop is gekomen en een referentie naar een woning op basis van het objectnummer. Verkooptransacties hebben een verkoopprijs, een verkoopdatum en een referentie naar een aanbieding op basis van het aanbodnummer. We gaan er in het onderstaande vanuit dat geen enkele tabel leeg is. De vraagprijs van een woning kan aangepast worden. In dat geval wordt de vraagprijs in het bestaande record aangepast en blijft de datum ongewijzigd. Een woning kan meerdere keren verkocht worden (uiteraard verandert dan het bijbehorende aanbodnummer). Het databaseschema van de database is als volgt (primary keys zijn in cursief):

```
Woning ( objectnr, postcode, huisnr, type, woonopp, grondopp, bouwjaar )
Aanbod ( aanbodnr, objectnr, adatum, vraagprijs )
Verkoop ( aanbodnr, vdatum, verkoopprijs )
```

Studentnr:	**Achternaam:**

Gegeven zijn de volgende queries:

Q1: Geef per verkochte woning de verandering in waarde tussen de eerste en de laatste verkoop.
Q2: Geef het aanbodnummer en de verkoopprijs van de woningen die niet voor de vraagprijs verkocht zijn.
Q3: Geef de postcodes waarvoor geldt dat alle aangeboden woningen verkocht zijn.

Hieronder volgen expressies in de RA of in SQL. Geef voor elke query aan welke expressie(s) met de query corresponderen. **Vul dit hieronder in**. De relatie tussen queries en expressies is many-to-many en optioneel. Je mag ervan uitgaan dat de vergelijkingen op datumvelden syntactisch correct zijn.

Q1 =

Q2 =

Q3 =

E1: $\pi_{aanbodnr,verkoopprijs}((\pi_{aanbodnr}(Aanbod) - \pi_{aanbodnr}(Verkoop)) \bowtie Verkoop)$

E2: $\pi_{aanbodnr,verkoopprijs}(Verkoop) - (\pi_{aanbodnr,verkoopprijs}(Verkoop) \cup \pi_{aanbodnr,vraagprijs}(Aanbod))$

E3: $\pi_{aanbodnr,verkoopprijs}(Verkoop) - (\pi_{aanbodnr,verkoopprijs}(Verkoop) \cap \pi_{aanbodnr,vraagprijs}(Aanbod))$

E4: $\pi_{aanbodnr,verkoopprijs}(Verkoop) - \pi_{aanbodnr,vraagprijs}(Aanbod)$

E5: $\pi_{aanbodnr,verkoopprijs}((\pi_{verkoopprijs}(Verkoop) - \pi_{vraagprijs}(Aanbod)) \bowtie Verkoop)$

E6: $\pi_{postcode}(\pi_{postcode,aanbodnr}(Aanbod) \div \pi_{aanbodnr}(Verkoop))$

E7:

```
SELECT Verkoop.aanbodnr, Verkoop.verkoopprijs
FROM Verkoop, Aanbod
WHERE Verkoop.aanbodnr = Aanbod.aanbodnr
AND Verkoop.verkoopprijs <> Aanbod.vraagprijs
```

E8:

```
SELECT Verkoop.aanbodnr, Verkoop.verkoopprijs
FROM Verkoop
WHERE Verkoop.verkoopprijs NOT IN (
    SELECT Aanbod.vraagprijs
    FROM Aanbod
    WHERE Aanbod.aanbodnr = Verkoop.aanbodnr)
```

E9:

```
SELECT Verkoop.aanbodnr, Verkoop.verkoopprijs
FROM Verkoop
WHERE Verkoop.aanbodnr NOT IN (
    SELECT Aanbod.aanbodnr
    FROM Aanbod
    WHERE Aanbod.vraagprijs = Verkoop.verkoopprijs)
```

E10:

```
SELECT Verkoop.aanbodnr, Verkoop.verkoopprijs
FROM Verkoop
WHERE NOT EXISTS (
    SELECT *
    FROM Aanbod
    WHERE Aanbod.vraagprijs = Verkoop.verkoopprijs
    AND Aanbod.aanbodnr = Verkoop.aanbodnr)
```

E11:

```
SELECT Verkoop.aanbodnr, MAX(Verkoop.verkoopprijs) -
    MIN(Verkoop.verkoopprijs)
FROM Verkoop
GROUP BY Verkoop.aanbodnr
```

E12:

```
SELECT Aanbod.objectnr, MAX(Verkoop.verkoopprijs) -
    MIN(Verkoop.verkoopprijs)
FROM Verkoop, Aanbod
WHERE Verkoop.aanbodnr = Aanbod.aanbodnr
GROUP BY Aanbod.objectnr
```

E13:

```
SELECT Aanbod1.objectnr, Verkoop2.verkoopprijs - Verkoop1.verkoopprijs
FROM Verkoop AS Verkoop1, Verkoop AS Verkoop2,
    Aanbod AS Aanbod1, Aanbod AS Aanbod2
WHERE Aanbod1.objectnr = Aanbod2.objectnr
AND Aanbod1.aanbodnr = Verkoop1.aanbodnr
AND Aanbod2.aanbodnr = Verkoop2.aanbodnr
AND Verkoop1.vdatum <= ALL
AND Verkoop2.vdatum >= ALL
```

E14:

```
SELECT Aanbod1.objectnr, Verkoop2.verkoopprijs - Verkoop1.verkoopprijs
FROM Verkoop AS Verkoop1, Verkoop AS Verkoop2,
    Aanbod AS Aanbod1, Aanbod AS Aanbod2
WHERE Aanbod1.objectnr = Aanbod2.objectnr
AND Aanbod1.aanbodnr = Verkoop1.aanbodnr
AND Aanbod2.aanbodnr = Verkoop2.aanbodnr
AND Aanbod1.adatum <= ALL (
    SELECT Aanbod.adatum
    FROM Aanbod
    WHERE Aanbod.objectnr = Aanbod1.objectnr)
AND Aanbod2.adatum >= ALL (
    SELECT Aanbod.adatum
    FROM Aanbod
    WHERE Aanbod.objectnr = Aanbod2.objectnr)
```

E15:

```
SELECT Aanbod1.objectnr, Verkoop2.verkoopprijs - Verkoop1.verkoopprijs
FROM Verkoop AS Verkoop1, Verkoop AS Verkoop2,
    Aanbod AS Aanbod1, Aanbod AS Aanbod2
WHERE Aanbod1.objectnr = Aanbod2.objectnr
AND Aanbod1.aanbodnr = Verkoop1.aanbodnr
AND Aanbod2.aanbodnr = Verkoop2.aanbodnr
AND Verkoop1.vdatum <= ALL (
    SELECT Verkoop.vdatum
    FROM Aanbod, Verkoop
    WHERE Aanbod.aanbodnr = Verkoop.aanbodnr
    AND Aanbod.objectnr = Aanbod1.objectnr)
AND Verkoop2.vdatum >= ALL (
    SELECT Verkoop.vdatum
    FROM Aanbod, Verkoop
    WHERE Aanbod.aanbodnr = Verkoop.aanbodnr
    AND Aanbod.objectnr = Aanbod2.objectnr)
```

Studentnr:	**Achternaam:**

5 Query processing

In de extended relationele algebra die werkt op bags (collecties met duplicaten) wordt de operator δ gebruikt voor het verwijderen van duplicaten. Voor bags hebben de operatoren een afwijkende betekenis: het aantal voorkomens van de elementen spelt een rol.
Voorbeelden:
$\{a, a, a, b\} \cup \{a, b, c\} = \{a, a, a, a, b, b, c\}$
$\{a, a, a, b\} \cap \{a, a, b, b, c\} = \{a, a, b\}$
$\{a, a, a, b, d\} - \{a, a, b, b, c\} = \{a, d\}$

Hieronder zie je enkele regels. Geef voor elk van de regels aan of deze geldig is of niet (ja/nee). Toelichting wordt genegeerd.

1. $\delta(R \cup S) = \delta(R) \cup \delta(S)$
2. $\delta(R - S) = \delta(R) - \delta(S)$
3. $\delta(R \cap S) = \delta(R) \cap \delta(S)$
4. $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$

1:

2:

3:

4:

6 Recovery

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

```
<START T1>
<T1, B, 10, 11>
<COMMIT T1>
<START T3>
<START T2>
<T2, A, 5, 6>
<T3, C, 10, 11>
<START CKPT (T2, T3)>
<T2, D, 15, 16>
<START T4>
<START T5>
<T4, F, 25, 26>
<T5, H, 12, 13>
<COMMIT T2>
<COMMIT T3>
<END CKPT>
<COMMIT T4>
<T5, G, 30, 31>
```

Stel dat een crash optreedt direct na `<COMMIT T2>` . (Het resterende gedeelte wordt dus niet geschreven.)

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

(ii) Welke transacties worden undone?

(iii) Welke transacties worden redone?

Stel dat een crash optreedt direct na `<T5, G, 30, 31>` .

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

(v) Welke transacties worden undone?

(vi) Welke transacties worden redone?

7 Concurrency

We herhalen nog even de basis van concurrency. Een schedule dat de executievolgorde van een aantal transacties beschrijft, is correct als deze executievolgorde serializeerbaar (SR) is.
Een schedule H is SR dan en slechts dan als de precedentiegraaf $G(H)$ acyclisch is.
Een voorbeeld van een mechanisme om serializeerbaarheid af te dwingen is two-phase-locking.

We gaan nu naar een alternatief mechanisme kijken om de eigenschap SR af te dwingen: de *timestamp scheduler*. Elke transactie T_i krijgt een unieke timestamp $ts(T_i)$. De transactiemanager laat de timestamp counter oplopen in de tijd. Hoe hoger de timestamp, des te jonger de transactie.

Stel we hebben transacties T_i en T_j. We hebben een operatie $p_i(X)$ van T_i en een operatie $p_j(X)$ van T_j. Het gaat hier om read- of writeoperaties. Je kunt serializeerbaarheid afdwingen door de volgende eis te implementeren.
Als $p_i(X)$ en $p_j(X)$ conflicteren, dan moeten deze operaties in volgorde van timestamp afgehandeld worden. Dus voor alle conflicterende $p_i(X)$ en $p_j(X)$ geldt:
$p_i(X) \prec p_j(X) \Leftrightarrow ts(T_i) < ts(T_j)$, waarbij $\prec$ tijdsvolgorde aanduidt.
Om deze regel te implementeren houden we voor elk data-element X twee getallen bij: $RMAX$ en $WMAX$. Deze getallen zijn respectievelijk: de hoogste timestamp van de transacties die X gelezen hebben respectievelijk de hoogste timestamp van de transacties die X geschreven hebben.

(i) Stel een transactie T_i wil een operatie $R_i(X)$ uitvoeren. Beschrijf hoe het mechanisme beslist of deze operatie mag plaatsvinden. Bij weigering volgt een transaction abort, gevolgd door transaction herstart met een nieuwe timestamp. Geef aan of de $RMAX(X)$ en/of $WMAX(X)$ aangepast moeten worden (bij goedkeuring respectievelijk weigering).

(ii) Stel een transactie T_i wil een operatie $W_i(X)$ uitvoeren. Beschrijf hoe het mechanisme beslist of deze operatie mag plaatsvinden. Bij weigering volgt een transaction abort, gevolgd door transaction herstart met een nieuwe timestamp. Geef aan of de $RMAX(X)$ en/of $WMAX(X)$ aangepast moeten worden (bij goedkeuring respectievelijk weigering).

(iii) Beargumenteer waarom dit mechanisme serializeerbaarheid garandeert.

Einde