**TECHNISCHE UNIVERSITEIT DELFT**
**Faculteit Elektrotechniek, Wiskunde en Informatica**

Uitwerkingen van het Tentamen

# TI2505/TI2500 Informatie- en Datamodellering

Maandag, 14 april 2014
18u30-21u30

Dit tentamen bestaat uit 11 delen, elk met een aantal open vragen.
Van alle vragen dienen de antwoorden op het gewone antwoordblad(en) ingevuld te worden. Kladpapier wordt niet nagekeken

| Deel | Punten |
|---|---|
| 1 | 8 |
| 2 | 10 |
| 3 | 10 |
| 4 | 5 |
| 5 | 10 |
| 6 | 20 |
| 7 | 5 |
| 8 | 5 |
| 9 | 10 |
| 10 | 8 |
| 11 | 9 |
| **Totaal** | **100** |

Het gebruik van de boeken "*Database Systems: Global Edition, 6/E*" en "*Fundamentals of Database Systems*" door Ramez Elmasri en Shamkant Navathe is toegestaan, evenals slides, oude tentamens met uitwerkingen en eigen aantekeningen.

Vul je naam, studienummer en studierichting in op ieder antwoordblad, **en op het eerste antwoordblad of je dit tentamen doet voor TI2500 of T2505**. Als het tentamen voor TI2500 doet hoef je deel 11 niet te maken.

## Veel succes

## Deel 1 – Relationele Algebra (8 punten)

**Vraag 1.** Beschouw het volgende relationele schema zoals ook gebruikt in het boek:

*EMPLOYEE(Fname, Minit, Lname, Ssn, Bdate, Address, Sex, Salary, Super_ssn, Dno)*
*DEPARTMENT(Dname, Dnumber, Mgr_ssn, Mgr_start_date)*
*DEPT_LOCATIONS(Dnumber, Dlocation)*
*PROJECT(Pname, Pnumber, Plocation, Dnum)*
*WORKS_ON(Essn, Pno, Hours)*
*DEPENDENT(Essn, Dependent_name, Sex, Bdate, Relationship)*

Geef voor de volgende queries de algebra-expressie die deze queries uitdrukt met de operatoren zoals besproken in Hfdst. 6 van het boek. Dit mag net zoals in het boek in de vorm van een enkele expressie of voor ingewikkelde queries in de vorm
*<tabel> ← <algebra expressie>;*
*<tabel> ← <algebra expressie>;*
*... ;*
*RESULT ← <algebra expressie>.*

**a)** [4pt] Geef de namen van de departementen die een project hebben waarvan de locatie niet die van het departement is.

**Antwoord**:

$$\text{DEPPROJ} := (\text{DEPARTMENT} * \text{DEPT\_LOCATIONS}) \bowtie_{\text{Dnumber=Dnum}} \text{PROJECT};$$
$$\text{RESULT} := \pi_{\text{Dname}} (\sigma_{\text{Dlocation} \neq \text{Plocation}}(\text{DEPPROJ}));$$

**Uitleg**: De *natural join* koppelt departementen en hun locaties, omdat DEPARTMENT en DEPT_LOCATIONS exact de *foreign key* Dnumber gemeenschappelijk hebben. Merk op dat je daarom niet een gewone equijoin kunt gebruiken zonder een renaming toe te passen. Vervolgens koppelt de join met PROJECT de project-informatie eraan vast. In de volgende regel selecteren we dan de combinaties waarvoor de lokatie van het department en de lokatie verschillend zij, en tenslotte projecteren we alleen de naam van het departement.

**b)** [4pt] Geef de namen van de departementen waarin minstens 3 werknemers alleen vrouwelijke dependenten hebben.

**Antwoord**:

$$\text{MALEDEPS} := \pi_{\text{Ssn}}(\text{EMPLOYEE} \bowtie_{\text{Ssn=Essn}} \sigma_{\text{Sex="M"}}(\text{DEPENDENT}));$$
$$\text{NOMALEDEPS} := \pi_{\text{Ssn}}(\text{EMPLOYEE}) - \text{MALDEPS};$$
$$\text{DEPTNMDEMPS} := \text{DEPARTMENT} \bowtie_{\text{Dnumber=Dno}} \text{NOMALEDEPS};$$
$$\text{DEPTCOUNT} := {}_{\text{Dname}}\mathcal{F}_{\text{COUNT Dnumber}}(\text{DEPTNMDEMPS});$$
$$\text{RESULT} := \pi_{\text{Dname}} (\sigma_{\text{COUNT Dumber} \geq 3} (\text{DEPTCOUNT}));$$

**Uitleg**: In MALEDEPS combineren we alle werknemers met hun mannelijke dependenten en projecteren we vervolgens op Ssn. Dus we krijgen dan de Ssn's van employees met minstens één mannelijke dependent. Voor NAMELEDEPS trekken we deze af van de set van alle ssn's, en krijgen dan dus de ssn's van alle werknmers die geen mannelijke dependenten

hebben. Deze combineren we met de afdelingen waarvoor ze werken in DEPTNMDEMPS. Vervolgens groeperen we op Dname (want aannemlijk uniek is per department) en tellen het aantal records per groep in DEPTCOUNT. Tenslote selecteren we alleen die groepen waarvan het aantal 3 of groter is en projecteren tenslotte op de departementsnaam.

## **Deel 2 – Normaalvormen** (10 punten)

**Vraag 2.** [5pt] Beschouw een relatie met schema R(A, B, C, D, E) met sleutels AB en BC, en functionele dependencies {DE→C, ABC→D}. Wat is de hoogste normaalvorm waarin dit schema zit? Beargumenteer je antwoord.

**Antwoord:** 3NF. Het is in 3NF want voor alle *non-triviale* FDs die eindigt in een *non-prime attribute*, dat is alleen ABC→ D, geldt de linkerkant een superset van een sleutel is. Het is niet in BCNF, want er is een *non-trivial* FD, nl. DE→C, waarvan de linkerkant niet een superset is van een sleutel.

**Vraag 3.** [5pt] Stel we hebben een relatie R(A, B, C, D) met *multi-valued dependency* AB ->> C en sleutels (*candidate keys*) A en B. Is deze relatie in 4NF? Laat zien waarom wel of niet.

**Antwoord:** Ja. Het is in 4NF want voor alle *non-triviale* MVDs geldt dat de linkerkant een superset is van een sleutel.

## **Deel 3 – Normalizatie** (10 punten)

**Vraag 4.** [3pt] Beschouw een relatie met schema R(A, B, C, D, E, F) en de set van functionele dependencies S = {A→B, BC→D, AD→E}. Bewijs met de regels van Armstrong (dus alleen IR1, IR2 en IR3) dat de functionele dependency AC→E in $S^+$ zit.

**Antwoord**: Het bewijs:

1. A → B (gegeven)
2. AC → ABC (augmentatie IR2, toegepast op 1, toevoeging van AC)
3. BC → D (gegeven)
4. ABC → AD (augmentatie IR2, toegepast op 3, toevoeging van A)
5. AC → AD (transitiviteit IR3, toegepast op 2 en 4)
6. AD → E (gegeven)
7. AC → E (transitiviteit, IR3, toegepast op 5 en 6)

**Vraag 5.** [3pt] Beschouw een relatie met schema R(A, B, C, D, E) met de set van functionele dependencies S = {B→CD, D→A}. Bepaal of de decompositie {AB, BCDE} de *lossless join property* heeft. Laat zien hoe je dit bepaald hebt.

**Antwoord:** Het heeft inderdaad de *lossless join property*. Omdat het hier een binaire decompositie betreft kunnen we de *NJB* property gebruiken om dit te beslissen (zie blz. 541 in boek). Die zegt dat de *lossless join property* getest kan worden door te kijken of B→A of B→CDE in $S^+$ zit. Aangezien $\{B\}^+ = \{B, C_{[B\rightarrow CD]}, D_{[B\rightarrow CD]}, A_{[D\rightarrow A]}\}$ geldt **niet** $\{C, D, E\} \subseteq \{B\}^+$ maar **wel** $\{A\} \subseteq \{B\}^+$ en dus zit B→A in $S^+$.

Het kan ook met het algemenere algoritme 15.3 bepaald worden (wat werkt voor decomposities in meer dan 2 tabellen), en dan is het antwoord als volgt:

1. Initiële invulling van de tabel:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| R1 | $a_1$ | $a_2$ | $b_{1,3}$ | $b_{1,4}$ | $b_{1,5}$ |
| R2 | $b_{2,1}$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |

2. Toepassing van B→CD

| | A | B | C | D | E |
|---|---|---|---|---|---|
| R1 | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $b_{1,5}$ |
| R2 | $b_{2,1}$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |

3. Toepassing van D→A:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| R1 | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $b_{1,5}$ |
| R2 | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |

4. Verdere toepassing van de dependencies levert geen verdere veranderingen op.
5. Er is inderdaad een rij met alleen a'tjes: die voor R2. Dus concluderen we dat de decompositie de *lossless/nonadditive join property* heeft.

**Vraag 6.** Beschouw een relatie met schema R(A, B, C, D, E, F, G) en de set van functionele dependencies S = {AB→CDE, BCD→F, F→B}.

a) [3pt] Bepaal een decomposite naar een BCNF schema met de *nonadditive join property* volgens algoritme 15.5 in het boek en slides. Geef per component ook de sleutels.

**Antwoord**: We bepalen eerst de sleutels van de start-tabel. Dat zijn in dit geval: {AFG, ABG}. Alle dependencies overtreden de BCNF regel, dus we kunnen kiezen met welke we willen beginnen om af te splitsen. Elke keuze is goed, maar we geven hier maar 1 uitwerking.

We splitsen AB→CDE af, en bepalen per component de sleutels:

- R1(A, B, C, D, E) met sleutels {AB} en lokale dependencies { AB→CDE }
- R2(A, B, F, G) met sleutels {AFG} en lokale dependencies { F→B }

Binnen R2 hebben we nu nog de dependencies { F→B } en deze overtreedt BCNF, dus splitsen we R2, resulterend in:

- R3(B, F) met sleutels { F } en lokale dependencies { F→B}
- R4(A, F, G) met sleutels { AFG } en lokale dependencies { }

Binnen deze relaties gelden er verder geen FDs die BCNF overtreden, en dus zijn we in BCNF.

b) [1pt] Is deze decompositie *dependency preserving*? Beargumenteer je antwoord.

**Antwoord**: Die dependencies die als lokale dependencies zijn overgebleven zijn: { AB→CDE, F→B }.

Daaruit kunenn we niet BCD → F afleiden, immers $\{BCD\}^+ = \{BCD\}$ en bevat geen F. De decompositie is dus niet dependency preserving.

## **Deel 4 – File organizatie** (5 punten)

**Vraag 7.** [5pt] Een PARTS bestand met Part# als de hash sleutel bevat records met de volgende Part# waardes (binair gepresenteerd): 100001, 000110, 111011, 111101 en 101010. Laad deze records in een lege *expandable hash* file gebaseerd op *extendible hashing*. Neem aan dat buckets maximal twee records kunnen bevatten. Laat zowel de globale als lokale diepte zien. Begin met globale diepte $d$=0. Gebruik de hash-functie $h(K) = K$ **mod** 16.

**Antwoord:** Merk op dat alleen de laatste vier bits van de sleutel relevant zijn vanwege de definitie van $h$. Deze zijn telkens in **vet** aangegeven. Van deze bits nemen we als eerste de *most significant bits* als index voor de directory.

Na 10**00001**, 00**0110**, 11**1011**, 11**1101**:

Directory (d = 1)

| Val | Ptr |
|---|---|
| 0 | B1 |
| 1 | B2 |

B1 (d'=1)

| |
|---|
| 10**0001** |
| 00**0110** |

B2 (d'=1)

| |
|---|
| 11**1011** |
| 11**1101** |

Na 10**1010** (B2 splitst in B2 en B3)

Directory (d = 2)

| Val | Ptr |
|---|---|
| 00 | B1 |
| 01 | B1 |
| 10 | B2 |
| 11 | B3 |

B1 (d'=1)

| |
|---|
| 10**0001** |
| 00**0110** |

B2 (d'=2)

| |
|---|
| 11**1011** |
| 10**1010** |

B3 (d'=2)

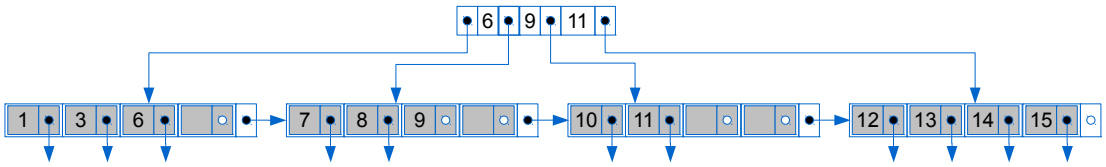
| |
|---|
| 10**1101** |
| |

## **Deel 5 – Indexering** (10 punten)

**Vraag 8.** [10pt] Een PARTS bestand met Part# als sleutelveld bevat records met de volgende Part# waardes: 11, 7, 6, 3, 9, 1, 10, 14, 15, 13, 12, 8. Stel dat deze zoekwaardes in deze volgorde ingevoegd worden in een lege $B^+$-boom van orde $p$=4 en $p_{leaf}$=4. Laat zien hoe de uiteindelijke $B^+$-boom er uitziet.

[11]

```
========
[7, 11]
========
[6, 7, 11]
========
[3, 6, 7, 11]
========
 . [7, 9, 11]
6
 . [3, 6]
========
 . [7, 9, 11]
6
 . [1, 3, 6]
========
 . [7, 9, 10, 11]
6
 . [1, 3, 6]
========
 . [10, 11, 14]
9
 . [7, 9]
6
 . [1, 3, 6]
========
 . [10, 11, 14, 15]
9
 . [7, 9]
6
 . [1, 3, 6]
========
 . [13, 14, 15]
11
 . [10, 11]
9
 . [7, 9]
6
 . [1, 3, 6]
========
 . [12, 13, 14, 15]
11
 . [10, 11]
9
 . [7, 9]
6
 . [1, 3, 6]
========
 . [12, 13, 14, 15]
11
 . [10, 11]
9
 . [7, 8, 9]
6
 . [1, 3, 6]
========
```




## **Deel 6 – Queryoptimalisatie** (20 punten)

**Vraag 9.** [2pt] Hoevel 'join orders' zijn er voor een query die 6 tabellen joint.

**Antwoord:** 6! = 720.

**Vraag 10.** [5pt] Een bestand van 4096 blocks moet gesorteerd worden met een beschikbare bufferruimte van 16 blocks. Hoeveel 'passes' zijn er nodig in de merge-fase van het 'external sort-merge' algoritme?

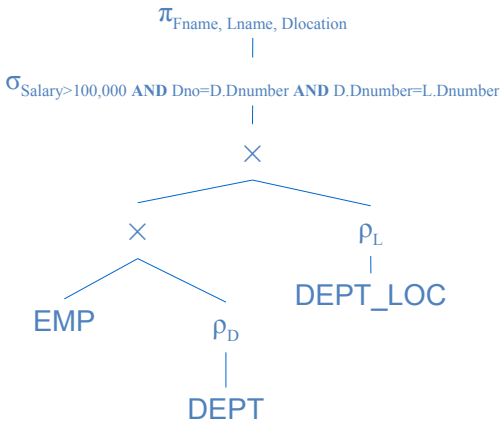
**Antwoord:** Na de eerste fase hebben we $\lceil 4096/16 \rceil = 256$ *initial runs*. Vervolgens kunnen we in elke pass 15 *runs mergen*. Dus zijn er $\lceil \log_{15}(256) \rceil = 3$ passes nodig.

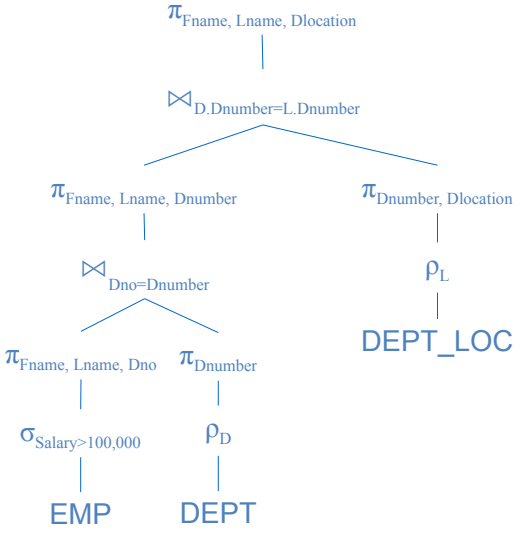
**Vraag 11.** [8pt] Beschouw de volgende SQL-query voor het schema van vraag 1:

> **SELECT** Fname, Lname, Dlocation
> **FROM** EMPLOYEE, DEPARTMENT D, DEPT_LOCATIONS L
> **WHERE** Salary>100,000 **AND** Dno=Dnumber **AND** D.Dnumber=L.Dnumber;

Geef twee verschillende *query trees* voor deze query.

**Antwoord:** Merk op dat we hier niet zomaar DEPARTMENT en DEPT_LOCATIONS mogen joinen omdat de headers kolommen gemeenschappelijk hebben. We zullen de RENAME operatie (ρ) moeten gebruiken om conflicten te vermijden.



**Vraag 12.** [5pt] In je relationele algebra is de join-operatie associatief (*associative*). Leg uit waarom dit belangrijk is voor query-optimalisatie.

**Antwoord:** Het is mogelijk door de associativiteitsregels om verschillende algebra-expressies te kiezen die een verschillende volgorde van joinen weergeven. Bijvoorbeeld de join $A \bowtie B \bowtie C$ kan uitgevoerd worden volgens $(A \bowtie B) \bowtie C$ of $A \bowtie (B \bowtie C)$. In het eerste geval worden eerst A en B gekoppeld, en het resultaat daarvan daarna met C, en in het tweede geval eerst B en C en daarna met A. Samen met commutativiteit (het feit dat $(A \bowtie B) \equiv (B \bowtie A)$) staat dit het systeem toe om de join-volgorde vrij te kiezen.

## Deel 7 – Databasetuning (5 punten)

**Vraag 13.** [5pt] Geef een mogelijke reden waarom het in een bepaalde situatie gunstig zou kunnen zijn om een *clustered index* om te zetten in een non-*clustered index*.

**Antwoord:** Voor een non-clustered index is niet nodig om de records in the tabel zelf in een bepaalde volgorder gesorteerd te houden. Dus de updates zullen efficiënter worden als de index non-clustered wordt.

## Deel 8 – Transacties (5 punten)

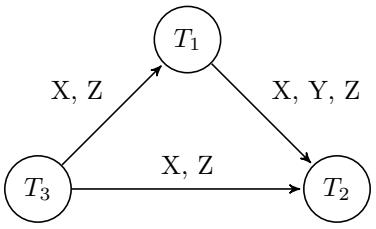
**Vraag 14.** [5pt] Beschouw de drie transacties $T_1$, $T_2$ en $T_3$, en het schedule S van deze transacties zoals beneden gegeven. Teken de *precedence graph* voor S, en bepaal of het wel of niet serialiseerbaar is. Als het schedule serialiseerbaar is, geef dan een equivalent *serial schedule*.

$T_1$: $r_1(Y)$; $w_1(Z)$; $w_1(X)$; $r_1(X)$;
$T_2$: $r_2(X)$; $w_2(Z)$; $w_2(Y)$;
$T_3$: $r_3(X)$; $w_3(Z)$; $w_3(X)$;

S: $r_3(X)$; $r_1(Y)$; $w_3(Z)$; $w_3(X)$; $w_1(Z)$; $w_1(X)$; $r_2(X)$; $w_2(Z)$; $w_2(Y)$; $r_1(X)$;

**Antwoord:** Links de lijst van de conflicterende operaties die een pijl in de *precedence* graaf veroorzaken, en rechts de graaf.

$r_3(X)$ .. $w_1(X)$ => $T_3 \rightarrow T_1$
$r_1(Y)$ .. $w_2(Y)$ => $T_1 \rightarrow T_2$
$w_3(Z)$ .. $w_1(Z)$ => $T_3 \rightarrow T_1$
$w_3(Z)$ .. $w_2(Z)$ => $T_3 \rightarrow T_2$
$w_3(X)$ .. $w_1(X)$ => $T_3 \rightarrow T_1$
$w_3(X)$ .. $r_2(X)$ => $T_3 \rightarrow T_2$
$w_3(X)$ .. $r_1(X)$ => $T_3 \rightarrow T_1$
$w_1(Z)$ .. $w_2(Z)$ => $T_1 \rightarrow T_2$
$w_1(X)$ .. $r_2(X)$ => $T_1 \rightarrow T_2$




Er is geen cykel in de graaf, dus het schedule is serialiseerbaar. Een mogelijke volgorde is $T_3 \rightarrow T_1 \rightarrow T_2$ (dus eerst alle operaties van $T_3$, dan die van $T_2$ en tenslotte die van $T_1$).

## **Deel 9 – Concurrency protocollen** (10 punten)

**Vraag 15.** [5pt] Beschouw het schedule S uit vraag 14. Stel we houden ons aan het *shared / exclusive locking scheme* en aan het ***strict*** *two-phase locking protocol.*

*a)* Geef aan na elke operatie welke locks er liggen.

**Antwoord:** We geven in een regel aan welke locks er liggen na de genoemde operatie, inclusief de locks aangevraagd voor die operatie die tussen haakjes staan. We nemen aan dat als een transactie klaar is er meteen een commit operatie volgt, en dus volgens *strict two-phase locking* alle locks vrijgeeft. Als een read-lock promoveert tot een write-lock geven we alleen het write-lock aan. De rode gevallen geven aan waar een gevraagd lock conflicteert met al aanwezige locks.

| Oper. | X | Y | Z |
|---|---|---|---|
| $r_3(X)$ | $(r_3)$ | | |
| $r_1(Y)$ | $r_3$ | $(r_1)$ | |
| $w_3(Z)$ | $r_3$ | $r_1$ | $(w_3)$ |
| $w_3(X)$ | $(w_3)$ | $r_1$ | $w_3$ |
| $c_3$ | | $r_1$ | |
| $w_1(Z)$ | | $r_1$ | $(w_1)$ |
| $w_1(X)$ | $(w_1)$ | $r_1$ | $w_1$ |
| $r_2(X)$ | $w_1$, $(r_2)$ | $r_1$ | $w_1$ |
| $w_2(Z)$ | $w_1$, $r_2$ | $r_1$ | $w_1$, $(w_2)$ |
| $w_2(Y)$ | $w_1$, $r_2$ | $r_1$, $(w_2)$ | $w_1$, $w_2$ |
| $c_2$ | $w_1$ | $r_1$ | $w_1$ |
| $r_1(X)$ | $w_1$, $(r_1)$ | $r_1$ | $w_1$ |
| $c_1$ | | | |

*b)* Bepaal of S wordt toegestaan onder de genoemde protocollen. Zo nee, geef aan waarom niet.

**Antwoord:** Het is niet toegestaan. Vlak voor $r_1(Z)$ conflicteren op Z de locks $w_3$ en $r_1$.

**Vraag 16.** [5pt] Beschouw opnieuw het schedule S uit vraag 14. Bepaal of deze toegestaan is volgens het *basic timestamp ordering* (TO) algoritme. Laat zien hoe je dit hebt bepaald.

**Antwoord:**
We volgen de executie van S. De timestamps van de transacties worden aangenomen te zijn bepaald door hun eerste operatie, dus $TS(T_1) = 2$, $TS(T_2) = 7$ en $TS(T_3) = 1$. Na elke stap registreren we de read timestamp (Rd_ts(*D*)) en write timestamp (Wr_ts(*D*)) van elk data item (*D*=*X,Y,Z*).

| T | Operat. | Rd_ts(*X*) | Wr_ts(*X*) | Rd_ts(*Y*) | Wr_ts(*Y*) | Rd_ts(*Z*) | Wr_ts(*Z*) |
|---|---|---|---|---|---|---|---|
| 1 | $r_3(X)$ | 1 | - | - | - | - | - |
| 2 | $r_1(Y)$ | 1 | - | 2 | - | - | - |
| 3 | $w_3(Z)$ | 1 | - | 2 | - | - | 1 |
| 4 | $w_3(X)$ | 1 | 1 | 2 | - | - | 1 |
| 5 | $w_1(Z)$ | 1 | 1 | 2 | - | - | 2 |
| 6 | $w_1(X)$ | 1 | 2 | 2 | - | - | 2 |
| 7 | $r_2(X)$ | 7 | 2 | 2 | - | - | 2 |
| 8 | $w_2(Z)$ | 7 | 2 | 2 | - | - | 7 |
| 9 | $w_2(Y)$ | 7 | 2 | 2 | 7 | - | 7 |
| 10 | $r_1(X)$ | 7 | 2 | 2 | 7 | - | 7 |

Er is op geen enkel moment een conflict, dus dit schedule is toegestaan.

## **Deel 10 – Recovery protocollen** (8 punten)

**Vraag 17.** [8pt] Beschouw de volgende geabstraheerde *system log* na een crash:

[**start**, T1] [**write**, T1, *D*, 15, 20] [**start**, T2] [**write**, T2, *B*, 15, 12] [**start**, T4] [**write**, T4, *B*, 25, 15] [**start**, T3] [**write**, T3, *A*, 20, 30] [**write**, T4, *A*, 10, 20] [**commit**, T1] [**commit**, T4] [**checkpoint**] [**commit**, T3] [ **write**, T2, *D*, 20, 25]

De log-records hebben de volgende betekenis:
[**start**, *Ti*] = start van transactie *Ti*
[**write**, *Ti*, *X*, *old*, *new*] = update van data item *X* van waarde *old* naar *new*
[**commit**, *Ti*] = commit van transactie *Ti*
[**checkpoint**] = checkpoint

Veronderstel dat we voor de recovery de procedure RIU_M (UNDO/REDO with checkpoints) volgen (zie blz. 797). Welke operaties worden dan in welke volgorde uitgevoerd? Gebruik dezelfde notatie als voor de getoonde system log.

**Antwoord**: We volgen de beschreven procedure van drie stappen:

1. We bepalen eerst de twee lijsten:
   a. Committed transactions since last checkpoint: T3
   b. Active transactions: T2
2. Vervolgens doen we een undo voor de write operaties van de actieve transacties, in de omgekeerde volgorde van waarin ze in de log staan. Dit zijn in de log de volgende operaties voor T2 en T4:

   [**write**, T2, *B*, 15, 12] [ **write**, T2, *D*, 20, 25]

   Dus dat keren we nu om, en draaien daarbij ook oude en nieuwe waarde om:

[**write**, T2, D, 25, 20] [**write**, T2, B, 12, 15]

3. Vervolgens doen we een redo voor write operaties van de committed transactions, (T3) in de volgorde waarin ze in de log voorkomen na de laatste checkpoint. Dat zijn de volgende operaties:

   - geen -

In totaal levert dat dus het volgende UNDO/REDO schedule:

[**write**, T2, D, 25, 20] [**write**, T2, B, 12, 15]

## **Deel 11 – Semantic Web en RDF** (9 punten)

**Vraag 18.** [4pt] Beschouw de volgende RDF graaf:

```
ex:VegetarianPizza rdfs:subClassOf ex:Pizza .
ex:NonVegetarianPizza rdfs:subClassOf ex:Pizza .
ex:QuattroFormaggi rdf:type ex:VegetarianPizza .
```

Geef **alle** triples die hier uit afgeleid worden door de RDFS inferentieregels (*inference rules*). Gebruik eventueel voor de efficiëntie afkortingen voor de URIs zoals ex:VegP, rdfs:sCO, ex:QF, etc.

**Antwoord:**
Regel: reflexiviteit van rdfs:subClassOf

```
ex:VegP rdfs:sCO ex:VegP
ex:Pizza rdfs:sCO ex:Pizza
ex:nonVegP rdfs:sCO ex:nonVegP
```

Regel: transitiviteit van rdfs:subClassOf

–

Regel: type overerving

```
ex:QF rdf:type ex:Pizza .
```

Regel: property type

```
rdfs:sCO  rdf:type  rdf:Property .
rdf:type  rdf:type  rdf:Property .
```

**Vraag 19.** [5pt] Beschouw de volgende RDF graaf:

```
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
                   rdfs:label "Etna" ;
                   p:location dbpedia:Italy .
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano ;
                   p:location dbpedia:United_States .
dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;
                   rdfs:label "Beerenberg"@en ;
                   p:location dbpedia:Norway .
```

Formuleer in SPARQL de volgende query over dergelijke RDF data: *"Op welke locaties*

*staat een vulkaan met namen in minstens twee verschillend talen?"* Maak eventuele aannames over de structuur van de RDF-graaf expliciet.

**Antwoord:**

```
SELECT ?loc WHERE {
   ?v rdf:type umbel-sc:Volcano .
   ?v p:location ?loc .
   ?v rdfs:label ?lab1 .
   ?v rdfs:label ?lab2 .
   FILTER (LANG(?lab1) != LANG(?lab2) )
}
```

**Einde van het tentamen**