TECHNISCHE UNIVERSITEIT DELFT
Faculteit Elektrotechniek, Wiskunde en Informatica

Uitwerkingen van het Tentamen

TI2505/TI2500 Informatie- en Datamodellering

Dinsdag, 28 januari 2014
14u00-17u00

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**.

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, dept-sdate, 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) Geef de namen van de departementen waarvan de manager (aangegeven door Mgr_ssn) in een ander departement zit dan waar hij of zij leiding aan geeft.

Antwoord:
DEPMGR := DEPARTMENT $\bowtie_{Mgr\_ssn=Ssn}$ EMPLOYEE;
RESULT := $\pi_{Dname}(\sigma_{Dnumber \neq Dno}(DEPMGR))$;

Uitleg: DEPMGR bevat de combinaties van departementen en werknemers zodat de werknemer de manager is van het departement. Vervolgens selecteren we uit DEPMGR de combinaties waar het departnummer van het departement en de manager verschillen zijn. Tenslotte projecteren we de overgebleven records op de naam van het departement.

b) Geef de namen van de departementen waar de meerderheid van de werknemers vrouw is.

Antwoord:
MANCNT := $\rho_{(Dno,\ ManCnt)}({}_{Dno}\mathcal{F}_{\ COUNT\ Ssn}(\sigma_{Sex=M}(EMPLOYEE)))$;
FEMCNT := $\rho_{(Dno,\ FemCnt)}({}_{Dno}\mathcal{F}_{\ COUNT\ Ssn}(\sigma_{Sex=F}(EMPLOYEE)))$;
FEMDEPT := $\pi_{Dno}(\sigma_{ManCnt\ <\ FemCnt}(MANCNT * FEMCNT))$;
RESULT := $\pi_{Dname}(FEMDEPT * DEPARTMENT)$

Uitleg: We nemen aan dat in elk departement minstens 1 vrouw en 1 man werkt. In MANCNT worden het aantal mannelijke werknemers per departement bepaald. In FEMCNT het aantal vrouwelijke werknemers per departement. Voor FEMDEPT nemen we de *natural join* en dus joinen we op de kolom Dno. Vervolgens selecteren we de records waar het totaal aantal werknemers minder is dan twee maal het aantal vrouwelijke werknemers. Tenslotte, voor RESULT *joinen* we met DEPARTMENT om de naam van het departement erbij op te zoeken en daar op het eind op te projecteren.

Vraag 2. Uitgaande van het relationele schema van de vorige vraag, geef een voorbeeld van een algebra expressie waarin de *division operator* (÷) voorkomt, en leg de betekenis van de expressie uit.

Antwoord: Bijvoorbeeld:
DEPT3PROJ := $\rho_{Pno}(\pi_{Pnumber}(\sigma_{Dnum=3}(PROJECT)))$;
WORKS := $\pi_{Essn,\ Pno}(WORKS\_ON)$;
RESULT := WORKS ÷ DEPT3PROJ

Uitleg: In DEP3PROJ selecteren we alle projecten van departement 3. In WORKS zit wie voor welk project werkt. Tenslotte in RESULT wordt dan bepaalt wie voor alle projecten van departement 3 werkt.

Deel 2 – Normaalvormen (10 punten)

Vraag 3. Beschouw een relatie met schema R(A, B, C, D, E) met kandidaatsleutels (*candidate keys*) AB en BC, en waar bovendien de functionele dependencies A→BC en AC→DE gelden. Leg uit waarom een dergelijk relatie niet kan bestaan.

Antwoord: Als AB en CD candidate keys zijn, dan geldt AB → ABCDE en CD→ABCDE. Maar dan volgt A^+ = {A, B, C, D, E}. En dus is AB niet minimal want er is een kleinere deelverzameling die ook sleutel is, en AB kan dus geen *candidate key* zijn.

Vraag 4. Stel we hebben een relatie R(A, B, C) met *join dependency* JD(AB, AC, BC) en sleutels (*candidate keys*) A en B. Is deze relatie in 5NF volgens de definitie gegeven tijdens de les in de slides? Laat zien waarom wel of niet.

Antwoord: We beginnen met decompositie volgens de JD die we willen afleiden:
{{A, B}, {A, C}, {B, C}}.
Vervolgens passen we de regel toe:
{{A, B, C}, {B, C}} want CK A in {A, B} ∩ {A, C}
{{A, B, C}} want CK B in {A, B, C} ∩ {B, C}
Dit is de hele relatie, dus de JD overtreed niet 5NF.

Deel 3 – Normalizatie (10 punten)

Vraag 5. Beschouw een relatie met schema R(A, B, C, D, E, F) en de set van functionele dependencies S = {CF→E , C→D, CD→F}. Bewijs met de regels van Armstrong (dus alleen IR1, IR2 en IR3) dat de functionele dependency C→F in S^+ zit.

Antwoord: Het bewijs:
1. C → D (gegeven)

2. C → CD (augmentatie, IR2, toegepast op 1)
3. CD → F (gegeven)
4. C → F (transitiviteit, IR2, toegepast op 2 en 3)

Vraag 6. Beschouw een relatie met schema R(A, B, C, D, E) met de set van functionele dependencies S = {A→BCDE, B→A}. Bepaal of de decompositie {AB, ACD, BCE} de *lossless join property* heeft. Laat zien hoe je dit bepaald hebt.

We passen algoritme 15.3 toe, en dan is het antwoord als volgt:

1. Initiele invulling van de tabel:

	A	B	C	D	E
R1	a_1	a_2	$b_{1,3}$	$b_{1,4}$	$b_{1,5}$
R2	a_1	$b_{2,2}$	a_3	a_4	$b_{2,5}$
R3	$b_{1,1}$	a_2	a_3	$b_{3,4}$	a_5

2. Toepassing van A→BCDE

	A	B	C	D	E
R1	a_1	a_2	a_3	a_4	$b_{2,5}$
R2	a_1	a_2	a_3	a_4	$b_{2,5}$
R3	$b_{1,1}$	a_2	a_3	$b_{3,4}$	a_5

3. Toepassing van B→A:

	A	B	C	D	E
R1	a_1	a_2	a_3	a_4	$b_{2,5}$
R2	a_1	a_2	a_3	a_4	$b_{2,5}$
R3	a_1	a_2	a_3	$b_{3,4}$	a_5

4. Toepassing van A→BCDE:

	A	B	C	D	E
R1	a_1	a_2	a_3	a_4	a_5
R2	a_1	a_2	a_3	a_4	a_5
R3	a_1	a_2	a_3	a_4	a_5

5. Verdere toepassing van de dependencies levert geen verdere veranderingen op. Dat kan ook niet, want elke kolommen bevat intern hetzelfde symbool.
6. Er is inderdaad een rij met alleen a'tjes, in zowel R1, R2 als R3, dus concluderen we dat de decompositie de *lossless/nonadditive join property* heeft.

Vraag 7. 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) Bepaal een *dependency preservering* decomposite naar een 3NF schema met de

nonadditive join property volgens algoritme 15.6 in het boek en slides. Geef per component ook de sleutels.

Antwoord: We beginnen met het bepalen van een minimal cover. Beschouw de gegeven set van dependencies:

AB→CDE
BCD→F
F→B

We vereenvoudigen eerst door de rechterkant unair te maken:

AB→C
AB→D
AB→E
BCD→F
F→B

Dan kijken of een linkerkant kleiner kunnen maken. Dat is niet het geval.
Dan kijken we of een dependency kunnen weglaten. Dat is ook niet het geval.
Dit is dus al een *minimal cover*.

We laten nu zien hoe we alle *candidate keys* kunnen afleiden. (Dit is strict genomen niet nodig voor de opgave.) Dit doen we hier door een verzameling van superkeys bij te houden, waarin we telkens deze vervangen door kleinere superkeys als we die kunnen afleiden met de dependencies. We beginnen met de triviale *superkey* met alle kolommen en werken de boom uit van alle verkleiningen. De superkeys die niet kunnen worden verkleind zijn in **bold** aangegeven:

ABCDEFG
- ABDEFG (want AB→C)
  - ABEFG (want AB→D)
    - **ABFG** (want AB→E)
    - **AEFG** (want F→B)
  - ABDFG (want AB→E)
    - ABFG (want AB→E) (al eerder afgeleid)
    - **ADFG** (want F→B)
  - **ADEFG** (want F→B)
- ABCEFG (want AB→D)
  - ABEFG (want AB→C) (al eerder afgeleid)
  - ABCFG (want AB→E)
    - ABFG (want AB→C) (al eerder afgeleid)
    - **ACFG** (want F→B)
  - **ACEFG** (want F→B)
- ABCDFG (want AB→E)
  - ABDFG (want AB→C) (al eerder afgeleid)
  - ABCFG (want AB→D) (al eerder afgeleid)
  - ABCDG (want BCD→F)
    - ABDG (want AB→C)
      - **ABG** (want AB→D)
    - ABCG (want AB→D)
      - ABG (want AB→C) (al eerder afgeleid)
  - **ACDFG** (want F→B)
- ABCDEG (want BCD→F)

ABDEG (want AB→C)
ABEG (want AB→D)
ABG (want AB→E) (al eeder afgeleid)
ABDG (want AB→E) (al eerder afgeleid)
ABG (want AB→D) (al eerder afgeleid)
ABCEG (want AB→D)
ABEG (want AB→C) (al eerder afgeleid)
ABCG (want AB→E) (al eerder afgeleid)
ABCDG (want AB→E) (al eerder afgeleid)
ACDEFG (want F→B)

De lijst van niet-verkleinbare superkeys is dus:
ABFG, AEFG, ADFG, ADEFG, ACFG, ACEFG, ABG, ACDFG, ACDEFG
Als we nu de superkeys verwijderen waarvoor er een kleinere in deze lijst is, krijgen we:
AEFG, ADFG, ACFG, ABG
Dit zijn nu dus alle candidate keys van deze relatie.

We hebben nu een minimal cover, en kunnen het algoritme gaan toepassen. Als we de dependencies uit het minimal cover clustern op hun linkerkant krijgen we:
AB→CDE
BCD→F
F→B
En dus de relaties:
R1(A, B, C, D, E) met CKs AB
R2(B, C, D, F) met CKs BCD
R3(B, F) met CKs F
Vervolgens moeten we controleren of er ergens een van de oorspronkelijke CKs voorkomt. Dat kan eenvoudig als we deze al bepaalt hebben, maar het is voldoende om te controleren of in een van de tabellen de kolommen een *superkey* vormen. Met andere woorden, is 1 van de volgende beweringen waar: (1) ABCDE → ABCDEFG, (2) BCDEF → ABCDEFG of (3) BF → ABCDEFG. Geen van deze volgt uit de gegeven dependencies, dus we moeten een relatie toevoegen die bestaat uit een *candidate key* die je zelf mag kiezen. Als je deze al bepaalt hebt je er een selecteren, of anders algoritme 5.2 gebruiken om er eentje af te leiden als volgt:

1. Je begint met ABCDEFG
2. Je kijkt of je een attribuut kunt weglaten en nog steeds een superkey hebt. Dat is niet het geval voor A, maar wel voor B. Immers $ACDEFG^+ = ABCDEFG$
3. Dus we hebben nu ACDEFG.
4. Daaruit kunnen we als eerste C weglaten, en hebben dan nog steeds een superkey. Immers $ADEFG^+ = ABCDEFG$
5. Dus we hebben nu ADEFG.
6. Daaruit kan als eerste D worden weggelaten. Immers $AEFG^+ = ABCDEFG$.
7. Dus we hebben nu AEFG.
8. Daar kan niets uit worden weggelaten zonder dat het geen superkey meer is. Dus dit is een CK.

Dus we voegen de volgende relatie toe aan de decompositie:
R4(A, E, F, G) met CK AEFG

Tenslotte moeten we controleren of een relatie niet een subrelatie is van een ander (of alle

kollommen als samen voorkomen in een andere relatie). Dat is het geval want de kolommen van R3 komen voor in R2, dus verwijderen we R3.

b) Is deze decompositie in BCNF? Beargumenteer je antwoord.

Antwoord: Ja. We controleren welke dependencies er overal lokaal liggen:
R1(A, B, C, D, E) met AB→CDE en CK AB
R2(B, C, D, F) met BCD→F, F→B en CKs BCD en CDF
R4(A, E, F, G) zonder dependencies en CK AEFG
In R2 overtreedt F→B BCNF want F is geen superkey.

Deel 4 – File organizatie (5 punten)

Vraag 8. Een PARTS bestand met Part# als de hash sleutel bevat records met de volgende Part# waardes (binair gepresenteerd): 101101, 111011, 001110, 101111 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 de structuur zien bij elke stap, evenals de globale en lokale diepte. Begin met globale diepte d=0. Gebruik de hash-functie $h(K) = K$ **mod** 16.

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**1101** en 11**1011**:
Directory (d = 1)

Val	Ptr
0	B1
1	B2

B1 (d'=1)

B2 (d'=1)

10**1101**
11**1011**

Na 00**1110** (B2 splitst in B2 en B3)
Directory (d = 2)

Val	Ptr
00	B1
01	B1
10	B2
11	B3

B1 (d'=1)

B2 (d'=2)

11**1011**

B3 (d'=2)

10**1101**
00**1110**

Na 10**1111** (B3 splitst in B3 en B4)

Directory (d = 3)

Val	Ptr
000	B1
001	B1
010	B1
011	B1
100	B2
101	B2
110	B3
111	B4

B1 (d'=1)

B2 (d'=2)

111**011**

B3 (d'=3)

10**1101**

B4 (d'=3)

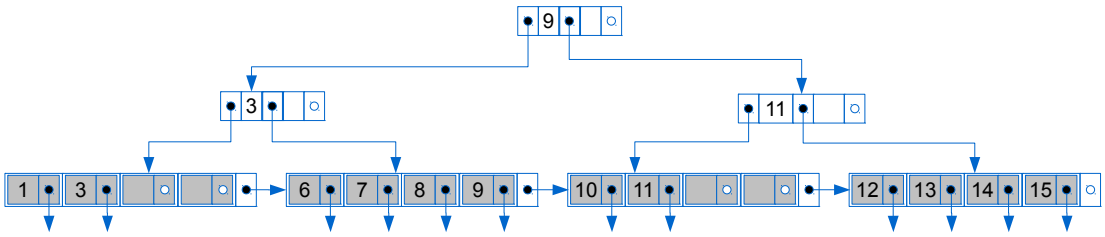
001**110**
101**111**

Deel 5 – Indexering (10 punten)

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

```
[12]
========
[8, 12]
========
[3, 8, 12]
========
[3, 8, 9, 12]
========
 . [8, 9, 12]
3
 . [1, 3]
========
 . [8, 9, 10, 12]
3
 . [1, 3]
========
 . [10, 11, 12]
9
 . [8, 9]
3
 . [1, 3]
========
 . [10, 11, 12]
```

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



Deel 6 – Queryoptimalisatie (20 punten)

Vraag 10. Moet er bij de uitvoering van een PROJECT operatie π altijd dubbele records verwijderd worden? Als wel, leg uit waarom, en als niet, leg uit wanneer dan wel en wanneer niet.

Antwoord: Nee, dat hoeft niet als de kollommen waarop geprojecteerd wordt een *superkey* zijn van de tabel (of in andere woorden: als het als deelverzameling een *candidate key* bevat). Immers, een *superkey* bevat per definitie altijd een unieke

combinatie binnen een tabel / relatie, en dus zullen er geen dubbelen zijn als de projectie dit als deel bevat.

Vraag 11. In de *merging phase* van de *sort-merge strategy* voor *external sort* worden telkens **min**(n_B – 1, n_R) *runs* gecombineerd in een enkele *run*. Waarom combineren we niet altijd n_B – 1 runs?

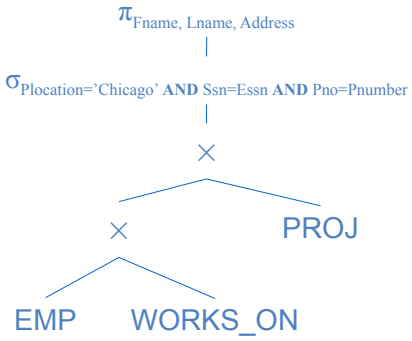
Antwoord: Als n_R kleiner is dan n_B -1, dan is er meer buffer-ruimte beschikbaar dan er runs (gesorteerde subfiles) zijn en kunnen ze dus allemaal ineens gemerged worden. Maar het aantal runs wat dan gemerged wordt is dan dus n_R (alle runs op dat moment) en niet n_B - 1.

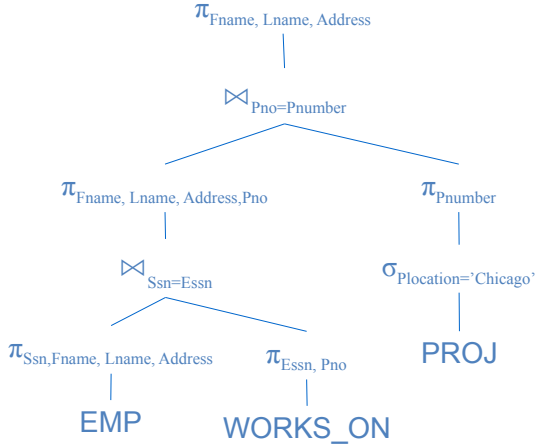
Vraag 12. Beschouw de volgende SQL-query voor het schema van vraag 1:

```
SELECT Fname, Lname, Address
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE Plocation='Chicago' AND Ssn=Essn AND Pno=Pnumber;
```

Geef twee verschillende *query trees* voor deze query.

Antwoord:





Vraag 13. Geef een tegenvoorbeeld van de regel $\pi_L(R - S) \equiv \pi_L(R) - \pi_L(S)$, dwz. een voorbeeld dat laat zien dat deze regel niet altijd geldt.

Antwoord:

R:

A	B
1	2

S:

A	B
1	3

$\pi_A(R - S)$

A
1

$\pi_A(R) - \pi_A(S)$

A

Deel 7 – Databasetuning (5 punten)

Vraag 14. Geef een mogelijke reden waarom het in een bepaalde situatie gunstig zou kunnen zijn om een *non-clustered index* om te zetten in een *clustered index*.

Antwoord: Range-queries worden veel efficiënter. Of als we vaak op basis van de geïndexeerde kolom(men) zoeken en er dan vaak veel records per specifieke waarde kunnen zijn. Tenslotte worden queries waarin een GROUP BY of ORDER BY gebeurt op basis van de geïndexeerde kolom(men) sneller.

Deel 8 – Transacties (5 punten)

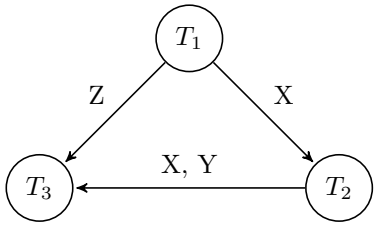
Vraag 15. Beschouw de drie transacties T_1, T_2 en T_3, en het schedule S van deze transacties zoals beneden gegeven. Teken de *precedence graph* voor S, en bepaal of het wel of niet serialiseerbaar is. Als het schedule serialiseerbaar is, geef dan een equivalent *serial schedule*.

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

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

Antwoord: We geven de graaf, en waarom er welke pijl moet liggen.

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



Een mogelijk equivalent serial-schedule is: $T_1 \rightarrow T_2 \rightarrow T_3$.

Deel 9 – Concurrency protocollen (10 punten)

Vraag 16. Beschouw het schedule S uit vraag 15. 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 aangenomen is dat alle locks pas op het eind worden vrijgegeven is dat ook goedgerekend.) Als een read-lock promoveert tot een write-lock geven we alleen het write-lock aan.

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

b) Bepaal of S wordt toegestaan onder de genoemde protocollen.

Antwoord: Het is niet toegestaan. Vlak voor $r_3(Y)$ conflicteren op Y de locks w_2 en r_3.

Vraag 17. Beschouw opnieuw het schedule S uit vraag 15. Bepaal of deze toegestaan is volgens het *basic timestamp ordering* (TO) algoritme.

Antwoord:
We volgen de executie van S. De timestamps van de transacties worden aangenomen te zijn bepaald door hun eerste operatie, dus $TS(T_1) = 2$, $TS(T_2) = 1$ en $TS(T_3) = 6$. 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	$r_2(Y)$	-	-	1	-	-	-
2	$r_1(X)$	2	-	1	-	-	-
3	$w_2(Y)$	2	-	1	1	-	-
4	$r_1(Z)$	2	-	1	1	2	-
5	$w_1(Z)$	2	-	1	1	2	2
6	$r_3(Z)$	2	-	1	1	6	2
7	$r_2(X)$	2	-	1	1	6	2

8	$w_3(Z)$	2	-	1	1	6	6
9	$r_3(Y)$	2	-	6	1	6	6
10	$w_2(X)$	2	**1**	6	1	6	6
11	$r_3(X)$	6	1	6	1	6	6
12	$w_3(Y)$	6	1	6	6	6	6

Op T=10 is er een conflict want Rd_ts(X) = 2 > $TS(T_2)$ = 1. Dus dit schedule is niet toegestaan.

Deel 10 – Recovery protocollen (8 punten)

Vraag 18. 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 RDU_M (NO-UNDO/REDO with checkpoints) volgen (zie blz. 794). Welke operaties worden dan in welke volgorde uitgevoerd? Gebruik dezelfde notatie als voor de getoonde *system log*.

Antwoord: We volgen de beschreven procedure van twee stappen:

1. We bepalen eerst de twee lijsten:
   a. Gecommitteerde transacties sinds laatste checkpoint: T3
   b. Actieve transacties: T2
2. Vervolgens doen we een redo voor de write operaties van de gecommitteerde transacties in de volgorde waarin ze in de log staan. Dit zijn in de log de volgende operaties voor T3:

   [**write**, T3, *A*, 20, 30]

   Merk op dat we de ook de operaties voor de checkpoint opnieuw moeten doen, want onder *deferred update* zijn deze updates nog niet op schijf gezet.

Deel 11 – Semantic Web en RDF (9 punten)

Vraag 19. Beschouw de volgende RDF graaf:

```
ex:Van rdfs:subClassOf ex:MotorVehicle .
ex:MiniVan rdfs:subClassOf ex:Van .
ex:MyRedVWT3 rdf:type ex:MiniVan .
```

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:V, rdfs:sCO, ex:MotV, etc.

Antwoord:

Regel: reflexiviteit van rdfs:subClassOf

```
ex:V rdfs:sCO ex:V
ex:MotV rdfs:sCO ex:MotV
ex:MinV rdfs:sCO ex:MinV
```

Regel: transitiviteit van rdfs:subClassOf

```
ex:MinV rdfs:sCO ex:MotV
```

Regel: type overerving

```
ex:MRVWT3 rdf:type ex:V .
ex:MRVWT3 rdf:type ex:MotV .
```

Regel: property type

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

Vraag 20. 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: *"Welke vulkanen in de VS (United States) en het VK (United Kingdom/Verenigd Koninkrijk) hebben een niet-Engelstalige naam?"* Maak eventuele aannames over de structuur van de RDF-graaf expliciet.

Antwoord:

```
SELECT ?v WHERE {
   ?v rdf:type umbel-sc:Volcano .
   ?v rdfs:label ?lab .
   FILTER (LANG(?lab) != en )
   { ?v p:location dbpedia:United_States }
    UNION
   { ?v p:location dbpedia:United_Kingdom }
                                  }
```

Einde van het tentamen