TECHNISCHE UNIVERSITEIT DELFT
Faculteit Elektrotechniek, Wiskunde en Informatica

Uitwerkingen van het tentamen

TI2500 Informatie en Datamodellering

Maandag, 16 april 2012
14u00-17u00

Dit tentamen bestaat uit 10 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	10
2	10
3	10
4	5
5	10
6	20
7	5
8	10
9	10
10	10
Totaal	100

Het gebruik van het boek "*Database Systems: Global Edition, 6/E*" door Ramez Elmasri en Shamkant Navathe is toegestaan, evenals de slides die voor het vak op Blackboard zijn gezet.

Vul je naam, studienummer en studierichting in op ieder antwoordblad.

Deel 1 – Relationele Algebra (10 punten)

Vraag 1. Wat is 'union compatibility' en waarom is dit een voorwaarde voor de UNION, INTERSECTION en DIFFERENCE?

ANTWOORD:
Union compatibility is het feit dat twee relaties dezelfde attributen hebben met dezelfde attribuutdomeinen.
Dit is een voorwaarde voor de UNION omdat anders het resultaat niet een correcte relatie is waarin alle tuples dezelfde structuur moeten hebben. Voor de INTERSECTIE is het een vorwaarde omdat in relaties met verschillen attributen nooit gemeenschappelijke records kunnen zitten en dus zou de doorsnede altijd leeg zijn. Voor de DIFFERENCE zou om dezelfde reden het resultaat altijd gelijk zijn aan de eerste operand.

Vraag 2. 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 alle employees in departement 5 die meer dan 10 uur per week werken op het ProductX project.
b) Voor elk project, geef de projectnaam en het totaal aantal uren per week (van alle employees) besteed aan dat project.

ANTWOORD

a) $DEPT5 \leftarrow \sigma_{dno=5}(EMPLOYEE)$
$XPROJ \leftarrow \sigma_{Pname="ProductX"}(PROJECT)$
$TENHR \leftarrow \sigma_{Hours>10}(WORKS\_ON)$
$RESULT \leftarrow \pi_{Fname,Minit,Lname}(DEPT5 \bowtie_{Ssn=Essn} XPROJ \bowtie_{Pnumber=Pno} TENHR)$

b) $RESULT \leftarrow \pi_{Pname,\ Hours}({}_{Pnumber}\Im_{SUM\ Hours}(WORKS\_ON) \bowtie_{Pno=Pnumber} PROJECT)$
Of (als je aanneemt dat Pname ook key is)
$RESULT \leftarrow {}_{Pname}\Im_{SUM\ Hours}(WORKS\_ON \bowtie_{Pno=PNumber} PROJECT)$

Deel 2 – Normaalvormen (10 punten)

Vraag 3. Geef de definities van de zowel de 'nonadditive join property' en de 'dependency preservation property' en leg uit waarom deze wenselijk zijn voor relationele schemas.

ANTWOORD

De 'nonadditive join property' is dat als we een decompositie toepassen op een relatie en de resulterende relaties daarna weer met elkaar combineren via de natural join we geen extra tupels ('spurious tuples') krijgen ten opzichte van de oorspronkelijke relatie. Oftewel, voor een decompositie $D=\{R1, .., Rn\}$ voor een relatie R geldt voor elke instantie r van R dat $*(\pi_{R1}(r), \ldots, \pi_{Rn}(r)) = r$. De Dit is wenselijk omdat anders de decompositie niet lossless is, i.e., we raken informatie kwijt en kunnen blijkbaar niet alle oorspronkelijke informatie in dit schema representeren.
De 'dependency preservation property' (DPP) zegt dat elke functionele dependency over de oorspronkelijke relatie logisch volgt uit de vereniging van de functionele dependencies die gelden over elk van de relaties in de decompostie. Oftewel als F de set van dependencies van R is dan geldt voor de decompositie $D=\{R1, .., Rn\}$ dat $(F^+[R1] \cup \ \ldots \cup F^+[Rn])^+ = F^+$. Dit is wenselijk omdat het betekent dat het bewaken van alle dependencies door de database beperkt kan worden tot de lokale dependencies.

Vraag 4. Beschouw een relatie met schema R(A, B, C, D, E) met sleutels BE, AB en DE, en de set van functionele dependencies { AB→CDE, DE→ABC, B→D }. In welke normaalvorm is dit schema? Beargumenteer je antwoord.

ANTWOORD

We zijn in 3NF want bij alle dependencies behalve B→D is de linkerkant een superkey, en B→D eindigt in een key attribuut, dus kan niet 3NF overtreden. Maar we zijn niet in BCNF want daarvoor zou ook voor deze dependency moeten gelden dat de linkerkant een superkey is. Dus: 3NF.

Vraag 5. Stel we hebben een relatie R(A, B, C) met 'multivalued dependencies' A↠B en A↠C. Als we weten dat deze relatie in een bepaalde state de volgende rijen bevat: (a1, b1, c1), (a1, b2, c2). Welke rijen moeten er dan noodzakelijkerwijs nog meer in die state zitten?

ANTWOORD

De tupels (a1, b1, c2) en (a1, b2, c1).

Deel 3 – Normalizatie (10 punten)

Vraag 6. Beschouw een relatie met schema R(A, B, C, D, E, F, G) en de set van functionele dependencies S = { AB→C, A→DE, B→F, F→G }. Bereken $\{A,B\}^+$.

ANTWOORD
$\{A, B\}^+ = \{A, B, _{[start]}, C _{[want\ AB\rightarrow C]}, D, E _{[want\ A\rightarrow DE]}, F _{[want\ B\rightarrow F]}, G _{[want\ F\rightarrow G]} \}$

Vraag 7. Beschouw een relatie met schema R(A, B, C, D, E) en de set van functionele dependencies S = { AB→C, A→D, B→E }. Bewijs met de regels van Armstrong (dus alleen IR1, IR2 en IR3) dat de functionele dependency ABC→E in S^+ zit.

ANTWOORD
(1) B→E want gegeven
(2) ABC→B want trivial; (IR1)
(3) ABC→E vanwege transitiviteit en (1) en (2); (IR3)

Vraag 8. Beschouw een relatie met schema R(A, B, C, D, E) met de set van functionele dependencies S = {A→C, AC→D, AD→E}. Bepaal of de decompositie {ABC, ADE} de 'lossless join property' heeft. Laat zien hoe je dit bepaald hebt.

ANTWOORD
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 NJB geldt als A→DE of A→BC in S^+ moet zitten. Aangezien $\{A\}^+ = \{A, C _{[A\rightarrow C]}, D _{[AC\rightarrow D]}, E _{[AC\rightarrow E]} \}$ geldt A→DE, en dus de NJB property en heeft de decompositie de lossless join property.

Als je het algemene algoritme (Alg. 15.3 op blz. 538/9) hebt toegepast is het antwoord als volgt:

1. Initiele invulling van de tabel:

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

2. Toepassing van A→C

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

3. Toepassing van AC→D:

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

4. Toepassing van AD→E:

	A	B	C	D	E
R1	a_1	a_2	a_3	a_4	**a_5**
R2	a_1	$b_{2,2}$	a_3	a_4	**a_5**

5. Verdere toepassing van de dependencies levert geen verdere veranderingen op.
6. Er is inderdaad een rij met alleen a'tjes, nl. R1, dus concluderen we dat de decompositie de 'lossless join property' heeft.

Vraag 9. Beschouw een relatie met schema R(A, B, C, D, E, F, G) en de set van functionele dependencies S = {A→B, A→C, B→D, B→F, AD→E}. Bepaal ineens een decomposite tot een 3NF schema die zowel 'dependency preserving' als 'lossless join' is. Zie algoritme 15.6 in boek op blz. 544. Je mag niet veronderstellen dat S reeds een minimal cover is. Geef per component de sleutels en de lokale dependencies. Is elke component in het resultaat in BCNF? Geef aan waarom wel of niet.

ANTWOORD

We bepalen eerste een 'minimal cover'. Daarbij kan geconstateerd worden AD→E verkleind kan worden want A→E kan afgeleid worden. We krijgen dan {A→B, A→C, B→D, B→F, A→E} waarin verder niet verkleind of weggelaten kan worden. Vervolgens bepalen we de set van sleutels, die in dit geval 1 element bevat: { AG }. Dan passen we alg. 15.6 toe en groeperen de dependencies per linkerkant:

1. A→B en A→C en A→E leiden tot R1(A, B, C, E)
2. B→D en B→F leiden tot R2(B, D, F)

Vervolgens controleren we of minstens 1 van de compenten een key bevat:

3. De key AG is nog niet bevat in een relatie, dus: R3(A, G)

Sleutels en dependencies per relatie:

1. R1(A, B, C, E) keys: A lokale deps: A→B, A→C, A→E
2. R2(B, D, F) keys: B lokale deps: B→D, B→F
3. R3(A, G) keys: AG lokale deps: -

Elke component is in BCNF want de linkerkant van elke lokale FD is een superkey.

Deel 4 – File organizatie (5 punten)

Vraag 10. Een PARTS bestand met Part# als de hash key bevat records met de volgende Part# waardes: 2369, 3760, 4692, 4871, 5659, 1821, 1074, 7115, 1620, 2428, 3943, 4750, 6975, 4981, en 9208. Het bestand gebruikt tien 'buckets' genummerd 0 t/m 9. Elke bucket is een 'disk block' en bevat twee records. Laad deze records in het bestand in de aangegeven volgorde, gebruik makende van de hash-functie $h(K) = K \bmod 10$. Bereken het gemiddelde aantal 'block accesses' dat nodig is voor het ophalen van een record op basis van een gegeven Part#.

ANTWOORD

We gaan er van uit dat bij een overflow van een bucket er een overflow bucket wordt aangemaakt waarnaar deze met een pointer verwijst. Dus we krijgen:

1. Bucket 0 [3760, 1620] -overflow→ [4750, ..]
2. Bucket 1 [4871, 1821] -overflow→ [4981, ..]
3. Bucket 2 [4692, ..]
4. Bucket 3 [3943, ..]
5. Bucket 4 [1074, ..]
6. Bucket 5 [7115, 6975]
7. Bucket 6 [.., ..]
8. Bucket 7 [.., ..]
9. Bucket 8 [2428, 9208]
10. Bucket 9 [2369, 5659]

We nemen aan dat de hash-tabel zich in het geheugen bevindt en dus geen block access veroorzaakt. We berekenen het gemiddelde voor als 1 van de ingevoegde sleutels wordt opgevraagd. Dan moeten we in 6 gevallen 2 blocks ophalen, en in 9 gevallen maar 1. Dat is dus gemiddeld $(6\times2 + 9\times1) / (6 + 9) = 1{,}4$.

Deel 5 – Indexering (10 punten)

Vraag 11. Waarom kan er maximaal 1 ‘primary’ of ‘clustering’ index op een file zijn, maar meerdere ‘secondary’ indexen.

ANTWOORD
De primary en clustering index verondestellen dat de records van de tabel in een bepaalde volgorde op schijf gesorteerd staan, en dat kan maar in 1 volgorde tegelijk. In het geval van secondary indexen kunnen records willekeurig gesorteerd worden opgeslagen.

Vraag 12. Een PARTS bestand met Part# als sleutelveld bevat records met de volgende Part# waardes: 23, 65, 37, 60, 46, 92, 48, 71, 56, 59, 18, 21. Stel dat de deze zoekwaardes in deze volgorde ingevoegd worden in een lege B^+-boom van orde p=4 en p_{leaf}=3; laat zien hoe de uiteindelijke boom er uitziet.

ANTWOORD
Voor de volledigheid geven we hier de stapsgewijze opbouw van de B+ boom. Merk op dat de boom op zijn kant wordt gerepresenteerd met de wortel naar links en kleinst waardes aan de onderkant. De ‘leaf nodes’ zijn met vierkante haken aangegeven, en de interne knopen door vertikale reeksen van punten en getallen.

[23]

[23, 65]

[23, 37, 65]

. [60, 65]
37
. [23, 37]

. [46, 60, 65]
37
. [23, 37]

. [65, 92]
60
. [46, 60]
37
. [23, 37]

. [65, 92]
60
. [46, 48, 60]
37
. [23, 37]

. [65, 71, 92]
60
. [46, 48, 60]
37
. [23, 37]

. [65, 71, 92]
60
. [56, 60]
48
. [46, 48]
37
. [23, 37]

. [65, 71, 92]
60
. [56, 59, 60]
48
. [46, 48]
37
. [23, 37]

. [65, 71, 92]
60
. [56, 59, 60]
48
. [46, 48]
37
. [18, 23, 37]

Eindresultaat:

. . [65, 71, 92]
. 60
. . [56, 59, 60]
. 48
. . [46, 48]
37
. . [23, 37]
. 21
. . [18, 21]

Deel 6 – Queryoptimalisatie (20 punten)

Vraag 13. Hoevel 'join orders' zijn er voor een query die 10 tabellen joint.

ANTWOORD
10! = 3.628.800

Vraag 14. Een bestand van 4096 blocks moet gesorteerd worden met een beschikbare bufferruimte van 64 blocks. Hoeveel 'passes' zijn er nodig in de merge-fase van het 'external sort-merge' algoritme?

ANTWOORD
Dit is de waar van p in het algoritme in figuur 18.2 op blz. 664: $\lceil \log_{k-1}(m) \rceil$ met $k = 64$ en $m = \lceil (j/k) \rceil$ waarbij $j = 4096$. We berekenen eerst $m = \lceil (4096/64) \rceil = 64$. Dus $p = \lceil \log_{k-1}(m) \rceil = \lceil \log_{63}(64) \rceil = 2$.

Vraag 15. Ontwikkel kost-functies voor het UNION algoritme zoals besproken in sectie 18.4 op basis van sort-merge. Zie ook figuur 18.3 op blz. 671 voor het algoritme, en sectie 18.8.3 voor voorbeelden van kost-functies. Net zoals in 18.8.3 dienen de kostfuncties een schatting te geven van de hoeveelheid block-transfers.

ANTWOORD
De kosten voor sorteren wordt geschat door (zie blz. 664) de formule: sort(b, n_B) = $2b + 2b.\log_{dM}(n_R)$. Hierbij veronderstellen we: b = aantal blocks van de te sorteren file, dM = graad van merging en n_R = aantal initiele runs. Deze worden op hun beurt als volgt berekend: dM = **min**(n_B-1, n_R) en $n_R = \lceil (b/n_B) \rceil$ waar n_B de beschikbare bufferruimte in blocks is. We vatten deze formule samen als sort(b, n_B). Voor files van grootte b_1 en b_2 krijgen we dan dus de volgende kostenfunctie voor de UNION:

sort(b_1, n_B) + sort(b_2, n_B) + $2b_1$ + $2b_2$

De eerste twee componenten ort(b_1, n_B) + sort(b_2, n_B) komen voort uit het feit dat we de twee bestanden apart sorten, en daarvoor waarschijnlijk dezelfde bufferruimte beschikbaar hebben. De volgende component $2b_1 + 2b_2$ komt doordat we alle blocks moet lezen en (in de "worst case" als er geen dubbelen zijn) ook weer terug moeten schrijven.

Vraag 16. In stap 2 van het heuristische algebraïsche optimalisatiealgoritme op blz. 688 in sectie 18.7.2 worden SELECT operaties zo ver mogelijk naar beneden in de query tree verplaatst. Leg kort uit waarom dit inderdaad waarschijnlijk een optimalere executie tot gevolg heeft.

ANTWOORD
De selecties naar beneden verplaatsen betekent dat we zo snel mogelijk beginnen met de data te selecteren die we voor het eindresultaat nodig hebben. Dit voorkomt onnodige berekeningen en maakt de tussenresultaten kleiner, wat beide waarschijnlijk leidt tot een snellere executie.

Deel 7 – Databasetuning (5 punten)

Vraag 17. Geef een voorbeeld van 'horizontal partitioning' en leg uit waarom dit de database onder sommige omstandigheden efficiënter maakt.

ANTWOORD
Een tabel voor personeelsleden kan gesplitst worden in aparte tabellen waarbij elke afdeling een eigen tabel krijgt met daarin de werkenemers van die afdeling. Queries die dan alleen bepaalde afdelingen betreffen kunnen gericht over de betreffende kleine(re) tabellen uitgevoerd worden, wat sneller gaat dan de query uitvoeren over de oorspronkelijke grote tabel waarin alle personeelsleden zitten.

Deel 8 – Transacties (10 punten)

Vraag 18. Beschouw de drie transacties T1, T2, and T3, en de schedules S1 en S2 zoals beneden gegeven. Teken de 'precedence graph' voor S1 en S2, en bepaal welk schedule serialiseerbaar is. Als een schedule serialiseerbaar is, geef dan een equivalent 'serial schedule'.

T1: r1(X); r1(Z); w1(X);
T2: r2(Z); r2 Y); w2(Z); w2(Y);
T3: r3(X); r3(Y); w3(Y);

S1: r1(X); r2(Z); r1(Z); r3(X); r3(Y); w1(X); w3(Y); r2(Y); w2(Z); w2(Y);
S2: r1(X); r2(Z); r3(X); r1(Z); r2(Y); r3(Y); w1(X); w2(Z); w3(Y); w2(Y);

ANTWOORD
De precedence graaf voor S1 moet de volgende pijlen bevatten:
- r1(Z) gevolgd door w2(Z): T1→T2
- r3(X) gevolgd door w1(X): T3 → T1
- r3(Y) gevolgd door w2(Y): T3 → T2
- w3(Y) gevolgd door w2(Y): T3 → T2 (hadden we al)

Deze graaf bevat geen gericht cykels, dus S1 is serialiseerbaar.
Een mogelijk serial schedule is: T3, T1, T2.

De precedence graaf voor S2 moet de volgende pijlen bevatten:
- r3(X) gevolgd door w1(X): T3 → T1
- r1(Z) gevolgd door w2(Z): T1 → T2
- r2(Y) gevolgd door w3(Y): T2 → T3
- r3(Y) gevolgd door w2(Y): T3 → T2

Deze graaf bevat diverse gericht cykels, dus S1 is niet serialiseerbaar. Voorbeelden van cykels: T2 → T3 → T2, T1 → T2 → T3 → T1.

Deel 9 – Concurrency protocollen (10 punten)

Vraag 19. Beschouw de volgende abstracte beschrijving van de lees- en schrijfstappen in een transactie:

[1] r(X); [2] r(Y); [3] w(X); [4] r(Z); [5] w(Y); [6] r(U); [7] w(Z); [8]

Geef aan op welke plekken (aangegeven met vierkante haken) we welke shared/exclusive lock-operaties in welke volgorde moeten plaatsen voor het aanvragen en vrijgeven van locks willen we (1) ons aan het 'shared/exclusive locking scheme houden', (2) ons aan het 'two-phase locking protocol' houden en (3) telkens zo min mogelijk resources bezet houden.

ANTWOORD

Merk op dat gevraagd is om een locking schema waarbij aan alle drie de eisen voldaan is. De standaard oplossing waarbij locks worden aangevraagd als ze nodig zijn en weer zo vroeg mogelijk losgelaten (dwz. nadat de laatst lock aanvraag is gebeurd, om aan eis (2) te voldoen) is de volgende:

- [1] : read_lock(X)
- [2] : read_lock(Y)
- [3] : write_lock(X)
- [4] : read_lock(Z)
- [5] : write_lock(Y)
- [6] : read_lock(U)
- [7] : write_lock(Z); unlock(U); unlock(X); unlock(Y)
- [8] : unlock(Z)

In dit geval is het aantal stappen dat de resources worden bezet: U(1), X(6), Y(5), Z(5).

Maar het kan beter als we locks soms al eerder aanvragen, zodat het moment van de laatste aanvraag eerder is, zodat we dan eerder weer kunnen beginnen met vrijgeven:

- [1] : read_lock(X)
- [2] : read_lock(Y)
- [3] : write_lock(X)
- [4] : write_lock(Y); read_lock(U); write_lock(Z); unlock(X),
- [5] :
- [6] : unlock(Y)
- [7] : unlock(U)
- [8] : unlock(Z)

In dit geval is het aantal stappen dat de resources worden bezet: U(3), X(3), Y(4), Z(4). Dat is dus waarschijnlijk beter dan het voorgaande schema.

Vraag 20. Beschouw opnieuw de twee schedules S1 en S2 uit vraag 18. Bepaal welk van deze zijn toegestaan zijn volgens het 'basic timestamp ordering' (TO) algoritme.

ANTWOORD

We volgen de executie van S1. De timestamps van de transacties worden aangenomen te zijn bepaald door hun eerste operatie, dus TS(T1) = 1, TS(T2) = 2 en TS(T3) = 4. 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*)
0		-	-	-	-	-	-
1	r1(X)	1	-	-	-	-	-
2	r2(Z)	1	-	-	-	2	-
3	r1(Z)	1	-	-	-	2	-
4	r3(X)	4	-	-	-	2	-
5	r3(Y)	4	-	4	-	2	-
6	w1(X)	!!!					
7	w3(Y)						
8	r2(Y)						
9	w2(Z)						
10	w2(Y)						

Op *T*=6 is ere een conflict want Rd_ts(X) > TS(T1). Dus dit schedule is niet toegestaan.

We beschouwen op dezelfde manier de uitvoering van schedule S2. De timestamps van de transacties worden opnieuw aangenomen te zijn bepaald door hun eerste operatie, dus TS(T1) = 1, TS(T2) = 2 en TS(T3) = 3.

T	Operat.	Rd_ts(*X*)	Wr_ts(*X*)	Rd_ts(*Y*)	Wr_ts(*Y*)	Rd_ts(*Z*)	Wr_ts(*Z*)
0		-	-	-	-	-	-
1	r1(X)	1	-	-	-	-	-
2	r2(Z)	1	-	-	-	2	-
3	r3(X)	3	-	-	-	2	-
4	r1(Z)	3	-	-	-	2	-
5	r2(Y)	3	-	2	-	2	-
6	r3(Y)	3	-	3	-	2	-
7	w1(X)	!!!					
8	w2(Z)						
9	w3(Y)						
10	w2(Y)						

Op T=7 is er een conflict want Rd_ts(X) > TS(T1). Dus dit schedule is niet toegestaan.

Deel 10 – Recovery protocollen (10 punten)

Vraag 21. Beschouw de volgende geabstraheerde 'system log' na een crash:

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

   [**write**, T2, *B*, 15, 12] [**write**, T3, *A*, 20, 30] [**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**, T3, *A*, 30, 20] [**write**, T2, *B*, 12, 15]

3. Vervolgens doen we een undo voor write operaties van de committed transactions, in de volgorde waarin ze in de log voorkomen. Dat zijn de volgende operaties:

   [**write**, T4, *B*, 25, 15] [**write**, T4, *A*, 10, 20]

In totaal levert dat dus het volgende UNDO schedule:

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

Einde van het tentamen