Antwoorden

Opgave 1

Antwoord a)

Patient (patientnr , adres, naam, … , primary key (patientnr))
Factuur (patientnr , datum, bedrag, … ,
foreign key (patientnr) references Patient (patientnr) on update cascade);

Toelichting

- In Factuur is “primary key (patientnr, datum)” onnodig voor het voldoen aan regels 1 en 2.
- In Factuur mag worden toegevoegd “on delete no action”, dit is de default.
- Het opnemen in *Patient* van: “foreign key (patientnr) references Factuur(patientnr)” is fout, want *(i)* in Factuur is (*patientnr*) geen key (maar slechts een component van een key), en *(ii)* het zou impliceren dat iedere patiënt tenminste een factuur heeft (en dat is in de casus niet gegeven).

Antwoord b)

```
create trigger regel_a
before delete on Patient
referencing old as O
for each row
when (O.patientnr in (select patientnr from Factuur ))
rollback;

create trigger regel_b
after update of patientnr on Patient
referencing old as O new as N
for each row
update Factuur F
set F .patientnr = N .patientnr
where F .patientnr = O.patientnr;
```

Opgave 2

Antwoord a)

$read_1(x)$; $read_2(y)$; $read_2(x)$; ... $write_2(y)$; $commit_2$; $write_1(y)$; $write_1(x)$; $commit_1$

NB1. Omdat $read_1(x)$ commuteert met ieder van $read_2(y)$; $read_2(x)$; en $write_2(y)$, is snel in te zien dat het schedule equivalent is met T_2; T_1.

NB2. Net nadat het gegeven beginstuk $read_1(x)$; $read_2(y)$; $read_2(x)$; is uitgevoerd, geldt het volgende.

1. $T2$ heeft een readlock op x en y, dus kan $T1$ de writelock op y niet verkrijgen (en moet dus wachten), en dat blijft zo totdat $T2$ bij zijn commit de lock (inmiddels een writelock) op y weer vrij geeft.

2. $T1$ heeft een readlock op x (tegelijk met $T2$!), dus kan $T2$ de writelock op y verkrijgen en zijn executie voltooien.

NB3. Een alternatieve motivatie is de volgende. Genereer alle mogelijke schedules, en laat dan zien dat ze allemaal (met uitzondering van het antwoord), niet serialiseerbaar zijn. Stel dat we het schedule afmaken als volgt:

$$read_1(x); read_2(y); read_2(x); ... write_1(y); write_2(y); commit_2; write_1(x); commit_1$$

Dan geldt: dit schedule is niet equivalent met T2; T1, omdat $write_1(y)$; niet commuteert met $write_2(y)$ (je kunt ze niet zomaar omwisselen). Het is ook niet equivalent met T1; T2 omdat $read_2(x)$ niet commuteert met $write_1(x)$.

Antwoord b) Ja, de schedule is equivalent met T3; T4. De volgende redenen moeten genoemd worden:

1. $write_4(y)$ commuteert met $write_3(x)$;
2. $read_4(y)$ commuteert met $read_3(y)$ en $write_3(x)$;
3. $read_4(z)$ commuteert met $read_3(x)$, $read_3(y)$ en $write_3(x)$.

NB1. Het schedule is niet equivalent met T4; T3, want $read_3(y)$ commuteert **niet** met $write_4(y)$.

NB2. Pas op: de volgorde binnen de transacties mag niet veranderen, bijvoorbeeld: $read_4(z)$ en $write_4(y)$ commuteren **niet**. Het kan bijvoorbeeld zijn dat de applicatie de waarde z gebruikt om de waarde van y aan te passen. In verschillende transacties kan dit niet, omdat we aannemen dat de transacties niets van elkaar weten. Voor het serialiseerbaar maken van een schedule is het omwisselen van twee acties binnen dezelfde transactie je trouwens ook niet dichter bij een serieel schedule.

Opgave 3

Antwoord a) 'Deployment' is het installeren (kopiëren) van alle nodige resources (code, pagina's, stylesheets) van een web applicatie in (naar) de Application Server. Dat is nodig want de web applicatie heeft een runtime environment nodig om geëxecuteerd te worden, en de Application Server biedt deze runtime environment (een web applicatie heeft geen `main()` methode die aangeroepen kan worden).

Ook acceptabel: nodig want de Application Server biedt toegang tot de web applicatie.

Antwoord b) Het scheiden van de model, control en view functionaliteiten voor hergebruik, zodat bijvoorbeeld verschillende views van een model vertoond kunnen worden zonder dat het model aangepast moet worden.

Ook acceptable: views, model en control apart van elkaar kunnen veranderen zonder 'side effects'.

Antwoord c) Servlets kunnen de rol van control (en ook model) uitoefenen want ze handelen de inouts af, Java Beans kunnen als (delen van) een model gezien worden want ze bevatten informatie over (delen van) de toestand van de applicatie, en JSP spelen zeker de rol van views want we genereren de pagina's die aan de eindgebruikers worden vertoond. Een preciezer antwoord is dat de Java Beans verbinden de control (en model) functionaliteit van de Servlets met de view

functionaliteit van de JSPs.

Antwoord d) Elke CRUD actie is afgebeeld op één HTTP request bericht methode. Op deze manier wordt de actie aangeroepen. Een populaire afbeelding is Create – POST, Read – GET, Update – PUT en Delete – DELETE.

Antwoord e)

```
<xml version="1.0"?>
<hello>HelloJersey</hello>
```

Dit bericht wordt gestuurd want methode `sayXMLHello()` wordt daadwerkelijk geëxecuteerd. Dat komt door de matching tussen de `GET` methode in het HTTP request bericht en de `@GET` annotatie, en de matching tussen de `Accept: text/xml` header element en de `@Produces(MediaTypee.TEXT_XML)` annotatie in de `sayXMLHello()` methode.

Opgave 4

Antwoord a)

De kortste antwoorden zijn

//movie[title='King Kong']/year of //year[../title='King Kong']

hoewel de laatste meer aannames doet over de structuur, namelijk dat een element met kinderen 'year' en 'title' een film betreft. Oplossingen met FOR-WHERE-RETURN zijn ook goed (maar niet 'zo kort mogelijk'), zoals

```
FOR $m in //movie
WHERE $m/title = 'King Kong'
RETURN $m/year
```

NB 1: als je in de WHERE-clause ook nog eens rechte haken gebruikt, bijvoorbeeld WHERE $m[title='King Kong'] dan is dat dubbelop. De WHERE-clause is al een predicaat.

NB 2: heel fout is //movie/year[title='King Kong']. Hierbij zou 'title' een kind moeten zijn van 'year'.

Antwoord b)

```
FOR $a in distinct-values(//actor)        (: We itereren over alle actors (distinct-values zorgt
                                             ervoor dat we dezelfde actor niet meerdere keren
                                             krijgen: <actor>foo</actor> op twee verschillende
                                             plekken in het document zijn verschillende
                                             elementen) :)
LET $m := //movie[.//actor = $a]          (: Dit is de eigenlijke groepering: alle films van actor $a :)

RETURN
  <actor name="{$a}">                     (: We maken een XML-fragmentje per actor met alle
    <aantal>{count($m)}</aantal>             informatie erin. :)
    <films>{$m/title}</films>             (: Ook $m hier is natuurlijk prima. :)
  </actor>
```

of

```
FOR $m in //movie                         (: Let op: we itereren hier eerst over movies! :)
```

FOR $a in $m//actor	*We willen groeperen naar actor, maar een movie heeft meerdere actoren, dus hier moet een FOR. Als je die niet hebt, dan heb ik dat niet foutgerekend.*
GROUP BY $a	*Dit doet de eigenlijke groepering. $a moet een enkel element zijn.*
RETURN <actor name="{$a}"> <aantal>{count($m)}</aantal> <films>{$m/title}</films> </actor>	*Hetzelfde XML-fragmentje*

Antwoord c)

FOR $m in //movie

RETURN
  INSERT NODE <genre>Romance</genre> INTO $m/genres

Antwoord d)

//movie[plot contains text {year}] of //movie[./plot contains text {./year}]

of

FOR $m IN //movie
WHERE $m/plot contains text $m/year
RETURN $m