Answers to G51DBS 2006-7

1.

(a)

(i)

```
pID
111
222
333
444
```

(ii)

```
pID
111
222
333
444
```

(iii)

```
pID
444
```

(iv)

```
pID
111
222
333
```

(v)

```
pID  Name
111 Tom
333 Sue
```

(vi)

```
pID pName pID dID Date
111 Tom   111  2  12-03-07
111 Tom   333  2   5-03-07
```

(b) $\pi_{\texttt{Date}}\sigma_{\texttt{(pName=Tom)AND(dName=Jones)AND(Patient.pID=Treatment.pID)AND(Treatment.dID=Doctor.dID)}}(\texttt{Patient}\times\texttt{Treatment}\times\texttt{Doctor})$

(c) $\pi_{\texttt{pName,dName}}\sigma_{\texttt{Patient.pID=Treatment.pID)AND(Treatment.dID=Doctor.dID)}}((\texttt{Patient}\times\texttt{Treatment}\times\texttt{Doctor})$

(d) (i) is more computationally expensive, because we compute a product of a larger relation with Patient.

2.

(a) Entities: Bottle, Wine, Wine-maker.
Attributes of Bottle: volume, price.
Attributes of Wine: Name, Type, Year, Grape.
Attributes of Wine-maker: Name?, Region, Country. (the latter two can be entities. Name is optional).
Relationships: Wine is contained in a bottle (one-to-many).
Wine-maker makes wines (one-to-many).
(b) Tables:
Wine(wID, Name, Type, Year, Grape, pID) where pID is producer (wine-maker) ID.
Bottle(bID, wID, Volume, Price)
Producer(pID, Region, Country).
(c) Uniqueness and minimality.
(d) Wine: wID; {Name,Year,pID}
Bottle: bID
Producer: pID
(e) pID in Wine is a foreign key for Producer. wID in Bottle is a foreign key for Wine.
(f) Entity integrity: No part of primary key should be NULL. Referential integrity: foreign key should either match a candidate key of some tuple in its home relation or be wholly null.
(g) Create a sequence which will produce a new integer value which can be used for ID; the first generated value is n, and every next one is incremented by m:

```
CREATE SEQUENCE tableSeq START WITH n INCREMENT BY m;
```

The nextval function on the sequence can be called each time when we need a new ID:

```
INSERT INTO Table(ID, ...) VALUES (tableSeq.nextval, ...);
```

3.

(a)

```
SELECT Name
FROM Staff
```

(b)

```
SELECT Funder
FROM Project
WHERE Budget > 100000
```

(c)

```
SELECT Name
FROM Staff, Allocation, Project
WHERE Staff.sID = Allocation.sID AND
      Allocation.pID = Project.pID AND
      Funder = 'EPSRC'
```

(d)

```
SELECT Name, SUM(Allocation.Time)
FROM Staff, Allocation
WHERE Staff.sID = Allocation.sID
GROUP BY Staff.Name
```

(e)

```
UPDATE Allocation SET Time = Time + 1
WHERE pID IN (SELECT pID
              FROM Project
              WHERE Funder = 'EPSRC'
```

(f)

```
INSERT INTO Staff(sID, Name, Manager)
VALUES (5, 'Edith', (SELECT sID FROM Staff WHERE Name = 'Bill'))

INSERT INTO Allocation(sID, pID, Time)
VALUES (5, 2, 10)
```

(g)

```
CREATE TABLE Deliverable
dID int,
pID int,
Description varchar(250),
Pages int NULL,
CONSTRAINT pkDeliverable PRIMARY KEY (dID)
CONSTRAINT fkDeliverable FOREIGN KEY (pID)
                         REFERENCES Project (pID)
```

4.

(a)

A relation R has a functional dependency X → Y if for every two tuples s and t in R, if s and t have the same values for the attributes in X, then they have the same values for the attributes in Y.

(b)

A relation is in 2NF if there is not functional dependency X → Y where Y is a non-key attribute and X is a stirct subset of a candidate key.

(c)

A relation is in 3NF if there are no non-trivial functional dependencies X → Y, Y → Z where X is a candidate key, Y ≠ Z, and Z is a non-key attribute.

(d)

A relation is in BCNF, if for every non-trivial functional dependency X → Y, X is a (super) key.

(e)

Insertion anomaly is a situation when it is impossible to insert information about some attributes X because this would make part of a primary key to be null, violating entity integrity. Deletion anomaly is when it is impossible to delete information about some objects without removing unrelated information. Update anomaly arises when in order to change a value of some attribute for we need to update many tuples.

(f)

No, because primary key is (StudentID, ModuleID) and there are fds ModuleID → ModuleName and StudentID → StudentName.

(g)

R1(StudentID, StudentName), R2(ModuleID, ModuleName), R3(StudentID, ModuleID, Mark). R1 and R2 are in BCNF because ID is the key in both, the other attribite isn't, and there clearly is no fd with determinant which is not a (super) key. In R3 the only key is (StudentID,ModuleID,Mark), and there are no fds with determinants StudentID, ModuleID on their own or Mark, or Mark with one of IDs.

(h)

Decomposition is lossless if the resulting tables can be join together and yield exactly the original tables:

$$\texttt{Marks} = R1 \bowtie R2 \bowtie R3$$

This is indeed the case because each time we decompose a table $R(X, Y, Z)$ into $R(X, Y)$ and $R(X, Z)$ in the presence of a functional dependency X → Y. In such case decomposition is guaranteed to be lossless. (X = StudentID, Y = StudentName, Z = ModuleID,ModuleName,Mark, and then X = ModuleID, Y = ModuleName, Z = StudentID,Mark).

5.
(a) A transaction log stores information about all the transactions that are run on a database, including when they start and finish, and what operations they make on the database .

Periodically a check point is taken, and the current database state is written to disk. The transaction log is also written directly to disk, so in the event of a system failure the DBMS has access to the checkpoint and the transaction log. When the system has failed transactions that had completed before the last checkpoint they are fine, as they are stored on disk, however those that completed after the checkpoint and before the system failure need to be redone, since their effects will have been lost, and those that were still running when the system failed need to be rolled back and restarted.
(b) In a serial schedule each transaction runs entirely before the next one starts. In a serialisable schedule the operations of transactions may be interleaved, but the end effect is the same as some serial schedule of the transactions involved.
(c) Shared (read) lock: several transactions can have this lock on the same data item at the same time. They can read the data item, but nobody can update it. Exclusive (write) lock: only one transaction at a time can have this lock. It can read the data item and update it. No other transaction can read or update this item.
(d) A transaction follows a 2PL if all locking operations precede the first unlock operation in the transaction. First a transaction acquires all the necessary locks (and waits if the locks are not available) (this is *growing phase*), and only releases the locks when no new locks are needed (*shrinking phase*).
(e) A deadlock occurs when two transactions are waiting for each other and none can make progress. This can happen if two transactions are each waiting for locks to be released that are held by the other. For example, suppose that two transactions both need exclusive locks on A and B. The first one gets a lock on A, and the second one on B. Then they are waiting for the second lock, and since both are in the growing phase none releases its lock, so they are deadlocked.
(f) A WFG for a particular schedule has transactions as nodes, and a link from t1 to t2 if t2 is waiting for a lock held by t1. If there is a cycle in the graph, then there is a deadlock. Example:

```
Time  T1                  T2
1     begin               begin
2     acquire w_lock(A)   acquire w_lock(B)
3     update(A)           request w_lock(A)
4     request w_lock(B)   wait
5     wait                wait
```

at time 3, we create a link from T2 to T1 and on line 4 a link from T1 to T2: a cycle.
(g) A classical concurrency way is to use lock ordering and always acquire and release locks in this order.

DBMS use lock timeouts. They also use transaction orderings (by timestamps).