Answers to G51DBS exam 2007-8

1.

(a)

(i)

```
Title
Java
Haskell
Databases
Robotics
```

(ii)

```
ID     Code    Mark
111   G51PRG    60
222   G51PRG    70
333   G51PRG    50
```

(iii)

```
ID
111
222
333
```

(iv)

```
ID
444
```

(v)

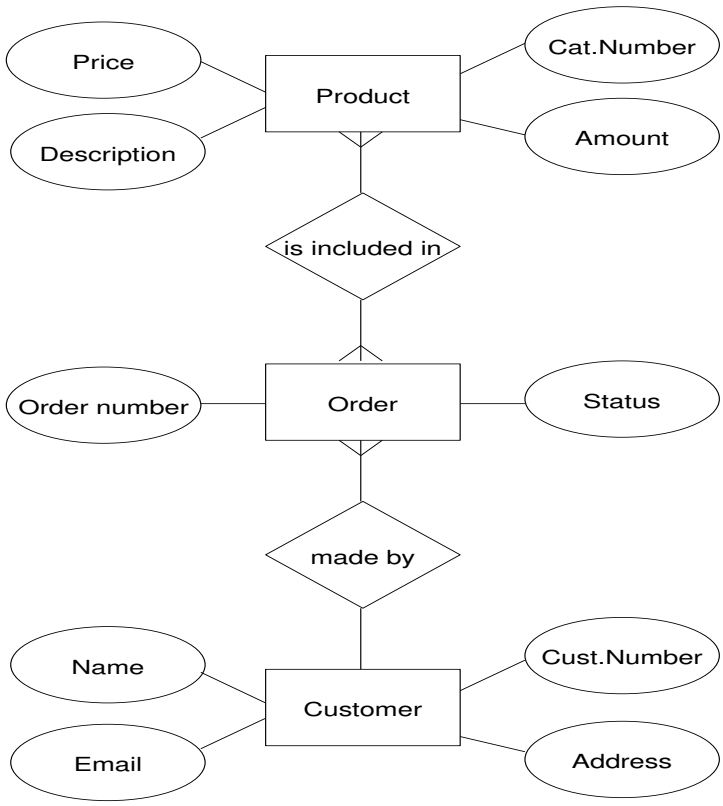
```
ID
111
```

(vi)

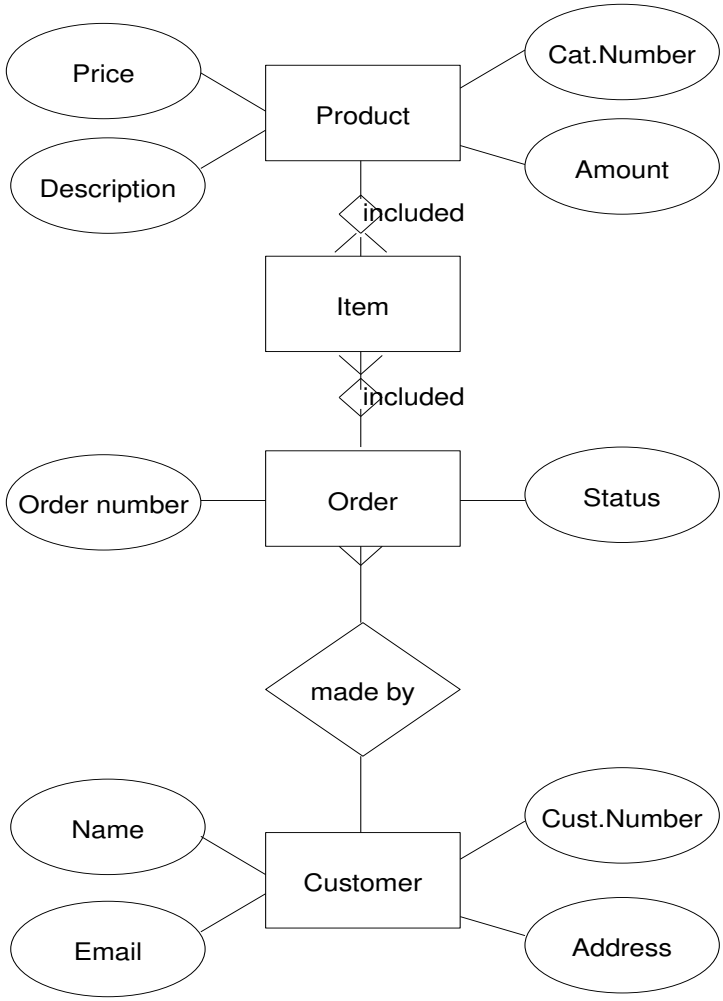
```
ID    Name   ID  Code     Mark
222   John   111  G51DBS    70
222   John   222  G51DBS    80
```

(b) $\pi_{\texttt{Mark}}\sigma_{\texttt{Student.ID=Marks.ID AND Marks.Code=Module.Code}}$
$(\sigma_{\texttt{Name=James}}(\texttt{Student}) \times \texttt{Marks} \times (\sigma_{\texttt{Title=Computer Graphics}}(\texttt{Module})$
(c) $\pi_{\texttt{Name}}\sigma_{\texttt{Student.ID=Marks.ID AND Marks.Mark>50}}(\texttt{Student} \times \texttt{Marks})$
(d) $\pi_{\texttt{Name,Title,Mark}}\sigma_{\texttt{Student.ID=Marks.ID AND Marks.Code=Module.Code}}(\texttt{Student} \times \texttt{Marks} \times \texttt{Module})$

2.(a)



or (with an extra table to break a many-to-many relationship)



(b)

```
CREATE TABLE Customer (
   cNumber INT NOT NULL,
   Name VARCHAR(50) NOT NULL,
   Email VARCHAR(20),
   Address VARCHAR(100) NOT NULL,
   CONSTRAINT pk_customer PRIMARY KEY (cNumber)
);
```

```
CREATE TABLE Product (
   pNumber INT NOT NULL,
   Description VARCHAR(100),
   Price FLOAT,
   Amount INT,
   CONSTRAINT pk_product PRIMARY KEY (pNumber)
);

CREATE TABLE Order (
   oNumber INT NOT NULL,
   cNumber INT NOT NULL,
   Status VARCHAR(10),
   CONSTRAINT pk_order PRIMARY KEY (oNumber),
   CONSTRAINT fk_orderCustomer FOREIGN KEY (cNumber) REFERENCES Customer
);

CREATE TABLE Item (
   oNumber INT NOT NULL,
   pNumber INT NOT NULL,
   CONSTRAINT pk_item PRIMARY KEY (oNumber, pNumber),
   CONSTRAINT fk_itemProduct FOREIGN KEY (pNumber) REFERENCES Product,
   CONSTRAINT fk_itemCustomer FOREIGN KEY (oNumber) REFERENCES Order
);
```

(c)

```
CREATE SEQUENCE productSeq START WITH 1000 INCREMENT BY 1;

INSERT INTO Product VALUES (productSeq.nextval, 'suitcase', 59.99, 100);

INSERT INTO Product VALUES (productSeq.nextval, 'backpack', 12.00, 100);
```

3.

(a)

```
SELECT Director
FROM Film
```

(b)

```
SELECT First, Last
FROM Actor
WHERE Gender = 'M'
```

(c)

```
SELECT First, Last
FROM Actor, Part
WHERE Actor.First = Part.First AND
      Actor.Last = Part.Last AND
      Actor.Gender = 'F' AND
      Part.Title = 'Magnolia' AND
      Part.Year = 1999
```

(d)

```
SELECT Genre, AVG(Price)
FROM Film, DVD
WHERE Film.Title = DVD.Title AND
      Film.Year = DVD.Year
GROUP BY Genre
```

(e)

```
SELECT First, Last
FROM Actor NATURAL JOIN Part NATURAL JOIN Film
WHERE Director = 'Anderson'
```

(f)

```
UPDATE DVD SET Price = Price + 1
```

(g)

```
INSERT INTO DVD VALUES ('Memento', 2000, 10)
```

(h) (Title, Year) is a foreign key to the Film table, so if there is no tuple with values (Memento, 1999) there, referential integrity will be violated - insertion will not succeed.

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 no functional dependency X → Y where Y is a non-key attribute and X is a strict 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, 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 there is a partial dependency of a non-key attribute City on the candidate key (Publisher → City).

(g)

No, because there is a transitive dependency of a non-key attribute Country on the candidate key: (Author, Title, Publisher, Year) → City → Country.

(h)

No, because it is not in in 3NF and also there is a non-trivial functional dependency where determinant is not a superkey: City → Country. Decomposition:

```
Book1(Author,Title,Publisher,Year,ISBN)

Location(Publisher,City)

Geography(City,Country)
```

the first table is in BCNF because there are no FDs where determinant is not a superkey (no proper subset of Author,Title,Publisher, Year determines another subset or ISBN). The second and the third tables are in BCNF because each has only one non-trivial dependency where the determinant is the key. Marks will not be withdrawn for pointing out that some of the decomposition here is rather pointless.

5.

(a)

A transaction is a group of database operations which is a unit of work which must be executed atomically.

(b)

Atomicity means that transactions should behave as if they are a single 'action' which is not interleaved with any other transaction.

(c)

Lost update happens when two transactions access the same resource in an interleaved fashion and one of them overwrites the update made by the first. In the example below, two transactions are both trying to decrement X by 10. The first update is lost, and X ends up decremented only once.

```
Transaction 1                Transaction 2

Read X (X = 100)

                             Read X (X = 100)

Write X = 90 (X = X - 10)

                             Write X = 90 (X = X - 10)
```

(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)

If both transactions follow 2PL, either Transaction 1 will acquire the read-lock and write-lock for X first (and Transaction 2 will wait until the locks are released and execute after 1 is finished) or both transactions will acquire the read locks for X and will deadlock, both waiting for a write lock for X (unless some deadlock-prevention mechanism is used).

(f)

Each transaction T is given a different timestamp $TS(T)$ (usually given by system time when transaction starts); transactions which start earlier have a smaller timestamp.

Each resource X is given two timestamps, $R(X)$ which is the largest timestamp of any transaction that has read X, and $W(X)$ the largest timestamp of any transaction that has written X.

If T tried to read X and $TS(T) < W(X)$, T is rolled back and restarted with a new timestamp. Otherwise read succeeds and $R(X)$ becomes $max(TS(T), R(X))$. If T tries to write X and $TS(T) < W(X)$ or $TS(T) < R(X)$, T is rolled back and restarted with a new timestamp. Otherwise write succeeds and $W(X)$ is set to be $TS(T)$.

(g)

The advantages are: less overhead to do with controlling locks; no waiting; no deadlocks. Disadvantages are: long transactions may be repeatedly rolled back (starvation).