Problem 1.

1. SELECT A.name
   FROM Artists A, Albums AL, Labels L
   WHERE A.artist_id = AL.artist_id AND AL.label_id = L.label_id AND AL.year = '2014' AND
   L.country = 'Canada'

2. SELECT A.name
   FROM Artists A
   WHERE 100 < (SELECT COUNT(S.song_id)
                  FROM Songs S, Albums AL
                  WHERE AL.artist_id = A.artist_id AND S.album_id = AL.album_id)

3. SELECT A.name
   FROM Artists A
   WHERE A.artist_id IN (SELECT AL.artist_id
                           FROM Albums AL
                           WHERE album_id IN (SELECT S.album_id
                                                FROM Songs S
                                                GROUP BY S.album_id
                                                HAVING COUNT(*) > 12)
                           GROUP BY AL.artist_id
                           HAVING COUNT(*) > 2)

4. SELECT A.name
   FROM Artists A
   WHERE NOT EXISTS (SELECT L.label_id
                       FROM Labels L
                       WHERE L.country = 'USA')
   EXCEPT
   (SELECT AL.label_id
    FROM Albums AL
    WHERE AL.artist_id = A.artist_id)
5. (Hint: You can decompose the query to simpler queries to understand the query better.)

```sql
SELECT A.name
FROM (SELECT Pf.artist_id, AVG(P.profit) AS avg_profit
      FROM (SELECT Al.artist_id, SUM(Sl.copies*Al.price) AS profit
            FROM Sales Sl, Albums Al
            WHERE Sl.album_id = Al.album_id
            GROUP BY Al.album_id, Al.artist_id) AS Pf
      GROUP BY Pr.artist_id) AS Temp, Artist A
WHERE Temp.avg_profit = (SELECT MAX(T.avg_profit) FROM Temp)
    AND Temp.artist_id = Ar.artist_id
```

**Problem 2.**

1. ```sql
SELECT  S.sname
FROM    Students S, Enroll E1, Enroll E2
WHERE S.sid = E1.side AND S.sid = E2.sid
       AND E1.dname="Computer Science" AND E2.dname="Biology"
```

2. ```sql
SELECT  S.sname
FROM    Students S
WHERE    S.year_started = 2015 AND
          S.age = ( SELECT MAX(S1.age)
                    FROM    Students S1
                    WHERE   S1.year_started = 2015)
```

3. ```sql
SELECT  M.dname, AVG(S.gpa) AS avggp
INTO    TEMP1
FROM     Major M, Student S
WHERE    M.sid = S.sid
GROUP BY M.dname

SELECT  T.dname, T.avggp
FROM     TEMP1 T
WHERE    T.avggp = ( SELECT MAX(T1.avggp) FROM TEMP1 T1)
```
4.

```
SELECT S.sid, S.sname, S.gpa
FROM Student S
WHERE NOT EXISTS ((SELECT DISTINCT S.cno
                      FROM Section S, Prof P
                      WHERE P.pname = "Prof. Smith" AND S.pid = P.pid)
                      EXCEPT
                      (SELECT E.cno
                      FROM Enroll E
                      WHERE E.sid = S.sid))
```

5.

```
SELECT avg(S.gpa) as avggpa
INTO T1
FROM Student S
WHERE S.gpa >= 3.0 AND 3.2 <= (SELECT AVG(grade)
                            FROM Enroll E
                            WHERE E.sid = S.sid AND E.dname="Computer Science")
```

```
SELECT S.sname, S.gpa, T1.avggpa
FROM Student S, T1
WHERE S.gpa >= 3.0 AND 3.2 <= (SELECT AVG(grade)
                            FROM Enroll E
                            WHERE E.sid = S.sid AND E.dname="Computer Science")
```

**Problem 3.**

There are many ways to create the constraints. Here we give some examples.

1. This constraint can be added by modifying the Emp table:

```
CREATE TABLE Emp
         (  eid INTEGER,
            ename CHAR(20),
            age INTEGER,
            salary REAL,
            PRIMARY KEY (eid),
            CHECK (salary>1000))
```
2. Create an assertion as follows:

```sql
CREATE ASSERTION ManagerIsEmployee
CHECK ( ( SELECT COUNT(*)
    FROM Dept D
    WHERE D.managerid NOT IN ( SELECT eid FROM Emp))=0)
```

Another option is to create a constraint on the Dept table.

3. This constraint can be added by modifying the Works table:

```sql
CREATE TABLE Works ( eid INTEGER,
   did INTEGER,
   pcttime INTEGER,
   PRIMARY KEY (eid, did),
   CHECK ( NOT EXISTS( SELECT W.eid
                              FROM Works W
                              GROUP BY W.eid
                              HAVING Sum(pcttime)>100))
```

4) Create an assertion as follows:

```sql
CREATE ASSERTION ManagerHigherSalary
CHECK (NOT EXISTS (SELECT E.eid
                        FROM Emp E, Emp M, Works W, Dept D
                        WHERE E.eid = W.eid AND W.did = D.did
                        AND D.managerid = M.eid
                        AND E.salary > M.salary))
```

**Problem 4.**

1) a) BD. b) The decomposition is not lossless join since the intersection of the two relations is empty. However, it is dependency preserving.

2) a) AB and CB. b) It is lossless join since the intersection is C and it is a key in ACD. It is not DP since the FD AB → C cannot be tested using single tables.

3) a) A and C. b) It is lossless join since the intersection is A that is the key in both relations. It is also not DP since from the functional dependencies in each table (A → BC, C → A, A → C) we cannot infer also C → D.

4) a) A. b) It is lossless join since the intersection is A that is the key in both relations. It is not DP since B → C cannot be inferred by the FDs in each table.
5) a) It is not lossless join. Either we join first AB and AD to get ABD (which is lossless join) and then join with CD or we join first AD and CD and then with AB. In both cases the common attribute is D and D is not a key in the relations that are joined, so the decomposition is not lossless join. It is not DP since B \rightarrow C cannot be inferred by the FDs in each table.

**Problem 5.**

a) Minimal cover is: \( G = \{XZ \rightarrow YB, YA \rightarrow C, C \rightarrow W, B \rightarrow G\} \)

b) Yes. We compute \( XZA^+ = \{XZAYBCGW\} \), so \( XZA \rightarrow YB \) can be implied from F.

c) The decomposition is lossless join since the intersection is YAB, which is the key in the second relation.

d) Yes it is. The only FD that does not exist in one of the tables is \( XZ \rightarrow G \) and this can be implied by the other FDs.