1. Consider the following instance of a relation R(A,B,C,D)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a2</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a3</td>
<td>b1</td>
<td>c2</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c1</td>
<td>d2</td>
</tr>
</tbody>
</table>

State whether this instance of R is consistent with the following functional dependencies. If not, state which dependencies do not hold, and give an example to illustrate why each does not hold.

a. A → B, C → A, B → D

A → B does **not** hold because a2 → {b1, b2}

C → A is consistent

B → D is consistent

b. \{A,B\} → C, D → B

\(\{A,B\} \rightarrow C\) is consistent

D → B is consistent

c. \{B,D\} → A, \{A,B\} → D

\(\{B,D\} \rightarrow A\) does **not** hold because (b1, d1) → {a2, a3}

\(\{A,B\} \rightarrow D\) is consistent

2. Consider the relation R(A,B,C,D,E,F,G,H,I,J) and the functional dependencies

\(\{A,B\} \rightarrow C, A \rightarrow \{D,E\}, B \rightarrow F, F \rightarrow \{G,H\}, D \rightarrow \{I,J\}, G \rightarrow B\)

a. Determine all candidate keys for R.

From \(\{A,B\} \rightarrow C\) and \(A \rightarrow \{D,E\}\) we can deduce \(\{A,B\} \rightarrow \{C, D, E\}\)

From B → F and F → {G,H} we can deduce B → {F,G,H}, whence \(\{A,B\} \rightarrow \{C, D, E, F, G, H\}\)

From D → {I,J} we finally conclude \(\{A,B\} \rightarrow \{C, D, E, F, G, H, I, J\}\) whence \(\{A,B\}\) is a candidate key

From G → B we get \(\{A,G\} \rightarrow \{A,B\} \rightarrow \{B, C, D, E, F, H, I, J\}\) whence \(\{A,G\}\) is a candidate key

From F → BG we get \(\{A,F\} \rightarrow \{A,G\} \rightarrow \{B, C, D, E, G, H, I, J\}\) whence \(\{A,F\}\) is a candidate key

Thus there are three candidate keys: \(\{A,B\}, \{A,F\}, \{A,G\}\)

b. If R is not in 2NF decompose it into 2NF relations.
Recall that a relation is in second normal form (2NF) if it is in 1NF and every non-prime attribute is fully functionally dependent on every candidate key of R. Here the non-prime attributes (those that are not part of any candidate key) are C, D, E, H, I, J.

Note that R is not in 2NF because D and E are not fully functionally dependent on any of the candidate keys (they only depend on A); because H only depends on F (or B, or G); and because ultimately I and J only depend on A.

We noted that a relation which is not in 2NF can be transformed into a set of relations which are in 2NF (without losing information from the original relation) by identifying each non-full functional dependency and forming projections to remove the attributes that depend on the determinants of each of the non-full dependencies. The determinant and their dependent attributes are placed in a separate relation. Applying this to the relation R we get the following decomposition

\begin{align*}
R & (A, B, C, F, G) \\
R1 & (A, D, E, I, J) \\
R2 & (F, H)
\end{align*}

c. If the relations from part b above are not in 3NF, decompose them into 3NF relations.

Recall that a relation scheme is in third normal form (3NF) if it is in 2NF and no non-prime attribute that is transitively dependent on a candidate key. Let us apply this criterion to each of the relations R, R1, and R2 above.

\begin{align*}
R & (A, B, C, F, G) : \text{ The candidate keys are } \{A,B\}, \{A,F\} \text{ and } \{A,G\}. \text{ The only non-prime attribute is } C \text{ and this is not transitively dependent on any of the candidate keys, so } R \text{ is in 3NF.} \\
R1 & (A, D, E, I, J) : \text{ The only candidate key is } A. \text{ Note that } I \text{ and } J \text{ are transitively dependent on } A \text{ (via } D), \text{ so } R1 \text{ is not in 3NF.} \\
R2 & (F, H) : \text{ With only two attributes relation } R2 \text{ is automatically in 3NF}
\end{align*}

Thus, we need only decompose R1 to 3NF relations. We noted that a relation which is 2NF but not in 3NF can be transformed into an equivalent set of 3NF relations by finding any non-prime attributes which are transitively dependent on a key and placing them and their determinants in a new relation. In this case we would place D, I, and J in a new relation (but keep D in R1), say R11, yielding the decomposition

\begin{align*}
R1 & (A, D, E) \\
R11 & (D, I, J)
\end{align*}

Thus our complete 3NF decomposition (with associated functional dependencies) is

\begin{align*}
R & (A, B, C, F, G) \quad \text{with dependencies } \{A,B\} \rightarrow C, \ B \rightarrow F, \ F \rightarrow G, \ G \rightarrow B \\
R1 & (A, D, E) \quad \text{with dependencies } A \rightarrow \{D,E\} \\
R11 & (D, I, J) \quad \text{with dependencies } D \rightarrow \{I,J\} \\
R2 & (F, H) \quad \text{with dependency } F \rightarrow H
\end{align*}

We note that from these dependencies we can re-derive all of the given dependencies.

3. Consider the following relation R that represents enrollments and schedules of course sections at a university

\[ R(Course\#, \ Sec\#, \ StID, \ FacID, \ Dept, \ Credits, \ Term, \ Time, \ Room, \ Grade) \]

Suppose also the following functional dependencies hold on R

\begin{align*}
\text{CourseNo} \rightarrow \{\text{Dept, Credits}\} \\
\{\text{CourseNo, SecNo, Term}\} \rightarrow \{\text{Time, Room, FacID}\} \\
\{\text{CourseNo, SecNo, Term, StID}\} \rightarrow \text{Grade} \\
\{\text{Room, Time, Term}\} \rightarrow \{\text{FacID, CourseNo, SecNo}\} \\
\{\text{Room, Time}\} \rightarrow \text{Dept} \\
\text{FacID} \rightarrow \text{Dept}
\end{align*}
a. Determine all the candidate keys for R.
   
   From \( (\text{CourseNo}, \text{SecNo}, \text{Term}) \rightarrow (\text{Time}, \text{Room}, \text{FacID}) \) and  
   \( (\text{CourseNo}, \text{SecNo}, \text{Term}, \text{StID}) \rightarrow \text{Grade} \)
   we derive that \( (\text{CourseNo}, \text{SecNo}, \text{Term}, \text{StID}) \rightarrow (\text{Time}, \text{Room}, \text{FacID}, \text{Grade}) \)
   
   Now using the dependency \( \text{CourseNo} \rightarrow (\text{Dept}, \text{Credits}) \) we get  
   \( (\text{CourseNo}, \text{SecNo}, \text{Term}, \text{StID}) \rightarrow (\text{Time}, \text{Room}, \text{FacID}, \text{Grade}, \text{Dept}, \text{Credits}) \)
   
   Thus \( (\text{CourseNo}, \text{SecNo}, \text{Term}, \text{StID}) \) is a candidate key for R
   
   Now, however, we note that  
   \( (\text{Room}, \text{Time}, \text{Term}) \rightarrow (\text{FacID}, \text{CourseNo}, \text{SecNo}) \)
   
   whence \( (\text{Room}, \text{Time}, \text{Term}, \text{StID}) \rightarrow (\text{CourseNo}, \text{SecNo}, \text{Term}, \text{StID}) \) and hence  
   \( (\text{Room}, \text{Time}, \text{Term}, \text{StID}) \) is also a candidate key for R

b. Normalize this relation to BCNF.
   
   Recall that a relation is in Boyce-Codd Normal Form (BCNF) if every determinant is a candidate key. R is not in BCNF because it has the following determinants that are not candidate keys:
   
   \( \text{CourseNo} \), because of \( \text{CourseNo} \rightarrow (\text{Dept}, \text{Credits}) \)
   
   \( (\text{CourseNo}, \text{SecNo}, \text{Term}) \) because of \( (\text{CourseNo}, \text{SecNo}, \text{Term}) \rightarrow (\text{Time}, \text{Room}, \text{FacID}) \)
   
   \( (\text{Room}, \text{Time}, \text{Term}) \) because of \( (\text{Room}, \text{Time}, \text{Term}) \rightarrow (\text{FacID}, \text{CourseNo}, \text{SecNo}) \)
   
   \( (\text{Room}, \text{Time}) \) because of \( (\text{Room}, \text{Time}) \rightarrow \text{Dept} \)
   
   FacID because of \( \text{FacID} \rightarrow \text{Dept} \)
   
   To find a BCNF decomposition for R we start with the dependency  
   \( (\text{CourseNo}, \text{SecNo}, \text{Term}) \rightarrow (\text{Time}, \text{Room}, \text{FacID}) \)
   
   and use this to form the decomposition  
   
   \( R1(\text{Course#}, \text{Sec#}, \text{Term}, \text{Time}, \text{Room}, \text{FacID}) \)
   
   \( R(\text{Course#}, \text{Sec#}, \text{StID}, \text{Dept}, \text{Credits}, \text{Term}, \text{Grade}) \)
   
   Now \( R1 \) is in BCNF because while it has the dependency \( (\text{Room}, \text{Time}, \text{Term}) \rightarrow \text{FacID} \) both  
   \( (\text{Room}, \text{Time}, \text{Term}) \) \( (\text{CourseNo}, \text{SecNo}, \text{Term}) \) are candidate keys for \( R1 \). \( R \) is not in BCNF however, because it has the dependency \( \text{CourseNo} \rightarrow (\text{Dept}, \text{Credits}) \) and  
   \( (\text{CourseNo}, \text{SecNo}, \text{Term}, \text{StID}) \) is now the only candidate key of \( R \). Thus we decompose \( R \) into \( R1 \) and \( R2 \), and end up with the following schemas
   
   \( R1(\text{Course#}, \text{Sec#}, \text{Term}, \text{Time}, \text{Room}, \text{FacID}) \) with candidate keys \{\text{Course#}, \text{Sec#}, \text{Term}\} and \{\text{Room}, \text{Time}, \text{Term}\}
   
   \( R2(\text{Course#}, \text{Dept}, \text{Credits}) \) with candidate key \text{Course#}
   
   \( R(\text{Course#}, \text{Sec#}, \text{StID}, \text{Term}, \text{Grade}) \) with candidate key \{\text{Course#}, \text{Sec#}, \text{StID}, \text{Term}\}
   
   Note, however we have lost the following functional dependencies
   
   \( (\text{Room}, \text{Time}) \rightarrow \text{Dept} \)
   
   \( \text{FacID} \rightarrow \text{Dept} \)

c. Normalize this relation to 3NF, preserving all of the above functional dependencies.
   
   Given the candidate keys \( (\text{CourseNo}, \text{SecNo}, \text{Term}, \text{StID}) \) and \( (\text{Room}, \text{Time}, \text{Term}, \text{StID}) \) the non-prime attributes are \text{FacID}, \text{Dept}, \text{Credits}, and \text{Grade}. We also note that since the
We begin by listing all of the determinants of each non-prime attribute that are not candidate keys.

- **FacID**: \(\{\text{CourseNo, SecNo, Term}\}, \{\text{Time, Room, Term}\}\)
- **Dept**: \(\text{CourseNo}, \{\text{Time, Room, Term}\}\)
- **Credits**: \(\text{CourseNo}, \{\text{Time, Room, Term}\}\)
- **Grade**: none beside the candidate keys

From this we form the following decompositions:

- **R(Course#, Sec#, StID, Room, Time, Term, Grade)**
  - dependencies:
    - \(\{\text{Room, Time, Term}\} \rightarrow \{\text{CourseNo, SecNo}\}\)
    - \(\{\text{CourseNo, SecNo, Term}\} \rightarrow \{\text{Room, Time}\}\)
    - \(\{\text{CourseNo, SecNo, Term, StID}\} \rightarrow \text{Grade}\)

- **R1(Term, Time, Room, FacID, Dept, Credits)**
  - dependencies:
    - \(\{\text{Room, Time, Term}\} \rightarrow \{\text{FacID, Credits}\}\)
    - \(\{\text{Room, Time}\} \rightarrow \text{Dept}\)
    - \(\text{FacID} \rightarrow \text{Dept}\)

- **R2(CourseNo, Dept, Credits)**
  - dependencies:
    - \(\text{CourseNo} \rightarrow \{\text{Dept, Credits}\}\)

We note, however that **R1** is not in 2NF because **Dept** is partially dependent on the (only) candidate key \(\{\text{Room, Time, Term}\}\). We thus decompose **R1** to the following

- **R11(Time, Room, Dept)**
  - dependencies:
    - \(\{\text{Room, Time}\} \rightarrow \text{Dept}\)

- **R1(Term, Time, Room, FacID, Credits)**
  - dependencies:
    - \(\{\text{Room, Time, Term}\} \rightarrow \{\text{FacID, Credits}\}\)

Alas we have now placed **FacID** and **Dept** in separate relations, losing the dependency \(\text{FacID} \rightarrow \text{Dept}\). To recapture it we form yet another relation:

- **R12(FacID, Dept)**
  - dependencies:
    - \(\text{FacID} \rightarrow \text{Dept}\)

Thus our decomposition is:

- **R(Course#, Sec#, StID, Room, Time, Term, Grade)**
  - dependencies:
    - \(\{\text{Room, Time, Term}\} \rightarrow \{\text{CourseNo, SecNo}\}\)
    - \(\{\text{CourseNo, SecNo, Term}\} \rightarrow \{\text{Room, Time}\}\)
    - \(\{\text{CourseNo, SecNo, Term, StID}\} \rightarrow \text{Grade}\)

- **R11(Time, Room, Dept)**
  - dependencies:
    - \(\{\text{Room, Time}\} \rightarrow \text{Dept}\)

- **R1(Term, Time, Room, FacID, Credits)**
  - dependencies:
    - \(\{\text{Room, Time, Term}\} \rightarrow \{\text{FacID, Credits}\}\)

- **R12(FacID, Dept)**
  - dependencies:
    - \(\text{FacID} \rightarrow \text{Dept}\)

- **R2(CourseNo, Dept, Credits)**
  - dependencies:
    - \(\text{CourseNo} \rightarrow \{\text{Dept, Credits}\}\)
We note that these relations are also in 3NF and that all of the original functional dependencies can be re-derived from the ones shown.

4. An enterprise has a database of information about various health clubs, but currently keeps all its records in a single table containing five attributes: Club#, Location, Manager, Facility, and Rate.
   - The **location** is a city such as “Charleston,” and several clubs can have the same location. You may assume that two different cities will have different names.
   - The **Club#** value is unique within a given city but not across cities. The combination of Club# and location uniquely determine a club, however.
   - A **manager** is a person assigned to a particular location, and s/he manages all of the clubs in that city.
   - A **facility** is a subunit of a club such as a swimming pool, a sauna, or a tennis court.
   - The **rate** is the charge per hour for using a particular facility and is the same for a given facility across all clubs in the same location.

   a. Express the above constraints of the health club database by means of functional dependencies.

   `(Club#, Location, Facility) → Manager, Rate
   Location → Manager
   (Location, Facility) → Rate`

   b. Identify all of the candidate keys of the table.

   The only candidate key is `{Club#, Location, Facility}`

   c. Decompose the health club relation into a set of relations that is 3NF and where all of the functional dependencies in part a. should be preserved.

   `Club(Club#, Location, Facility)
   candidate key: {Club#, Location, Facility}
   Managers(Location, Manager)
   dependency: Location → Manager
   FacilityRates(Location, Facility, Rate)
   dependency: (Location, Facility) → Rate`

5. Consider the following relations from a database that keeps track of business trips of its sales people (attributes in italics represent foreign keys – the referenced primary key should be obvious from the name used):

   `SPERSON(EmpID, Name)
   TRIP(TripID, EmpID, ToCity, StartDate, EndDate)
   EXPENSE(TripID, Item#, Cost)
   ITEM(Item#, Description, MaxAllowed)`

   Express the following queries in SQL:

   a. Give the details (all TRIP attributes) for all trips to “Miami”.

   ```sql
   SELECT *
   FROM TRIP
   WHERE ToCity = 'Miami'
   ```

   b. Give the names of the salespeople who took a trip to “Chicago”.

   ```sql
   SELECT s.Name
   FROM SPERSON s, TRIP t
   WHERE (t.ToCity = 'Chicago') AND (s.EmpID = t.EmpID)
   ```
c. For every trip in which the cost of an expense item exceeded the maximum allowable cost, give the name of the sales person, the item’s description, the cost, and the city visited.

```
SELECT s.Name, i.Description, e.Cost, t.ToCity
FROM SPERSON s, TRIP t, EXPENSE e, ITEM i
WHERE (s.EmpID = t.EmpID) AND (t.TripID = e.TripID) AND
(e.ItemID = i.ItemID) AND (e.Cost > i.MaxAllowed)
```

d. Give the id numbers of salespeople who took no trips.

```
SELECT s.EmpID
FROM SPERSON s
WHERE NOT s.EmpID IN
  ( SELECT t.EmpID
    FROM TRIP t )
```

6. Consider the following ER diagram, which is based on the one we used in Test 1. Once again, assume here that except for STUDENT and FACULTY, each entity set has an attribute whose name is the first letter of the entity set name followed ID (e.g. the primary key of DEPARTMENT is DID). Assume also that each entity set also has a “Name” attribute that follows the same convention (for example DEPARTMENT has the attribute DName). STUDENT and FACULTY are subclasses of PERSON and besides the ID and Name attributes inherited from PERSON (not shown below) they have attributes Address and Rank, respectively. Assume however that the type for Address is both multi-valued (for example reflecting a student’s permanent and campus addresses, though any number of other addresses are permissible as well) and also is composite, with components Street, City, State, and ZipCode.

Represent this ER diagram in the ODL of the ODMG standard for object-oriented databases. For the base types of each of the simple attributes, use an appropriate type of your choosing.

```
interface Department
{
```

attribute string DID;
attribute string DName;
relationship Set<Faculty> deptFaculty
inverse Faculty::memberOf;
relationship Set<Students> majors
inverse Student::majorsIn;
}

interface Person
{
attribute string PID;
attribute string PName;
}

interface Faculty: Person
{
attribute string rank;
relationship <Department> memberOf
inverse Department::deptFaculty;
relationship Set<Faculty> researchGroup
inverse Faculty:: director;
relationship <Faculty> director
inverse Faculty:: researchGroup;
relationship set<Class> classTaught
inverse Class:: instructor;
}

interface Student: Person
{
attribute Set<struct AddressType
   (string street, string city, string state, string zipCode)> addresses;
relationship Set<Department> majorsIn
inverse Department::majors;
relationship Set<Class> enrolledIn
inverse Class::classRoll;
}

interface Course
{
attribute string CID;
attribute string CName;
relationship Set<Class> sections
inverse Class:: courseOffered;
}

interface Class
{
attribute string term;
relationship Set<Student> classRoll
inverse Student::enrolledIn;
relationship Set<Course> courseOffered
inverse Course::sections;
relationship <Faculty> instructor
inverse Faculty::classTaught;
}
7. Same as problem 6, but this time represent the ER diagram using the appropriate object-oriented features of SQL-3. Once again, for the base types of each of the simple attributes, use an appropriate type of your choosing. You must define row types in your solution as use them as the basis for creating the appropriate tables.

```sql
CREATE ROW TYPE DepartmentType
(
    DID char(5);
    DName varchar(15);
)

CREATE ROW TYPE PersonType
(
    PID char(11);
    PName varchar(20);
)

CREATE ROW TYPE FacultyType:
(
    rank char(5);
    memberOf REF (DepartmentType);
    directedBy REF (FacultyType);
) UNDER PersonType

CREATE ROW TYPE AddressType
(
    street char(5);
    city char(15);
    state char(2);
    zipCode char(5);
)

CREATE ROW TYPE StudentType
(
    addresses SetOf (AddressType);
) UNDER PersonType

CREATE ROW TYPE MajorsInType
(
    majorStudent REF (StudentType);
    studentMajor REF (DepartmentType);
)

CREATE ROW TYPE CourseType
(
    CID char(8) PRIMARY KEY;
    CName varchar(20);
)

CREATE ROW TYPE ClassType
(
    term varchar(6);
    classRoll REF (StudentType);
    courseOffered REF (Course);
    instructor REF (Faculty);
)

CREATE TABLE DEPARTMENT OF TYPE DepartmentType;
CREATE TABLE PERSON OF TYPE PersonType;
CREATE TABLE FACULTY OF TYPE FacultyType;
CREATE TABLE STUDENT OF TYPE StudentType;
CREATE TABLE COURSE OF TYPE CourseType;
CREATE TABLE MAJORS_IN OF TYPE MajorsInType;
CREATE TABLE CLASS OF TYPE ClassType;
```