Review of Normal forms:

**First Normal Form (1NF):** Only atomic attributes are allowed. This means that each attribute (or column in a table) will contain only atomic values. You cannot have a table like:

<table>
<thead>
<tr>
<th>OrderNum</th>
<th>OrderDate</th>
<th>PartNum</th>
<th>NumOrdered</th>
</tr>
</thead>
<tbody>
<tr>
<td>21608</td>
<td>10/20/2003</td>
<td>AT94</td>
<td>11</td>
</tr>
<tr>
<td>21610</td>
<td>10/20/2003</td>
<td>DR93</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DW11</td>
<td>1</td>
</tr>
</tbody>
</table>

A **key** is an attribute that determines the values of other attributes in a table. A **candidate key** is a key that uniquely determines a row in a relation. It is the Primary Key of a table. It could be combination of two or more attributes. For example, the relation:

```
starsin(star_id, movie_id)
```

contains a candidate key composed of two attributes.

A **primary key** is the best candidate key for a table.

**Second Normal Form (2NF):** No partial key dependency.

Simple definition -&gt; Every attribute should depend directly on the entire primary key. If it doesn’t, it should go to a separate table.

This implies that if an attribute depends only on a part of the primary key, it should go to a separate table.

Example:
Relation (Proj_Num, Proj_Name, Emp_Num, Emp_Name, Job, Hours)

In the above relation, there are two keys that make up the primary key (Proj_Num and Emp_Num).

Proj_Name depends only on the Proj_Num and not on the Emp_Num. It needs to be moved to a separate table.
To evaluate the dependencies in a relation, we introduce the idea of Functional Dependency (FD):

**Functional Dependencies**

- **Functional Dependency**: The value of one attribute (the *determinant*) determines the value of another attribute.
  - \( A \rightarrow B \) reads “Attribute B is functionally dependent on A”
  - \( A \rightarrow B \) means if two rows have same value of A they necessarily have same value of B
  - FDs are determined by *semantics*: You can’t say that a FD exists just by looking at data. But can say whether it does not exist by looking at data.

Consider a student relation, the FDs can be evaluated as:

\[
\begin{array}{cccccc}
\text{StudentId} & \text{StuName} & \text{CourseId} & \text{CourseName} & \text{Grade} \\
\end{array}
\]

Can represent FDs with arrows as above, or

- \( \text{StudentId} \rightarrow \text{StuName} \),
- \( \text{CourseId} \rightarrow \text{CourseName} \)
- \( \text{StudentId, CourseId} \rightarrow \text{Grade} \) (and \( \text{StuName, CourseName} \))

StudentId determines StuName,
CourseId determines CourseName
StudentId and CourseId determine the Grade (and of course they can be used to determine StuName and CourseName also).
To get this relation into 2NF (Second Normal Form), split up as follows:

![Diagram illustrating 2NF normalization]

**Third Normal Form**: 2NF + No transitive dependency

Transitive key dependency occurs when a non-key attribute depends on another non-key attribute (instead of depending directly on the primary key attribute).

In the relation below, the capacity attribute depends on the classroom attribute. Both of them are non-key attributes.

![Diagram illustrating 3NF normalization]

To normalize it, split up the relation as follows:

**3NF Normalization**
Classroom Assignment

1. Consider the following data table showing the surgery appointment dates and times in a hospital setting.

<table>
<thead>
<tr>
<th>DoctorID</th>
<th>DoctorName</th>
<th>PatientID</th>
<th>PatientName</th>
<th>Appointment Date</th>
<th>Appointment Time</th>
<th>SurgeryNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1011</td>
<td>Tony Smith</td>
<td>P100</td>
<td>Gillian White</td>
<td>12-Aug-13</td>
<td>10:00AM</td>
<td>S10</td>
</tr>
<tr>
<td>D1011</td>
<td>Tony Smith</td>
<td>P105</td>
<td>Jill Bell</td>
<td>13-Aug-13</td>
<td>12:00PM</td>
<td>S11</td>
</tr>
<tr>
<td>D1024</td>
<td>Helen Pearson</td>
<td>P108</td>
<td>Ian McKey</td>
<td>12-Aug-13</td>
<td>10:00AM</td>
<td>S12</td>
</tr>
<tr>
<td>D1024</td>
<td>Helen Pearson</td>
<td>P110</td>
<td>Ian McKey</td>
<td>13-Aug-13</td>
<td>11:00AM</td>
<td>S13</td>
</tr>
<tr>
<td>D1032</td>
<td>Robin Plevin</td>
<td>P105</td>
<td>Jill Bell</td>
<td>14-Aug-13</td>
<td>2:00PM</td>
<td>S14</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

The table is not normalized and can lead to several anomalies, eg. insertion, deletion, modification anomalies.

**Insertion Anomaly:** To create a new appointment, you need to enter details of the doctor over and over. What happens if a patient doesn’t have a doctor yet?

**Deletion Anomaly:** If you delete an appointment, you lose a lot of data. What if a doctor has only one patient and he cancels. For example, if patient Ian McKey wants to cancel and you delete the two rows for him, you lose the DoctorID value for Dr. Helen Pearson.

**Modification Anomaly:** If you wanted to change the name of a doctor, it would involve changing multiple rows.

Your job is to normalize this table **in steps up to the 3NF**.

**Hints:**

1. The attribute **appointment** as it occurs right now is not a single valued attribute. It contains two values (date and time). To get this table into 1NF, you need to separate them.

2. Evaluation of FDs is subjective, but you need to figure out what would make sense in a real world. You are given the following information to start with:

   DoctorID -> DoctorName
   DoctorID, AppointmentDate, AppointmentTime -> PatientID, PatientName
   DoctorID, AppointmentDate -> SurgeryNum (this condition implies a doctor can only perform one surgery per day).
2. You own an IT staffing company that provides programmers to various clients. The clients sign a contract with you that specifies the programmer’s name, the number of hours per week that the programmer will work with the client and the programmer’s rate ($/hour). Your database also keeps track of the programmer’s SSN and the client’s location in the same table.

The relation is as follows:

Staffing(SSID, ContractID, ProgrammerName, Rate, HoursPerWeek, ClientID, ClientAddress)

The field ClientAddress is multivalued and stores data as follows:

<table>
<thead>
<tr>
<th>..</th>
<th>ClientID</th>
<th>ClientAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>..</td>
<td>101</td>
<td>123 Main Street, Dallas TX 75101</td>
</tr>
<tr>
<td>..</td>
<td>102</td>
<td>123 Campus Street, Dallas TX 75205</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

The table is not normalized and can lead to several anomalies, eg. insertion, deletion, modification anomalies. List the three types of anomalies and normalize this table in steps up to the 3NF.

**Hints:**

1. The attribute **ClientAddress** as it occurs right now is not a single valued attribute. It contains multiple values. To get this table into 1NF, you need to separate them.

2. Evaluation of FDs is subjective, but you need to figure out what would make sense in a real world. For example, it is safe to assume:

SSN, ContractID -> HoursPerWeek, Rate  
SSN -> ProgrammerName  
ContractID -> ClientID, ClientAddress  
ClientID -> ClientAddress

**Further questions:**

After normalizing, run the following queries:

1. Find the average number of hours per week for each client.  
2. Find the programmer with the highest rate.  
3. Find the highest rate for each client.  
4. Which programmer is not assigned to a client.  
5. Create an list of clients and the number of programmers working for them. Sort by the client with most clients first i.e., by the number of programmers in descending order.