Users:
R(User_ID, Gender, DOB, Password, City, Province, Photo_ID, Name)
Bernstein's Algorithm:
Step 1: Write out all the functional dependencies

1. User_id → Gender
2. User_id → DOB
3. User_id → Password
4. User_id → City
5. User_id → Province
6. User_id → Photo_id
7. User_ID → Name
8. DOB, Name, City, Province → Gender
9. DOB, Name, City, Province → Sexual Orientation
10. DOB, Name, City, Province → Password
11. DOB, Name, City, Province → Photo_id
12. City → Province
13. Photo_id → User_id

Functional dependencies 8 through 11 were made based on the assumption that it is incredibly unlikely to have 2 people with the same name and birthday who are from the same city.

Step 2: Check for redundancies in each of the functional dependencies:

1. User_id → Gender
    a. User_id+ = {User_id, DOB, Password, city, Province, Photo_id, Name, gender}
    b. The above includes gender, so the functional dependency is redundant
2. User_id → DOB
    a. User_id+ = {User_id, gender, Password, city, Province, Photo_id, Name}
    b. The above does not include DOB, so the functional dependency is not redundant
3. User_id → Password
    a. User_id+ = {User_id, DOB, gender, city, Province, Photo_id, Name, Password}
    b. The above does include Password, so the functional dependency is redundant
4. User_id → City
    a. User_id+ = {User_id, DOB, Password, gender, Province, Photo_id, Name}
    b. The above does not include city, so the functional dependency is not redundant
5. User_id → Province
    a. User_id+ = {User_id, DOB, Password, city, gender, Photo_id, Name, Province}
    b. The above does include Province, so the functional dependency is redundant
6. User_id → Photo_id
    a. User_id+ = {User_id, DOB, Password, city, Province, Name, Photo_id}
    b. The above does include Photo_id, so the functional dependency is redundant
7. User_id → Name
    a. User_id+ = {User_id, DOB, Password, city, Province, Photo_id, gender}

    b. The above does not include Name, so the functional dependency is not redundant
8. DOB, Name, City, Province → Gender
    a. {DOB, Name, City, Province}+ = {DOB, Name, City, Province, Sexual Orientation, Password, Photo_id, User_id, gender}
    b. The above does include Gender, so the functional dependency is redundant
9. DOB, Name, City, Province → Sexual Orientation
    a. {DOB, Name, City, Province}+ = {DOB, Name, City, Province, gender, Password, Photo_id, User_id, Sexual Orientation}
    b. The above does include Sexual Orientation, so the functional dependency is redundant
10. DOB, Name, City, Province → Password
    a. {DOB, Name, City, Province}+ = {DOB, Name, City, Province, gender, Sexual Orientation, Photo_id, User_id, Password}
    b. The above does include Password, so the functional dependency is redundant
11. DOB, Name, City, Province → Photo_id
    a. {DOB, Name, City, Province}+ = {DOB, Name, City, Province, gender, Sexual Orientation, Password}
    b. The above does not include Photo_id, so the functional dependency is not redundant
12. City → Province
    a. City+ = {city}
    b. The above does not include Province, so the functional dependency is not redundant
13. Photo_id → User_id
    a. Photo_id+ = {Photo_id}
    b. The above does not include User_id, so the functional dependency is not redundant.

Note: Since city → Province, the determinants for dependencies 8 through 11 can be reduced as follows:

8. DOB, Name, City → Gender
9. DOB, Name, City → Sexual Orientation
10. DOB, Name, City → Password
11. DOB, Name, City → Photo_id

Final set of functional dependencies:
User_id → DOB
User_id → City
User_id → Name
DOB, Name, City → Photo_id
City → Province
Photo_id → User_id
Step 3: Determine the keys

Sub-steps:

1. Determine if any attributes are not used in any functional dependencies
    a. These are present in all of the keys
2. Determine if any attributes are only present on the right side of a functional dependency
    a. These are not present in any of the keys
3. Test the remaining attributes to see if they are keys.

Sub-Step 1:
No such attributes exist. All attributes are present in some functional dependency

Sub-Step 2:
{Gender , Sexual Orientation, Password)
The above attributes will not be present in any of the keys.

Sub-Step 3:
Remaining attributes: {User_id, Photo_id, DOB, name, city, Province}. Check each:

User_id + = {User_id, gender, DOB, Password, City, Province, Photo_id, Name} → User_id is a key!

Photo_id+ = { Photo_id, User_id, gender, DOB, Password, City, Province, Name} → Photo_id is a key!

Name, City, DOB, and province, by themselves are not keys.

{Name, City, DOB}+ = {Name, City, DOB, gender, Province, Photo_id, User_id, Password} → {Name, City, DOB} is a key!

Step 4: Derive the final relational schema:
R.1 (User_id, DOB, City, Name)
R.2 (DOB, Name, City, Photo_id)
R.3 (City, Province)
R.4(Photo_id, User_id)

BCNF Algorithm:

R.1(User_id, DOB, city, name):
User_id → DOB
User_id → city
User_id → Name
Has 1 candidate key: User_id, and since every attribute is determined by a candidate key in this relation, R.1 is in the BCNF form.

R.2(DOB, city, name, Photo_id):

DOB, city, Name → Photo_id

Has 1 candidate key: {DOB, city, Name}, and since every attribute is determined by a candidate key in this relation, R.2 is in the BCNF form.

R.3(city, Province):

City → Province

Has 1 candidate key: city,, and since every attribute is determined by a candidate key in this relation, R.3 is in the BCNF form.

R.4(Photo_id, User_id):

Photo_id → User_id

Has 1 candidate key: Photo_id, and since every attribute is determined by a candidate key in this relation, R.4 is in the BCNF form.

Dates:

R(User_id, match_id,Date_Number, match_Date, anniversary, username, Location, Number_of_Dates)

Bernstein's Algorithm:

Step 1: Write out all the functional dependencies

User_id → Username
User_id, Match_id → Number_of_Dates
User_id, Match_id → Anniversary
User_id, Match_id → match_Date
match_Date → Anniversary
Date_number → Location

Step 2: Check for redundancies in each of the functional dependencies:

1. User_id → Username
    a. User_id+ = {User_id}
    b. The above does not include Username, so the functional dependency is not redundant
2. User_id, Match_id → Number_of_Dates
    a. {User_id, Match_id}+={User_id, Match_id, Anniversary, match_Date}
    b. The above does not include Number_of_Dates, so the functional dependency is not redundant
3. User_id, Match_id → Anniversary
    a. {User_id, Match_id}+ = {User_id, Match_id, Date_Number, location, match_Date, Anniversary}
    b. The above includes Anniversary, so the functional dependency is redundant
4. User_id, Match_id → match_Date
    a. {User_id, Match_id}+ = {User_id, Match_id, Date_Number, location, Anniversary}
    b. The above does not included match_Date, so the functional dependency is not redundant

5. match_Date → Anniversary
    a. match_Date+ = {match_Date}
    b. The above does not included Anniversary, so the functional dependency is not redundant
6. Date_Number → Location
    a. Date_Number+ = {Date_Number}
    b. The above does not included Location, so the functional dependency is not redundant

The final set of functional dependencies:
User_id → Username
User_id, Match_id → Number_of_Dates
User_id, Match_id → match_Date
match_Date → Anniversary
Date_number → Location

Step 3: Determine the keys

Sub-Step 1:
No such attributes exist that are not present in any functional dependencies.

Sub-Step 2:
{Username, Number_of_Dates, Anniversary, Location}
The above attributes will not be present in any of the keys

Sub-Step 3:
Remaining attributes:
{User_id, Match_id, match_Date, Date_Number}

User_id+ = {User_id, Username} → Not a key!
Match_id+ = {Match_id} → Not a key!

{User_id, Match_id}+ = {User_id, Match_id, Username, Number_of_Dates, match_Date, Anniversary} → Not a key! (Missing Date_Number, Location)

{User_id, Match_id, Date_Number}+ = {User_id, Match_id, Date_Number, Username, Number_of_Dates, match_Date, Anniversary, location} → A key!

match_Date+ = {match_Date, Anniversary} → not a key!

Date_Number+ = {Date_Number, Location} → not a key!

Step 4: Derive the final relational schema:

R.1 (User_id, Username)
R.2 (User_id, Match_id, Number_of_Dates, match_Date)
R.3 (match_Date, Anniversary)
R.4 (Date_Number, Location)
R.5 (User_id, Match_id, Date_Number) → Added so that one of the relations contains the candidate key.

All the above relations are already in BCNF.

BCNF Algorithm
Since both of our above answers are already in BCNF, we have added an extra example here to show our understanding of how to use the algorithm. The table used is NOT a part of our database

Administration:
R(Admin_ID,email, Password)

FD: {Admin_ID → Email
Email → password}

Admin_ID+ = {Admin_ID,Email, Password} → A key!
Email+ = {Email, Password} → Not a key

Candidate key: Admin_ID
Violation of BCNF: Email → Password

Separate R into R.1 and R.2 where:
R.1(Admin_ID, Email)
R.2(Email, password)