Give the periods during which two users with ID “1” and “2” are located in the same region.

```
create or replace function minDate(one date, two date)
returns date
language plpgsql
as
$$
begin
    return CASE WHEN one < two then one else two end;
end;
$$;

create or replace function maxDate(one date, two date)
returns date
language plpgsql
as
$$
begin
    return CASE WHEN one > two then one else two end;
end;
$$;


SELECT UL1.point, maxDate(UL1.Start, UL2.Start), minDate(UL1.End, UL2.End)
FROM User U1, User U2, UserLocation UL1, UserLocation UL2,
WHERE U1.ID=1 AND U2.ID=2
AND UL1.UID=1 AND UL2.UID=2
AND UL1.point = UL2.point
AND maxDate(UL1.Start, UL2.Start) < minDate(UL1.End, UL2.End)
```

By using table “R1(Start, End)” containing the results of the first query, give the periods during which the user with ID “1” is subscribed to user with ID “2” but is not located in the same region

```
SELECT UL.Start, UL.End
FROM UserLocation UL
WHERE UL.SID=1 AND UL.ID=2
AND NOT EXISTS (
    SELECT *
    FROM R1
    WHERE UL.Start < R1.End AND UL.End < H.Start
)

UNION

-- Case 2: No R1 covering the start of UL
SELECT UL.Start, UL.End
FROM UserLocation UL, R1 R11
WHERE UL.SID=1 AND UL.ID=2
AND UL.Start < R11.Start AND UL.End > R11.Start
AND NOT EXISTS (
    SELECT *
    FROM R1 R12
    WHERE R12.Start < R11.Start AND R12.End > UL.Start
)

UNION
```

```sql
-- Case 3: No R1 covering the end of UL
SELECT UL.Start, UL.End
FROM UserLocation UL, R1 R11
WHERE UL.SID=1 AND UL.ID=2
AND UL.Start < R11.End AND UL.End > R11.End
AND NOT EXISTS (
    SELECT *
    FROM R1 R12
    WHERE R12.Start < UL.End AND R12.End < R11.End
)

UNION

-- Case 4: There is a hole in UL not covered by any R1
SELECT UL.Start, UL.End
FROM UserLocation UL, R1 R11, R1 R12
WHERE UL.SID=1 AND UL.ID=2
AND UL.Start < R11.End AND R11.End < R12.Start AND R12.Start < UL.End
AND NOT EXISTS (
    SELECT *
    FROM R1 R13
    WHERE R11.End < R13.End AND R13.Start < R12.Start
)


-- 1.3
-- First step: Construct intervals during which the number of subscriptions of a user
does not change
WITH Instants(ID, Instant) AS (
    SElECT DISTINCT ID, Start FROM UserSubscription
    UNION
    SElECT DISTINCT ID, End FROM UserSubscription
),

Intervals(ID, Start, End) AS (
    SELECT DISTINCT I1.ID, I1.Instant, I2.Instant
    FROM Instans I1, Instants I2
    WHERE I1.ID = I2.ID AND I1.Instant < I2.Instant
    AND NOT EXISTS (
        SELECT * FROM Instants I3
        WHERE I1.ID = I3.ID
        AND I1.Instant < I3.Instant AND I3.Instant < I2.Instant
    )
)
```

Give the number of subscriptions each user has. Do not coalesce the results.

```sql
TempCountUser(ID, NbSub, Start, End) AS (
    SELECT US.ID, COUNT(US.SID), I.Start, I.End
    FROM UserSubscription US, Intervals I
    WHERE US.ID = I.ID
    AND US.Start <= I.Start AND US.End >= I.End
    GROUP BY I.Start, I.End
)
```

With table “R3(Name, Count, Start, End)” containing the result of the answer of Question 3, coalesce R3.

```
-- Third step: Coalescing the results of 1.3
SElECT
FROM TempCountUser F, TempCountUser L
WHERE F.ID = L.ID and F.NbSub = L.NbSub AND F.Start < L.End
AND NOT EXISTS (
    SELECT * FROM TempCount M
    WHERE M.ID = F.ID AND M.NbSub = F.NbSub
    AND NOT EXISTS (
        SELECT FROM TempCount T1
        WHERE T1.ID = F.ID AND T1.NbSub = F.NbSub
        AND T1.Start < M.Start AND M.Start <= T1.End
    )
)
AND NOT EXISTS(
    SELECT * FROM TempCount T2
    WHERE T2.ID = F.ID and T2.NbSub = F.NbSub
    AND ((T2.Start < F.Start AND F.Start <= T2.End)
    OR (T2.Start <= L.End AND L.End < T2.End))
)
```

Give the server on which the highest number of messages were posted and are still active on “01/01/2023”. To know on which server a message is stored, use the location of the user at the time of posting the message.

```
WITH User_Location(ID, Loc, Start, End) AS (
    SELECT U.ID, UL.point, maxDate(U.Start, UL.Start), minDate(U.End, M.End)
    FROM User U, UserLocation UL
    WHERE U.ID = UL.UID
    AND maxDate(U.Start, UL.Start) < minDate(U.end, UL.End)
),

-- Join User_Location and Message
User_Location_Message(UID, point, MID, Start, End) AS (
    SELECT UL.ID, UL.point, M.ID, maxDate(UL.Start, M.Start), minDate(UL.End, M.End)
    FROM User_Location UL, Message M
    WHERE UL.ID = M.UID
    AND maxDate(UL.Start, M.Start) < minDate(UL.End, M.End)
),

-- Join Server and ServerStore
Server_Location(SID, point, RID) AS (
    SElECT S.ID, S.point, SS.RID
    FROM Server S, ServerStore SS
    WHERE S.ID = SS.SID
),

-- Join User_Location_Message and Server_Location
User_Message_Server_Region(UID, MID, Start, End, SID, RID) AS (
    SElECT ULM.UID, ULM.MID, ULM.Start, ULM.End, SL.SID, SL.RID
    FROM User_Location_Message ULM, Server_Location SL, Region R
    WHERE ST_Contains(R.region, ULM.point) = True AND ST_Contains(R.region, SL.point) =
True
)
```

```
SELECT SID, COUNT(MID)
FROM User_Message_Server_Region
WHERE DATE(Start) <= '01-01-2023' AND DATE(End) >= '01-01-2023'
GROUP BY UMSR.SID
ORDER BY DESC
LIMIT BY 1
```

Give, for each user, the server on which is stored the last active message posted by the user

```
SELECT UMSR1.UID, UMSR2.SID
FROM User_Message_Server_Region UMSR1
WHERE End > getTime()
WHERE NOT EXIST (
    SELECT *
    FROM UMSR2
    WHERE UMSR1.End < UMSR2.End
)
```

By using a table “R5(UID, SID) containing the results of the previous query, give the ID of the user who is currently located the farthest of the server hosting her last active message.

```
WITH R5_UserLoc_ServerLoc(UID, Upoint, SID, Spoin) AS (
    SELECT R5.UID, UL.point, R5.SID, SS.point
    FROM R5, UserLocation UL, ServerStore SS
    WHERE R5.UID = UL.UID AND R5.SID = SS.SID
)

SELECT UID
FROM R5_UserLoc_ServerLoc R5US1
WHERE NOT EXIST (
    SELECT *
    FROM R5_UserLoc_ServerLoc R5US2
    AND ST_Distance(R5US1.Upoint, R5US1.Spoint) < ST_Distance(R5US2.Upoint, R5US2.Spoint)
)
```

Give, for each server, its distance with the centroid of all the regions for which it is hosting messages.

```
SELECT SID, ST_Distance(S.point, ST_Centroid(ST_Union(R.region)))
FROM Server S, ServerStore SS, Region R
WHERE S.ID = SS.SID and R.ID = SS.RID
GROUP BY SID
```

The location of users is only kept during the time they are registered on the platform

```
CREATE TRIGGER UserLocTimeInsideUserTime_1
ON User
AFTER Update, Delete
AS
IF EXISTS (
    SELECT * FROM UserLocation UL
    WHERE UL.UID IN (
        SELECT ID FROM Deleted
    )
    AND NOT EXISTS (
```

```
        SELECT * FROM User U
        WHERE U.ID = UL.UID
        AND U.Start <= UL.Start and U.End >= UL.End
    )
)
BEGIN
    raiserror "The lifecycle of UserLocation must be included in the lifecycle of User"
    rollback
END


CREATE TRIGGER UserLocTimeInsideUserTime_2
ON UserLocation
AFTER Insert, Update
AS
IF EXISTS (
    SELECT * FROM Inserted I
    WHERE NOT EXISTS (
        SELECT * FROM User U
        WHERE U.ID = I.UID
        AND U.Start <= I.Start and U.End >= I.End
    )
)
BEGIN
    raiserror "The lifecycle of UserLocation must be included in the lifecycle of User"
    rollback
END
```

At each point in time a user has a single location.

```
CREATE TRIGGER UserSingleLocation
ON UserLocation
AFTER Insert, Update
AS
IF EXISTS (
    SELECT * FROM Inserted I
    WHERE 1 < (
        SELECT COUNT(*) FROM UserLocation UL
        WHERE I.UID = UL.UID
        AND I.Start < UL.End AND UL.Start < I.End
    )
)
BEGIN
    raiserror "At any point in time a user can have only one location"
    rollback
End
```

The location of a server is contained in its region.

```
-- Case 1: Update point in Server
CREATE TRIGGER ServerLocInsideRegion_1
ON Server
AFTER Update
AS
IF EXISTS (
    SELECT * FROM Inserted I, ServerStore SS, Region R
    WHERE I.ID = SS.SID and SS.RID = R.ID
```

```
    AND ST_Contains(R.region, I.point) = False
)
BEGIN
    raiserror "Server location must be contained in its region"
    rollback
End

-- Case 2: Update region in Region
CREATE TRIGGER ServerLocInsideRegion_2
ON Region
AFTER Update
AS
IF EXISTS (
    SELECT * FROM Server S, ServerStore SS, Inserted I
    WHERE S.ID = SS.SID and SS.RID = I.ID
    AND ST_Contains(I.region, S.point) = False
)
BEGIN
    raiserror "Server location must be contained in its region"
    rollback
End

-- Case 3: Insert or update ServerStore
CREATE TRIGGER ServerLocInsideRegion_3
ON Region
AFTER Insert, Update
AS
IF EXISTS (
    SELECT * FROM Server S, Inserted I, Region R
    WHERE S.ID = I.SID and I.RID = R.ID
    AND ST_Contains(R.region, S.point) = False
)
BEGIN
    raiserror "Server location must be contained in its region"
    rollback
End
```

There are at most 3 servers located in a region.

```
CREATE TRIGGER RegionNbServer
ON ServerStore
AFTER Insert, Update
AS
IF EXISTS (
    SELECT * FROM ServerStores W
    GROUP BY W.RID
    HAVING COUNT(*) >3
)
BEGIN
    raiserror "A region can have at most 3 servers"
    rollback
End
```