Université libre de Bruxelles
Faculté des Sciences Appliquées

January 23, 2024

INFO-H-415 – Advanced databases

First session examination

A shipping company put a database in place to keep track of their boats. An inventory of all the past and currently used shipping lanes is kept (i.e., routes taken by boat from one port to another). When moving, each boat has its real position updated every hour. The company also keeps track of the crew present on its boat. Each crew member has a role on the boat that can change throughout her career.

Next is an excerpt of the relational schema used.

- **Boat** (boatId, fromDate, toDate, name, capacity)
  - fromDate is the commissioning date of the boat
  - toDate is the decommissioning date (it is NULL if the boat is still in service)
- **ShippingLane** (laneId, geom)
  - geom is a MULTILINE geometry
- **Position** (boatId, time, laneId, geom)
  - boatId references Boat.boatId
  - laneId references ShippingLane.laneId
  - time is a timestamptz datatype
  - geom is a POINT geometry
- **Crew** (crewId, firstName, lastName, aStreet, aCity, aZip, aCountry, telephone)
- **Managed** (crewId, fromDate, toDate, manager)
  - crewId references Crew.crewId
  - manager references Crew.crewId
  - fromDate is the date upon which the crew member was put under her manager
  - toDate is the date upon which he stopped being under her manager
- **Role** (crewId, fromDate, toDate, role)
  - crewId references Crew.crewId
  - fromDate marks the start of the role of a crew member
  - toDate marks the end of the role
- **PartOf** (crewId, fromDate, toDate, boatId)
  - crewId references Crew.crewId
  - boatId references Boat.boatId
  - fromDate is the date the crew member has arrived on the boat
  - toDate is the date when the crew member left

1 Spatial Databases

For the following questions, suppose you are using a PostgreSQL database. We also suppose that the geometries are in the SRID 4326 (WGS84) spatial reference system.

a. List the name of boats that have deviated from their shipping lane by more than a kilometre as well as the most recent time when they did.

```
SELECT b.name, MAX(p.time) as date
FROM Boat b, Position p, ShippingLane s
WHERE b.boatId = p.boatId AND p.lane = s.laneId AND
    ST_Distance(p.geom, s.geom) >= 1000
GROUP BY b.name
```

b. For each boat, find how far it has gotten on its latest shipping lane. This progress will be presented as a percentage of the total length of the shipping lane.

```
WITH MaxTime(mTIme) AS (
    SELECT boatId, MAX(time)
    FROM Position
    GROUP BY boatId )
SELECT p.boatId, ST_LineLocatePoint(s.geom, p.geom) * 100 AS progress
FROM Position p, MaxTime m, ShippingLane s
WHERE p.boatId = m.boatId AND p.time = m.mTime AND p.lane = s.laneId
```

c. For each boat, count how many other boats it has crossed so far (we suppose two boats have crossed each other if they have been ever less than 500 meters from each other).

```
SELECT p.boatId, COUNT(DISTINCT(p.boatId, p2.boatId))
FROM Position p1, Position p2
WHERE p.boatId < p2.boatId AND p.time = p2.time AND
    ST_Distance(p.geom , p2.geom) <= 500
GROUP BY p.boatId
```

d. Find the shipping lanes that cross the shipping lane where the oldest boat (that is still running) has last been.

```
WITH OldestBoat(BoatId) AS (
    SELECT boadId
    FROM Boat
    WHERE fromDate = (
        SELECT MIN(fromDate)
        FROM Boat
        WHERE toDate is NULL )
        LIMIT 1 ),
OldestLane(LaneId) AS (
    SELECT Lane
    FROM Position b, OldestBoat O
    WHERE b.boadId = O.BoatId
    ORDER BY time DESC LIMIT 1 ),
SELECT s.laneId
FROM ShippingLane S, OldestLane O
WHERE ST_Intersects(S.geom, O.geom)
```

2 Temporal Databases

a. Give the role and boat history of the crew members.

```
SELECT c.firstName, c.lastName, greatest(p.fromDate, r.fromDate) as start,
    least(p.toDate, r.toDate) as end
FROM Crew c, PartOf p, Role r
WHERE c.crewId = p.crewId AND c.crewId = r.crewId AND
    greatest(p.fromDate, r.fromDate) < least(p.toDate, r.toDate)
```

b. Give the name and last role of crew members that are not on any boat at this time (suppose that if a crew member is currently on a boat, the toDate of its entry in the PartOf table is NULL).

```
WITH LastRole(crewId, role) AS (
    SELECT crewId, role, MAX(fromDate)
    FROM Role
    GROUP BY crewId )
SELECT c.firstName, c.lastName, role
FROM Crew c, LastRole l
WHERE r.crewId = l.crewId AND role = r.role AND NOT EXISTS (
    SELECT *
    FROM PartOf p
    WHERE p.crewId = c.crewId AND p.toDate IS NULL )
```

c. Give the history of the boat with the highest number of crew members (do not coalesce the results).

```
WITH Instants(instant) AS (
    SELECT DISTINCT fromDate FROM PartOf
    UNION
    SELECT DISTINCT toDate FROM PartOf ),
Intervals(fromDate, toDate) AS (
    SELECT DISTINCT i1.instant, i2.instant
    FROM Instants i1, Instants i2
    WHERE i1.instant < i2.instant AND NOT EXISTS (
        SELECT *
        FROM Instants i3
        WHERE i1.instant < i3.instant AND i3.instant < i2.instant ) )
SELECT b.boatId, i.fromDate, i.toDate
FROM Boat b, Intervals i, PartOf p
WHERE b.boatId = p.boat AND p.fromDate <= i.fromDate AND p.toDate >= i.toDate
GROUP BY b.boatId, i.fromDate, i.toDate
HAVING COUNT(*) >= ALL(
    SELECT COUNT(*)
    FROM PartOf p1, Boat b1
    WHERE b1.boatId = p1.boat AND p1.fromDate <= i.fromDate AND
        p1.toDate >= i.toDate
    GROUP BY b1.boatId, i.fromDate, i.toDate)
```

d. Coalesce the result of your previous answer.

First, let us suppose we put the result of the previous query in a temporary table BiggestBoat(boatId, fromDate, toDate)

```
SELECT DISTINCT f.boatId, f.fromDate, l.toDate
FROM BiggestBoat f, BiggestBoat l
```

```
WHERE f.boatId = l.boatId AND f.fromDate < l.toDate AND NOT EXISTS (
    SELECT *
    FROM BiggestBoat m
    WHERE m.boatId = f.boatId AND f.toDate < m.fromDate AND
        m.fromDate <= l.fromDate AND NOT EXISTS (
        SELECT *
        FROM BiggestBoat t1
        WHERE t1.boatId = f.boatId AND t1.fromDate < m.fromDate AND m.fromDate <= t1.toDate ) )
    AND NOT EXISTS (
        SELECT *
        FROM BiggestBoat t2
        WHERE t2.boatId = f.boatId AND
            ((t2.fromDate < f.fromDate AND f.fromDate <= t2.toDate) OR
            (t2.fromDate <= l.toDate AND l.toDate < t2.toDate)) )
ORDER BY f.fromDate
```

3 Active Databases

For the following questions, suppose you are using a SQLServer database.

a. A shipping lane cannot be fully contained in another shipping lane.

```
CREATE TRIGGER notContainsLanes ON ShippingLane
AFTER INSERT, UPDATE AS
IF EXISTS (
    SELECT *
    FROM INSERTED AS i, ShippingLane l
    WHERE ST_Contains(i.geom, l.geom)
BEGIN
    RAISERROR 13000 'A shipping lane cannot be contained into another one'
    ROLLBACK
END
```

b. Only active ships (i.e., ships that are not decommissioned) can perform trips and send their position.

```
CREATE TRIGGER activeShips ON Positions
AFTER INSERT, UPDATE AS
IF EXISTS (
    SELECT *
    FROM INSERTED AS i
    WHERE NOT EXISTS (
        SELECT * FROM Boat AS b
        WHERE i.boatId = b.boatId AND b.fromDate <= i.fromDate AND
            i.fromDate < b.toDate )
    OR NOT EXISTS (
        SELECT * FROM Boat AS b
        WHERE i.boatId = b.boatId AND b.fromDate < i.toDate AND
            i.toDate <= b.toDate )
    OR EXISTS (
        SELECT * FROM Boat AS b
        WHERE i.boatId = b.boatId AND i.fromDate < b.toDate AND
            b.toDate < i.toDate AND NOT EXISTS (
            SELECT * FROM Boat AS b2
            WHERE b2.boatId = b.boatId AND b2.fromDate <= b.toDate AND
                b.toDate < b2.toDate ) ) )
```

```
BEGIN
    RAISERROR 13000 'A ship must be active to send its position'
    ROLLBACK
END
```

c. At each point in time a crew member has a single role.

```
CREATE TRIGGER singleRole ON Role
AFTER INSERT, UPDATE AS
IF EXISTS (
    SELECT i1.crewId FROM Role AS i1 WHERE 1 < (
        SELECT COUNT(i2.crewId) FROM Role AS i2
        WHERE i1.crewId = i2.crewId AND i1.fromDate < i2.toDate AND
            i2.fromDate < i1.toDate ) )
    OR EXISTS (
        SELECT * FROM Role AS I
        WHERE I.crewId IS NULL )
BEGIN
    RAISERROR 13000 'Violation of sequenced unique constraint'
    ROLLBACK
END
```

d. There cannot be cycles in the manager relationship represented in table Managed.

```
CREATE TRIGGER noncyclicSubordinates ON Managed
AFTER INSERT, UPDATE AS
BEGIN
    CREATE TABLE #Supervision AS
        SELECT crewId, superID
        FROM Managed
        WHERE superID IS NOT NULL;
    WHILE @@rowcount != 0 -- while previous iteration affected some rows
    BEGIN
        IF EXISTS (
            SELECT *
            FROM #Supervision
            WHERE crewId = superID )
        BEGIN
            RAISERROR 'The manager relationship is cyclic'
            ROLLBACK
        END
        INSERT INTO #Supervision
            SELECT DISTINCT s1.crewId, s2.superID
            FROM #Supervision s1, #Supervision s2
            WHERE s1.superID = s2.crewId AND NOT EXISTS (
                SELECT *
                FROM #Supervision S
                WHERE S.crewId = s1.crewId AND S.superID = s2.superID )
    END
END
```

4 Mobility Databases

For this section we will use another database implemented using MobilityDB. This database contains the trips various cars have been doing within Brussels. The database has information about the communes in Brussels as well as various points of interest within those communes. It is supposed that the geometries are in the SRID 3812 (ETRS89 / Belgian Lambert 2008).

- **Vehicles** (vehId, licence, type)
- **Trips** (tripId, vehId, day, trip)
  - tripId is the identifier of the trip
  - vehId references Vehicle.vehId
  - day is the date of the trip
  - trip is a MobilityDB type tgeompoint
- **Communes** (communeId, name, population, geom)
  - geom is the geometry of the commune
- **Points** (pointId, geom)
  - geom is a geometry POINT representing the location of the point of interest

a. Give the first time when each vehicle visited a point of interest in table Points.

```
SELECT T.VehId, p.PointId, MIN(startTimestamp(atValues(t.trip, p.geom))) AS Instant
FROM Trips t, Points p
WHERE ST_Contains(trajectory(t.Trip), p.Geom)
GROUP BY t.vehId, p.pointId;
```

Alternative solution using distance less than 100 meters.

```
SELECT T.VehId, p.PointId, MIN(startTimestamp(atValues(t.trip, p.geom))) AS Instant
FROM Trips t, Points p
WHERE ST_Distance(trajectory(t.Trip), p.Geom) < 100
GROUP BY t.vehId, p.pointId;
```

b. Give the minimal distance between each pair of vehicles whenever their time intervals overlap.

```
SELECT t1.vehId AS car1Id, t2.vehId AS car2Id,
    t1.trip <-> t2.trip AS minDistance
FROM Trips t1, Trips t2
WHERE timeSpan(t1.Trip) && timeSpan(t2.Trip)
```

c. Give for each vehicle the duration of the trip of maximum length over all of its trips. The result should show three columns: vehId, duration of the trip, trajectory of the trip. If a vehicle has multiple trips with the same maximum length, it must appear multiple times in the result set (one for each duration time).

```
WITH longest AS (
    SELECT vehId, MAX(ST_Length(trajectory(t.trip))) AS maximum
    FROM trips t
    GROUP BY vehId )
SELECT t.vehid, duration(timeSpan(t.trip)) AS duration, trajectory(trip)
FROM trips t, longest l
WHERE t.vehId = l.vehId AND ST_Length(trajectory(trip)) = l.maximum
ORDER BY t.vehId;
```

d. For every pair of adjacent communes, give the trips that cross each of them. You should only show the trips restricted by these boundaries and the time when the intersection occurred.

```
-- Adjacent communes
WITH adjacentCommunes AS (
    SELECT c1.communeId AS commune1Id, c2.communeId AS commune2Id
    FROM communes c1, communes c2
    WHERE c1.communeId < c2.communeId AND ST_Touches(c1.geom, c2.geom) ),
-- Geometries of adjacent communes
adjacentCommunesGeom AS (
    SELECT ac.*, c1.geom AS geom1, c2.geom AS geom2,
        ST_Union(c1.geom, c2.geom) AS geomUnion
    FROM adjacentCommunes ac
    LEFT JOIN communes c1 ON ac.commune1Id = c1.communeId
    LEFT JOIN communes c2 ON ac.commune2Id = c2.communeId ),
-- Restrict each trip to the pairs of adjacent communes.
restrictedTrips AS (
    SELECT acg.commune1Id, acg.commune2Id, t.tripid,
        atGeometry(t.trip, acg.geomUnion) AS tripRestricted
    FROM adjacentCommunesGeom acg, trips t
    WHERE ST_Intersects(trajectory(t.trip), acg.geom1) AND
        ST_Intersects(trajectory(t.trip), acg.geom2) ),
-- Times of each trip in adjacent communes
SELECT tripid, timeSpan(tripRestricted) AS startTime, tripRestricted
FROM restrictedTrips
```

Consider the temporary table adjacentCommunesGeom. For example, if commune 1 is adjacent to 2, and commune 2 is adjacent to commune 3 (but 1 and 3 are not adjacent), and trip1 passed through communes 1, 2 and 3, then in one row we show trip1 restricted to communes 1 and 2, and in another row it will appear trip 1 restricted to 2 and 3. If the trip passes through 1 and 2, exits, and then re-enters at 1 and/or 2, then both parts of the trip will appear in the same row when passed through 1 and/or 2 as two different sequences.

You can use the following PostGIS functions:

- **ST_Area(geometry)**: Return the area of the geometry if it is a Polygone or Multipolygone.
- **ST_Centroid(geometry)**: Return the geometric center of a geometry
- **ST_Contains(geometry,geometry)**: Returns true if the first geometry contains the second one
- **ST_Distance(geomA, geomB)**: Return the 2D Cartesian distance between two geometries
- **ST_DumpPoints(geometry)**: Return a set of all points that make up a geometry
- **ST_Intersection(geomA, geomB)**: Return a geometry that represents the shared portion of geomA and geomB.
- **ST_Intersects(geomA, geomB)**: Return TRUE if the Geometries share any portion of space and FALSE if they do not.
- **ST_Length(geometry)**: Return the length of the geometry if it is a Line or MultiLine.
- **ST_LineLocatePoint(geomA, geomB)**: Return a float between 0 and 1 representing the location of the closest point on a line geomA to the given Point geomB, as a fraction of 2d line length.
- **ST_LineInterpolatePoint(geomA, fraction)**: Return a point interpolated along a line geomA at a fractional location.
- **ST_Segmentize(geometry)**: Return a modified geometry having no segment longer than the given distance
- **ST_Touches(geometry, geometry)**: Return true if two geometries have at least one point in common, but their interiors do not intersect
- **ST_Union(geometry)**: Aggregating function, return a geometry that represents the point set union of the geometries.
- **ST_Value(geometry, raster)**: Return the value of a given band of a raster at a given geometry point.

You can use the following MobilityDB functions:

- Return the lower or upper bound
  - lower(spans) → base
  - upper(spans) → base
- Return the value or time span ignoring the potential gaps
  - valueSpan(tnumber) → numspan
  - timeSpan(ttype) → tstzspan
- Return the trajectory
  - trajectory(tpoint) → geo
- Return the start, end, or n-th timestamp
  - startTimestamp(ttype) → timestamptz
  - endTimestamp(ttype) → timestamptz
  - timestampN(ttype,integer) → timestamptz
- Restrict to (the complement of) a set of values
  - atValues(ttype,values) → ttype
  - minusValues(ttype,values) → ttype
- Return the duration
  - duration({datespan,tstzspan}) → interval
  - duration({datespanset,tstzspanset},boundspan bool=false) → interval
- Return the smallest distance ever
  - {geo,tpoint} |=| {geo,tpoint} → float
- Return the temporal distance
  - {point,tpoint} <-> {point,tpoint} → tfloat
- Restrict to (the complement of) a geometry and a Z span
  - atGeometry(tgeompoint,geometry,zspan=NULL) → tgeompoint
  - minusGeometry(tgeompoint,geometry,zspan=NULL) → tgeompoint