UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences Appliquées

INFO-H-415 – Advanced databases

First session examination

Name: **Last name:**

A postal service uses a database to keep track of its letters in transit. The postal service has multiple post offices. Each trip taken by the letter is recorded in a table "Trip". The letter can pass through multiple post offices before being delivered to the customer. Thus, a letter can have multiple trip and the last one (i.e. when delivered to the customer) will have an origin post office, but the destination office will be null. Finally, the database also tracks the employees and vehicles of the postal service (not all employees are drivers) as well as the route taken by the vehicle during a drive (a route can pass by multiple post offices and customers).

The database is based on the following relational schema:

- **Customer** (CustomerID, Name, Surname, StreetAdress, City, PostCode, Country)
- **PostOffice** (PostOfficeID, Name, Location)
  - Location is a geometrical POINT
- **Employee** (EmployeeId, StartDate, EndDate, Role, Salary, BaseOfficeID)
  - StartDate is the DATE of hiring
  - EndDate is the DATE of firing (null if still working for the company)
  - BaseOfficeID references PostOffice(PostOfficeID)
- **Vehicle** (VehicleID, PlateNumber, InService, PurchaseDate)
  - PurchaseDate is the DATE of purchase
- **Letter** (LetterID, PostedAt, ExpectedDeliveryDate, SenderID, RecipientID)
  - PostedAt is the TIMESTAMPTZ of the posting
  - ExpectedDeliveryDate is the DATE by which we expect the letter to be delivered.
  - SenderID references Customer(CustomerID)
  - RecipientID references Customer(CustomerID)
- **Driving** (DrivingID, DriverID, StartTS, EndTS, VehicleID, RouteID)
  - DriverId references Employee(EmployeeID)
  - StartTS is the TIMESTAMPTZ marking the start of the drive
  - EndTS is the TIMESTAMPTZ marking the end of the drive
  - VehicleID references Vehicle(VehicleID)
  - RouteID references Route(RouteID)

- **Route** (RouteID, OriginOfficeID, FinalOfficeID, RouteGeo)
  - OriginOfficeID references PostOffice(PostOfficeID)
  - FinalOfficeID references PostOffice(PostOfficeID)
  - RouteGeo is geometrical MULTILINESTRING
- **Trip** (LetterID, DepartTime, ArriveTime, DrivingID, OriginOfficeID, DestinationOfficeID, TempTrip, Trajectory)
  - LetterID references Letter(LetterID)
  - DepartTime is the TIMESTAMPTZ of departure
  - ArriveTime is the TIMESTAMPTZ of arrival (null if in transit)
  - DrivingID references Driving(DrivingID)
  - OriginOfficeID references PostOffice(PostOfficeID)
  - DestinationOfficeID references PostOffice(PostOfficeID) (can be null if going to the customer)
  - TempTrip is a temporal point with type TGEOMPOINT
  - Trajectory is a geometrical LINESTRING

Note: When writing SQL queries, you can use the first letter of the table (e.g. R for Route, L for Letter or V for Vehicle).

Name: **Last name:**

1 Spatial Databases

For the following questions, suppose you are using a PostgreSQL database with the PostGIS extension added. We also suppose working in 4326 (WGS84) spatial reference system.

1. For each letter, we want to have a single entry for the trip table. For each letter, give the aggregated trajectory and the ID of the corresponding letter.
2. For each letter, we want to have the complete distance it has covered.
3. For each Post Office, we want the list of customers less than 5 km away.
4. The vehicle still in service that has covered the longest distance.

Name: **Last name:**

1 Spatial Databases

Name: **Last name:**

2 Temporal Databases

For the following questions, suppose you are using a PostgreSQL database (No extensions apart from PostGIS were added!). If you are using an algorithm seen during the course, you can specify the steps of the algorithm and write the parts that had to be modified in each step (in SQL). But you do not need to rewrite the full algorithm.

1. Give the number of different letters in transit each month last year.
2. Give the history of the number of letters the postal service has handled.
3. Give the intervals where a driver was not assigned a vehicle.
4. List the letters that were late in their deliveries and by how much time they were.

Name: **Last name:**

2 Temporal Databases

3 MobilityDB Databases

Suppose we have, in addition to the tables above, the following table that contains the hourly weather measures for the overall geographical extent of the databases.

- WeatherHourly(Time timestamptz PRIMARY KEY, Temperature float, Rain float, Wind float, Humidity float)

Write the following in MobilityDB.

- Add to the Trip table a column TripWeather of type TFLOAT that associates with each instant of the TempTrip the temperature at that instant.
- Give the letters such that in all the trips from an origin to a destination office (that means, from the first office to the final one), the average temperature was above 25 degrees.
- Give the three (origin) post offices with the largest distances traversed by the letters they send, until the letters arrive at the destination post office.
- For each driver, compute the average speeds of all her deliveries.

Name: **Last name:**

3 MobilityDB Databases

Name: **Last name:**

4 Active Databases

Write triggers ensuring the following constraints. Whenever multiple triggers are needed to enforce a single integrity constraint, write the code in full for only one of them and list the remaining ones. You can choose whether you write the triggers using SQL Server or PostgreSQL.

1. In table Route, ensure that the locations of the origin and final office correspond, respectively, to the first and the last point of RouteGeo.
2. A letter cannot be at the same time in two different trips.
3. A driver can only drive in routes starting at her base office.
4. Add to table Letter a derived attribute ActualDeliveryDate that is automatically updated when the trip that delivers the letter to the customer is introduced in the database.

Name: **Last name:**

4 Active Databases

You can use the following PostGIS functions:

ST_Centroid(geometry) Returns the geometric center of a geometry

ST_Distance(geomA, geomB) Returns the 2D Cartesian distance between two geometries

ST_DumpPoints(geometry) Returns a set of all points that make up a geometry

ST_EndPoint(geometry) Returns the last point of a Linestring or CircularLinestring geometry as a POINT.

ST_StartPoint(geometry) Returns the first point of a Linestring or CircularLinestring geometry as a POINT.

ST_Intersection(geomA, geomB) Returns a geometry that represents the shared portion of geomA and geomB.

ST_Intersects(geomA, geomB) Returns TRUE if the Geometries share any portion of space and FALSE if they do not.

ST_Length(geometry) Returns the length of the geometry if it is a Line or MultiLine.

ST_Area(geometry) Returns the area of the geometry if it is a Polygone or Multipolygone.

ST_Union(geometry) Aggregating function, returns a geometry that represents the point set union of the Geometries.

ST_Segmentize(geometry) Return a modified geometry having no segment longer than the given distance

ST_Value(geometry, raster) Returns the value of a given band of a raster at a given geometry point.

ST_Within(geomA, geomB) Returns TRUE if geometry A is within geometry B

ST_LineLocatePoint(geomA, geomB) Returns 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) Returns a point interpolated along a line geomA at a fractional location.

ST_Collect(geomA, geomB) Collects geometries into a geometry collection. The result is either a Multi* or a GeometryCollection, depending on whether the input geometries have the same or different types

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
- Return the time when the temporal Boolean takes the value true
  - whenTrue(tbool) → tstzspanset
- Spatial relationships
  - eContains, aContains, tContains: Ever, always, or temporal contains
  - eDisjoint, aDisjoint, tDisjoint: Ever, always, or temporal disjoint
  - eDwithin, aDwithin, tDwithin: Ever, always, or temporal at distance within
  - eIntersects, aIntersects, tIntersects: Ever, always, or temporal intersects
  - eTouches, aTouches, tTouches: Ever, always, or temporal touches

  The above ever/always relationships have one of the following signatures

  - eaSpatialRel({geo,tpoint}, {geo,tpoint}) → bool
  - eaDwithin({geo,tpoint}, {geo,tpoint}, float) → bool

  The above temporal relationships have one of the following signatures

  - tSpatialRel({geo,tpoint}, {geo,tpoint}) → tbool
  - tDwithin({geo,tpoint}, {geo,tpoint}, float) → tbool