Solutions for Session 10 - Geographical Databases (1/3)

4 Importing shapefiles

By creating an sql script then executing it with psql:

```
$ shp2pgsql -W "latin1" bel_city.shp > shp_insert_script.sql
$ shp2pgsql -W "latin1" bel_dist.shp >> shp_insert_script.sql
$ shp2pgsql -W "latin1" bel_prov.shp >> shp_insert_script.sql
$ shp2pgsql -W "latin1" bel_regn.shp >> shp_insert_script.sql
$ shp2pgsql -W "latin1" belriver.shp >> shp_insert_script.sql
$ psql <database_name> -f shp_insert_script.sql
```

4.1 Update SRID

```
SELECT UpdateGeometrySRID('bel_regn', 'geom', 4326);
SELECT UpdateGeometrySRID('bel_city', 'geom', 4326);
SELECT UpdateGeometrySRID('bel_dist', 'geom', 4326);
SELECT UpdateGeometrySRID('bel_prov', 'geom', 4326);
SELECT UpdateGeometrySRID('belriver', 'geom', 4326);
```

5 Requests

1. Get the SRID from table bel_city.

```
SELECT ST_SRID(geom) FROM bel_city LIMIT 1;
```

2. Get a textual description for this SRID.

```
SELECT srtext from spatial_ref_sys where srid=4326;
```

3. Get the dimension of geographical objects in that table.

```
-- Inherent dimension:
SELECT ST_Dimension(geom) from bel_city limit 1;
-- Coordinate dimension:
SELECT ST_CoordDim(geom) from bel_city limit 1;
-- or
SELECT coord_dimension from geometry_columns WHERE f_table_name = 'bel_city';
-- You can check the coherence of the above results with:
SELECT ST_AsText(ST_POINTS(geom)) from bel_city limit 1;
-- Each entity is composed of one point -> inherent dimension = 0
-- but the coordinates of these points are in two dimensions
```

4. Get the geometry type of these objects.

```
-- easy way, look at the type of each geometry object of each row in
-- the bel_city table
SELECT ST_GeometryType(geom) FROM bel_city limit 3;
-- more efficient way, use the geometry_columns table
SELECT type FROM geometry_columns WHERE f_table_name = 'bel_city';
```


5. Compute the distance between the cities of IXELLES and BRUGES.

```
SELECT ST_Distance((SELECT geom FROM bel_city where name= 'Ixelles'),
                   (SELECT geom FROM bel_city where name='Brugge'));
```

ST_Distance gives us a result which is in the units of our spatial reference (SRID 4326 which correspond to WGS 84) and is therefore an angle. It is not really "human useful". A first possibility to obtain readable results is to change the spatial reference system of the geometries.

```
SELECT ST_Distance(
     ST_Transform((SELECT geom FROM bel_city where name= 'Ixelles'), 3812),
     ST_Transform((SELECT geom FROM bel_city where name='Brugge'), 3812));
```

SRID 3812 corresponds to Belgian Lambert 2008 which is a conic representation adapted for Belgium. Since it is a conic (plane) representation, its units are meters.

```
SELECT ST_DistanceSphere((SELECT geom FROM bel_city where name= 'Ixelles'),
                         (SELECT geom FROM bel_city where name='Brugge'));
```

ST_DistanceSphere gives an approximation of the distance on a sphere without changing the spatial reference system.

Further information can be found with the following links:

- `http://postgis.net/workshops/postgis-intro/geography.html` (Calculating distances with geographic instead of geometric objects.)
- `http://postgis.net/docs/ST_Distance.html`

6. Compute the bounding rectangle for the BRABANT province.

```
SELECT ST_AsText(ST_Envelope(geom)) FROM bel_prov WHERE name='Brabant';
```

If you want to be able to visualise the result using qgis, insert the result in a new table:

```
CREATE TABLE Temp As
SELECT ST_Envelope(geom) FROM bel_prov WHERE name='Brabant';
```

The same procedure can be used to visualize the output of other queries.

7. Compute the geographical union of the bel_regn and bel_prov tables.

```
SELECT ST_AsText(ST_Union(geom))
FROM (SELECT gecm FROM bel_regn
      UNION SELECT geom FROM bel_prov) AS g;
```

8. Compute the length of each river

```
SELECT name, ST_Length2DSpheroid(geom,
                                 'SPHEROID["GRS_1980",6378137,298.257222101]')
FROM belriver;
```

9. Create a table containing all cities that stand less than 1000m from a river.

```
SELECT DISTINCT ON (c.gid) c.gid, c.id, c.name, c.geom
FROM bel_city c JOIN belriver r ON ST_DistanceSphere(c.geom, r.geom) < 1000;
```

You can also make use of the geography entity proposed by PostGIS:

```
CREATE TABLE belriver_geog AS
SELECT gid, id, name, Geography(geom) AS geog FROM belriver;
CREATE TABLE bel_city_geog AS
SELECT gid, id, name, Geography(geom) AS geog FROM bel_city;
CREATE TABLE river_cities AS
SELECT DISTINCT ON (c.gid) c.gid, c.id, c.name, c.geog FROM bel_city_geog c
    JOIN belriver_geog r ON ST_DWithin(c.geog, r.geog, 1000);
```


10. For each river, compute the length of its path inside each province it traverses.

```
SELECT r.name,
       p.name,
       ST_Length2D_Spheroid(ST_Intersection(r.geom, p.geom),
                            'SPHEROID["GRS_1980", 6378137, 298.257222101]')
FROM belriver r, bel_prov p WHERE ST_Intersects(r.geom, p.geom);
```


5 Spatial queries

Write down and execute the following queries:

1. Get the SRID from table cities. (ST_SRID)
2. Get a textual description for this SRID.
3. Get the dimension of geographical objects in that table. (ST_Dimension, ST_CoordDim)
4. Get the geometry type of these objects. (*Hint: explore the geometry_columns table,* ST_GeometryType)
5. Compute the distance between the cities of IXELLES and BRUGES. (ST_DistanceSphere, ST_Distance, ST_Transform)
6. Compute the bounding rectangle for the BRABANT province. (ST_Envelope)
7. Compute the geographical union of the bel_regn and bel_prov tables. (ST_Union)
8. Compute the length of each river. (ST_Length2DSpheroid)
9. Create a table containing all cities that stand less than 1000m from a river. (ST_DistanceSphere)
10. For each river, compute the length of its path inside each province it traverses. (ST_Intersection, ST_Intersects)

3 Raster importing in PostGIS

```
$ initdb                                    # necessary depending on your installation
$ pg_ctl start
$ createdb tp11                             # Create the database
$ psql tp11 -c "CREATE EXTENTION postgis;"  # make it postgis
$ psql tp11 -f generate.sql                 # populate it with the data

$ cd bel_alt/                               # move to bel_alt directory
$ raster2pgsql BEL_alt.vrt > insert_bel_alt.sql # export raster to sql in a new sql file
$ psql tp11 -f insert_bel_alt.sql           # execute the insertion file

$ cd ../alt_10m_bil/                        # move to alt_10m_bil directory
$ raster2pgsql alt.bil > insert_alt.sql     # export raster to sql in a new sql file
$ psql tp11 -f insert_alt.sql               # execute the insertion file
$ cd ..                                     # return to home directory
```

4 Spatial Queries

To perform the exercices, you will need to update the SRID of the raster tables:

```
SELECT UpdateRasterSRID('alt', 'rast', 4326);
SELECT UpdateRasterSRID('bel_alt', 'rast', 4326);
```

You can check that the change has been applied:

```
SELECT ST_SRID(rast) FROM alt;
SELECT ST_SRID(rast) FROM bel_alt;
```

Search in the documentation for the following functions:

- ST_Clip
- ST_DumpPoints
- ST_Intersects
- ST_MapAlgebra
- ST_Resample
- ST_Segmentize
- ST_SummaryStats
- ST_UNION
- ST_Value

Write down and execute the following queries:

1. Compute the difference between the two altitude datasets.
   Try to perform this exercice first without using ST_Resample. Export the result and visualize it in QGIS:
   ```
   $ gdal_translate -of GTiff PG:"dbname=tp11 schema=public table=sol1" sol1.tiff
   ```
   (This commend should be runned directly in the terminal and not in psql)
2. Compute the maximum altitude in Belgium.
3. Get the altitudes of all cities in Belgium.
4. Compute the maximum and minimum altitudes for each province.

5. Create a new raster table restraining the alt_16 raster to Belgium. (Hint: use ST_Intersection.)
6. Compute the altitude along each river.

1.
– Executing the MapAlgebra function without resampeling will not work as the two raster
– do not have the same granularity

```
create table sol1 as
SELECT ST_MapAlgebra( a.rast, 1, b.rast, 1, '[rast2] - [rast1]') AS rast
FROM
        (select b.RID, ST_Resample(b.rast, a.rast) as rast
        from bel_alt b,  alt a)
as b, alt a;

– To execute in the terminal, then drag and drop the sol1.tiff in qgis
– $ gdal_translate -of GTiff PG:"dbname=tp11 schema=public table=sol1" sol1.tiff
```

2.
```
SELECT (stats).max
FROM (SELECT ST_SummaryStats(rast, 1) AS stats
        FROM bel_alt WHERE rid=1) AS foo;
```

3.
```
SELECT name, ST_Value(rast, geom) FROM bel_alt JOIN bel_city ON ST_Intersects(geom, rast);
```

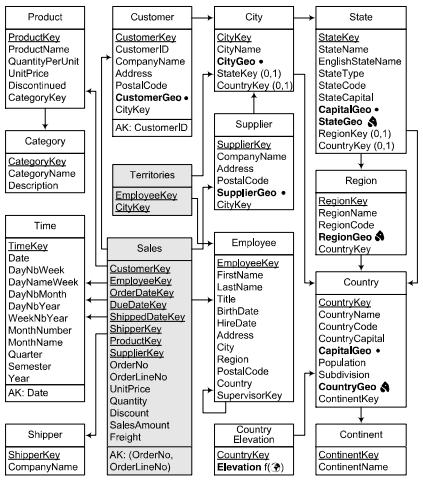
4.
```
SELECT name, (stats).min, (stats).max
FROM (SELECT name, ST_SummaryStats(ST_Clip(rast, 1, geom, TRUE)) AS stats
        FROM alt JOIN bel_prov ON ST_Intersects(geom,rast)) AS foo;
```

5.
```
SELECT name, (stats).min, (stats).max
FROM (SELECT name, ST_SummaryStats(ST_Clip(rast, 1, geom, TRUE)) AS stats
        FROM alt_16_belgium JOIN bel_prov ON ST_Intersects(geom,rast)) AS foo;
```

6.
```
SELECT name, ST_AsText((position).geom), ST_Value(rast, (position).geom)
FROM (SELECT name, rast, ST_DumpPoints(ST_Segmentize(geom, 0.1)) AS position
        FROM belriver JOIN bel_alt on ST_Intersects(geom, rast)) AS dp;
```


Session 12 - Spatial Databases (3/3)

For this session, we consider the following spatial relational diagram of the GeoNorthwind database.



You can load this database using the following commands:

```
$ createdb NW
$ pg_restore -d NW -x -O geonorthwind.backup
```

The restoration process could lead to some unimportant errors. Please check that you have a database loaded by exploring it with psql.

For this exercise session, we provided a "spatial reference guide" which describe the different functions that you may need.

Exercise 1. Give the total sales in 1997 to customers located in cities that are within an area whose extent is a polygon drawn by the user. For the purpose of the exercise, we will consider the following coordinates for the polygon's opposite corners:
(5, 40) and (100, 90).

You can create the rectangle with

```
'SRID=4326;POLYGON((5.0 40.0, 100 40.0, 100.0 90.0, 5.0 90.0, 5.0 40.0))'::geometry
```

For instance:

```
SELECT ST_contains('SRID=4326;POLYGON((5 40, 100 40, 100 90, 5 9, 5 40))'::geometry,
                   'SRID=4326;POLYGON((0 0, 200 0, 200 200, 0 200, 0 0))'::geometry)
```

Will return `true`

Exercise 2. What are total sales to customers located in a state that contains the capital city of the country?

Exercise 3. Give the spatial union of the states in the USA where at least one customer placed an order in 1997.

Exercise 4. What is the distance between the customers' locations and the capital of the state in which they are located?

Exercise 5. For each customer, give the total sales amount to its closest supplier.

Exercise 6. Give the total sales amount for customers that have orders delivered by suppliers such that their locations are less than 200 km from each other.

Exercise 7. What is the distance between the customer and supplier for customers that have orders delivered by suppliers of the same country.

Exercise 8. Give the number of customers for European countries with an area larger than 50,000 km^2.
Note: the database provided during the lab does referer to the table "Continent" (on the schema) as "Area"

Exercise 9. For each supplier, give the number of customers located at more than 100 km from the supplier.

Exercise 10. For each supplier, give the distance between its location and the centroid of the locations of all its customers.

Solutions for Session 12 - Spatial Databases (3/3)

➤ Solution to Exercise 1

```
SELECT C.CompanyName AS Customer,
       Y.CityName AS City,
       SUM(S.SalesAmount) AS TotalSales
  FROM Sales S,
       Customer C,
       City Y,
       Time T
 WHERE S.CustomerKey = C.CustomerKey
   AND C.CityKey = Y.CityKey
   AND S.OrderDateKey = T.TimeKey
   AND T.Year = '1997'
   AND ST_Within(Y.CityGeo, 'SRID=4326;POLYGON((5.0 40.0, 100 40.0,
                                        100.0 90.0, 5.0 90.0, 5.0 40.0))'::geometry)
 GROUP BY C.CompanyName, Y.CityName
```

Beware that you have to specify the SRID of the polygone and to "cast" the Geography-typed result to a Geometry type (with the `::geometry` statement) because the functions `ST_Within` and `ST_Containsn` only accept geometry.
To check your restults, you could add the following column to the query: `ST_AsText(Y.CityGeo) AS City`.
Note that you could also use the `ST_GeographyFromText` function to interpret the polygon string into a geography, but you still would have to cast it into a geometry type. Practically, you could replace

```
'SRID=4326;POLYGON((5.0 40.0, 100 40.0, 100.0 90.0, 5.0 90.0, 5.0 40.0))'::geometry
```

by

```
ST_GeomFromText('SRID=4326;POLYGON((5.0   40.0, 100 40.0,
                                    100.0 90.0, 5.0 90.0, 5.0 40.0))')
```

➤ Solution to Exercise 2

```
SELECT C.CompanyName AS Customer,
       SUM(S.SalesAmount) AS TotalSales
  FROM Sales S,
       Customer C,
       City Y,
       State A,
       Country O
 WHERE S.CustomerKey = C.CustomerKey
   AND C.CityKey = Y.CityKey
   AND Y.StateKey = A.StateKey
   AND A.CountryKey = O.CountryKey
   AND ST_Contains(A.StateGeo,O.CapitalGeo)
 GROUP BY C.CompanyName
```

Note that, like for the previous exercise, you could also use the `ST_Within` function (and swap the parameters).

➤ Solution to Exercise 3

```
SELECT ST_AsText(ST_Union(DISTINCT A.StateGeo)) AS States
  FROM Sales S,
       Customer C,
       City Y,
       State A,
       Country O,
       Time T
 WHERE S.CustomerKey = C.CustomerKey
   AND C.CityKey = Y.CityKey
   AND Y.StateKey = A.StateKey
   AND A.CountryKey = O.CountryKey
   AND O.CountryName = 'United States'
   AND S.OrderDateKey = T.TimeKey
   AND T.Year = '1997'
```

➤ Solution to Exercise 4

```
SELECT C.CompanyName AS Customer,
       ST_Distance(C.CustomerGeo,A.CapitalGeo) AS Distance
  FROM Customer C,
       City Y,
       State A
 WHERE C.CityKey = Y.CityKey
   AND Y.StateKey = A.StateKey
 ORDER BY C.CompanyName
```

➤ Solution to Exercise 5

```
SELECT C.CompanyName AS Customer,
       SUM(S.SalesAmount) AS TotalSales
  FROM Sales S,
       Customer C,
       Supplier U
 WHERE S.CustomerKey = C.CustomerKey
   AND S.SupplierKey = U.SupplierKey
   AND ST_Distance(C.CustomerGeo,U.SupplierGeo)
         <= (SELECT MIN(ST_Distance(C.CustomerGeo,U1.SupplierGeo))
               FROM Sales S1,
                    Supplier U1
              WHERE S1.CustomerKey = C.CustomerKey
                AND S1.SupplierKey = U1.SupplierKey)
 GROUP BY C.CompanyName
```

Note that this query is assuming that we cannot have two closest suppliers for any customer. Should we want to take this case into account, then the query could be extended as follows:

```
SELECT C.CompanyName AS Customer,
       U.CompanyName AS Supplier,
       SUM(S.SalesAmount) AS TotalSales
```



```
  FROM Sales S,
       Customer C,
       Supplier U
 WHERE S.CustomerKey = C.CustomerKey
   AND S.SupplierKey = U.SupplierKey
   AND ST_Distance(C.CustomerGeo,U.SupplierGeo)
         <= (SELECT MIN(ST_Distance(C.CustomerGeo,U1.SupplierGeo))
               FROM Sales S1,
                    Supplier U1
              WHERE S1.CustomerKey = C.CustomerKey
                AND S1.SupplierKey = U1.SupplierKey)
 GROUP BY C.CompanyName, U.CompanyName
```

➤ Solution to Exercise 6

```
SELECT C.CompanyName AS Customer,
       SUM(S.SalesAmount) AS TotalSales
  FROM Sales S,
       Customer C,
       Supplier U
 WHERE S.CustomerKey = C.CustomerKey
   AND S.SupplierKey = U.SupplierKey
   AND ST_Distance(C.CustomerGeo,U.SupplierGeo) < 200
 GROUP BY C.CompanyName
```

➤ Solution to Exercise 7

```
SELECT DISTINCT C.CompanyName AS Customer,
       U.CompanyName AS Supplier,
       ST_Distance(C.CustomerGeo,U.SupplierGeo) AS Distance
  FROM Sales S,
       Customer C,
       City Y,
       State A,
       Supplier U,
       City AS SY,
       State AS SA
 WHERE S.CustomerKey = C.CustomerKey
   AND C.CityKey = Y.CityKey
   AND Y.StateKey = A.StateKey
   AND S.SupplierKey = U.SupplierKey
   AND U.CityKey = SY.CityKey
   AND SY.StateKey = SA.StateKey
   AND SA.CountryKey = A.CountryKey
 ORDER BY C.CompanyName, U.CompanyName
```

➤ Solution to Exercise 8

```
SELECT O.CountryName AS Country,
       COUNT(DISTINCT S.CustomerKey) AS NbCustomers
  FROM Sales S,
       Customer C,
       City Y,
       State A,
       Country O,
       Area R
 WHERE S.CustomerKey = C.CustomerKey
   AND C.CityKey = Y.CityKey
   AND Y.StateKey = A.StateKey
   AND A.CountryKey = O.CountryKey
   AND ST_Area(O.CountryGeo) > 5
   AND O.AreaKey = R.AreaKey
   AND TRIM(R.AreaName) = 'Europe'
 GROUP BY O.CountryName
```

The table "Area" can be referred as "Conitinent" if the exercise is done on paper.

➤ Solution to Exercise 9

```
SELECT U.CompanyName AS Supplier,
       COUNT(DISTINCT C.CustomerKey) AS CustomerCount
  FROM Sales S,
       Supplier U,
       Customer C
 WHERE S.SupplierKey = U.SupplierKey
   AND S.CustomerKey = C.CustomerKey
   AND ST_Distance(U.SupplierGeo,C.CustomerGeo) > 100
 GROUP BY U.CompanyName
```

➤ Solution to Exercise 10

```
SELECT U.CompanyName AS Supplier,
       ST_Distance(U.SupplierGeo, ST_Centroid(ST_Union(DISTINCT C.CustomerGeo))) AS Dist
  FROM Sales S,
       Supplier U,
       Customer C
 WHERE S.SupplierKey = U.SupplierKey
   AND S.CustomerKey = C.CustomerKey
 GROUP BY U.CompanyName,
          U.SupplierGeo
```


MobilityDB Data Model and Type System

An instance of a temporal type represents a function from time to the base type. Temporal types are defined based on their base type and time type.

MobilityDB defines six built-in temporal types on top of the base and **spatial** types bool, int, float, text, **geometry point (uses planar coordinate system), and geography point (uses spherical coordinate system)** data types. Formally, the temporal data types are: tbool, tint, tfloat, ttext, **tgeompoint, and tgeogpoint**.

Instants represent **valid time**, i.e. the time in the real world when the event occurred.

For instance, we can create a table with a temporal data type as follows:

CREATE TABLE ranking(star **tint**);

CREATE TABLE IoT(temperature **tfloat**);

CREATE TABLE gps(position **tgeompoint**);

In addition, temporal data types have three subtypes: instant, sequence and sequence set. Any temporal type column can store an instance of any of these three sybtypes.

1. **Instant Subtype**: represents a value at a time instant.

As an example of a **tfloat instant**, temperature sensors measure the amount of heat energy in a source at different time instants:

17@2018-01-01 08:00:00

This can be inserted in the IoT table as follows (note that it is not necessary to cast to tfloat):

INSERT INTO IoT VALUES (tfloat '17@2018-01-01 08:00:00');

INSERT INTO IoT VALUES ('18.4@2018-01-01 08:00:05');

MobilityDB transforms the literal (the "@" symbol that separates the value from the time instant) to the corresponding temporal data type.

We can also insert these data using the appropriate type constructor which has two parameters: base type and the timestamptz data type (timestamp with time zone). The constructor for creating, and instant is defined as follows:

ttype_inst(base, timestamptz): ttype_inst

Note that timestamptz is an abbreviation for timestamp with time zone. Standard SQL also requires timestamp to be equivalent to timestamp with zone. You can use any of them.

Thus, we can insert both previous tuples by using the following alternative expressions:

INSERT INTO IoT VALUES (**tfloat_inst**(17::double precision, '2018-01-01 08:00:00'::timestamp with time zone));

INSERT INTO IoT VALUES (**tfloat_inst**(18.4::double precision, '2018-01-01 08:00:05'::timestamptz));

When we have *instant* values, the 'Discrete Interpolation method' is applied, i.e. we cannot infer data types beyond the data provided.

If we run

SELECT **tempSubType**(temperature), interp(temperature), temperature

FROM IoT;

We obtain

	tempsubtype text	interp text	temperature tfloat
1	Instant	None	17@2018-01-01 08:00:00-03
2	Instant	None	18.4@2018-01-01 08:00:05-03

Since the attribute temperature corresponds to an **instant** subtype (necessarily with discrete interpolation), if we ask for its value at a moment different to this instant, we obtain null value

SELECT valueattimestamp(temperature, '2018-01-01 08:00:00'), *

from IoT;

valueattimestamp double precision	temperature tfloat
17	17@2018-01-01 08:00:00-03
[null]	18.4@2018-01-01 08:00:05-03

GPS sensors provide geolocation positions at different instants. For storing these data in the GPS table, **tgeompoint** or **tgeogpoint** would be more appropriate.

Notice that if locations are provided as latitude-longitude pairs, the use of a constructor instead of literals is more convenient when inserting data into the table.

For instance, consider a gps table defined as follows:

CREATE TABLE gps(position tgeompoint, lat float, lon float, t timestamp);

If the three attributes lat, lon and t contain data, in order to populate the "position" column we can run the following query

UPDATE gps SET position=tgeompoint_inst (ST_MakePoint(lat, lon), t);

It is also possible to generate the expected literal string from scratch, but this method is error-prone:

UPDATE gps SET position = tgeompoint ('POINT(' || lat || ' ' || lon || ')@' || t) ;

2. Sequence Subtype: represent the evolution of the value during a sequence of time instants where **the values between these instants are interpolated using discrete (i.e., none)/stepwise/linear manner**.

A temporal sequence is composed of N instants in ascendant order, i.e. t1 < t2 <… < tN.

The expression of a sequence is a comma-separated list of instants enclosed in curly brackets, square brackets or parentheses as explained next.

2.1 Set (Curly brackets). Represents the evolution of a value at a **set** of time instants where **the values between these instants are unknown, i.e. the discrete (or none) interpolation method is used.**

We next show an example of a **tfloat sequence value** with discrete interpolation method.

{17@2018-01-01 08:00:00, 17.4@2018-01-01 08:05:00, 18@2018-01-01 08:10:00};

We can insert a tuple containing a sequence value with discrete interpolation method as follows:

INSERT INTO IoT VALUES ('{17@2018-01-01 08:00:00, 17.4@2018-01-01 08:05:00, 18@2018-01-01 08:10:00}');

It is also possible to insert these data using the appropriate constructor.

If we have same value at different instants we can use:

ttype (base, timestampt_set): ttype

For instance,

INSERT INTO IoT VALUES (tfloat_seq(17,

tstzset ('{2018-01-01 08:00:00,

2018-01-01 08:05:00,

2018-01-01 08:10:00}')));

Where "tstzset" refers to a set of timestamps.

After both insertions, if we run

SELECT tempSubType(temperature), interp (temperature),

temperature

FROM IoT

We obtain

tempsubtype text	interpolation text	temperature tfloat
Sequence	Discrete	{17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03}
Sequence	Discrete	{17@2018-01-01 08:00:00-03, 17@2018-01-01 08:05:00-03, 17@2018-01-01 08:10:00-03}

A more general form is to use an array of instants. The elements of this array are considered isolated but **need to be provided in temporal ascending order**. As previously discussed, the use of a constructor could be more convenient. Here, we show the alternative expression for inserting the first tuple above:

ttype_seq(ttype []): ttype

INSERT INTO iot values(

tfloat_**seq**(ARRAY [

tfloat_inst(17, '2018-01-01 08:00:00'::timestamp),

tfloat_inst(17.4, '2018-01-01 08:05:00'::timestamp),

tfloat_inst(18, '2018-01-01 08:10:00'::timestamp)

], '**Discrete**', true, true));

If we run

SELECT tempSubType(temperature), interp(temperature),

temperature

FROM IoT;

We obtain three tuples

tempsubtype text	interpolation text	temperature tfloat
Sequence	Discrete	{17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03}
Sequence	Discrete	{17@2018-01-01 08:00:00-03, 17@2018-01-01 08:05:00-03, 17@2018-01-01 08:10:00-03}
Sequence	Discrete	{17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03}

Since we have an instant set, if we ask for a value at a timestamp different for those present in the (discrete) sequences, we obtain the *null* value.

SELECT valueattimestamp(temperature, '2018-01-01 08:05:00'), *

FROM IoT;

valueattimestamp double precision	temperature tfloat
17.4	{17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03}
17	{17@2018-01-01 08:00:00-03, 17@2018-01-01 08:05:00-03, 17@2018-01-01 08:10:00-03}
17.4	{17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03}

SELECT valueattimestamp(temperature, '2018-01-01 08:04:00'), *

FROM IoT;

valueattimestamp double precision	temperature tfloat
[null]	{17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03}
[null]	{17@2018-01-01 08:00:00-03, 17@2018-01-01 08:05:00-03, 17@2018-01-01 08:10:00-03}
[null]	{17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03}

However, if we write a sequence of instant values with step interpolation:

INSERT INTO iot values(
tfloat_seq(ARRAY [
tfloat_inst(18, '2018-01-01 08:00:00'::timestamp),
tfloat_inst(15.4, '2018-01-01 08:05:00'::timestamp),
tfloat_inst(18.8, '2018-01-01 08:10:00'::timestamp)
] , 'Step', true, true));

SELECT tempSubType(temperature), interp (temperature),

temperature

FROM IoT

tempsubtype text	interp text	temperature tfloat
Sequence	Discrete	{17@2018-01-01 08:00:00-03, 17.4@2018-01-01 0…
Sequence	Step	Interp=Step;[18@2018-01-01 08:00:00-03, 15.4@2…

The query

SELECT valueattimestamp(temperature,'2018-01-01 08:04:00'), *
FROM IoT;

returns:

	valueattimestamp double precision	temperature tfloat
1	[null]	{17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03}
2	18	Interp=Step;[18@2018-01-01 08:00:00-03, 15.4@2018-01-01 08:05:00-03, 18.8@2018-01-01 08:10:00-…

2.2 Span (parentheses or squared brackets). Represents a range of values. The parenthesis is used when the boundary of the interval is exclusive. On the contrary, the squared bracket is used when the boundary is inclusive. Both can be combined in the same expression. In any case, we can use **stepwise o linear interpolation** depending of the ttype base.

In the case of a temporal sequence with stepwise or linear interpolation, the N instants t1 < t2 <… < tN produce a partition.

For instance, [v1@t1, v2@t2, v3@t3, v4@t4) defines a temporal partition composed of three intervals: [t1, t2), [t2, t3) and [t3, t4).

This is an example of tint sequence:

(10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]

For these sequences, MobilityDB provides **stepwise** and/or **linear interpolation**, depending on the subjacent (base/spatial) data type:

2.2.1. Stepwise interpolation. Discrete data (**tbool, tint, ttext**) can only use *stepwise interpolation*. On the contrary, continuous data (**tfloat, tegompoint, tgeogpoint**) can choose between stepwise o linear (default) interpolation. In this subsection, we show only how to use stepwise interpolation with any data type.

Consider the following **tint sequence** (10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]. A sequence with discrete base type only accepts stepwise interpolation. Thus, we can infer that ∀ t ∈ (2018-01-01 08:00:00, 2018-01-01 08:05:00) the value is 10, ∀ t ∈ [2018-01-01 08:05:00, 01-01 08:10:00] the value is 20 and the value is 15 for t=01-01 08:10:00

We can insert a tuple with a **tint sequence** as a String:

```
INSERT INTO ranking VALUES ('(10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]');
```

We can also explicitly indicate the interpolation method:

```
INSERT INTO ranking VALUES ('interp=step;(10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]');
```

An error is raised when we indicate an invalid interpolation method for a tint data type, which accepts only stepwise interpolation, as in the following statement:

```
INSERT INTO ranking VALUES ('interp=linear; (10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]');
```

It is also possible to insert data using the appropriate constructor. You can provide an array of instants (temporal limits are inclusive unless you provided explicitly by using the last two parameters).

tbase_seq(ttype_inst[], lower boolean, upper boolean): tbase

Here we show the alternative expression for the same tuple insertion with values [10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]:

```
INSERT INTO ranking VALUES (tint_seq( ARRAY [
    tint_inst ( 10,  '2018-01-01 08:00:00'::timestamp),
    tint_inst ( 20,  '2018-01-01 08:05:00'::timestamp),
    tint_inst ( 15,  '2018-01-01 08:10:00'::timestamp) ]
                    ) );
```

In this case, the first value has a closed left bound.

And if you want to provide an equivalent to the first insertion (left bound not included):

```
INSERT INTO ranking VALUES (tint_seq(ARRAY [
    tint_inst (10,  '2018-01-01 08:00:00'::timestamp),
    tint_inst (20,  '2018-01-01 08:05:00'::timestamp),
    tint_inst (15,  '2018-01-01 08:10:00'::timestamp) ],
    [Step, false, true ) );
```

After these three insertions, we can show both the sequence and the interpolation method:

```
SELECT  tempSubType(star),  interp(star), *
FROM ranking;
```

We obtain:

tempsubtype text	interpolation text	star tint
Sequence	Step	(10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]
Sequence	Step	[10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]
Sequence	Step	(10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]

Since this sequence has stepwise interpolation, we can ask for values different from the instants used.

```
SELECT valueattimestamp(star, '2018-01-01 08:00:00'), *
FROM ranking
```

valueattimestamp integer	star tint
[null]	(10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]
10	[10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]
[null]	(10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]

The null value obtained when the left instant is not included in the sequence.

If, instead, we ask for a value at a time in the interval of the sequence, we obtain either the value or the interpolated value.

```
SELECT valueattimestamp(star, '2018-01-01 08:10:00'), *
FROM ranking
```

valueattimestamp integer	star tint
15	(10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]
15	[10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]
15	(10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]

In this case, the value 15 corresponds to the instant explicitly included in the sequence

```
SELECT valueattimestamp(star, '2018-01-01 08:09:00'), *
FROM ranking
```

valueattimestamp integer	star tint
20	(10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]
20	[10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]
20	(10@2018-01-01 08:00:00-03, 20@2018-01-01 08:05:00-03, 15@2018-01-01 08:10:00-03]

The value 20 is valid ∀ t ∈ [2018-01-01 08:05:00, 2018-01-01 08:10:00)

In the case that we want to use a stepwise interpolation with a tbase data type that admits both, stepwise and linear interpolation, we should indicate that explicitly, since the latter is the default.

Let's try with a tfloat.

We first delete the information in table IoT:

```
DELETE FROM IoT;
```

Now, we insert some **tfloats** in different formats.

```
INSERT INTO IoT VALUES ('interp=step;(17@2018-01-01 08:00:00, 17.4@2018-01-01 08:05:00, 18@2018-01-01 08:10:00]' );
```

It is also possible to insert these data using the appropriate constructor. You can provide an array of instants (temporal limits are inclusive unless you provided explicitly by using the last two parameters). Note that this statement differs from the one above due to the last parameter which indicates the interpolation method (***true*** **is the default, i.e. linear interpolation**).

tbase_seq(ttype_inst[], text interpolationmethod, lower boolean, upper boolean): tbase

Here we show the alternative expression for the same tuple insertion with values '[17@2018-01-01 08:00:00, 17.4@2018-01-01 08:05:00, 18@2018-01-01 08:10:00]':

Thus, we should run

INSERT INTO IoT VALUES (**tfloat_seq**(ARRAY [

tfloat_inst (17, '2018-01-01 08:00:00'::timestamp),

tfloat_inst (17.4, '2018-01-01 08:05:00'::timestamp),

tfloat_inst (18, '2018-01-01 08:10:00'::timestamp)],

'step', **false, true**));

After these two insertions, we can show both the sequence and the interpolation method:

SELECT tempSubType(temperature), interp(temperature), temperature

FROM IoT;

We obtain

tempsubtype text	interpolation text	temperature tfloat
Sequence	Step	Interp=Step[17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03]
Sequence	Step	Interp=Step[17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03]

Since this sequence has stepwise interpolation, we can ask for values different from the instants used.

SELECT valueattimestamp(temperature, '2018-01-01 08:00:00'), *

FROM IoT

valueattimestamp double precision	temperature tfloat
[null]	Interp=Step[17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03]
[null]	Interp=Step[17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03]

The null value obtained since the left instant is not included in the sequence.

If, instead, we ask for a value at a time in the interval of the sequence, we obtain either the value or the interpolated value.

SELECT valueattimestamp(temperature, '2018-01-01 08:10:00'), *

FROM IoT

valueattimestamp double precision	temperature tfloat
18	Interp=Step[17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03]
18	Interp=Step[17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03]

In this case, the value 18 corresponds to the instant explicitly included in the sequence

SELECT valueattimestamp(temperature, '2018-01-01 08:09:00'), *

FROM IoT

valueattimestamp double precision	temperature tfloat
17.4	Interp=Step[17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03]
17.4	Interp=Step[17@2018-01-01 08:00:00-03, 17.4@2018-01-01 08:05:00-03, 18@2018-01-01 08:10:00-03]

The value 17.4 is valid $\forall$ t $\in$ [2018-01-01 08:05:00, 2018-01-01 08:10:00)

2.2.2. Linear interpolation. In linear interpolation the temporal value evolves continuously in a linear way. Only continuous data (**tfloat, tgeompoint or tgeogpoint)** use **linear (default) interpolation**.

The following sentence inserts a tuple with a tgeompoint sequence as a String:

INSERT INTO gps VALUES ('(Point(1 1)@2021-01-01 08:00:00, Point(2 3)@2021-01-01 08:05:00, Point(3 3)@2021-01-01 08:10:00]');

linear interpolation is used in each of the 3-temporal partitions (2021-01-01 08:00:00, 2021-01-01 08:05:00) , [2021-01-01 08:05:00, 2021-01-01 08:10:00) and [2021-01-01 08:10:00, 2021-01-01 08:10:00].

It is also possible to insert these data using the appropriate constructor.

We can use again the array of instants:

tbase_seq(ttype_inst[], text interpolationmethod, lower boolean, upper boolean): tbase

For the String

(Point(1 1)@2021-01-01 08:00:00, Point(2 3)@2021-01-01 08:05:00, Point(3 3)@2021-01-01 08:10:00]

We have:

INSERT INTO gps VALUES (**tgeompoint_seq**(ARRAY [

tgeompoint_inst (ST_MakePoint(1, 1),

'2021-01-01 08:00:00'::timestamp),

tgeompoint_inst(ST_MakePoint(2, 3),

'2021-01-01 08:05:00'::timestamp),

tgeompoint_inst(ST_MakePoint(3, 3),

'2021-01-01 08:10:00'::timestamp)],

'Linear', false, true));

We can omit the last two parameters and use the default option.

We show now how to generate an array of tgeompoints from scratch. A very useful alternative consists of building an array of tgeompoints that comes from some column of another table. We next explain this option.

Consider a table with three columns: x, y and date. The table (called **rawdata**) contains:

RawData		
X (int)	Y (int)	Date (timestamp)
1	1	2021-01-01 08:00:00
2	3	2021-01-01 08:05:00
3	3	2021-01-01 08:10:00
…	…	…

PostgreSQL offers a function that returns an array from an expression:

array_agg(expression): array

Thus, instead of ARRAY[] we can use array_agg(), which is used to generate a **tgeompoint** in temporal ascending order. To dynamically insert dynamically tuples in the **gps table** we write:

INSERT INTO gps

SELECT

tgeompoint_contseq (array_agg(

tgeompoint (ST_MakePoint(?? ??), ??) ORDER BY ??, false)

FROM rawdata

where we need to replace every "??" symbol with the correct column of the source table. Since we need to generate a sequence, these values need to be inserted sorted by the **date column**. Summarizing, the expression should be:

INSERT INTO gps

SELECT

tgeompoint_seq(array_agg(

tgeompoint_inst (ST_MakePoint(x, y),date) ORDER BY date

), 'Linear', false, true)

FROM rawdata

We showed different ways of populating a **tsequence** column: building a string, from a static array or from an array generated dynamically from other table. Now, we can ask:

SELECT tempSubType(position), interp (position), *

FROM gps;

Temp subtype Text	Interpol. text	Position tgeompoint
Sequence	Linear	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 010100000000000000000000400000000000000840@2021-01-01 08:05:00-03, 010100000000000000000008400000000000000840@2021-01-01 08:10:00-03]
Sequence	Linear	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 010100000000000000000000400000000000000840@2021-01-01 08:05:00-03, 010100000000000000000008400000000000000840@2021-01-01 08:10:00-03]
Sequence	Linear	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 010100000000000000000000400000000000000840@2021-01-01 08:05:00-03, 010100000000000000000008400000000000000840@2021-01-01 08:10:00-03]

Since we have linear interpolation, we can ask for values different from the instants used.

SELECT valueattimestamp(position, '2021-01-01 08:09:00'), *

FROM gps

Valueattimestamp geometry	Position tgeompoint
0101000000666666666666064 00000000000000840	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 010100000000000000000000400000000000000840@2021-01-01 08:05:00-03, 010100000000000000000000840000000000000840@2021-01-01 08:10:00-03]
0101000000666666666666064 00000000000000840	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 010100000000000000000000400000000000000840@2021-01-01 08:05:00-03, 010100000000000000000000840000000000000840@2021-01-01 08:10:00-03]
0101000000666666666666064 00000000000000840	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 010100000000000000000000400000000000000840@2021-01-01 08:05:00-03, 010100000000000000000000840000000000000840@2021-01-01 08:10:00-03]

We reformulate the query for a human readable result

SELECT

ST_AsText(valueattimestamp(position, '2021-01-01 08:09:00')), *

FROM gps

St_astext text	Position tgeompoint
POINT(2.8 3)	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 010100000000000000000000400000000000000840@2021-01-01 08:05:00-03, 010100000000000000000000840000000000000840@2021-01-01 08:10:00-03]
POINT(2.8 3)	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 010100000000000000000000400000000000000840@2021-01-01 08:05:00-03, 010100000000000000000000840000000000000840@2021-01-01 08:10:00-03]
POINT(2.8 3)	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 010100000000000000000000400000000000000840@2021-01-01 08:05:00-03, 010100000000000000000000840000000000000840@2021-01-01 08:10:00-03]

This point is obtained by linear interpolation within the corresponding subinterval ['2018-01-01 08:05:00', '2018-01-01 08:10:00')

DELETE FROM IoT;

INSERT INTO IoT VALUES (

tfloat_seqset(ARRAY [

tfloat_seq(ARRAY [

tfloat_inst (10.3 , '2018-01-01 08:00:00'::timestamp with time zone),

tfloat_inst(20.4, '2018-01-01 08:05:00'::timestamp with time zone)

], 'Linear', true, true),

tfloat_seq(ARRAY [

tfloat_inst(15.1, '2018-01-01 08:10:00'::timestamp with time zone),

tfloat_inst(3.5, '2018-01-01 08:12:00'::timestamp with time zone)

], 'Linear', false, true)]));

We can show both the sequence and the interpolation method:

SELECT tempSubType(temperature), interp(temperature), *

FROM IoT;

Data Output Explain Messages Notifications

	tempsubtype text	interpolation text	temperature tfloat
1	SequenceSet	Linear	{[10.3@2018-01-01 08:00:00-03, 20.4@2018-01-01 08:05:00-03], (...

Tempsubtype Text	Interpolation text	Position tgeompoint
SequenceSet	Linear	{[10.3@2018-01-01 08:00:00-03, 20.4@2018-01-01 08:05:00-03], (15.1@2018-01-01 08:10:00-03, 3.5@2018-01-01 08:12:00-03]}

3. **Spanset (a set of spans, i.e., curly brackets enclosing parentheses or squared brackets).** Represents a **set of ranges of values**. More precisely, represents the evolution of the value at a set of spans where **the values between these spans are unknown, i.e., there exist temporal gaps where there is no value information**. The expression of a sequence is a comma-separated list of spans enclosed in curly brackets. This is an example of tfloat sequence set:

{

[10.3@2018-01-01 08:00:00, 20.4@2018-01-01 08:05:00],

(15.1@2018-01-01 08:10:00, 3.5@2018-01-01 08:12:00]

}

We can insert a tuple with a **tfloat** sequence set as a String, for instance, with linear interpolation method:

INSERT INTO IoT VALUES (

'{[10.3@2018-01-01 08:00:00, 20.4@2018-01-01 08:05:00],

(15.1@2018-01-01 08:10:00, 3.5@2018-01-01 08:12:00]}');

It is also possible to insert these data using the appropriate constructor.

tbase_seqset (tbase[]): tbase

where the array can be built on top of elements of type tbase_discseq or tbase_contseq.

Here we show the alternative expression for the same tuple insertion with values:

{

[10.3@2018-01-01 08:00:00, 20.4@2018-01-01 08:05:00],

(15.1@2018-01-01 08:10:00, 3.5@2018-01-01 08:12:00]

}

Asking for values of instant outside any of the sequences belonging to the set returns a *null* value.

SELECT valueattimestamp(temperature, '2018-01-01 08:07:00'), *

FROM Iot

valueattimestamp double precision	temperature tfloat
[null]	{[10.3@2018-01-01 08:00:00-03, 20.4@2018-01-01 08:05:00-03], (15.1@2018-01-01 08:10:00-03, 3.5@2018-01-01 08:12:00-03]}

Important Whenever possible, MobilityDB merges data.

Activity 1: Vessels

AIS, standing for Automatic Identification System, is the location-tracking system for sea vessels. AIS equipment on board of ships produces data that are collected by a shore-based AIS system operated by the Danish Maritime Authority. AIS data are published in CSV file format. Check the site https://dma.dk/safety-at-sea/navigational-information/ais-data for more details.

You must be aware that this information is produced by the ships' own instruments and contains errors, such as erroneous ship's speeds and/or positions. We should analyze data quality and, if necessary, define and apply some rules to fix or discard data with problems (ETL). Trajectories will be built from raw AIS Data.

In this assignment we propose to analyze trajectory patterns to learn some basic MobilityDB functionality. Whenever possible we also visualize data in some GIS tool such as QGIS.

Exercise 1. Creating and populating the database

1.1 Create a mobilityDB-enabled database

```
CREATE DATABASE  vessels;
```

Choose it and run the following sentence:

```
CREATE EXTENSION IF NOT EXISTS MobilityDB CASCADE;
```

1.1.1 Download the file https://web.ais.dk/aisdata/aisdk-2018-04.zip which contains temporal data occurred during April-2018. More precisely, we will use only information generated during 1 day, 01/04/2018.

You must create the table that stores the information in the csv.

For this, execute the following sentence:

```
CREATE TABLE AISInput(
T timestamp,
TypeOfMobile varchar(50),
MMSI integer,
Latitude float,
Longitude float,
navigationalStatus varchar(60),
ROT float,
SOG float,
COG float,
Heading integer,
IMO varchar(50),
Callsign varchar(50),
Name varchar(100),
ShipType varchar(50),
CargoType varchar(100),
Width float, Length float,
TypeOfPositionFixingDevice varchar(50),
Draught float,
Destination varchar(50),
ETA varchar(50),
DataSourceType varchar(50),
SizeA float,
SizeB float,
SizeC float,
SizeD float
);
```

1.1.2. Populate the table.

Note that the COPY sentence is server oriented. Thus, the path needs to be solved on the server side. Assure that the timestamp format is the correct one according to the data in the csv file. If the file is in the server we can use the following sentence to import data in the target table. Run it. After some minutes, we will obtain more than one million tuples (**10619231** rows). (change the path accordingly)

```
SET datestyle TO postgres, dmy;

COPY AISInput(T, TypeOfMobile, MMSI, Latitude, Longitude, NavigationalStatus, ROT, SOG, COG, Heading, IMO, CallSign, Name, ShipType, CargoType, Width, Length, TypeOfPositionFixingDevice, Draught, Destination, ETA, DataSourceType)
FROM '/software/data/ais/aisdk_20180401.csv' DELIMITER ',' CSV HEADER;
```

When the size of the file is huge, it is convenient to follow these steps: put the file in the server and use the server-oriented COPY command.

If the file were small we could use the client-side oriented \COPY command. In this case, the file path should be found on the client side. The problem with this is that, to load the data into the table, the file should be transferred from the client to the server during the execution of this command.

Exercise 2. Cleaning the database

The csv file has at least two major problems:

- Presence of null values: The 'Unknown' text is used for representing null values. We need to convert this text into the null database value.
- Points as latitude-longitude values: The csv does not contain points. Latitude and longitude values are stored in two table columns. For spatial queries we must transform the lat/lon pairs into point geometries. In order to fix this problem, we add an extra column in the table that the csv does not contain: **geom**. Two PostGIS functions allow constructing a Point data from lat/lon: ST_MakePoint and ST_Point.

At this point, it is important to remark that PostGIS offers **geography** and **geometry** data types. The former is based on a spherical model and uses a geodetic coordinates (lat/lon) system (unit of measurement: degrees). The latter uses the Cartesian coordinate system (unit of measurement: meters/feets). The execution performance of functions such as distance, area, intersects, among others, are less complicated when working on the plane than on the spherical mode because the underlying mathematics used is simpler. Although PostGIS offers conversion between these both types, fewer functions are defined on geography than on geometry types.

Both data types above are associated with a Spatial Reference System (SRS) via a Spatial Reference System Identifier (SRID) which defines how the object is referenced to locations on the Earth's surface. On the one hand, **Geodic SRS** uses angular coordinates (lat/lon) which map directly to the surface of the earth. On the other hand, **Projected SRS** uses a mathematical transformation to flatten the spherical surface onto the plane. In general, the last one needs to be limited to a **bounded area** to avoid distortions.

For our extra column we may use geography or geometry type. *SRID 4326 is used for worldwide* geodesic shapes and *SRID 25832 is used for geometries that belong to Denmark zone.*

Fig 1- (https://epsg.io/25832) bounded zone for SRID 25832

We define the attribute

geom geometry(Point, 25832)

2.1. Considering the above discussion, run the following query.

It takes five minutes, approximately, on standard hardware.

```
ALTER TABLE AISInput
ADD COLUMN geom geometry(Point, 25832)
```

We can check with:

```
SELECT Find_SRID('public', 'aisinput', 'geom');
```

2.2. Now, we fix ***null*** values and generate points from the lat/lon values. When we populate the geom column using the statement:

```
Geom = ST_Transform(
ST_SetSRID(ST_MakePoint(Longitude, Latitude), 4326), 25832),
```

we would obtain an error because the ST_Transform function checks that latitude and longitude ranges from -90 to 90 and from -180 to 180, respectively. Run the following query and check that there are values outside the expected range.

```
SELECT min(latitude) as minlatitude, max(latitude) as maxlatitude,
 min(longitude) as minlongitude, max(longitude) as maxlongitude
FROM aisInput
```

	minlatitude double precision	maxlatitude double precision	minlongitude double precision	maxlongitude double precision
1	-108.482315	91	-217.806473	181

Thus, we need to exclude those values.

But, in fact, there exist other points that are outside the minimum bounding box corresponding to Denmark. According to https://epsg.io/25832 the limits should be

Latitude: 40.18 and 84.73

Longitude: -16.1 and 32.88

For this, we execute the following statement:

```
UPDATE AISInput
SET
   NavigationalStatus = CASE NavigationalStatus
                        WHEN 'Unknown value' THEN NULL END,
  IMO = CASE IMO WHEN 'Unknown' THEN NULL END,
  ShipType = CASE ShipType WHEN 'Undefined' THEN NULL END,
  TypeOfPositionFixingDevice = CASE TypeOfPositionFixingDevice
                               WHEN 'Undefined' THEN NULL END,
  Geom = ST_Transform(ST_SetSRID(ST_MakePoint(Longitude,
         Latitude), 4326), 25832)
WHERE latitude between 40.18 and 84.73 AND
      longitude between -16.1 AND 32.88
```

The execution takes approximately 10 minutes on standard hardware.

When the process finishes the points outside the boundaries will contain ***null*** values (97340 in total). Check with

```
SELECT
   NavigationalStatus,
   IMO,
   ShipType,
   TypeOfPositionFixingDevice,
   geom
FROM AISInput
WHERE latitude between 40.18 and 84.73 AND
     longitude between -16.1  and 32.88
```

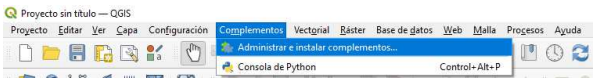
Exercise 3 Visualization and data exploration

3.1. Download QGIS V 3.22 from

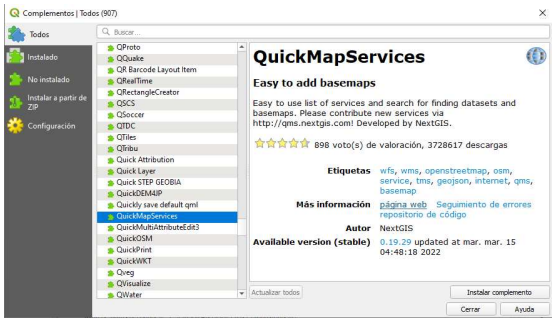
https://www.qgis.org/en/site/forusers/download.html

Open the QGIS Desktop application.

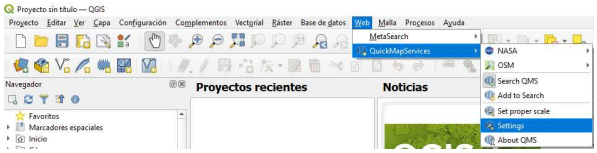
To add a map layer we first need to add the QuickMapServices QGIS complement.



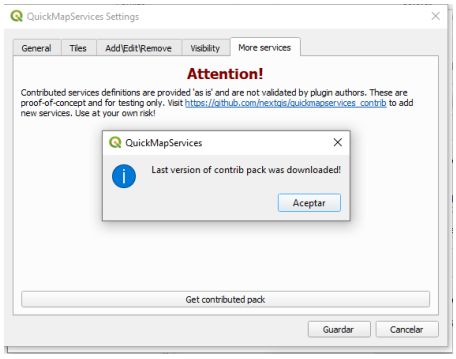
Choose the option Install Complement:



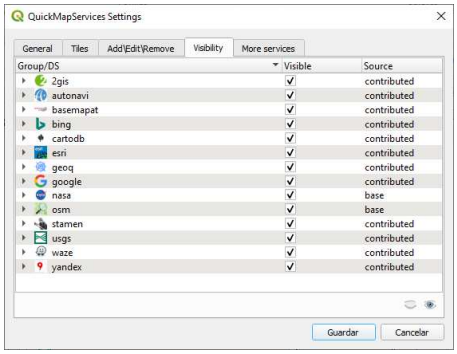
To visualize services such as Google ESRI, among others, choose the "Web-QuickMapServices-Settings" sub-menu.



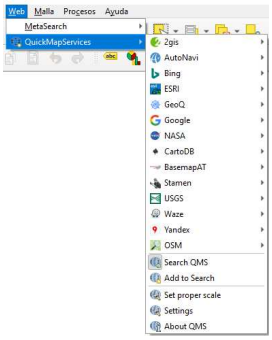
In the pop-up dialog choose the "Get Contributed Pack" option.



In the "Visibility" panel you can see the new enabled services. You can choose which ones to use. Close the dialog box.

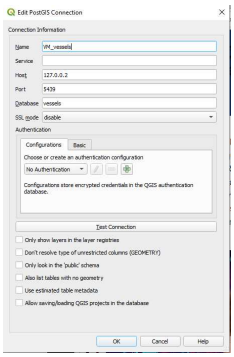


Now, the sub menu "Web-QuickMapServices" displays enabled services.



3.2. Add the Waze World map layer or the ESRI Topo map layer

3.3. Generate a remote PostGIS connection

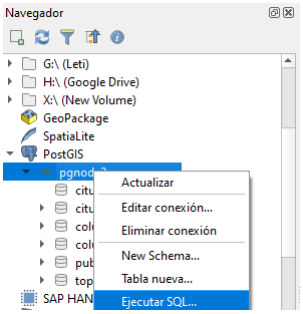


3.4. Add the geographic layers

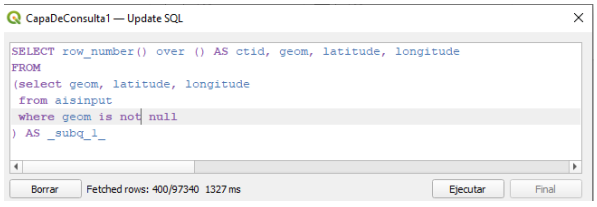
To create a layer from an SQL query, we need to make sure that there exist at least two special columns: one that identifies each feature to be drawn and one to be displayed (geometry).

If a table contains a Primary Key/Unique and a geometry column the process is straightforward.

However, in our case study, we do not have a primary key. Thus, we need to generate an ID ad hoc. Select the SQL option:



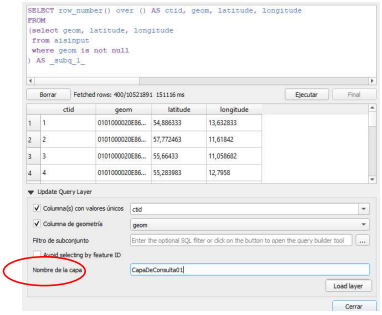
First, we run the query.

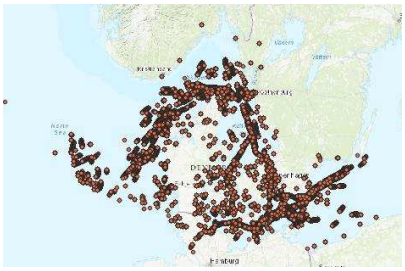


We need at least two columns: ctid (the ID) and the geometry (geom).

```
SELECT row_number() over () AS ctid, geom, latitude, longitude
FROM
(select geom, latitude, longitude
 from aisinput
 where geom is not null   -- exclude points outside Denmark zone
) AS _subq_1_
```

Once the execution is finished, we need to load the result in a layer. Check the "Column with unique values" and "Geometry Column" boxes, and choose *ctid* and *geom* values. Then click the "Load Layer" option. After several minutes we will obtain the result set over the map. This can also take a while.





3.5. Due to the lack of a data quality analysis, there are other problems that have not been detected and should be fixed before generating the trajectories.

The MMSI is the vessel identifier. Thus, it is not possible that the same vessel can report more than one position at the same instant. To find out the vessels with more than one location at the same time instant, we run the following query:

```
SELECT mmsi, t, count(geom)
FROM aisinput
WHERE geom is not null
GROUP BY mmsi, t
HAVING count(geom) > 1
```

We found that 161117 tuples have problems.

3.6. Calculate the percentage of vessels with this problem

```
select count(distinct mmsi)
from aisinput
where geom is not null
-- There are 2995 vessels.
SELECT  count(distinct mmsi)
FROM (
   SELECT mmsi, t, count(geom)
   FROM aisinput
   WHERE geom is not null
   GROUP BY mmsi, t
   HAVING count(geom) > 1
   ) as a
-- Returns  1873 vessels
-- 62,53% of vessels have problems.
```

3.7. The analysis scenario could be divided in two cases:

- If a vessel (MMSI) at the same time reports the same location severañ times, the solution is straightforward: keep only one of them.

Check that:

```
SELECT mmsi, t, count(geom)
FROM aisinput
WHERE geom is not null
GROUP BY mmsi, t
HAVING count(distinct geom) = 1  AND count(geom) > 1
```

There are 121859 tuples that have this problem, which correspond to 1749 vessels.

- If a vessel (MMSI) at the same time reports more than one location, not necessarily the same one, the solution is more involved.

We have 161117 tuples with problems and only 121859 with a simple solution. Therefore, 39258 tuples have a bigger problem.

We now analyze some possible solutions. In each option, we generate a new table, named AISInputFilteredX, which contains the fixed tuples

- Option A: Assuming that the values are close to each other, we can consider a representative lat/lon pair as the average of latitudes and the average of longitudes, respectively. The query for this, reads:

```
CREATE TABLE AISInputFiltered1 AS
select t, max(typeofmobile) as typeofmobile, mmsi,
avg(latitude) as latitude, avg(longitude) as longitude,
max(navigationalstatus) as navigationalstatus,
max(rot) as rot, max(sog) as sog, max(cog) as cog,
max(heading) as heading, max(imo) as imo, max(callsign) as callsign,
max(name) as name, max(shiptype) as shiptype,
max(width) as width, max(length) as length,
max(typeofpositionfixingdevice) as typeofpositionfixingdevice,
max(draught) as draught, max(destination) as destination,
max(eta) as eta, max(datasourcetype) as datasourcetype,
ST_Transform(ST_SetSRID(ST_MakePoint(avg(Longitude), avg(Latitude)
), 4326), 25832) as geom
FROM aisinput
WHERE geom is not null
GROUP  BY mmsi, t
```

After several minutes, the AISInputFilter1 table contains 10357785 representative tuples.

- Option B: Ignore those points. We assume that the vessels' instruments had problems during the transmission. The decision is to ignore those tuples where for the same MMSI there is more than one location reported (either the same or not), considering that instruments were not reliable at those instants. We run the following query:

```
CREATE TABLE AISInputFiltered2 AS
SELECT *
FROM aisinput
WHERE geom IS NOT NULL AND
(mmsi, t) IN
(
        SELECT mmsi, t
        FROM aisinput
        WHERE geom IS NOT NULL
        GROUP BY mmsi, t
        HAVING count(geom) = 1
)
```

As a result, the AISInputFilter2 table contains 10196668 tuples.

That is, from the total of 10357785 (MMSI, t) pairs, we have excluded the 161117 pairs that have some problem.

- Option C: Choose one representative element. For instance, for those (MMSI, t) pairs with more than one location, choose the first one. The Postgres SQL: "SELECT DISTINCT ON (expression) LISTAATTR from TABLE" returns the first tuple in each group. If we want to control which tuple we want to keep, an ORDER BY clause should be used. Usually, we sort the table with respect to the time dimension, and keep the first occurrence. But in our case, that is not useful because the temporal attribute "t" is part of the grouping attributes. Thus, we cannot write "SELECT DISTINCT ON(MMSI,T) * FROM AISInput ORDER BY t"

In our case, there is no information that can help us to decide which tuple is preferable to keep. Thus, we do not use ORDER BY clause. The query for this solution is as follows:

```
CREATE TABLE AISInputFiltered3 AS
SELECT DISTINCT ON(MMSI,T) *
FROM AISInput
WHERE geom IS NOT NULL
```

After several minutes, AISInputFilter3 table contains 10357785 representative tuples.

Exercise 4 Trajectory generation

For a specific MMSI, the list of 2D-points, sorted by their timestamp values, defines a trip. We assume that the vessel located at position pi at instant ti moved linearly to position pi+1 at instant ti+1.

Any of the previously AISInputFilteredX could be used, and we choose the last one, namely AISInputFiltered3.

To represent ships and their spatiotemporal trajectory (trip) we select the following attributes of interest from **AISInputFiltered3**:

- MMSI: integer invariant
- geom: geom changes over time. We use the mobilityDB temporal data type **tgeompoint**. From tgeompoints we can build trajectories.
- sog: float that changes over time. We use the mobilityDB **tfloat**.
- cog: float that changes over time. We use the mobilityDB **tfloat**

As we discussed in "Mobilitydb data model", the table Ships should be defined as follows:

```
CREATE TABLE Ships(
MMSI integer,
Trip tgeompoint,
SOG tfloat,
COG tfloat
)
```

The three attributes Trip, SOG and COG represent the evolution of a value during a sequence of time instants where the values between these instants are obtained using **linear interpolation**. Thus, we use the MobilityDB **sequence subtype** when populating these attributes.

4.1. Run the following query which builds temporal sequences for each MMSI value:

```
CREATE TABLE Ships(MMSI, Trip, SOG, COG) AS
SELECT MMSI,
    tgeompoint_seq(array_agg(
        tgeompoint_inst(geom , t) ORDER BY t )),
    tfloat_seq(array_agg(
        tfloat _inst(SOG , t) ORDER BY t
                ) FILTER (WHERE SOG IS NOT NULL ) ),
    tfloat_seq(array_agg(
        tfloat _inst (COG, t) ORDER BY t
                ) FILTER (WHERE COG IS NOT NULL ) )
FROM AISInputFiltered3
GROUP BY MMSI;
```

We obtain 2995 tuples.

Finally, to display the trajectories in QGIS, we need to generate a geom column (QGIS does not interpret temporal data types). Let us name this column **traj** and populate it with the geometry of each trip.

4.2. Generate a column to store the trajectory of the ships, that is, the spatial projection of the spatiotemporal trip. Run the following query:

```
ALTER TABLE Ships ADD COLUMN Traj geometry;
UPDATE Ships SET Traj= trajectory(Trip);
```

4.3. We can compute, for example, the length or speed of trips.

Next, we list the vessels whose trips have the maximal length among all trips.

```
SELECT mmsi, length(trip)
FROM ships
WHERE length(trip) = (SELECT max(length (trip)) FROM ships)
```

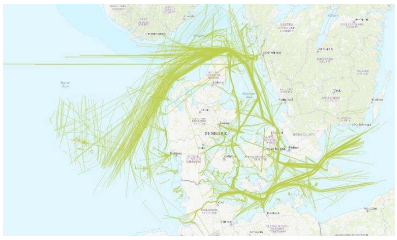
4.4. We now classify vessels by their trip length. Show a histogram where for different ranges of trip lengths (measured in Km), and display number of vessels in each bucket. We want to obtain the following output:

bucketno integer	rangekm floatspan	freq bigint	bar text
1	[0, 0]	303	#####
2	(0, 50)	1691	##############################
3	[50, 100)	265	#####
4	[100, 200)	274	#####
5	[200, 500)	357	######
6	[500, 1500)	87	##
7	[1500, 10000)	15	

We run the following query:

```
WITH
buckets (bucketNo, RangeKM) AS (
        SELECT 1, floatspan '[0, 0]'
 UNION SELECT 2, floatspan '(0, 50)'
 UNION SELECT 3, floatspan '[50, 100)'
 UNION SELECT 4, floatspan '[100, 200)'
 UNION SELECT 5, floatspan '[200, 500)'
 UNION SELECT 6, floatspan '[500, 1500)'
 UNION SELECT 7, floatspan '[1500, 10000)' ),
 histogram AS
       (SELECT bucketNo, RangeKM, count(MMSI) as freq
        FROM buckets LEFT OUTER JOIN
                          Ships ON (length(Trip)/1000) <@ RangeKM
-- The operator "<@" tests if the first argument is contained in the second one.
           GROUP BY bucketNo, RangeKM
           ORDER BY bucketNo, RangeKM )
SELECT bucketNo, RangeKM, freq, repeat('#', ( freq::float / max(freq)
          OVER () * 30 )::int ) AS bar
FROM histogram;
```

The trips to be considered are visualized as follows:



4.6. We can calculate the *speed* of the ships from the *trip* column and compare the results against the stored SOG value. Calculated speed is measured in m/s whereas SOG is expressed in knots (1 knot is equivalent to 1.852 km/h). We convert both to the same unit: km/h.

Since time interval durations differ from each other, we use the speed time-weighted average for comparing them. The mobilityDB function twavg computes the time-weighted average.

Run the following query

```
SELECT ABS(twavg(SOG) * 1.852 - twavg(speed(Trip))* 3.6 ) SpeedDifference
FROM Ships
ORDER BY SpeedDifference DESC;
```

You should obtain:

Null

Null

…

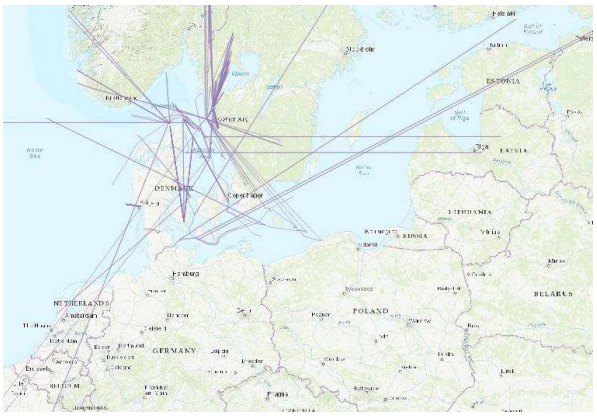
107.86110006787943

83.01385241476066 (tuple 35)

…..

4.5. There are trips that are too long, outside the range above. To compute them, we write:

```
SELECT mmsi, traj
FROM ships
WHERE length(trip) > 1500000
```



Also trips with length 0 should be excluded.

Run the following SQL. 321 tuples where deleted (either with trips of length 0 or too long trips.

```
DELETE FROM Ships
WHERE length(Trip) = 0 OR length(Trip) >= 1500000;
```

You will note that the first 34 tuples show a difference or null or more than 100 km/h. We show these trajectories:

They do not seem to be real trajectories. Thus, delete them and check that they were deleted:

```
Answer
```

4.7. Now we analyse trips between the ports of Rodby and Puttgarden.

Check that port of Rodby UTM coordinates are 652129 6059729

https://geohack.toolforge.org/geohack.php?pagename=R%C3%B8dbyhavn&params=54_39_43_N_11_21_31_E_type:city_region:DK-85

We can build a rectangle very close to the entrance of the harbour with:

ST_MakeEnvelope(651135, 6058230, 651422, 6058548).

Analogously, the UTM coordinates of the port of **Puttgarden** are **643543 6041415**.

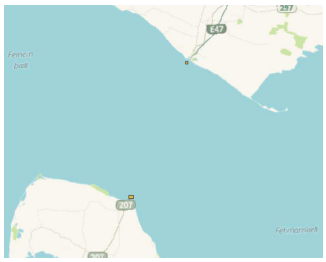
https://geohack.toolforge.org/geohack.php?pagename=Puttgarden&params=54_30_N_11_13_E_region:DE_type:city

We can build a rectangle very close to the entrance of the port:

ST_MakeEnvelope(644339, 6042108, 644896, 6042487)

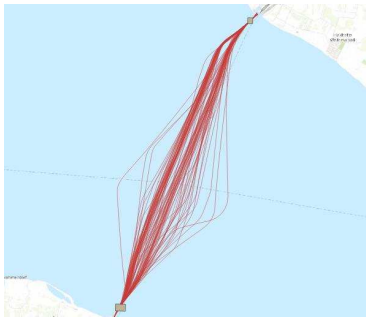
The PostGIS statement next, corresponds to the definition of both rectangles:

```
SELECT  -- Rodby harbour
1 as id, ST_MakeEnvelope(651135, 6058230, 651422, 6058548, 25832) as g
UNION ALL
SELECT  -- Puttgarden port
2 as id, ST_MakeEnvelope(644339, 6042108, 644896, 6042487, 25832) as g
```



We want to analyse the trajectories that intersect both places above.

```
WITH Ports(Rodby, Puttgarden) AS
(SELECT ST_MakeEnvelope(651135, 6058230, 651422, 6058548, 25832), ST_MakeEnvelope(644339, 6042108, 644896, 6042487, 25832) )
SELECT S.*, Rodby, Puttgarden
FROM Ports P, Ships S
WHERE eintersects(S.Trip, P.Rodby) AND eintersects(S.Trip, P.Puttgarden)
```



Notice that the MobilityDB eintersects function checks whether a temporal point has ever intersected a geometry. We obtain **four ships** that continuously go from one port to the other. The figure above shows the trajectories of these four ships.

Spatio-temporal queries can be improved if an appropriate spatio-temporal index has been created. Build a GiST index over trajectories and rerun the query:

```
CREATE INDEX Ships_Trip_Idx ON Ships USING GiST(Trip);
```

4.8. The previous result set contains four vessels and their trips. Calculate how many times each of these ships repeated a one-way trip.

We can ask, for each trip, how many times the vessel touches both ports. For this, MobilityDB has two functions: **atGeometry** and **numSequences**. The former restricts the temporal point to the parts where it is inside the given geometry and returns **sequence set values**. The latter returns the number of sequences. For example, we can see that MMSI = 219000431 has a sequence set containing 32 sequences:

{[0101000020E8640000C33ED38D36E123410D0300008D1C5741@2018-04-01 04:16:42.423054-03,…],

…,

[…,
0101000020E86400003E866CAAC3E02341FA667A63791C5741@2018-04-01 20:01:49-03,
0101000020E8640000D2EA9DA408E123418C3C2415811C5741@2018-04-01 20:01:59-03,
0101000020E8640000F9B6FFFF3BE123415176C7A0861C5741@2018-04-01 20:02:08.000808-03]

}

Below, we display the last three points within the last sequence:

In red we show the zone around the port of Rodby. In green, we can see three points of the same sequence. The ship then moves to another zone that has no intersection with the red zone. Later, it enters the zone around the other port, although we are no displaying this.

Thus, we have as **many sequence**s as times the ship intersects one of those zones. In this case 16 sequences correspond to the port of Rodby and 16 sequences to the port of Puttgarden.

We can check how many sequences correspond to Rodby or Puttgarden running the following query

```
WITH Ports(g) AS (
SELECT  ST_MakeEnvelope(651135, 6058230, 651422, 6058548, 25832)
UNION
SELECT ST_MakeEnvelope(644339, 6042108, 644896, 6042487, 25832))
SELECT MMSI, array_length(sequences(atGeometry(S.Trip,g)),1), sequences(atGeometry(S.Trip, g))
FROM Ports P, Ships S
WHERE eintersects(S.Trip, g)
ORDER BY MMSI
```

We show only a portion of the result set:

7	219000429	24	{"[0101000020E8640000A578631303DF23417DF4FF7F3D1C5741@2018-04-01 01:04:03.32978
8	219000429	24	{"[0101000020E864000004707B85522AC2341E40E00C0DD0C5741@2018-04-01 00:03:11.69105
9	219000431	16	{"[0101000020E8640000090735743337AC2341650900C0DD0C5741@2018-04-01 05:00:00.05574
10	219000431	16	{"[0101000020E8640000C33ED38D36E123410D0300008D1C5741@2018-04-01 04:16:42.42305
11	219004263	4	{"[0101000020E8640000DA8BDD383FE02341A4DFFF7F3D1C5741@2018-04-01 07:08:59.4935
12	219016144	1	{"[0101000020E864000097FACC31F5DF23413EFEFF7F3D1C5741@2018-04-01 20:38:45.83438

The previous query does not check that the same trajectory intersects both ports. For example, the ship with MMSI 219000431 has intersected both zones, but the MMSI 219004263 had not.

The correct solution must check both intersections and can be expressed as follows (note the use of cross product instead of union):

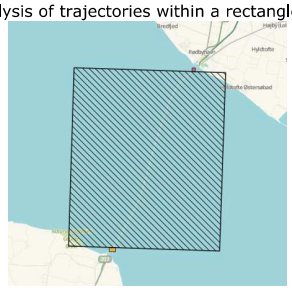
```
WITH Ports(Rodby, Puttgarden) AS (
SELECT ST_MakeEnvelope(651135, 6058230, 651422, 6058548, 25832), ST_MakeEnvelope(644339, 6042108, 644896, 6042487, 25832) )
SELECT MMSI,
(numSequences(atGeometry(S.Trip, P.Rodby)) +
numSequences(atGeometry(S.Trip, P.Puttgarden)))/2.0 AS NumTrips
FROM Ports P, Ships S
WHERE eintersects(S.Trip, P.Rodby) AND eintersects(S.Trip, P.Puttgarden)
```

We obtain

mmsi integer	numtrips numeric
211188000	24.0000000000000000
211190000	25.0000000000000000
219000429	24.0000000000000000
219000431	16.0000000000000000

4.9. The zone between both ports is frequently visited by lots of vessels. This zone could be dangerous if two vessels become at less than 300m from each other at some point in time.

Taking into account the rectangles at the entrance of the ports, we can restrict the analysis of trajectories within a rectangle between them



This rectangle can be built by the expression ST_MakeEnvelope(640730, 6058230, 654100, 6042487, 25832).

Restrict the analysis of trajectories exclusively to this zone.

```
SELECT MMSI, atGeometry(S.Trip, belt) AS Trip,
        trajectory(atGeometry(S.Trip, belt)) AS Traj
FROM Ships S, ST_MakeEnvelope(640730, 6058230, 654100, 6042487, 25832) as belt
WHERE eintersects(S.Trip, belt)
```

Or, alternatively

```
WITH B(Belt) AS (SELECT ST_MakeEnvelope(640730, 6058230, 654100, 6042487, 25832) )
SELECT MMSI, atGeometry(S.Trip, belt) AS Trip,
        trajectory(atGeometry(S.trip, belt)) AS Traj
FROM Ships S, b as belt
WHERE eintersects(S.Trip, belt)
```

We obtain

4.10. We now compute the trajectories that have been at less than 300 meters from each other.

In the visualization, **we want to highlight the minimum distance between ships at some point in time** (probably, ships had been close to each other many times during the trajectory). To show this situation, we compute and draw the shortest line between ships.

```
WITH
B(Belt) AS (SELECT ST_MakeEnvelope(640730, 6058230, 654100, 6042487, 25832) ),
BeltShips AS
(
SELECT MMSI, atGeometry(S.Trip, B.Belt) AS Trip,
trajectory(atGeometry(S.Trip, B.Belt)) AS Traj
FROM Ships S, B WHERE eintersects(S.Trip, B.Belt)
)
SELECT S1.MMSI || '..' || S2.MMSI as both,  S1.MMSI, S2.MMSI, S1.Traj,
S2.Traj, shortestLine(S1.trip, S2.trip) Approach
FROM BeltShips S1, BeltShips S2
WHERE S1.MMSI > S2.MMSI AND edwithin(S1.trip, S2.trip, 300)
```

The method "edwithin" is other "ever relationship", i.e., the generalization of the st_dwithin for temporal points. It returns "true" if the ships have ever been at 300m from each other at some point in time.

Exercise 5

Summarize the most important **MobilityDB functions** we have used. Explain briefly their objective and enumerate all the item exercises where we have used them.

Function	Goal	Used in items…
speed(tpoint): tfloat_seqset		
length(tpoint): float		
eintersects({geo,tpoint},{geo,tpoint}): Boolean		
edwithin({geo,tpoint},{geo,tpoint},float): Boolean		
atGeometry(tgeompoint,geometry): tgeompoint		
shortestLine({geo,tpoint},{geo,tpoint}): geo		
twAvg(tnumber): float		
sequences(ttype_seqset): ttype_seq[] or in a more general form: sequences({ttype_seq,ttype_seqset}): ttype_seq[]		
numSequences(ttype_seqset): integer or in a more general way numSequences({ttype_seq,ttype_seqset}): integer		

Activity 3: Flight Data

The data used for this assignment is provided by The OpenSky Network. We will use data from a 24hr-period corresponding to June 1, 2020 (dataset link). The raw data are provided in separate CSV documents for each hour of the day.

As in the previous assignment, we analyse the data quality and, if necessary, define and apply some rules to fix or discard data with problems (ETL). Again, we build trajectories from the raw data.

Exercise 1. Creating and populating the database

1.1 Create a mobilityDB-enabled database

```
CREATE DATABASE OpenSky;
```

Choose it and run the following sentence:

```
CREATE EXTENSION IF NOT EXISTS MobilityDB CASCADE;
```

1.1.1 Download the data from the link above https://opensky-network.org/datasets/states/2020-06-01/

If you are using the provided VM, you will find the files in /software/data/opensky/

The raw data will be stored in a table called *flights*, with the following structure:

```
CREATE TABLE flights(
et bigint,
icao24 varchar(20),
lat float,
lon float,
velocity float,
heading float,
vertrate float,
callsign varchar(10),
onground boolean,
alert boolean,
spi boolean,
squawk integer,
baroaltitude numeric(7,2),
geoaltitude numeric(7,2),
lastposupdate numeric(13,3),
lastcontact numeric(13,3)
);
```

1.1.2. Populate the *flights* table.

Load the data into the database using the following command. Replace the <path_to_file> with the actual path of the CSV file. Do this for all files (change the path, if necessary)

```
COPY flights(et, icao24, lat, lon, velocity, heading,
vertrate, callsign, onground, alert, spi, squawk,
baroaltitude, geoaltitude, lastposupdate, lastcontact)
FROM '<path_to_file>' DELIMITER ',' CSV HEADER;
```

You can run the following script which iterates over the 23 files and executes dynamic SQL statements:

```
DO
$$DECLARE
prefixpath text= '/software/data/opensky/states_2020-06-01-
';
path text;

BEGIN
    FOR rec in 0..23 LOOP
       path:= prefixpath || trim(to_char(rec, '09')) ||
       '.csv'; -- fill with 0s

      EXECUTE format('COPY flights(et, icao24, lat, lon,
              velocity, heading, vertrate, callsign,
              onground, alert, spi, squawk, baroaltitude,
              geoaltitude, lastposupdate, lastcontact)
      FROM %L WITH DELIMITER '','' CSV HEADER', path);
     COMMIT;
     Raise Notice 'inserting %', path;
    END LOOP;
END
$$;
```

All the times in this dataset are in Unix timestamp (an integer) with timezone being UTC. Thus, we need to convert them to PostgreSQL timestamp type. This is done as follows:

```
ALTER TABLE flights
ADD COLUMN et_ts timestamp,
ADD COLUMN lastposupdate_ts timestamp,
ADD COLUMN lastcontact_ts timestamp;

UPDATE flights
SET et_ts = to_timestamp(et), lastposupdate_ts =
            to_timestamp(lastposupdate),
            lastcontact_ts = to_timestamp(lastcontact);
```

Check the size of the database with:

```
SELECT pg_size_pretty( pg_total_relation_size('flights') );
```

Exercise 2. Cleaning the database

2.1 Delete the NULL values for latitude.

```
-- icao24_with_null_lat is used to indicate the list of
   rows to be deleted

WITH icao24_with_null_lat AS (
SELECT icao24, COUNT(lat)
FROM flights
GROUP BY icao24
HAVING COUNT(lat) = 0
)

DELETE
FROM flights
WHERE icao24 IN
-- this SELECT statement is needed for the IN statement to
   compare against a list
            (SELECT icao24 FROM icao24_with_null_lat);
```

2.2. Explore the database and propose and execute new cleaning tasks (in what follows we assume that only the cleaning tasks in 2.1. have been done).

Exercise 3. Raw data visualization using QGIS

We now visualize the raw data using different tools.

For example, we visualize the points of a single flight using QGIS as follows:

Load the Waze(world) map service and run

```
SELECT row_number() over() as ctid, st_setsrid(geom, 4326)
as geom
FROM(
  SELECT et_ts, icao24, ST_MAKEPOINT(lat, lon) AS geom
-- TABLESAMPLE SYSTEM (n) returns only n% of the data from
-- the table.
FROM flights TABLESAMPLE SYSTEM (5)
WHERE icao24 IN ('738286') AND
      et_ts between '2020-06-01 2:30:00' and '2020-06-01
      4:30:00') as Subq1
```

The results looks like this:

Exercise 4. MobilityDB Data Visualization

We now create the MobilityDB database for flight trajectories. We first create a geometry point. We did this in the SELECT clause of the QGIS query in Exercise 3.1. This treats each latitude and longitude as a point in space. 4326 is the SRID.

```
ALTER TABLE flights
ADD COLUMN geom geometry(Point, 4326);
UPDATE flights SET
geom = ST_SetSRID( ST_MakePoint( lon, lat ), 4326);
```

4.1. Airframe trajectories

Each "icao24" field in the dataset represents a single airplane. We create a composite index on icao24 (unique to each plane) and et_ts (timestamps of observations) to help improving the performance of the trajectory generation.

```
CREATE INDEX icao24_time_index ON flights (icao24, et_ts);
```

We first create trajectories for a single airframe (the plane itself), later on we create another trajectory table for flights (that is, a single aircraft is used for different flights in the same day, and this will be identified by icao24, CallSign

```
CREATE TABLE airframe_traj(icao24, trip, velocity, heading,
vertrate, callsign, squawk, geoaltitude) AS (
SELECT icao24, tgeompoint_seq(array_agg
(tgeompoint_inst(geom,et_ts) ORDER BY et_ts)
FILTER (WHERE geom IS NOT NULL)),
tfloat_seq(array_agg(tfloat_inst(velocity, et_ts)
ORDER BY et_ts)
FILTER (WHERE velocity IS NOT NULL)),
tfloat_seq(array_agg(tfloat_inst(heading, et_ts) ORDER BY
et_ts)
FILTER (WHERE heading IS NOT NULL)),
tfloat_seq(array_agg(tfloat_inst(vertrate, et_ts) ORDER BY
et_ts) FILTER (WHERE vertrate IS NOT NULL)),
ttext_seq(array_agg(ttext_inst(callsign, et_ts) ORDER BY
et_ts) FILTER (WHERE callsign IS NOT NULL)),
tint_seq(array_agg(tint_inst(squawk, et_ts) ORDER BY et_ts)
FILTER (WHERE squawk IS NOT NULL)),
tfloat_seq(array_agg(tfloat_inst(geoaltitude, et_ts) ORDER
BY et_ts) FILTER (WHERE geoaltitude IS NOT NULL))
FROM flights
GROUP BY icao24;
```

This is similar to what we did with AIS data (Assignment 2), that is:

- `tgeompoint_inst`: Combines each geometry point(lat, long) with the timestamp where that point existed
- `array_agg`: aggregates all the instants together into a single array for each item in the group by. In this case, it will create an array for each icao24
- `tgeompoint_seq`: Constructs the array as a sequence which can be manipulated with mobilityDB functionality. In general, *tbase_seq* constructs a temporal base type from an array of sequences of a temporal base type *tbase*. For example, we build a temporal integer type *tint* invoking the corresponding constructor *tint_inst*. With this, we build the sequence.

4.2. Flight trajectories

Now we have, in a single row, an airframe's (where an airframe is a single physical airplane) entire day's trip information. We will now partition this information per flight (**an airframe flying under a specific callsign**). The following query segments the airframe trajectories (in temporal columns) based on the time period of the callsign. Below we show and explain the query.

```
CREATE TABLE flight_traj(icao24, callsign, flight_period,
trip, velocity, heading, vertrate, squawk, geoaltitude)
AS
-- callsign sequence unpacked into rows to split all other
temporal sequences.
WITH airframe_traj_with_unpacked_callsign AS
  (SELECT icao24, trip, velocity, heading, vertrate,
   squawk, geoaltitude,
   startValue(unnest(segments(callsign))) AS
   start_value_callsign,
   unnest(segments(callsign))::tstzspan AS
   callsign_segment_period
   FROM airframe_traj)
SELECT icao24 AS icao24,
  start_value_callsign AS callsign,
  callsign_segment_period AS flight_period,
  atTime(trip, callsign_segment_period) AS trip,
  atTime(velocity, callsign_segment_period) AS velocity,
  atTime(heading, callsign_segment_period) AS heading,
  atTime(vertrate, callsign_segment_period) AS vertrate,
  atTime(squawk, callsign_segment_period) AS squawk,
  atTime(geoaltitude, callsign_segment_period) AS
        geoaltitude
FROM airframe_traj_with_unpacked_callsign;
```

Note: We could have tried to create the table "flight_traj" by simply including "callsign" in the GROUP BY statement in the query used to create the previous airframe_traj table (GROUP BY icao24, callsign).

The problem with this solution is that it would put the trajectory data of two distinct flights where the airplane and flight number are the same in a single row, which is not correct.

MobilityDB functions help us avoid the use of several hardcoded conditions that depend on user knowledge of the data. This approach is very generic and can be applied anytime we want to split a trajectory by the inflection points in time of some other trajectory.

4.2.1. Compute the average velocity per flight

The query which computes this is:

```
SELECT callsign,twavg(velocity) AS average_velocity
FROM flight_traj
WHERE twavg(velocity)IS NOT NULL
     --  drops rows without velocity data
     AND twavg(velocity) < 1500 -- removes erroneous data
ORDER BY twavg(velocity) desc;
```

`Twavg` computes the time-weighted average of a temporal value.

4.2.2. Flights taking-off in some (user-defined) time interval

First, a `flight_traj_time_slice` CTE clips all the temporal columns to a user-specified time range.

```
WITH
flight_traj_time_slice (icao24, callsign,
time_slice_trip, time_slice_geoaltitude,
time_slice_vertrate) AS
(SELECT icao24, callsign,
atTime(trip, '[2020-06-01 03:00:00, 2020-06-01
04:00:00)'::tstzspan),
atTime(geoaltitude, '[2020-06-01 03:00:00, 2020-06-01
04:00:00)'::tstzspan),
atTime(vertrate,'[2020-06-01 03:00:00, 2020-06-01
04:00:00)'::tstzspan)
FROM flight_traj TABLESAMPLE SYSTEM (20)
WHERE atTime(trip, '[2020-06-01 03:00:00, 2020-06-01
      04:00:00)'::tstzspan) IS NOT NULL),
flight_traj_time_slice_ascent(icao24, callsign,
ascending_trip, ascending_geoaltitude,
ascending_vertrate) AS
(SELECT icao24, callsign,
atTime(time_slice_trip, sequenceN(
atValues(time_slice_vertrate, '[1,20]'::floatspan),
1)::tstzspan),
atTime(time_slice_geoaltitude, sequenceN(atValues(
time_slice_vertrate, '[1,20]'::floatspan)
,1)::tstzspan),
atTime(time_slice_vertrate, sequenceN(atValues
(time_slice_vertrate, '[1,20]'::floatspan)
,1)::tstzspan)
FROM flight_traj_time_slice
WHERE atTime(time_slice_trip, sequenceN(
      atValues(time_slice_vertrate,
      '[1,20]'::floatspan), 1)::tstzspan) IS NOT NULL)
```

In the CTE above:

1. **atTime:** Clips the temporal data to create ranges where the vertrate was between '[1, 20]'. This vertrate means an aircraft was ascending.

2. **sequenceN:** Selects the first sequence (because N = 1) from the generated sequences.

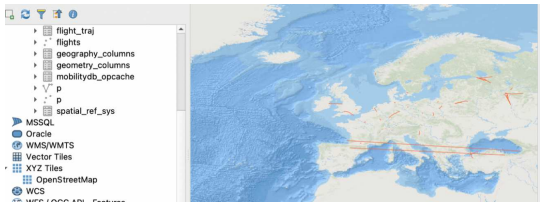
This first sequence is the take off and eliminates mid-flight ascents.

3. **tstzspan:** Returns the part of the temporal values corresponding to the specified period.

Finally, after the CTEs above:

```
SELECT icao24, callsign, trajectory(ascending_trip) AS
geom, ST_GeometryType(trajectory(ascending_trip))
FROM flight_traj_time_slice_ascent
```

4.2.3. Show in QGIS the result of the previous query



Exercise 5. Write the following queries and display their result in QGIS

5.1. ***Trajectories of the flights of the aircraft '000001'.***

```
SELECT ROW_NUMBER()  OVER() as cid, icao24,callsign, traj
FROM (
SELECT icao24, callsign, trajectory(trip) AS traj
FROM flight_traj
WHERE icao24='000001' and callsign > 'A'   AND
        ST_GeometryType(trajectory(trip))='ST_LineString'
) as subq1
```

5.2. ***Flights taking off in Australia in the interval 2020-06-01 03:00:00, 2020-06-01 04:00:00.***

```
SELECT row_number() over() AS cid, geom
FROM (
WITH
flight_traj_time_slice (icao24, callsign, time_slice_trip, time_slice_geoaltitude,
time_slice_vertrate) AS
(
  SELECT icao24,callsign,
  atTime(trip, '[2020-06-01 03:00:00, 2020-06-01  04:00:00)'::tstzspan),
  atTime(geoaltitude, '[2020-06-01 03:00:00, 2020-06-01  04:00:00)'::tstzspan),
  atTime(vertrate, '[2020-06-01 03:00:00, 2020-06-01  04:00:00)'::tstzspan)
  FROM flight_traj TABLESAMPLE SYSTEM (10)
  WHERE atTime(trip, '[2020-06-01 03:00:00, 2020-06-01   04:00:00)'::tstzspan)
        IS NOT NULL),
-- ASCENDING PLANES
flight_traj_time_slice_ascent(icao24, callsign, ascending_trip,
ascending_geoaltitude, ascending_vertrate) AS
  (SELECT icao24, callsign,
   atTime(time_slice_trip, sequenceN(
   atValues(time_slice_vertrate, '[1,20]'::floatspan), 1)::tstzspan),
   atTime(time_slice_geoaltitude,  sequenceN(atValues(time_slice_vertrate,
                                        '[1,20]'::floatspan), 1)::tstzspan),
  atTime(time_slice_vertrate,  sequenceN(atValues(time_slice_vertrate,
                                        '[1,20]'::floatspan), 1)::tstzspan)
   FROM flight_traj_time_slice
  WHERE atTime(time_slice_trip, sequenceN( atValues(time_slice_vertrate,
                                        '[1,20]'::floatspan), 1)::tstzspan) IS NOT NULL),

Australia(geomaustralia) AS
  (SELECT ST_Transform(ST_makeEnvelope(-1758447, -
          4666808,2281883,-1535490,3112),4326) )
--3112 is the SRID of Australia
SELECT icao24, callSign, trajectory(ascending_trip) as geom
FROM flight_traj_time_slice_ascent f, Australia
WHERE atGeometry(f.ascending_trip, geomaustralia) IS NOT NULL  AND
      ST_GeometryType(trajectory(ascending_trip)) = 'ST_LineString'
)  AS subq11
```

5.3. ***Altitude of aircraft icao24='06a0af' during take-off in the interval 2020-06-01 03:00:00, 2020-06-01 04:00:00.***

This query allows reviewing important functions

```
WITH
flight_traj_time_slice (icao24, callsign, time_slice_trip, time_slice_geoaltitude,
time_slice_vertrate) AS(
SELECT icao24,callsign,
  atTime(trip, '[2020-06-01 03:00:00, 2020-06-01
        04:00:00)'::tstzspan),
  atTime(geoaltitude, '[2020-06-01 03:00:00, 2020-06-01 04:00:00)'::tstzspan),
```

```
 atTime(vertrate, '[2020-06-01 03:00:00, 2020-06-01 04:00:00)'::tstzspan)
FROM flight_traj
WHERE  icao24='0100a3'  AND atTime(trip, '[2020-06-01 03:00:00, 2020-06-01
                                    04:00:00)'::tstzspan)  IS NOT NULL
),
-- ASCENDING PLANES
flight_traj_time_slice_ascent(icao24, callsign, ascending_trip,
ascending_geoaltitude, ascending_vertrate) AS
(SELECT icao24, callsign,
 atTime(time_slice_trip, sequenceN(atValues(time_slice_vertrate,
                                    '[1,20]'::floatspan), 1)::tstzspan),
 atTime(time_slice_geoaltitude, sequenceN(atValues(time_slice_vertrate,
                                    '[1,20]'::floatspan),1)::tstzspan),
 atTime(time_slice_vertrate, sequenceN(atValues(time_slice_vertrate,
                                    '[1,20]'::floatspan), 1)::tstzspan)
FROM flight_traj_time_slice
WHERE
atTime(time_slice_trip, sequenceN(atValues(time_slice_vertrate,
                                    '[1,20]'::floatspan), 1)::tstzspan) IS NOT NULL),
StatsTable AS(
  SELECT icao24, callsign, unnest(instants(ascending_geoaltitude)),
         startTimestamp(unnest(instants(ascending_geoaltitude))) AS timeasc,
         getValue(unnest(instants(ascending_geoaltitude))) AS altitude
-- instants return the (value,time) pairs as an array
-- unnest them as single value@time
-- getValue gets the value
-- startTimestamp obtains the initial timestamp (in this case, the only one)
-- startValue obtains the initial value (=getValue for a simple element)
FROM flight_traj_time_slice_ascent)
SELECT timeasc,altitude
FROM StatsTable
```

5.4. *All flights over Australia in the interval 2020-06-01 03:00:00, 2020-06-01 03:30:00*

```
SELECT row_number() over() AS ctid, geom
FROM(
WITH
Australia AS (SELECT ST_Transform(ST_makeEnvelope
        (-1758447, -4666808,2281883,-1535490,3112),4326) AS australia)
SELECT icao24, trajectory(atTime(atGeometry(trip,australia), '[2020-06-01
                    03:00:00, 2020-06-01 03:30:00)'::tstzspan)) AS geom
FROM airframe_traj, Australia
WHERE eintersects(trip,australia) IS NOT NULL AND
      atTime(atGeometry(trip,australia), '[2020-06-01 03:00:00, 2020-06-01
          03:30:00)'::tstzspan) IS NOT NULL
) AS Subq1
```

5.5. Duration of all flights over Australia, showing the trajectory, the number of points in the trajectory, and the duration computed in two different ways: using `timespan` and using `duration`.

```
WITH
Australia AS (SELECT ST_Transform(ST_makeEnvelope
        (-1758447, -4666808,2281883,-1535490,3112),4326) AS Australia)
SELECT icao24, atGeometry(trip,australia)::tstzspan,
       numtimestamps(atGeometry(trip,australia)),
       duration(atGeometry(trip,australia)), timespan(atGeometry(trip,australia)),
       trajectory(atGeometry(trip,australia))
FROM airframe_traj, Australia
WHERE eintersects(trip,australia) IS NOT null AND atGeometry(trip,australia) IS
      NOT NULL
```

5.6. ***Duration of flight icao24= 'c0175a' over Australia, together with the distance travelled within the country.***

```
WITH
Australia AS (SELECT ST_Transform(ST_makeEnvelope
        (-1758447, -4666808,2281883,-1535490,3112),4326) AS Australia)
SELECT icao24,length(transform((atGeometry(trip,australia)), 3112)),
       st_length(st_transform(trajectory(atGeometry(trip,australia))::geometry,
       3112)),
       atGeometry(trip,australia)::tstzspan,
       timespan(atGeometry(trip,australia)),
       trajectory(atGeometry(trip,australia))
FROM airframe_traj , Australia
WHERE eintersects(trip,australia) IS NOT NULL AND
       atGeometry(trip,australia) IS NOT NULL AND  icao24='c0175a'
```

5. 7. ***Compute the distance between two planes at all instants and visualize the results in QGIS. Use planes with icao24='06a1bc' and icao24='040039'.***

```
WITH mindist AS(
    SELECT transform(s1.trip, 3112) <-> transform(s2.trip,3112) AS distance
    FROM flight_traj s1, flight_traj s2
    WHERE s1.icao24 > s2.icao24 AND  atMin(transform(s1.trip,3112) <->
          transform(s2.trip,3112)) IS NOT NULL   AND s1.icao24='06a1bc' AND
          s2.icao24='040039')
SELECT startTimestamp(unnest(instants(distance))) as time,
       getValue(unnest(instants(distance)))/1000 as distance
FROM mindist
```

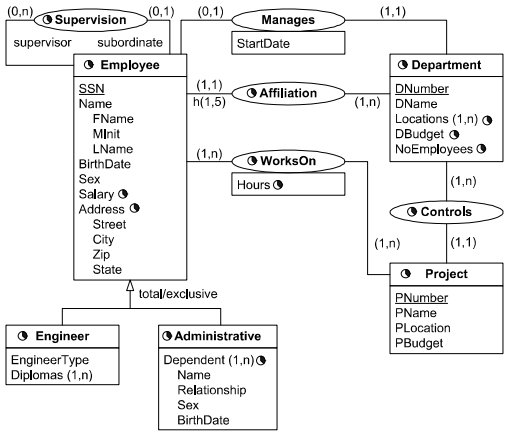
transform(s1.trip,3112) transforms the geometries in the trip to the SRID of Australia.

<-> returns the distance between two moving points at all timestamps in the trajectory

atMin(s1.trip <-> s2.trip) returns the minimum distance between two moving objects and the moment when that occurred

instants(distance) converts the sequence into an array that can later be unnested to build the series.

SQL Server
Temporal Databases



Consider the above temporal conceptual schema

- Define a relational schema corresponding to the conceptual schema.
- Write the following queries in SQL:
  1. Give the name of managers living currently in Houston
  2. Give the name of employees working currently in the 'Research' department having a salary greater or equal than 45000
  3. Give the name of current employees who does not work currently in any department
  4. Give the name of the employee(s) that had the highest salary in 1/1/2002
  5. Provide the salary and affiliation history for all employees
  6. Give the name of employees and the period of time in which they were supervisors but did not work in any project during the same period
  7. Give the name of supervisors who had work on a project at some time
  8. Give the name of employees and the date they changed their affiliation
  9. Give the name of employees and the periods they worked on any project
  10. Give the history of the maximum salary
  11. Give by department the history of the maximum salary
  12. Give the history of the number of projects of a department
  13. Give the name of employees and the periods they worked on all projects of their department

Relational Schema

- Employee(SSN, FName, MInit, LName, BirthDate, Sex)
- EmployeeLifecycle(SSN, FromDate, ToDate)
  SSN references Employee(SSN)
- EmployeeSalary(SSN, Salary, FromDate, ToDate)
  SSN references Employee(SSN)
- EmployeeAddress(SSN, Street, City, Zip, Country, FromDate, ToDate)
  SSN references Employee(SSN)
- Engineer(SSN, EngineerType, FromDate, ToDate)
  SSN references Employee(SSN)
  In this table the lifecycle of Engineer is kept as well as the attribute EngineerType. There will be redundancy if the lifecycle of Engineer is not continuous.
- EngineerDiplomas(SSN, Diploma)
  SSN references Engineer(SSN)
- AdministrativeLifecycle(SSN, FromDate, ToDate)
  SSN references Employee(SSN)
- AdminDependent(SSN, Name, Relationship, Sex, BirthDate, FromDate, ToDate)
  SSN references AdministrativeLifecycle(SSN)
  It is supposed that an employee does not have two dependents of the same name and the same relationship. An alternative will be to put BirthDate instead of Relationship as part of the key.
- Supervision(SSN, SuperSSN, FromDate, ToDate)
  SSN references Employee(SSN)
  SuperSSN references Employee(SSN)
- Affiliation(SSN, DNumber, FromDate, ToDate)
  SSN references Employee(SSN)
- Department(DNumber, DName, MgrSSN, MgrStartDate, FromDate, ToDate)
  MgrSSN references Employee(SSN)
  It is supposed that the lifecycle of departments is continuous. In this case an additional table for the lifecycle is not necessary.
- DeptLocations(DNumber, Location, FromDate, ToDate)
  DNumber references Department(DNumber)
- DepartmentBudget(DNumber, DBudget, FromDate, ToDate)
  DNumber references Department(DNumber)
- DepartmentNbEmp(DNumber, NbEmp, FromDate, ToDate)
  DNumber references Department(DNumber)
- Project(PNumber, PName, PLocation, PBudget, FromDate, ToDate)
  It is supposed that the lifecycle of Project is continuous.
- Controls(PNumber, DNumber, FromDate, ToDate)
  PNumber references Project(PNumber)
  DNumber references Department(DNumber)
- WorksOn(SSN, PNumber, Hours, FromDate, ToDate)
  PNumber references Project(PNumber)
  SSN references Employee(SSN)
  It is supposed that the temporality of attribute Hours is the same as the lifecycle of the association. In this case two different tables are not necessary. To obtain the lifecycle of the association independently of the attribute hours a temporal projection is needed.

Example Database

Partial schema where not all entities and attributes are taken into account.

Employee

SSN	FName	MInit	LName	BirthDate	Sex
123456789	John	B	Smith	09-05-1955	M
333445555	Franklin	T	Wong	08-12-1945	M
999887777	Alicia	J	Zelaya	19-07-1958	F
987654321	Jennifer	S	Wallace	20-06-1931	F
666884444	Ramesh	K	Narayan	15-09-1952	M
453453453	Joyce	A	English	31-07-1962	F
987987987	Ahmad	V	Jabbar	29-03-1959	M
888665555	James	A	Borg	10-11-1927	M

EmployeeLifecycle

SSN	FromDate	ToDate
123456789	01-01-1985	01-01-2079
333445555	01-01-1982	01-01-2079
999887777	01-01-1985	01-01-2079
987654321	01-01-1982	01-01-2079
666884444	01-01-1985	01-01-2079
453453453	01-01-1985	01-01-2079
987987987	01-01-1985	01-01-2079
888665555	01-01-1980	01-01-2079

EmployeeSalary

SSN	Salary	FromDate	ToDate
123456789	30000	01-01-1985	01-01-2079
333445555	40000	01-01-1982	01-01-1983
333445555	45000	01-01-1983	01-01-2079
999887777	25000	01-01-1985	01-01-2079
987654321	43000	01-01-1982	01-01-2079
666884444	38000	01-01-1985	01-01-2079
453453453	25000	01-01-1985	01-01-2079
987987987	25000	01-01-1985	01-01-2079
888665555	55000	01-01-1980	01-01-1981
888665555	58000	01-01-1981	01-01-2079

EmployeeAddress

SSN	Street	City	Zip	State	FromDate	ToDate
123456789	731 Fondren	Houston	1000	TX	01-01-1985	01-01-2079
333445555	638 Voss	Houston	1000	TX	01-01-1982	01-01-2079
999887777	3321 Castle	Spring	1000	TX	01-01-1985	01-01-2079
987654321	291 Berry	Bellaire	1000	TX	01-01-1982	01-01-2079
666884444	975 Fire Oak	Humble	1000	TX	01-01-1985	01-01-2079
453453453	5631 Rice	Houston	1000	TX	01-01-1985	01-01-2079
987987987	980 Dallas	Houston	1000	TX	01-01-1985	01-01-2079
888665555	450 Stone	Houston	1000	TX	01-01-1980	01-01-2079

Supervision

SSN	SuperSSN	FromDate	ToDate
123456789	333445555	01-01-1985	01-01-2079
333445555	888665555	01-01-1982	01-01-2079
999887777	987654321	01-01-1985	01-01-2079
987654321	888665555	01-01-1982	01-01-2079
666884444	333445555	01-01-1985	01-01-2079
453453453	333445555	01-01-1985	01-01-2079
987987987	987654321	01-01-1985	01-01-2079

Affiliation

SSN	DNumber	FromDate	ToDate
123456789	1	01-01-1985	01-01-1986
123456789	5	01-01-1986	01-01-2079
333445555	4	01-01-1982	01-01-1984
333445555	5	01-01-1984	01-01-2079
999887777	4	01-01-1985	01-01-2079
987654321	4	01-01-1982	01-01-2079
666884444	5	01-01-1985	01-01-2079
453453453	5	01-01-1985	01-01-2079
987987987	4	01-01-1985	01-01-2079
888665555	1	01-01-1980	01-01-2079

Department

DNumber	DName	MgrSSN	MgrStartDate	FromDate	ToDate
1	Headquarters	888665555	19-06-1980	01-01-1980	01-01-2079
4	Administration	987654321	01-01-1982	01-01-1981	01-01-2079
5	Research	333445555	22-05-1984	01-01-1982	01-01-2079

DepartmentNbEmp

DNumber	NbEmp	FromDate	ToDate
5	4	01-01-1980	01-01-2079
4	3	01-01-1980	01-01-2079
1	1	01-01-1980	01-01-2079

DeptLocations

DNumber	DLocation	FromDate	ToDate
1	Houston	01-01-1980	01-01-2079
4	Stafford	01-01-1980	01-01-2079
5	Bellaire	01-01-1980	01-01-2079
5	Sugarland	01-01-1980	01-01-2079
5	Houston	01-01-1980	01-01-2079

Project

PNumber	PName	PLocation	FromDate	ToDate
1	ProductX	Bellaire	01-01-1980	01-01-2079
2	ProductY	Sugarland	01-01-1980	01-01-2079
3	ProductZ	Houston	01-01-1980	01-01-2079
10	Computerization	Stafford	01-01-1980	01-01-2079
20	Reorganization	Houston	01-01-1980	01-01-2079
30	Newbenefits	Stafford	01-01-1980	01-01-2079

Controls

PNumber	DNumber	FromDate	ToDate
1	5	01-01-1980	01-01-2079
2	5	01-01-1980	01-01-2079
3	5	01-01-1980	01-01-2079
10	4	01-01-1980	01-01-2079
20	1	01-01-1980	01-01-2079
30	4	01-01-1980	01-01-2079

WorksOn

SSN	PNumber	Hours	FromDate	ToDate
123456789	1	32.5	01-01-1985	01-01-2079
123456789	2	7.5	01-01-1985	01-01-2079
333445555	1	10	01-01-1982	01-01-2000
333445555	2	10	01-01-1982	01-01-2002
333445555	3	20	01-01-2005	01-01-2079
453453453	1	20	01-01-1985	01-01-2079
453453453	2	20	01-01-1985	01-01-2079
666884444	3	40	01-01-1985	01-01-2079
888665555	20	30.0	01-01-1983	01-01-2079
987654321	10	5.0	01-01-1982	01-01-2000
987654321	20	15.0	01-01-1982	01-01-2001
987654321	30	20.0	01-01-1982	01-01-2002
987987987	10	35.0	01-01-1985	01-01-2079
987987987	30	5.0	01-01-1985	01-01-2079
999887777	10	10.0	01-01-1985	01-01-2079
999887777	30	30.0	01-01-1985	01-01-2079

Queries

1. Give the name of the managers living currently in Houston

```
select E.FName, E.LName
from Employee E, EmployeeAddress A, Department D
where E.SSN = A.SSN and E.SSN = D.MgrSSN
  and A.City = 'Houston'
  and A.FromDate <= current_date and current_date < A.ToDate
  and D.FromDate <= current_date and current_date < D.ToDate;
```

2. Give the name of employees working currently in the 'Research' department and having a salary greater or equal than 45000

```
select E.FName, E.LName
from Employee E, EmployeeSalary S, Affiliation A, Department D
where E.SSN = S.SSN and E.SSN = A.SSN and A.DNumber = D.DNumber
  and D.DName = 'Research' and S.Salary >= 45000
  and S.FromDate <= current_date and current_date < S.ToDate
  and A.FromDate <= current_date and current_date < A.ToDate;
```

3. Give the name of current employees who do not work currently in any department

```
select distinct E.FName, E.LName
    from Employee E, EmployeeLifecycle L
    where E.SSN = L.SSN
      and L.FromDate <= current_date and current_date < L.ToDate
      and not exists (
      select * from Affiliation A
                where E.SSN = A.SSN
                  and A.FromDate <= current_date and current_date < A.ToDate );
```

4. Give the name of the employee(s) that had the highest salary on 1/1/2002

```
select E.FName, E.LName
         from Employee E, EmployeeSalary S
         where E.SSN = S.SSN
         and salary = ( select max(salary) from EmployeeSalary
                        where FromDate <= '2002-01-01' and '2002-01-01' < ToDate )
         and S.FromDate <= '2002-01-01' and '2002-01-01' < S.ToDate;
```

5. Provide the salary and affiliation history for all employees

```
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 E.FName, E.LName, D.DName, S.Salary,
       maxDate(S.FromDate,A.FromDate) as "Start date",
       minDate(S.ToDate,A.ToDate) as "End date"
from Employee E, EmployeeSalary S, Affiliation A, Department D
where E.SSN = S.SSN and E.SSN = A.SSN and A.DNumber = D.DNumber
and maxDate(S.FromDate,A.FromDate) < minDate(S.ToDate,A.ToDate)
order by E.FName, E.LName;
```

6. Give the name of employees and the period of time in which they were supervisors but did not work in any project during the same period

```
--Case 1
select S.SuperSSN, S.FromDate, W1.FromDate as ToDate
from Supervision S, WorksOn W1
where S.SuperSSN = W1.SSN
  and S.FromDate < W1.FromDate and W1.FromDate < S.ToDate
  and not exists ( select * from WorksOn W2 where S.SuperSSN = W2.SSN
                   and S.FromDate < W2.ToDate and W2.FromDate < W1.FromDate )
union
--Case 2
select S.SuperSSN, W1.ToDate as FromDate, S.ToDate
from Supervision S, WorksOn W1
where S.SuperSSN = W1.SSN
  and S.FromDate < W1.ToDate and W1.ToDate < S.ToDate
  and not exists ( select * from WorksOn W2 where S.SuperSSN = W2.SSN
                   and W1.ToDate < W2.ToDate and W2.FromDate < S.ToDate )
union
--Case 3
select S.SuperSSN, W1.ToDate as FromDate, W2.FromDate as ToDate
from Supervision S, WorksOn W1, WorksOn W2
where S.SuperSSN = W1.SSN and S.SuperSSN = W2.SSN and W1.ToDate < W2.FromDate
  and S.FromDate < W1.ToDate and W2.FromDate < S.ToDate
  and not exists ( select * from WorksOn W3 where S.SuperSSN = W3.SSN
                   and W1.ToDate < W3.ToDate and W3.FromDate < W2.FromDate )
union
--Case 4
select SuperSSN, FromDate, ToDate from Supervision S
where not exists ( select * from WorksOn W where S.SuperSSN=W.SSN
                   and S.FromDate < W.ToDate and W.FromDate < S.ToDate );
```

7. Give the name of supervisors who had work on a project at some time

```
select distinct E.FName, E.LName
from Employee E, Supervision S, WorksOn W
where E.SSN = S.SuperSSN and E.SSN = W.SSN;
```

8. Give the name of employees and the date they changed their affiliation

```
select distinct E.FName, E.LName, A1.ToDate
from Employee E, Affiliation A1, Affiliation A2
where E.SSN = A1.SSN and E.SSN = A2.SSN
  and A1.ToDate = A2.FromDate and A1.DNumber <> A2.DNumber;
```

9. Give the name of employees and the periods they worked on any project

```
select distinct E.SSN, E.FName, E.LName, F.FromDate, L.ToDate
from Employee E, WorksOn F, WorksOn L
where E.SSN = F.SSN and F.SSN = L.SSN and F.FromDate < L.ToDate
  and not exists (
    select * from WorksOn M
```



```
    where M.SSN = F.SSN
      and F.FromDate < M.FromDate and M.FromDate <= L.ToDate
      and not exists ( select * from WorksOn T1
                       where T1.SSN = F.SSN
                         and T1.FromDate < M.FromDate and M.FromDate <= T1.ToDate ) )
  and not exists (
    select * from WorksOn T2
    where T2.SSN = F.SSN
     and ( ( T2.FromDate < F.FromDate and F.FromDate <= T2.ToDate )
       or ( T2.FromDate <= L.ToDate and L.ToDate < T2.ToDate ) ) );
```

10. Give the history of the maximum salary

```
-- First step: Construct intervals during which no salary change occurred
WITH Instants(Instant) AS (
select distinct E.FromDate from EmployeeSalary E
                           union select distinct E.ToDate from EmployeeSalary E ),
    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 ) ),

-- Second step: Compute the maximum salary for these intervals
TempMax(SalaryMax, FromDate, ToDate) AS (
select max(E.Salary), I.FromDate, I.ToDate
from EmployeeSalary E, Intervals I
where E.FromDate <= I.FromDate and I.ToDate <= E.ToDate
group by I.FromDate, I.ToDate )

-- Third step: Coalescing the above table
select distinct F.SalaryMax, F.FromDate, L.ToDate
from TempMax F, TempMax L
where F.FromDate < L.ToDate and F.SalaryMax =  L.SalaryMax
  and not exists ( '
    select * from TempMax M
    where M.SalaryMax =  F.SalaryMax
        and F.ToDate < M.FromDate and M.FromDate <= L.FromDate
        and not exists (
            select * from TempMax T1
            where T1.SalaryMax =  F.SalaryMax
              and T1.FromDate < M.FromDate and M.FromDate <= T1.ToDate ) )
  and not exists (
    select * from TempMax T2
    where T2.SalaryMax = F.SalaryMax
     and ( ( T2.FromDate < F.FromDate and F.FromDate <= T2.ToDate )
           or ( T2.FromDate <= L.ToDate and L.ToDate < T2.ToDate ) ) )
order by F.FromDate;
```

11. Give by department the history of the maximum salary

```
-- First step: Construct by department the intervals during
-- which the maximum salary must be calculated.
WITH Aff_Sal (DNumber, Salary, FromDate, ToDate) AS (
select distinct A.DNumber, S.Salary,
                maxDate(S.FromDate,A.FromDate),
                minDate(S.ToDate,A.ToDate)
```



```
from Affiliation A, EmployeeSalary S
where A.SSN = S.SSN
  and maxDate(S.FromDate,A.FromDate) < minDate(S.ToDate,A.ToDate) ),
    SalChanges(DNumber, Instant) AS (
        select distinct DNumber, FromDate from Aff_Sal
        union
        select distinct DNumber, ToDate from Aff_Sal ),
    SalIntervals(DNumber, FromDate, ToDate) AS (
        select distinct P1.DNumber, P1.Instant, P2.Instant
        from SalChanges P1, SalChanges P2
        where P1.DNumber=P2.DNumber and P1.Instant<P2.Instant
          and not exists (
            select * from SalChanges P3
            where P1.DNumber = P3.DNumber and P1.Instant < P3.Instant
              and P3.Instant < P2.Instant ) ),

-- Second step: Compute the maximum salary for the
-- above periods.
TempMaxDep(DNumber, MaxSalary, FromDate, ToDate) AS (
select P.DNumber, max(Salary), P.FromDate, P.ToDate
from Aff_Sal A, SalIntervals P
where A.DNumber = P.DNumber
  and A.FromDate <= P.FromDate and P.ToDate <= A.ToDate
group by P.DNumber, P.FromDate, P.ToDate  )

-- Third step: Coalescing the above table
select distinct F.DNumber, F.MaxSalary, F.FromDate, L.ToDate
from TempMaxDep F, TempMaxDep L
where F.DNumber = L.DNumber and F.MaxSalary = L.MaxSalary
  and F.FromDate < L.ToDate
  and not exists (
    select * from TempMaxDep M
    where F.DNumber = M.DNumber and F.MaxSalary = M.MaxSalary
      and F.ToDate < M.FromDate and M.FromDate <= L.FromDate
      and not exists ( select *
                       from TempMaxDep T1
                       where F.DNumber = T1.DNumber and F.MaxSalary = T1.MaxSalary
                         and T1.FromDate < M.FromDate and M.FromDate <= T1.ToDate ) )
  and not exists (
    select * from TempMaxDep T2
    where F.DNumber = T2.DNumber and F.MaxSalary = T2.MaxSalary
      and ( ( T2.FromDate < F.FromDate and F.FromDate <= T2.ToDate )
            or ( T2.FromDate <= L.ToDate and L.ToDate < T2.ToDate ) ) )
order by F.DNumber, F.FromDate;
```

12. Give the history of the number of projects of a department

```
-- First step: Construct intervals during which the number of
-- projects of a department does not change
WITH Instants(DNumber, Instant) AS (
select distinct DNumber, FromDate from Controls
union
select distinct DNumber, ToDate from Controls ),

Intervals(DNumber, FromDate, ToDate) AS (
select distinct I1.DNumber, I1.Instant, I2.Instant
from Instants I1, Instants I2
where I1.DNumber = I2.DNumber
  and I1.Instant < I2.Instant
```



```
  and not exists ( select * from Instants I3
                   where I1.DNumber = I3.DNumber
                     and I1.Instant < I3.Instant
                     and I3.Instant < I2.Instant ) ),

-- Second step: Compute the number of projects for these intervals
TempCountDep(DNumber, NbProjects, FromDate, ToDate) AS (
select I.DNumber, count(C.PNumber), I.FromDate, I.ToDate
from Controls C, Intervals I
where C.DNumber = I.DNumber
  and ( C.FromDate <= I.FromDate and I.ToDate <= C.ToDate)
group by I.DNumber, I.FromDate, I.ToDate )

-- Third step: Coalescing the above table
select distinct F.DNumber, F.NbProjects, F.FromDate, L.ToDate
from TempCountDep F, TempCountDep L
where F.DNumber = L.DNumber and F.FromDate < L.ToDate
  and F.NbProjects = L.NbProjects
  and not exists (
    select * from TempCountDep M
    where M.DNumber = F.DNumber and M.NbProjects = F.NbProjects
      and F.ToDate < M.FromDate and M.FromDate <= L.FromDate
      and not exists ( select *
                       from TempCountDep T1
                       where T1.DNumber = F.DNumber and T1.NbProjects = F.NbProjects
                         and T1.FromDate < M.FromDate and M.FromDate <= T1.ToDate ) )
  and not exists (
    select * from TempCountDep T2
    where T2.DNumber = F.DNumber and T2.NbProjects = F.NbProjects
      and ( ( T2.FromDate < F.FromDate and F.FromDate <= T2.ToDate )
            or ( T2.FromDate <= L.ToDate and L.ToDate < T2.ToDate ) ) )
order by F.DNumber, F.FromDate;
```

13. Give the name of employees and the periods they worked on all projects of their department

```
-- First step: Construct intervals during which the number of projects
-- of an employee does not change
WITH Aff_Cont(SSN, DNumber, PNumber, FromDate, ToDate) AS (
select distinct A.SSN, A.DNumber, C.PNumber,
                maxDate(A.FromDate,C.FromDate),
                minDate(A.ToDate,C.ToDate)
from Affiliation A, Controls C
where A.DNumber = C.DNumber
  and maxDate(A.FromDate,C.FromDate) < minDate(A.ToDate,C.ToDate) ),

Aff_Cont_WO(SSN, DNumber, PNumber, FromDate, ToDate) AS (
select distinct A.SSN, A.DNumber, W.PNumber,
                maxDate(A.FromDate,W.FromDate),
                minDate(A.ToDate,W.ToDate)
from Aff_Cont A, WorksOn W
where A.PNumber = W.PNumber and A.SSN = W.SSN
  and maxDate(A.FromDate,W.FromDate) < minDate(A.ToDate,W.ToDate) ),

ProjChanges(SSN, DNumber, Instant) AS (
select distinct SSN, DNumber, FromDate from Aff_Cont
union select distinct SSN, DNumber, ToDate from Aff_Cont
union select distinct SSN, DNumber, FromDate from Aff_Cont_WO
union select distinct SSN, DNumber, ToDate from Aff_Cont_WO
```

Relational model

Employee
SSN Name

- In this model the principal concept is the relation ($\sim$ table)
- The entities, the associations and multivalued attributes are translated by relations
- Model : Relation(Key(s), Attribute, *Optionnal Attribute*, ...)
- Translation :
  Employee(SSN, Name)

(1) multivalued attributes

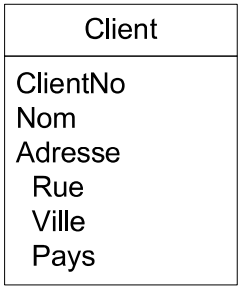
Livre
ISBN Auteur (1,n)


```
    union select SSN, DNumber, FromDate from Affiliation
    union select SSN, DNumber, ToDate from Affiliation ),

ProjIntervals(SSN, DNumber, FromDate, ToDate) AS (
select distinct P1.SSN, P1.DNumber, P1.Instant, P2.Instant
from ProjChanges P1, ProjChanges P2
where P1.SSN = P2.SSN
  and P1.DNumber = P2.DNumber and P1.Instant < P2.Instant
  and not exists (
    select * from ProjChanges P3
    where P1.SSN = P3.SSN and P1.DNumber = P3.DNumber
      and P1.Instant < P3.Instant and P3.Instant < P2.Instant ) ),

-- Second step: Compute the number of projects for these intervals
TempUnivQuant(SSN, FromDate, ToDate) AS (
select distinct P.SSN, P.FromDate, P.ToDate
from ProjIntervals P
where not exists (
    select * from Controls C
    where P.DNumber = C.DNumber
      and C.FromDate <= P.FromDate and P.ToDate <= C.ToDate
      and not exists (
        select * from WorksOn W
        where C.PNumber = W.PNumber and P.SSN = W.SSN
          and W.FromDate <= P.FromDate and P.ToDate <= W.ToDate ) ) )

-- Third step: Coalescing the above table
select distinct F.SSN, F.FromDate, L.ToDate
from TempUnivQuant F, TempUnivQuant L
where F.SSN = L.SSN and F.FromDate < L.ToDate
  and not exists (
    select * from TempUnivQuant M
    where M.SSN = F.SSN
      and F.ToDate < M.FromDate and M.FromDate <= L.FromDate
      and not exists (
        select * from TempUnivQuant T1
        where T1.SSN = F.SSN
          and T1.FromDate < M.FromDate and M.FromDate <= T1.ToDate ) )
  and not exists (
    select *
    from TempUnivQuant T2
    where T2.SSN = F.SSN
      and ( ( T2.FromDate < F.FromDate and F.FromDate <= T2.ToDate )
        or ( T2.FromDate <= L.ToDate and L.ToDate < T2.ToDate ) ) )
order by F.SSN, F.FromDate;
```

INFO-H-415 Advanced Databases
Temporal Databases Part 1
Traduction of EA schema to relational databases

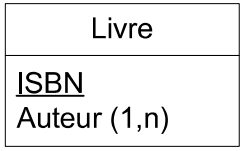
13 octobre 2021

(2) Translation of composed attributes



Client(ClientNo, Nom, AdresseRue, AdresseVille, AdressePays)

(1) multivalued attributes

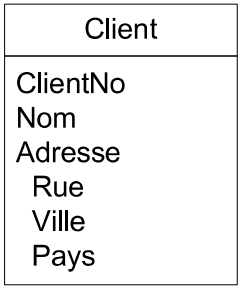


Livre(ISBN, . . .)
LivreAuteur(ISBN, Auteur)
LivreAuteur.ISBN references Livre.ISBN

- Question : why (ISBN,Auteur) and not (ISBN,Auteur) ?

(3) Translation of 'one to one' or 'one to many' associations

(2) Translation of composed attributes



(3) Translation of 'many to many' associations

Employee(SSN, Name)
Project(PNo, Name)
EmpProj(SSN,PNo)
EmpProj.SSN references Employee.SSN
EmpProj.PNo references Project.PNo

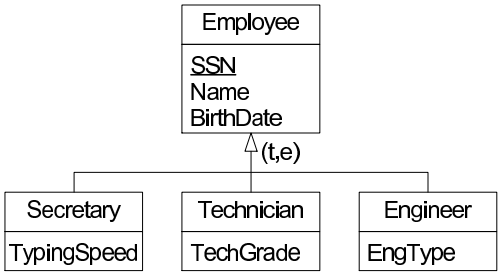
- Careful, (SSN,PNo) $\neq$ (SSN,PNo)

(3) Translation of 'one to one' or 'one to many' associations

Department(DNo, Name)
Employee(SSN, Name, DNo)
Employee.DNo reference Department.DNo

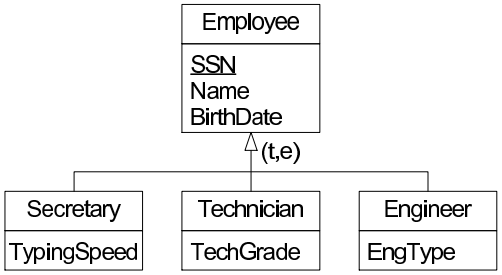
- 'one to one' association : if one is optional, the reference goes to the mandatory side !
- 'one to many' association : the reference goes to the 'one' side

(4) Translation of generalisations : solution 1



(3) Translation of 'many to many' associations

(4) Translation of generalisations : solution 2



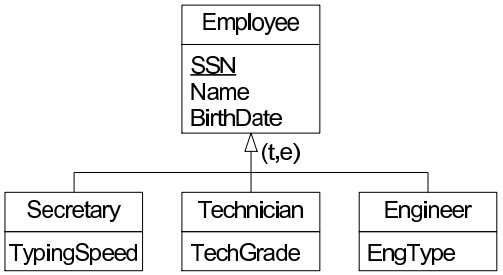
Secretary(SSN, FName, MInit, LName, BirthDate, Address, TypingSpeed)

Technician(SSN, FName, MInit, LName, BirthDate, Address,, TechGrade)

Engineer(SSN, FName, MInit, LName, BirthDate, Address, EngType)

- \+ integrity constraints

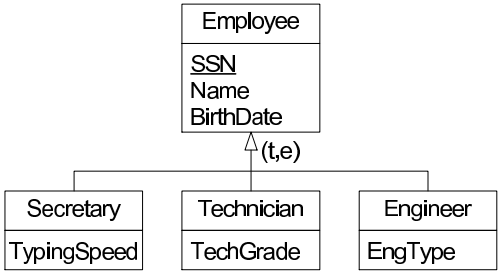
(4) Translation of generalisations : solution 1



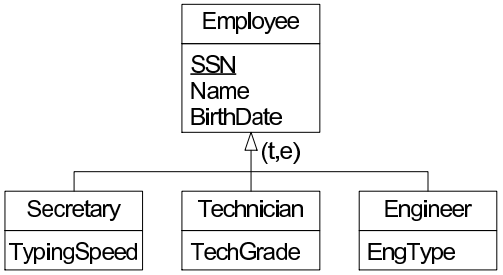
Employee(SSN, FName, MInit, LName, BirthDate, Address)
Secretary(SSN, TypingSpeed)
Secretary.SSN reference Employee.SSN
Technician(SSN, TechGrade)
Technician.SSN reference Employee.SSN
Engineer(SSN, EngType)
Engineer.SSN reference Employee.SSN

- \+ integrity constraints

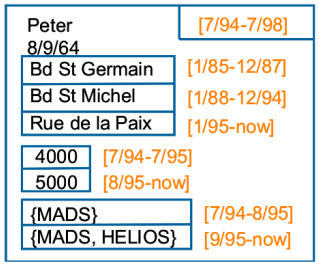
(4) Translation of generalisations : solution 3

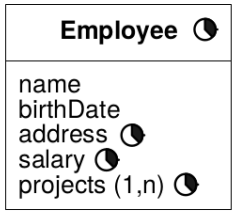


(4) Translation of generalisations : solution 2

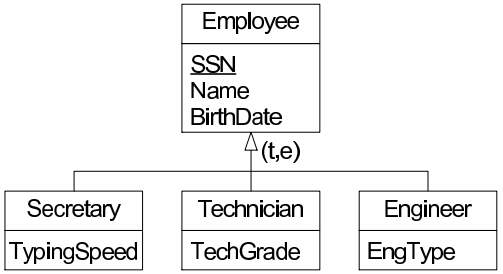


(5) Translation sequences relations





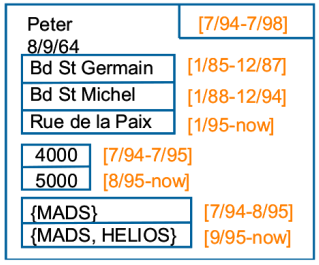
(4) Translation of generalisations : solution 3

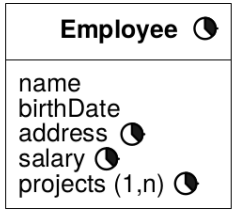


Employee(SSN, FName, MInit, LName, BirthDate, Address, TypingSpeed, TechGrade, EngType)

- \+ integrity constraints

(5) Translation sequences relations





Employee(name, startTime, birthDate, endTime)

EmployeeAddress(name, startTime, address, endTime)

EmployeeSalary(name, startTime, salary, endTime)

EmployeeProject(name, startTime, project, endTime)

- \+ integrity constraints

(4) Translation of generalisations

- What can we say about these generalisations ?
  - Total, non-exclusive ?
  - Partial, exclusive ?
  - Partial, non-exclusive ?

Temporal join Example

Thursday, October 18, 2018 9:24 AM

S

EmployeeSal

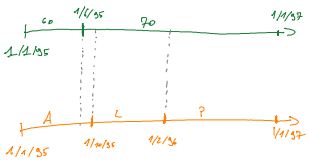
Name	Salary	FromDate	ToDate
John	60.000	1/1/95	1/6/95
John	70.000	1/6/95	1/1/97

EmployeeTitle

Name	Title	FromDate	ToDate
John	Assistant	1/1/95	1/10/95
John	Lecturer	1/10/95	1/2/96
John	Professor	1/2/96	1/1/97

EmployeeSal ⋈ EmployeeTitle

Name	Salary	Title	FromDate	ToDate
John	60.000	Assistant	1/1/95	1/6/95
John	70.000	Assistant	1/6/95	1/10/95
John	70.000	Lecturer	1/10/95	1/2/96
John	70.000	Professor	1/2/96	1/1/97



For each couple $t \in T$, $s \in S$
find the period where they are both valid

A. Extensive Method

4 cases

1)

```
SELECT s.SSN, s.salary, t.title, t.FromDate, t.ToDate
FROM S s, T t
WHERE t.SSN = s.SSN
      AND s.FromDate <= t.FromDate
      AND t.ToDate <= s.ToDate
```

2)

```
SELECT s.SSN, s.salary, t.title, s.FromDate, t.ToDate
FROM S s, T t
WHERE t.SSN = s.SSN
      AND s.FromDate > t.FromDate
      AND t.ToDate > s.FromDate
      AND t.ToDate <= s.ToDate
```

3)

```
SELECT s.SSN, s.salary, t.title, t.FromDate, s.ToDate
FROM S s, T t
WHERE t.SSN = s.SSN
      AND t.FromDate > s.FromDate
      AND t.FromDate < s.ToDate
      AND s.ToDate <= t.ToDate
```

4)

```
SELECT s.SSN, s.salary, t.title, s.FromDate, s.ToDate
FROM S s, T t
WHERE t.SSN = s.SSN
      AND t.FromDate < s.FromDate
      AND s.ToDate < t.ToDate
```

be sure all the select have the same columns

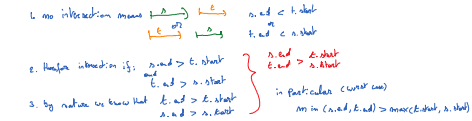
```
SELECT s.SSN, s.salary, t.title, t.FromDate as START, t.ToDate as END
      FROM S s, T t
      WHERE t.SSN = s.SSN
            AND s.FromDate <= t.FromDate
            AND t.ToDate <= s.ToDate
UNION ALL
SELECT s.SSN, s.salary, t.title, s.FromDate as START, t.ToDate as END
      FROM S s, T t
      WHERE t.SSN = s.SSN
            AND s.FromDate > t.FromDate
            AND t.ToDate > s.FromDate
            AND t.ToDate <= s.ToDate
UNION ALL
SELECT s.SSN, s.salary, t.title, t.FromDate as START, s.ToDate as END
      FROM S s, T t
      WHERE t.SSN = s.SSN
            AND t.FromDate > s.FromDate
            AND t.FromDate < s.ToDate
            AND s.ToDate <= t.ToDate
UNION ALL
SELECT s.SSN, s.salary, t.title, s.FromDate as START, s.ToDate as END
      FROM S s, T t
      WHERE t.SSN = s.SSN
            AND t.FromDate < s.FromDate
            AND s.ToDate < t.ToDate
```


B. Short method

Always [max(t.start), min(t.end)]

But check that they intersect!

```
SELECT s.SSN, s.salary, t.title, maxDate(s.FromDate, t.FromDate) AS StartDate,
                                 minDate(s.ToDate, t.ToDate) AS EndDate
FROM S s, T t
WHERE s.SSN = t.SSN
      AND maxDate(s.FromDate, t.FromDate) < minDate(s.ToDate, t.ToDate)
```

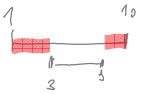


Temporal Difference Explained Example

Wednesday, October 23, 2019 8:42 AM

E

SSN	Role	Start	End
1	Teacher	1/1	1/10
1	HEAD	1/3	1/9
1	Teacher	1/11	1/12
1	Assistant	5/3	7/12



List the employees who are teachers but not head of departments

4 Cases (Some Cases can match different cases!)

1) ⇒ no h intersecting t

h intersect t if: max(t.start) < min(t.end)

```
SELECT t.ssn, t.start, t.end
FROM E t
WHERE t.role = "Teacher"
      AND NOT EXISTS (
            SELECT *
            FROM E h
            WHERE h.ssn = t.snn AND h.role = "HEAD"
                  AND h.end > t.start
                  AND h.start < t.end
```

⇔ $t.start < h.end$ and $h.start < t.end$

} only this because obviously $t.start < t.end$, $h.start < h.end$

2) t, h, result

= There is no h "covering" the start of t

⇒ we have to find an h such that $h.start > t.start$ and $h.start < t.end$

& check that no h_2 intersects $[t.start, h.start]$

⇔ $h_2.start < h.start$ or $t.start < h_2.end$

```
SELECT t.ssn, t.start, h.start
FROM E t, E h
WHERE h.ssn = t.ssn AND t.role = "Teacher" and h.role = "HEAD"
      AND h.start > t.start AND h.start < t.end
      AND NOT EXISTS (SELECT *
                      FROM E h2
                      WHERE h2.ssn = h.ssn AND h2.role = "HEAD"
                            AND h2.start < h.start
                            AND t.start <= h2.end)
```


3) t, result

There is no h covering the end of t

find h such that $t.start < h.end < t.end$

& h_2 that intersects $[h.end, t.end]$

⇔ $h_2.start < t.end$ and $h.end < h_2.end$

```
SELECT t.ssn, h.end, t.end
FROM E t, E h
WHERE h.ssn = t.ssn AND t.role = "Teacher" and h.role = "HEAD"
      AND h.end > t.start AND h.end < t.end
      AND NOT EXISTS (SELECT *
                      FROM E h2
                      WHERE h2.ssn = h.ssn AND h2.role = "HEAD"
                            AND h2.start <= t.end
                            AND h.end < h2.end)
```

4) t, h_1, h_2

There is a "Role" in t not covered by any h

find h_1 such that $t.start < h_1.end < t.end$

h_2 such that $h_2.end < h_2.start < t.end$

& and h_3 intersecting $h_4 = [h_2.end, h_2.start]$

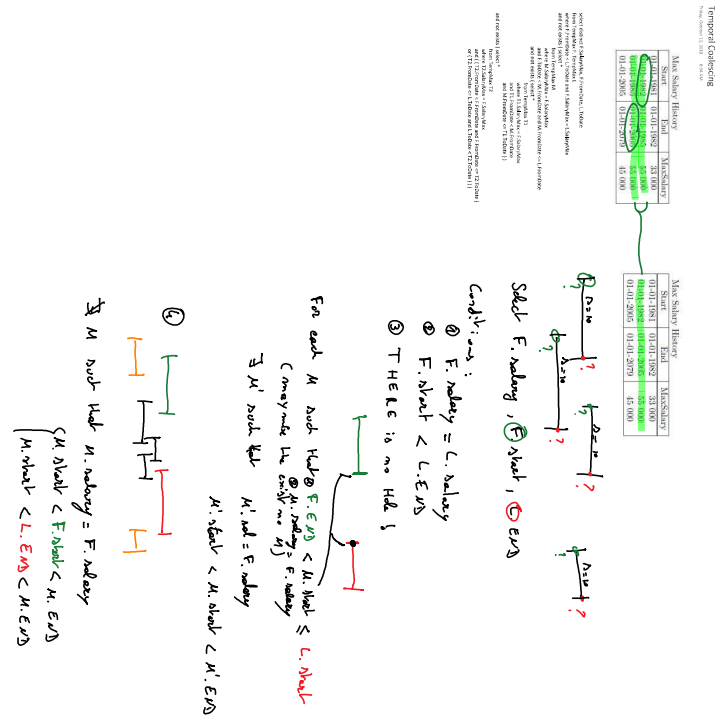
⇒ no h_3 such that: $h_2.end < h_3.end$, $h_3.start < h_2.start$

```
SELECT t.ssn, h1.end, h2.start
FROM E t, E h1, E h2
WHERE h1.ssn = t.ssn AND t.ssn = h2.ssn
      AND t.role = "Teacher" and h1.role = "HEAD" AND h2.role = "HEAD"
      AND t. start < h1.end  AND h1.end < h2.start AND h2.start < t.end
      AND NOT EXISTS (SELECT *
                      FROM E h3
                      WHERE h2.ssn = h3.ssn and h3.role = "HEAD"
                            AND h1.end < h3.end
                            AND h3.start < h2.start)
```

Give the history of the maximum salary

4 steps procedure

1. Find all temporal points of change
2. Build the intervals of constant value
3. Compute the aggregation on each interval
4. Coalesce the result

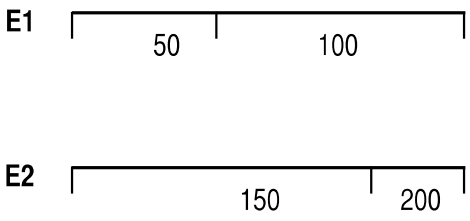


Exercice 10

Give the history of the maximum salary

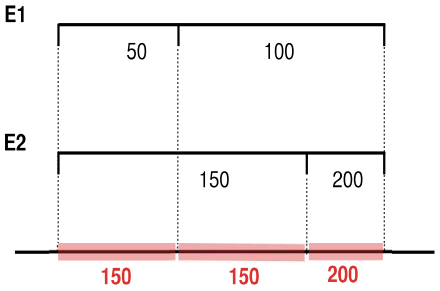
Give the history of the maximum salary

Example



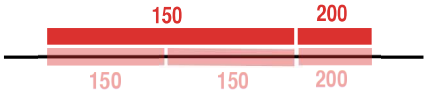
Step 3: Compute the aggregation on each interval

```
TempMax(SalaryMax, FromDate, ToDate) AS (
       select max(E.Salary), I.FromDate, I.ToDate
         from EmployeeSalary E,
              Intervals I
        where E.FromDate <= I.FromDate
          and I.ToDate <= E.ToDate
        group by I.FromDate, I.ToDate )
```



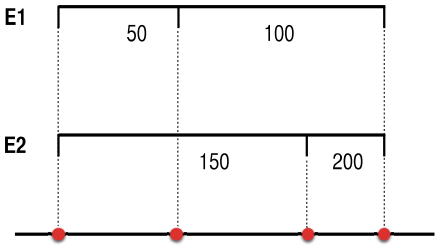
Step 4: Coalesce the result

```
select distinct F.SalaryMax, F.FromDate, L.ToDate
  from #TempMax F, #TempMax L
 where F.FromDate < L.ToDate and F.SalaryMax =  L.SalaryMax
   and not exists (
      select *
        from #TempMax M
       where M.SalaryMax =  F.SalaryMax
         and F.ToDate < M.FromDate and M.FromDate <= L.FromDate
         and not exists (
            select *
              from #TempMax T1
             where T1.SalaryMax =  F.SalaryMax
               and T1.FromDate<M.FromDate and M.FromDate<=T1.ToDate))
         and not exists (
            select *
              from #TempMax T2
             where T2.SalaryMax = F.SalaryMax
               and (T2.FromDate<F.FromDate and F.FromDate<=T2.ToDate)
                or (T2.FromDate<=L.ToDate and L.ToDate<T2.ToDate)))
         order by F.FromDate
```



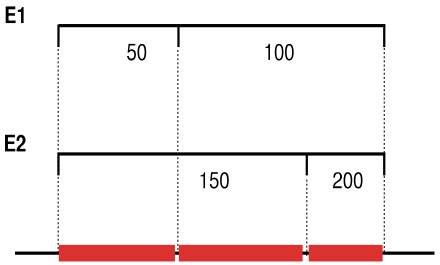
Step 1: Find all temporal points of change

```
Instants(Instant) AS (
        select distinct E.FromDate
          from EmployeeSalary E
        union
        select distinct E.ToDate
          from EmployeeSalary E ),
```



Step 2: Build the intervals of constant value

```
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 ) )
```

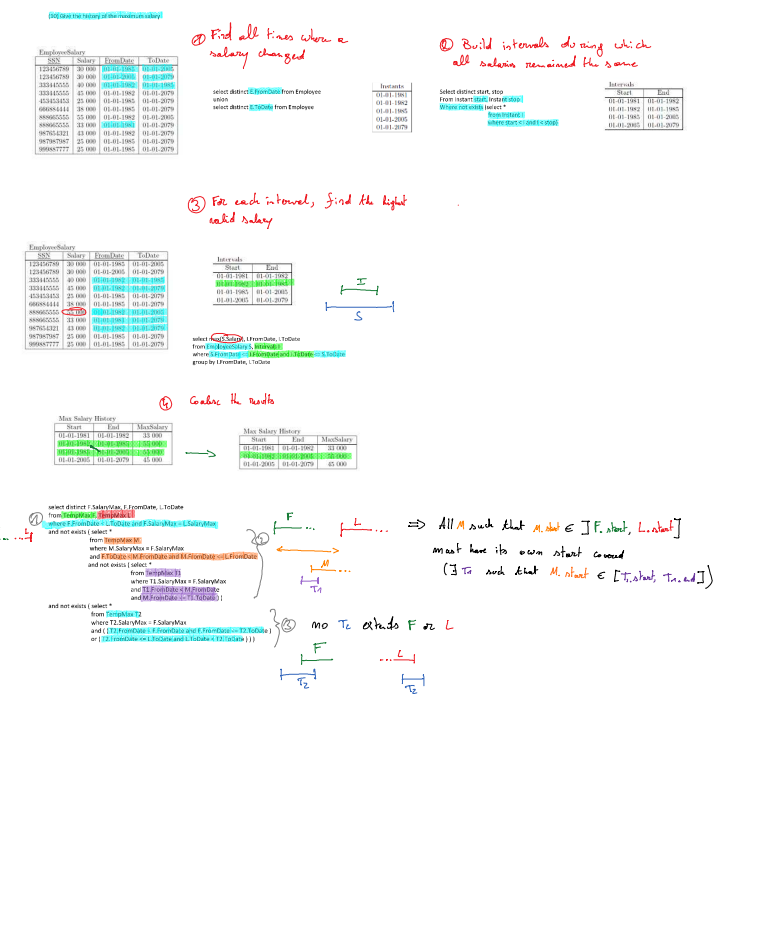


Give the history of the maximum salary

```
WITH
-- First step: Construct intervals during which no salary change occurred
    Instants(Instant) AS (
          select distinct E.FromDate from EmployeeSalary E
        union select distinct E.ToDate from EmployeeSalary E ),
    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 ) ),
-- Second step: Compute the maximum salary for these intervals
    TempMax(SalaryMax, FromDate, ToDate) AS (
        select max(E.Salary), I.FromDate, I.ToDate
          from EmployeeSalary E, Intervals I
          where E.FromDate <= I.FromDate and I.ToDate <= E.ToDate
          group by I.FromDate, I.ToDate )
-- Third step: Coalescing the above table
SELECT distinct F.SalaryMax, F.FromDate, L.ToDate
  from TempMax F, TempMax L
  where F.FromDate < L.ToDate and F.SalaryMax =  L.SalaryMax
    and not exists ( select *
                     from TempMax M
                     where M.SalaryMax =  F.SalaryMax
                       and F.ToDate < M.FromDate and M.FromDate <= L.FromDate
                       and not exists ( select *
                                        from TempMax T1
                                        where T1.SalaryMax =  F.SalaryMax
                                          and T1.FromDate < M.FromDate
                                          and M.FromDate <= T1.ToDate ) )
    and not exists ( select *
                     from TempMax T2
                     where T2.SalaryMax =  F.SalaryMax
                       and ( T2.FromDate < F.FromDate and F.FromDate <= T2.ToDate )
                       and ( T2.FromDate <= L.ToDate and L.ToDate < T2.ToDate ) )
order by F.FromDate
```

Aggregation

Monday, October 22, 2018 5:49 AM

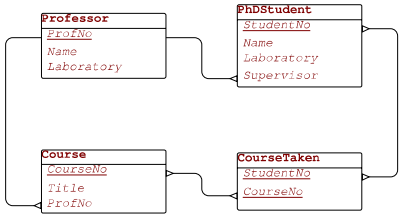


ADB Page 1

INFO-H-415 - Advanced Databases

Session 1 - Active Databases (1/3)

Consider the following database schema:



Define in SQL Server a set of triggers that ensure the following constraints:

Exercise 1. A PhD student must work in the same laboratory as his/her supervisor.

Exercise 2. A PhD student must take at least one course.

Exercise 3. A PhD student must take all courses taught by his/her supervisor.

Details of the database for the exercises

Table creation script

```
create database PhD;
GO
use PhD;

create table Professor (
  ProfNo char(5) not null,
  Name varchar(25) not null,
  Laboratory varchar(25) not null,
  constraint PK_Professor primary key (ProfNo)
)
create table PhDStudent (
  StudentNo char(9) not null,
  Name varchar(25) not null,
  Laboratory varchar(25),
  Supervisor char(5),
  constraint PK_PhDStudent primary key (StudentNo),
  constraint FK_PhDStudent_Professor
     foreign key (Supervisor) references Professor(ProfNo)
)
create table Course (
  CourseNo char(7) not null,
  Title varchar(30) not null,
  ProfNo char(5),
  constraint PK_Course primary key (CourseNo),
  constraint FK_Course_Professor
     foreign key (ProfNo) references Professor(ProfNo)
)
create table CourseTaken (
  StudentNo char(9) not null,
  CourseNo char(7) not null,
  constraint PK_CourseTaken primary key (StudentNo,CourseNo),
  constraint FK_CourseTaken_Student
     foreign key (StudentNo) references PhDStudent(StudentNo),
  constraint FK_CourseTaken_Course
     foreign key (CourseNo) references Course(CourseNo)
)
```

Initial data in the tables

Professor

ProfNo	Name	Laboratory
12345	John B. Smith	Databases
33344	Franklin T. Wong	Databases
99988	Alicia J. Zelaya	Networks
98765	Jennifer S. Wallace	Web Technologies
66688	Ramesh K. Narayan	Web Technologies

PhDStudent

StudentNo	Name	Laboratory	Supervisor
453453453	Joyce A. English	Databases	12345
987987987	Ahmad V. Jabbar	Databases	33344
888665555	James A. Borg	Web Technologies	66688

Course

CourseNo	Title	ProfNo
INFO364	Introduction to Databases	12345
INFO365	Advanced Databases	33344
INFO378	XML	66688
INFO379	Web Services	66688

CourseTaken

StudentNo	CourseNo
453453453	INFO364
453453453	INFO365
987987987	INFO364
987987987	INFO365
888665555	INFO378

Solutions for Session 1 - Active Databases (1/3)

➤ Solution to Exercise 1

Constraint

"A PhD student must work in the same laboratory as his/her supervisor."

Events that may violate the constraint

a) Insert into **PhDStudent**

b) Update of *Laboratory* or *Supervisor* in **PhDStudent**

c) Update of *Laboratory* in **Professor**

d) Delete from **Professor**

Actions

For each event, we have to choose whether to *abort* the transaction (by rolling back and generating a message) or to *repair* it (applying modifications that ensure that the constraint is satisfied). Note that for the latter, there are often several ways to ensure constraint satisfaction; choosing the way of applying changes that enforce the database's consistency therefore depends on the actual implementation context. In the following, we propose one of the often many possible ways of repairing the violated constraints.
For the present exercises, when applicable, a trigger creation statement for both scenarios – *aborting* or *repairing* – is proposed.

⚠ **CAUTION** Beware, however, that, for each constraint, only one single rule should be active at any given time. Thus, ensure that the first trigger does not exist or has been deactivated or deleted before testing the second one, and vice-versa.

Events a) and b)

1. *Aborting* the transaction

```
create trigger StudSameLabAsSuperv_PhDStud_InsUpd_Abort
on PhDStudent
after insert, update
as
if exists (
    select * from Inserted I, Professor P
     where P.ProfNo = I.Supervisor
       and P.Laboratory <> I.Laboratory )
begin
    raiserror ('Constraint Violation:
                A PhD student must work in the same
                laboratory as his/her supervisor', 1, 1)
    rollback
end
```


2. *Repairing* the transaction

```
create trigger StudSameLabAsSuperv_PhDStud_InsUpd_Repair
on PhDStudent
after insert, update
as
begin
    update PhDStudent
       set Laboratory = (
                select P.Laboratory
                  from Professor P
                 where P.ProfNo = Supervisor )
     where StudentNo in (
                select I.StudentNo
                  from Inserted I )
end
```

In this case, repairing by making the necessary updates in **PhDStudent** will probably be the better choice, as the value that *Laboratory* should take is univocally determined by the consistency rule.

Event c)

1. *Aborting* the transaction

```
create trigger StudSameLabAsSuperv_Prof_Upd_Abort
on Professor
after update
as
if exists (
    select * from Inserted I, PhDStudent S
     where I.ProfNo = S.Supervisor
       and I.Laboratory <> S.Laboratory )
begin
    raiserror ('Constraint Violation:
                A PhD student must work in the same
     laboratory as his/her supervisor', 1, 1)
    rollback
end
```

2. *Repairing* the violated constraints

```
create trigger StudSameLabAsSuperv_Prof_Upd_Repair
on Professor
after update
as
begin
    update PhDStudent
       set Laboratory = (
                select I.Laboratory
                  from Inserted I
                 where Supervisor = I.ProfNo )
     where Supervisor in (
                select I2.ProfNo
                  from Inserted I2 )
end
```

Again, *repairing* the violated constraints should be preferred in this case too. The assumption is that, when a professor leaves her lab for another one, her PhD students are following.

Event d)

In this case, there are several possibilities.

- The professor is deleted and the attributes *Laboratory* and *Supervisor* of the PhD students who worked for the deleted professor are set to `null`.
- The transaction is rolled back, preventing a professor to be deleted when there are PhD students associated to her. This is taken care of by the referential integrity.
- The professor is deleted and all PhD students associated with him are also deleted. This is taken care of by the referential integrity with the option `on update cascade`.

We here provide the trigger corresponding to the first of these cases.

```
alter table PhDStudent
drop constraint FK_PhDStudent_Professor
```

```
create trigger StudSameLabAsSuperv_Prof_Del_Repair
on Professor
after delete
as
begin
    update PhDStudent
       set Laboratory = null,
           Supervisor = null
     where Supervisor in (
                select ProfNo
                  from Deleted )
end
```

⚠ **CAUTION** As SQL Server does not implement the option `on delete set null` for the referential integrity, it is necessary to drop the foreign key constraint in the table **PhDStudent**.

➤ Solution to Exercise 2

Constraint

"A PhD student must take at least one course."

Contrarily to Exercise 1, this constraint is a "negative" one, in the sense that it prevents that there does not exist any PhD student that does not take any course, but it provides no information about a way of automatically determining the one course that should be taken "by default", should no other be provided. Consequently, it will not be possible without any additional assumption, to *repair* violated constraints. All violating events will thus simply result in aborting the transaction.

Events that may violate the constraint

a) Insert into **PhDStudent**

b) Update of *StudentNo* in **CourseTaken**

c) Delete from **CourseTaken**

d) Delete from **Course**

Actions

Event a)

```
create trigger PhDStudMinOneCourse_PhDStud_Ins_Abort
on PhDStudent
after insert
as
if exists (
   select * from Inserted I
    where not exists (
                select *
                  from CourseTaken
                 where StudentNo = I.StudentNo ) )
begin
   raiserror ('Constraint Violation:
   A PhD student must take at least one course', 1, 1)
   rollback
end
```

⚠ **CAUTION** This does not work in SQL Server, since a trigger is executed immediately after the triggering instruction. Thus, embedding several inserts (into **PhDStudent** and **CourseTaken**) into one transaction would not help.
Practically, thus, and without any further assumption, it will *not* be possible to ensure that this constraint verified, as the aborting trigger would prevent any insertion into the table **PhDStudent**.

Events b) and c)

```
create trigger PhDStudMinOneCourse_PhDStud_Ins_Abort
on CourseTaken
after update, delete
as
if exists (
    select * from Deleted D
     where D.StudentNo not in (
                select StudentNo
                  from CourseTaken ) )
begin
    raiserror ('Constraint Violation:
    A PhD student must take at least one course', 1, 1)
    rollback
end
```

Event d)

Removing an entry from **Course** could indirectly affect the number of courses taken by one or several PhD students. This case, however, should be handled with the `on update cascade` option of the referential integrity constraint on the *CourseNo* field of **CourseTaken**.

Solution to Exercise 3

Constraint

"A PhD student must take all courses taught by his/her supervisor."

Events that may violate the constraint

a) Insert into **PhDStudent**

b) Update of *Supervisor* in **PhDStudent**

c) Insert into **Course**

d) Update of *ProfNo* in **Course**

e) Update of *StudentNo* or *CourseNo* in **CourseTaken**

f) Delete from **CourseTaken**

Actions

Events a) and b)

1. *Aborting* the transaction

```
create trigger StudAllCoursesOfSuperv_Stud_InsUpd_Abort
on PhDStudent
after insert, update
as
if exists (
   select * from Inserted I
    where exists (
               select *
                 from Course C
                where C.ProfNo = I.Supervisor
                  and C.CourseNo not in (
                                    select T.CourseNo
                                      from CourseTaken T
                                     where T.StudentNo = I.StudentNo  ) ) )
begin
   raiserror ('Constraint Violation:
       A PhD student must take all the courses
       given by his supervisor', 1, 1)
   rollback
end
```

2. *Repairing* the transaction

```
create trigger StudAllCoursesOfSuperv_Stud_InsUpd_Repair
on PhDStudent
after insert, update
as
begin
    insert into CourseTaken (StudentNo, CourseNo)
    select I.StudentNo, C.CourseNo
      from Inserted I,
           Professor P,
           Course C
     where I.Supervisor = P.ProfNo
       and C.ProfNo = P.ProfNo
       and C.CourseNo not in (
                select T.CourseNo
                  from CoursTaken T
                 where T.StudentNo = I.StudentNo )
end
```

The rules implemented by this trigger can be challenged, for instance, with the following change of supervisor for the student named Joyce. The lab she belongs to will automatically be changed to "Web Technologies" by the trigger.

```
begin transaction
    update PhDStudent
       set Supervisor = 66688
     where StudentNo = 453453453
commit transaction
```

Events c) and d)

Aborting the transaction, particularly in SQL Server (where triggers are executed immediately after the triggering instruction), would not work (well).
The repairing rule being implicitly defined, or at least suggested, by the constraint (namely to automatically enrol the student in the added course), it will be the method of choice for this case.
Note that the `update` case, where a Professor would abandon a course, has not to be handled here explicitely, since letting a Student take courses from Professors that are not his supervisor is not forbidden.

```
create trigger StudAllCoursesOfSuperv_Course_InsUpd_Repair
on Course
after insert, update
as
begin
    insert into CourseTaken (StudentNo, CourseNo)
    select S.StudentNo, I.CourseNo
      from Inserted I,
           Professor P,
           PhDStudent S
     where C.ProfNo = P.ProfNo
       and S.Supervisor = P.ProfNo
       and I.CourseNo not in (
                select T.CourseNo
                  from CourseTaken T
                 where T.StudentNo = C.StudentNo )
end
```

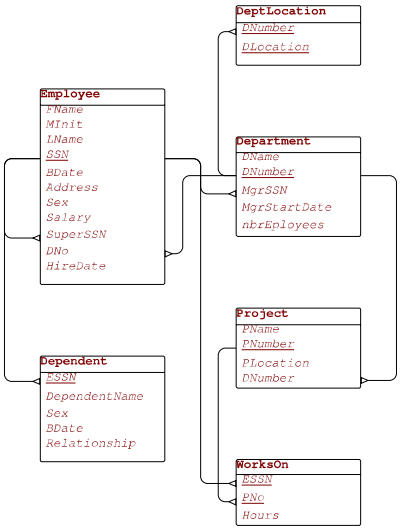
Events e) and f)

```
create trigger StudAllCoursesOfSuperv_CourseTaken_UpdDel_Abort
on CourseTaken
after update, delete
as
if exists (
    select *
      from Deleted D,
           Course C,
           PhDStudent S
     where D.CourseNo = C.CourseNo
       and C.ProfNo = S.Supervisor
       and D.StudentNo = S.StudentNo )
begin
   raiserror ('Constraint Violation:
       A PhD student must take all the courses
       given by his supervisor', 1, 1)
end
```

INFO-H-415 - Advanced Databases

Session 2+3 - Active Databases (2+3 of 3)

Consider the following database schema:



In SQL Server, enforce the following constraints using a set of `CHECK` constraints, referential integrity constraints, or triggers.

Exercise 1. The age of employees must be greater than 18.

Exercise 2. The supervisor of an employee must be older than the employee.

Exercise 3. The salary of an employee cannot be greater than the salary of his/her supervisor.

Exercise 4. The manager of a department must be an employee of that department.

Exercise 5. The location of a project must be one of the locations of its department.

Exercise 6. The hire date of employees must be greater than their birth date.

Exercise 7. A supervisor must be hired at least 1 year before every employee s/he supervises.

Exercise 8. The attribute Department.NbrEmployees is a derived attribute from Employee.DNo.

Exercise 9. An employee works at most in 4 projects.

Exercise 10. An employee works at least 30h/week and at most 50 h/week on all its projects.

Exercise 11. Among all employees working on a project, at most 2 can work for less than 10 hours.

Exercise 12. Only department managers can work less than 5 hours on a project.

Exercise 13. Employees that are not supervisors must work at least 10 hours on every project they work.

Exercise 14. The manager of a department must work at least 5 hours on all projects controlled by the department.

Exercise 15. The attribute Employee.SuperSSN is a derived attribute computed as follows. Department managers are supervised by the manager of Department 1 (Headquarters). Employees that are not managers are supervised by the manager of their department. Finally, the manager of Department 1 has a null value in attribute SuperSSN.

Exercise 16. The supervision relationship defined by Employee.SuperSSN must not be cyclic. (It is supposed that attribute Employee.SuperSSN is not derived as stated above.)

Details of the database for the exercises

Table creation script

```
create table Employee (
  FName varchar(15) not null,
  MInit char(1),
  LName varchar(15) not null,
  SSN char(9) not null,
  BDate smalldatetime null,
  Address varchar(30),
  Sex char(1),
  Salary decimal(18,2),
  SuperSSN char(9),
  DNo int not null,
  HireDate smalldatetime null,
  constraint PK_Employee primary key (SSN),
  constraint FK_Employee_Employee foreign key (SuperSSN) references Employee (SSN),
)
create table Department (
  DName varchar(15) not null,
  DNumber int not null,
  MgrSSN char(9) not null,
  MgrStartDate smalldatetime,
  nbrEmployees int,
  constraint PK_Department primary key (DNumber),
  constraint FK_Department_Employee foreign key (MgrSSN) references Employee (SSN)
    on delete cascade on update cascade
)
alter table Employee
  add constraint FK_Employee_Department foreign key (DNo) references Department (DNumber)
create table Project (
  PName varchar(15) not null,
  PNumber int not null,
  PLocation varchar(15),
  DNumber int not null,
  constraint PK_Project primary key (PNumber),
  constraint FK_Project_Department foreign key (DNumber) references Department (DNumber)
)
create table DeptLocations (
  DNumber int not null,
  DLocation varchar(15) not null,
  constraint PK_Dept_Locations primary key (DNumber,DLocation),
  constraint FK_Dept_Locations_Department foreign key (DNumber) references Department (DNumber)
)
create table Dependent (
  ESSN char(9) not null,
  DependentName varchar(15) not null,
  Sex char(1),
  BDate smalldatetime null,
  Relationship varchar(8),
  constraint PK_Dependent primary key (ESSN,DependentName),
  constraint FK_Dependent_Employee foreign key (ESSN) references Employee (SSN)
)
create table WorksOn (
  ESSN char(9) not null,
  PNo int not null,
  hours decimal(18,1) not null,
  constraint PK_WorksOn primary key (ESSN,PNo),
  constraint FK_WorksOn_Employee foreign key (ESSN) references Employee (SSN),
  constraint FK_WorksOn_Project foreign key (PNo) references Project (PNumber)
)
```


Initial data in the tables

Employee

FName	MInit	LName	SSN	BDate	Address	Sex	Salary	SuperSSN	DNo	HireDate
John	B	Smith	123456789	09-05-1955	731 Fondren, Houston, TX	M	30000	333445555	5	01-01-1985
Franklin	T	Wong	333445555	08-12-1945	638 Voss, Houston, TX	M	40000	888665555	5	01-01-1982
Alicia	J	Zelaya	999887777	19-07-1958	3321 Castle, Spring, TX	F	25000	987654321	4	01-01-1985
Jennifer	S	Wallace	987654321	20-06-1931	291 Berry, Bellaire, TX	F	43000	888665555	4	01-01-1982
Ramesh	K	Narayan	666884444	15-09-1952	975 Fire Oak, Humble, TX	M	38000	333445555	5	01-01-1985
Joyce	A	English	453453453	31-07-1962	5631 Rice, Houston, TX	F	25000	333445555	5	01-01-1985
Ahmad	V	Jabbar	987987987	29-03-1959	980 Dallas, Houston, TX	M	25000	987654321	4	01-01-1985
James	A	Borg	888665555	10-11-1927	450 Stone, Houston, TX	M	55000		1	01-01-1980

Department

DName	DNumber	MgrSSN	MgrStartDate	nbrEmployees
Research	5	333445555	22-05-1978	4
Administration	4	987654321	01-01-1985	3
Headquarters	1	888665555	19-06-1971	1

DeptLocations

DNumber	DLocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Project

PName	PNumber	PLocation	DNumber
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

WorksOn

ESSN	PNo	Hours
123456789	1	32.5
123456789	2	7.5
333445555	1	10
333445555	2	10
333445555	3	20
453453453	1	20
453453453	2	20
666884444	3	40
888665555	20	30.0
987654321	10	5.0
987654321	20	15.0
987654321	30	20.0
987987987	10	35.0
987987987	30	5.0
999887777	10	10.0
999887777	30	30.0

Dependent

ESSN	DependentName	Sex	BDate	Relationship
333445555	Alice	F	05-04-1976	Daughter
333445555	Theodore	M	25-10-1973	Son
333445555	Joy	F	03-05-1948	Spouse
987654321	Abner	M	29-02-1932	Spouse
123456789	Michael	M	01-01-1978	Son
123456789	Alice	F	31-12-1978	Daughter
123456789	Elizabeth	F	05-05-1957	Spouse

Solutions for Session 2+3 - Active Databases (2+3 of 3)

➤ Solution to Exercise 1

"The age of employees must be greater than 18."

Using a CHECK constraint

```
alter table Employee
add constraint employee_Age18
check ( dateadd(year,18,BDate) <= getdate() )
```

Using a trigger

```
create trigger age18
on Employee
after insert, update
as
if exists (
    select *
      from Inserted
     where dateadd(year,18,BDate) > getdate() )
begin
    raiserror('Constraint Violation: The age of an employee
        must be greater than 18', 1, 1)
    rollback
end
```

➤ Solution to Exercise 2

"The supervisor of an employee must be older than the employee"

Using a trigger

```
create trigger supervisorAge
on Employee
after insert, update
as
if exists (
    select *
      from Inserted I,
           Employee E
     where ( I.SuperSSN = E.SSN and I.BDate < E.BDate )
        or ( E.SuperSSN = I.SSN and E.BDate < I.BDate ) )
begin
    raiserror( 'Constraint Violation:
        The age of an employee must be less than
        the age of his/her supervisor', 1, 1)
    rollback
end
```

➤ Solution to Exercise 3

"The salary of an employee cannot be greater than the salary of his/her supervisor."

Using a trigger

```
create trigger supervisorSalary
on Employee
after insert, update
as
if exists (
    select *
      from Inserted I,
           Employee E
     where ( I.SuperSSN = E.SSN and I.Salary > E.Salary )
        or ( E.SuperSSN = I.SSN and E.Salary > I.Salary ) )
begin
    raiserror('Constraint Violation:
        The salary of an employee cannot be greater than
        the salary of his/her supervisor', 1, 1)
    rollback
end
```

➤ Solution to Exercise 4

"The manager of a department must be an employee of that department."

Using `UNIQUE` and foreign key constraints

```
alter table Employee
add constraint UN_Employee_SSN_DNo
unique( SSN, DNO )

alter table Department
add constraint FK_Employee_SSN_DNo
foreign key( MgrSSN, DNumber )
references Employee( SSN, DNo )
```

➤ Solution to Exercise 5

"The location of a project must be one of the locations of its department."

Using a foreign key constraint

```
alter table Project
add constraint FK_Project_DeptLocations
foreign key( DNumber, PLocation )
references DeptLocations( DNumber, DLocation )
```


➤ Solution to Exercise 6

"The hire date of employees must be greater than their birth date."

Using a `CHECK` key constraint

```
alter table Employee
add constraint HireDate_BDate
check( HireDate > BDate )
```

➤ Solution to Exercise 7

"A supervisor must be hired at least 1 year before every employee s/he supervises."

Using a trigger

```
create trigger hireSuperv
on Employee
after insert, update
as
if exists (
    select *
      from Inserted I,
           Employee E
     where ( I.SuperSSN = E.SSN and datediff(year, E.HireDate, I.HireDate) < 1
        or ( E.SuperSSN = I.SSN and datediff(year, I.HireDate, E.HireDate) < 1
) )
begin
    raiserror('Constraint Violation:
        A supervisor must be hired at least 1 year before
        every employee s/he supervises', 1, 1)
    rollback
end
```

➤ Solution to Exercise 8

"The attribute Department.NbrEmployees is a derived attribute from Employee.DNo"

Using value deriving triggers

```
create trigger DeptNbrEmp_Employee_InsUpdDel_Derive
on Employee
after insert, update, delete
as
begin
    update Department D
       set NbrEmployees = (
             select count(*)
               from Employee E
              where E.DNo = D.DNumber )
     where D.DNumber in (
             select distinct I.DNo
               from Inserted I )
        or D.DNumber in (
             select distinct D.DNo
               from Deleted D )
end
```

Incremental version

```
create trigger derived_Department_NbrEmployees_Employee
on Employee
after insert, update, delete
as
begin
    update Department
       set NbrEmployees = NbrEmployees +
             ( select count(*) from Inserted I where DNumber=I.DNo ) -
             ( select count(*) from Deleted D where DNumber=D.DNo )
     where DNumber in ( select DNo from Inserted )
        or DNumber in ( select DNo from Deleted )
end

create trigger derived_Department_NbrEmployees_Department
on Department
after insert, update
as
if exists ( select *
              from Inserted
             where NbrEmployees <> ( select count(*)
                                       from Employee E
                                      where E.DNo = DNumber ) )
begin
    raiserror('Constraint Violation:
        The attribute Department.NbrEmployees is a derived
        attribute from Employee.DNo', 1, 1)
    rollback
end
```

➤ Solution to Exercise 9

"An employee works at most in 4 projects"

Using a trigger

```
create trigger empNbrProj
on WorksOn
after insert, update
as
if exists (
    select *
      from WorksOn W
     group by W.ESSN
    having count(*) > 4 )
begin
    raiserror('Constraint Violation: An employee works at
        most in 4 projects', 1, 1)
    rollback
end
```


➤ Solution to Exercise 10

"An employee works at least 30h/week and at most 50 h/week on all its projects"

Using a trigger

```
create trigger workson_30_50
on WorksOn
after insert, update, delete
as
if exists ( select *
              from WorksOn
             group by ESSN
            having ( sum(Hours) < 30 )
                or ( sum(Hours) > 50 ) )
begin
    raiserror('Constraint Violation: An employee works at
        least 30 h/week and at most 50 h/week on all its projects', 1, 1)
    rollback
end
```

➤ Solution to Exercise 11

"A project can have at most 2 employees working on the project less than 10 hours"

Using a trigger

```
create trigger worksonLess10h
on WorksOn
after insert, update
as
if exists ( select *
              from WorksOn
             where Hours < 10
             group by PNo
            having count(*) > 2 )
begin
    raiserror('Constraint Violation: A project can have at
        most 2 employees working on the project less than 10 hours', 1, 1)
rollback end
```

➤ Solution to Exercise 12

"Only department managers can work less than 5 hours on a project"

Using a set of triggers

```
create trigger worksonLess5h_WorksOn
on WorksOn
after insert, update
as
if exists ( select *
              from Inserted
             where Hours < 5
```

```
               and ESSN not in (
                           select MgrSSN
                             from Department
                            where MgrSSN is not null ) )
begin
    raiserror('Constraint Violation: Only department managers
        can work less than 5 hours on a project', 1, 1)
    rollback
end

create trigger worksonLess5h_Department
on Department
after update, delete
as
if exists ( select *
              from Deleted
             where MgrSSN not in (
                           select MgrSSN
                             from Department )
               and MgrSSN in (
                           select ESSN
                           from WorksOn
                           where Hours < 5 ) )
begin
    raiserror('Constraint Violation: Only department managers
        can work less than 5 hours on a project', 1, 1)
    rollback
end
```

➤ Solution to Exercise 13

"Employees that are not supervisors must work at least 10 hours on every project they work"

Using a set of triggers

```
create trigger workson10h_WorksOn
on WorksOn
after insert, update
as
if exists ( select *
              from Inserted
             where Hours < 10
               and ESSN not in (
                           select SuperSSN
                             from Employee
                            where SuperSSN is not null ) )
begin
    raiserror('Constraint Violation: Employees that are not supervisors
        must work at least 10 hours on every project they work', 1, 1)
    rollback
end

create trigger workson10h_Employee
on Employee
after update, delete
as
if exists ( select *
```



```
              from Deleted
             where SuperSSN not in (
                           select SuperSSN
                             from Employee
                            where SuperSSN is not null )
               and SuperSSN in (
                           select ESSN
                             from WorksOn
                            where Hours < 10 ) )
begin
    raiserror('Constraint Violation: Employees that are not supervisors
        must work at least 10 hours on every project they work', 1, 1)
    rollback
end
```

Note about the second trigger. The logic behind that trigger is the following: A supervisor is a supervisor as long as there is at least one employee he supervises... When an employee is updated (changes the supervisor) or is deleted, there is a chance for a supervisor to loose his status and to become a "regular" employee again. When this happens, that former supervisor has to work at least 10 hours on every project.

Therefore, the query in the above trigger looks for an employee who had a supervisor (since it was a SuperSSN entry for at least one employee in the Deleted table), that is not supervisor anymore (since it is not in the set of current supervisors) and belongs to the set of employees that work less than 10 hours on a project.

➤ Solution to Exercise 14

"The manager of a department must work at least 5 hours on all projects controlled by the department."

Using a set of triggers

```
create trigger mgrProj_Department
on Department
after insert, update
as
if exists ( select *
              from ( Inserted I join Project P on I.DNumber = P.DNumber )
                   left outer join WorksOn on MgrSSN = ESSN and PNumber = PNo
             where Hours is null
                or Hours < 5 )
begin
    raiserror('Constraint Violation: A manager must work at least 5 hours
        on all projects controlled by his/her department', 1, 1)
    rollback
end
```

```
create trigger mgrProj_Project
on Project
after insert, update
as
if exists ( select *
              from ( Project P join Department D on D.DNumber = P.DNumber )
                   left outer join WorksOn on MgrSSN = ESSN and PNumber = PNo
             where P.PNumber in ( select PNumber
                                  from Inserted )
               and ( Hours is null
                     or Hours < 5 ) )
begin
    raiserror('Constraint Violation: A manager must work at least 5 hours
        on all projects controlled by his/her department', 1, 1)
    rollback
end

create trigger mgrProj_WorksOn
on WorksOn
after update, delete
as
if exists ( select *
              from ( Department D join Project P on D.DNumber=P.DNumber)
                   left outer join WorksOn on MgrSSN = ESSN and PNumber = PNo
             where D.MgrSSN in ( select ESSN
                                 from Deleted )
               and ( Hours is null
                     or Hours < 5 ) )
begin
    raiserror('Constraint Violation: A manager must work at least 5 hours
        on all projects controlled by his/her department', 1, 1)
    rollback
end
```


➤ Solution to Exercise 15

*"The attribute **Employee**.SuperSSN is a derived attribute computed as follows. Department managers are supervised by the manager of Department 1 (Headquarters). Employees that are not managers are supervised by the manager of their department. Finally, the manager of Department 1 has a NULL value in attribute SuperSSN."*

Using a set of triggers

```
create trigger derived_Employee_SuperSSN_Department
on Department
after insert, update
as
if update(MgrSSN)
begin
    update Employee
       set SuperSSN = (
                select case when SSN != D.MgrSSN
                                then D.MgrSSN
                           when SSN = D.MgrSSN and DNo != 1
                                then ( select MgrSSN
                                       from Department
                                       where DNumber = 1 )
                           else
                              null
                           end
                 from Department D
                where DNo = D.DNumber )
    -- if the department manager changes all employees of the department
    -- must be updated
     where ( DNo in (
                 select DNumber
                   from Inserted ) )
    -- if the manager of department 1 changes, all department managers
    -- must be updated
        or ( 1 in (
                 select DNumber
                 from Inserted )
         and SSN in (
                 select MgrSSN
                 from Department ) )
end

create trigger derived_Employee_SuperSSN_Employee
on Employee
after insert, update
as
if update(DNo)
begin
    update Employee
       set SuperSSN = (
                select case when SSN != MgrSSN
                                then D.MgrSSN
                           when SSN = MgrSSN and I.DNo != 1
                                then ( select MgrSSN
                                       from Department
                                       where DNumber = 1 )
                           else
                               null
                           end
```

```
                    from Inserted I,
                         Department D
                   where SSN = I.SSN
                     and I.DNo = D.DNumber )
    where SSN in (
                select SSN
                from Inserted )
end
```

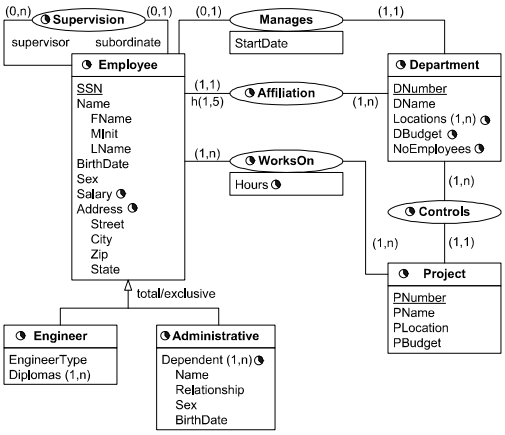
➤ Solution to Exercise 16

*"The supervision relationship in **Employee**.SuperSSN must not be cyclic"*

Using a trigger

```
create trigger noncyclic_subordinates
on Employee
after insert, update
as
begin
    create table #Supervision (
        SSN char(9),
        SuperSSN char(9)
        primary key (SSN,SuperSSN) )
    insert into #Supervision
        select SSN, SuperSSN
          from Employee
         where SuperSSN is not null
    while @@rowcount != 0 -- while previous operation affected some rows
    begin
        if exists ( select *
                      from #Supervision
                     where SSN = SuperSSN )
        begin
            raiserror('Constraint Violation: The supervision
                relationship is cyclic', 1, 1)
            rollback
        end
        insert into #Supervision
            select distinct S1.SSN, S2.SuperSSN
              from #Supervision S1 join #Supervision S2
                on S1.SuperSSN = S2.SSN
             where not exists (
                          select *
                            from #Supervision S
                           where S.SSN = S1.SSN
                             and S.SuperSSN = S2.SuperSSN )
    end
end
```


SQL Server
Temporal Databases: Constraints



Consider the above temporal conceptual schema.
Ensure the following integrity constraints.

1. An employee works in at most one department at any point in time.
2. At any point in time an employee cannot work more than once in a project.
3. The lifecycle of affiliation is included in the lifecycle of employee
4. The lifecycle of an employee is equal to the union of his/her affiliations.
5. Employees have a contiguous lifecycle.
6. The lifecycle of an employee is equal to the union of his/her affiliations, now taking into account that the lifecycle of employees is contiguous.

Constraints

1. An employee works in at most one department at any point in time.
   In other terms SSN is a sequenced primary key for Affiliation.

```
create trigger Seq_PK_Affiliation on Affiliation
  after insert, update  as
if exists ( select * from Inserted A1
  where 1 < ( select count(*) from Affiliation A2
    where A1.SSN = A2.SSN
    and A1.FromDate < A2.ToDate and A2.FromDate < A1.ToDate ) )
begin
  raiserror 13000
    'An employee works in at most one department at any point in time'
  rollback transaction
end
```

2. At any point in time an employee cannot work more than once in a project.
   In other terms (SSN,PNumber) is a sequenced primary key for WorksOn

```
create trigger Seq_PK_WorksOn on WorksOn
  after insert, update as
if exists ( select * from Inserted W1
  where 1 < ( select count(*) from WorksOn W2
    where W1.SSN = W2.SSN and W1.PNumber = W2.PNumber
    and W1.FromDate < W2.ToDate and W2.FromDate < W1.ToDate ) )
begin
  raiserror 13000
    'At any point in time an employee cannot work more than once in a project'
  rollback transaction
end
```

3. The lifecycle of affiliation is included in the lifecycle of employee.
   In the following triggers it is assumed that the table EmployeeLifecycle is coalesced.
   Therefore, every line in Affiliation must be covered by one line in EmployeeLifecycle.

```
create trigger Seq_FK_Affiliation_EmployeeLifecycle_1 on Affiliation
  after insert, update as
if exists ( select * from Inserted A
  where not exists ( select * from EmployeeLifecycle E
    where A.SSN = E.SSN
    and E.FromDate <= A.FromDate and A.ToDate <= E.ToDate ) )
begin
  raiserror 13000
    'The lifecycle of affiliation must be included in the lifecycle of employee'
  rollback transaction
end

create trigger Seq_FK_Affiliation_EmployeeLifecycle_2 on EmployeeLifecycle
  after update, delete as
if exists ( select * from Affiliation A
  where A.SSN IN ( select SSN from Deleted)
    and not exists ( select * from EmployeeLifecycle E
    where A.SSN = E.SSN
    and E.FromDate <= A.FromDate and A.ToDate <= E.ToDate ) )
begin
  raiserror 13000
    'The lifecycle of affiliation must be included in the lifecycle of employee'
  rollback transaction
end
```


4. The lifecycle of an employee is equal to the union of his/her affiliations. It is supposed that the previous trigger is activated, therefore it is sufficient to monitor that an employee must be affiliated to a department throughout his/her lifecycle.

```
create trigger Seq_FK_EmployeeLifecycle_Affiliation_1 on Affiliation
  after update, delete as
if exists ( select * from EmployeeLifecycle E
  where E.SSN in ( select SSN from Deleted )
  and not exists ( select * from Affiliation A
    where E.SSN = A.SSN
    and A.FromDate <= E.FromDate and E.FromDate < A.ToDate )
  or not exists ( select * from Affiliation A
    where E.SSN = A.SSN
    and A.FromDate < E.ToDate and E.ToDate <= A.ToDate )
  or exists ( select * from Affiliation A
    where E.SSN = A.SSN
    and E.FromDate < A.ToDate and A.ToDate < E.ToDate
    and not exists ( select * from Affiliation A2
      where A2.SSN = A.SSN
      and A2.FromDate <= A.ToDate and A.ToDate < A2.ToDate ) ) )
begin
  raiserror 13000
    'An employee must be affiliated to a department throughout his/her lifecycle'
  rollback transaction
end

create trigger Seq_FK_EmployeeLifecycle_Affiliation_2 on EmployeeLifecycle
  after insert, update as
if exists ( select * from Inserted E
  where not exists ( select * from Affiliation A
    where E.SSN = A.SSN
    and A.FromDate <= E.FromDate and E.FromDate < A.ToDate )
  or not exists ( select * from Affiliation A
    where E.SSN = A.SSN
    and A.FromDate < E.ToDate and E.ToDate <= A.ToDate )
  or exists ( select * from Affiliation A
    where E.SSN = A.SSN
    and E.FromDate < A.ToDate and A.ToDate < E.ToDate
    and not exists ( select * from Affiliation A2
      where A2.SSN = A.SSN
      and A2.FromDate <= A.ToDate and A.ToDate < A2.ToDate ) ) )
begin
  raiserror 13000
    'An employee must be affiliated to a department throughout his/her lifecycle'
  rollback transaction
end
```

5. Employees have a contiguous lifecycle.

```
alter table EmployeeLifecycle
drop constraint PK_EmployeeLifecycle
alter table EmployeeLifecycle
add constraint PK_EmployeeLifecycle primary key (SSN)
```

6. The lifecycle of an employee is equal to the union of his/her affiliations, now taking into account that the lifecycle of employees is contiguous.
   It is necessary to ensure that (1) the affiliations of an employee define a contiguous history, and (2) an employee must be affiliated to a department throughout his/her lifecycle.

The following trigger ensures that the affiliations of an employee define a contiguous history.

```
create trigger Contiguous_Hist_Affiliation on Affiliation
  after insert, update, delete as
if exists ( select * from Affiliation A1, Affiliation A2
  where A1.SSN = A2.SSN and A1.ToDate < A2.FromDate
  and not exists ( select * from Affiliation A3
     where A1.SSN = A3.SSN
     and ( ( A3.FromDate <= A1.ToDate and A1.ToDate < A3.ToDate )
        or ( A3.FromDate < A2.FromDate and A2.FromDate <= A3.ToDate ) ) ) )
begin
  raiserror 13000
    'The affiliations of an employee define a contiguous history'
  rollback transaction
end
```

The following two triggers replaces those of question (4).

```
alter trigger Seq_FK_EmployeeLifecycle_Affiliation_1 on Affiliation
  after update, delete as
if exists ( select * from EmployeeLifecycle E
  where E.SSN in ( select SSN from Deleted )
  and not exists ( select * from Affiliation A
    where E.SSN = A.SSN
    and A.FromDate <= E.FromDate and E.FromDate < A.ToDate )
  or not exists ( select * from Affiliation A
    where E.SSN = A.SSN
    and A.FromDate < E.ToDate and E.ToDate <= A.ToDate ) )
begin
  raiserror 13000
    'An employee must be affiliated to a department throughout his/her lifecycle'
  rollback transaction
end

alter trigger Seq_FK_EmployeeLifecycle_Affiliation_2 on EmployeeLifecycle
  after insert, update  as
if exists ( select * from Inserted E
  where not exists ( select * from Affiliation A
    where E.SSN = A.SSN
    and A.FromDate <= E.FromDate and E.FromDate < A.ToDate )
  or not exists ( select * from Affiliation A
    where E.SSN = A.SSN
    and A.FromDate < E.ToDate and E.ToDate <= A.ToDate ) )
begin
  raiserror 13000
    'An employee must be affiliated to a department throughout his/her lifecycle'
  rollback transaction
end
```


UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences Appliquées

January 21, 2020

INFO-H-415 – Advanced databases

First session examination

The examination is divided in three sections, which are graded by different persons. Please answer to these different sections on different pages.

Schema

Suppose a country that is divided in different regions which themselves are divided in communes. In this country, a smartphone application allows the rental of cars from a float of cars in free service. Citizens wanting to use these cars must register to the system. The history of the users' address should be kept. To be able to use a car in a given commune at a given instant, users must have a subscription for this commune valid at the moment of the rental. Users can have different subscriptions and one subsciption can give acces to more than one commune.

Here is an excerpt of the relational model used:

- **User** (UID, UserName, RegistrationTime)
- **UserAddress** (UID, StartDate, Address, Point, EndDate)
  - Point is a geometrical POINT. It is the location of user's address.
  - UID references User.UID
- **Subscription** (SID, UID, StartDate, EndDate)
  - UID references User.UID
- **CommuneSubscription** (SID, CommuneName)
  - SID references Subscription.SID
  - CommuneName references Commune.CommuneName
- **Country** (CountryName, Altitude, Geom)
  - Altitude is a PostGIS raster containing only one band
  - Geom is a geometrical MULTIPOLYGON
- **Region** (RegionName, Capital, Country)
  - Capital references Commune.CommuneName
  - Country references Country.CountryName
- **Commune** (CommuneName, Geom, RegionName)
  - Geom is a geometrical MULTIPOLYGON
  - RegionName references Region.RegionName
- **CarTrip** (CID, Start, UID, End, Itinerary)
  - Itinerary is a geometrical MULTILINE
  - UID references User.UID
  - CID references the CID of the concerned car (the corresponding table is not represented here).

"StartDate" and "EndDate" are of type Date, while "Start" and "End" are of type Datetime.

1 Temporal and Spatial Databases

For the following questions, suppose you are using a Microsoft SQL Server database. When you need spatial functions, assume the availability of spatial functions such as those provided by PostGIS, shown below.

(1) Give the period during which user with UID 1 was living in the commune called Ixelles but did not have any subscription valid for this commune.
(2) Give the history of the communes for which there was the most subscriptions. Do not coalesce the results.
(3) For each User, provide for each of the Commune the coalesced history for which the user had an active subscription in this Commune.
(4) Give for each region, the number of users which were living in this region during the complete $[01/01/2000, 01/01/2001]$ period. User that moved from on address to another (both address in the same region) should be counted. You should assume that the different addresses of any user form a continuous and uninterrupted lifecycle (if a user has a registred address at a time instant A and another or the same registered address at a later time instant B, then he also has a registered address for each time instant C such that $C \in [A, B]$). Furthermore, a user can not have two different addresses at the same time. We suppose you can compare time points to strings of the form `"dd/mm/yyyy"` using standard ordinal comparators ($>, <, =, \ldots$).
(5) List the different regions with their total area.
(6) For each commune, provide the distance between its centroid and the centroid of the capital of the region they are in.
(7) Give the car trip (CID and StartDate time) which has performed the longest segment in the commune of Ixelles.
(8) List for each trip the altitude of the lowest and highest point of the trip.

2 Active Databases

For the following questions, suppose you are using a Microsoft SQL Server database. When you need spatial functions, assume the availability of spatial functions such as those provided by PostGIS, shown below. If needed, you can create a trigger that can be applied on several tables.

(9) Ensure with a trigger that at any instant a user has a single address.
(10) Ensure with a trigger that at any instant a user has a single subscription to a commune.
(11) Ensure with a trigger that the communes in table Commune does not overlap.
(12) Ensure with a trigger that the geometry of a Country is equal to the union of its composing communes.

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 if these are points
ST_DumpPoints(geometry) Returns a set of all points that make up a geometry
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.

Advanced DB Exam 2020

January 23, 2023

1 Temporal and Spatial DB

1) Give the period during which user with UID 1 was living in the commune called Ixelles but did not have any subscription valid for this commune.

```
-- Case 4
SELECT UA.StartDate AS StartDate, UA.EndDate AS EndDate
FROM UserAddress UA, Commune C
WHERE UA.UID = 1 AND ST_Intersects(UA.Point, C.Geom) AND C.CommuneName = 'Ixelles'
  AND NOT EXISTS(
    SELECT *
    FROM Subscription S, CommuneSubscription CS
    WHERE S.UID = 1 AND CS.CommuneName = 'Ixelles' AND CS.SID = S.SID
      AND ((S.StartDate >= UA.StartDate AND S.StartDate < UA.EndDate)
        OR
        (S.EndDate > UA.StartDate AND S.EndDate <= UA.EndDate))

UNION
-- Case 1
SELECT UA.StartDate AS StartDate, S.StartDate AS EndDate
FROM UserAddress UA, Commune C, Subscription S, CommuneSubscription CS
WHERE UA.UID = 1 AND ST_Intersects(UA.Point, C.Geom) AND C.CommuneName = 'Ixelles'
  AND S.UID = 1 AND CS.CommuneName = 'Ixelles' AND CS.SID = S.SID
  AND UA.StartDate < S.StartDate AND S.StartDate < UA.EndDate
  AND NOT EXISTS (
    SELECT *
    FROM Subscription S2, CommuneSubscription CS2
    WHERE S2.UID = 1 AND CS.CommuneName = 'Ixelles' AND CS.SIS = S.SID
      AND UA.StartDate < S2.EndDate AND S2.FromDate < S.EndDate)

UNION
-- Case 2
SELECT S.EndDate AS StartDate, UA.EndDate AS EndDate
FROM UserAddress UA, Commune C, Subscription S, CommuneSubscription CS
WHERE UA.UID = 1 AND ST_Intersects(UA.Point, C.Geom) AND C.CommuneName = 'Ixelles'
  AND S.UID = 1 AND CS.CommuneName = 'Ixelles' AND CS.SID = S.SID
  AND UA.StartDate < S.EndDate AND S.EndDate < UA.EndDate
  AND NOT EXISTS (
    SELECT *
    FROM Subscription S2, CommuneSubscription CS2
    WHERE S2.UID = 1 AND CS.CommuneName = 'Ixelles' AND CS.SID = S.SID
      AND S.EndDate < S2.EndDate AND S2.FromDate < UA.EndDate)

UNION
--Case 3
SELECT S1.EndDate AS StartDate, S2.StartDate AS EndDate
FROM UserAddress UA, Commune C, Subscription S1, Subscription S2, Commune Subscription CS1,
    Commune Subscription CS2
WHERE UA.UID = 1 AND ST_Intersects(UA.Point, C.Geom) AND C.CommuneName = 'Ixelles'
  AND S1.UID = 1 AND CS1.CommuneName = 'Ixelles' AND CS1.SID = S1.SID
  AND S2.UID = 1 AND CS2.CommuneName = 'Ixelles' AND CS2.SID = S2.SID
  AND UA.StartDate < S1.EndDate AND S1.EndDate < UA.EndDate
  AND UA.StartDate < S2.StartDate AND S2.StartDate < UA.EndDate
  AND NOT EXISTS(
    SELECT *
    FROM Subscription S3, CommuneSubscription CS3
```

```
    WHERE S3.UID = 1 AND CS3.CommuneName = 'Ixelles' AND CS3.SID = S.SID
      AND S1.ToDate < S3.ToDate AND S3.FromDate < S2.FromDate)
```

2) Give the history of the communes for which there was the most subscriptions. Do not coalesce the results.

```
CREATE VIEW SubsChanges(Day) AS
  SELECT DISTINCT StartDate
  FROM Subscription

  UNION

  SELECT DISTINCT EndDate
  FROM Subscription

CREATE VIEW SubsPeriods(StartDate, EndDate) AS
  SELECT C1.Day, C2.Day
  FROM SubsChanges C1, SubsChanges C2
  WHERE C1.Day < C2.Day
    AND NOT EXISTS(
      SELECT *
      FROM SubsChanges C3
      WHERE C1.Day < C3.Day AND C3.Day < C2.Day)

CREATE VIEW TempTotal(CS.CommuneName, TotalSubs, StartDate, EndDate) AS
  SELECT CS.CommuneName, COUNT(DISTINCT UID), P.StartDate, P.EndDate
  FROM Subscription S, CommuneSubscription CS, SubsPeriods P
  WHERE S.SID = CS.SID
    AND S.StartDate <= P.StartDate AND P.EndDate <= S.EndDate
  GROUP BY CS.CommuneName, P.StartDate, P.EndDate

CREATE VIEW TempMax(MaxSubs, StartDate, EndDate) AS
  SELECT MAX(TotalSubs), StartDate, EndDate
  FROM TempTotal
  GROUP BY StartDate, EndDate

CREATE VIEW TempResult(CommuneName, MaxSubs, StartDate, EndDate) AS
  SELECT TT.CommuneName, TM.MaxSubs, StartDate, EndDate
  FROM TempTotal TT, TempMax TM
  WHERE TT.TotalSubs = TM.MaxSubs AND TT.StartDate = TM.StartDate AND TT.EndDate = TM.EndDate
```

3) For each user, provide for each of the commune the coalesced history for which the user had an active subscription in this commune.

```
CREATE VIEW TempJoin(UID, CommuneName, StartDate, EndDate) AS
  SELECT UA.UID, CS.CommuneName, UA.StartDate, UA.EndDate
  FROM UserAddress UA, Subscription S, CommuneSubscription CS
  WHERE S.UID = UA.UID AND S.SID = CS.SID
  AND S.StartDate < UA.StartDate AND UA.EndDate <= S.EndDate

  UNION ALL

  SELECT UA.UID, CS.CommuneName, UA.StartDate, S.EndDate
  FROM UserAddress UA, Subscription S, CommuneSubscription CS
  WHERE S.UID = UA.UID AND S.SID = CS.SID
    AND S.StartDate <= UA.StartDate AND UA.StartDate < S.EndDate AND UA.EndDate > S.EndDate

  UNION ALL

  SELECT UA.UID, CS.CommuneName, S.StartDate, UA.EndDate
  FROM UserAddress UA, Subscription S, CommuneSubscription CS
  WHERE S.UID = UA.UID AND S.SID = CS.SID
    AND UA.StartDate <= S.StartDate AND S.StartDate < UA.ToDate AND S.EndDate > UA.EndDate

  UNION ALL

  SELECT UA.UID, CS.CommuneName, S.StartDate, S.EndDate
  FROM UserAddress UA, Subscription S, CommuneSubscription CS
  WHERE S.UID = UA.UID AND S.SID = CS.SID
    AND UA.StartDate < S.StartDate AND S.EndDate <= UA.EndDate

SELECT DISTINCT F.UID, F.CommuneName, F.StartDate, L.EndDate
```



```
FROM TempJoin F, TempJoin L
WHERE F.StartDate < L.EndDate AND F.UID = L.UID AND F.CommuneName = L.CommuneName
  AND NOT EXISTS(
    SELECT *
    FROM TempJoin T
    WHERE T.UID = F.UID AND T.CommuneName = F.CommuneName
      AND F.StartDate < T.StartDate AND T.StartDate < L.EndDate
      AND NOT EXISTS(
        SELECT * FROM TempJoin as T1
        WHERE T1.UID = F.UID AND T1.CommuneName = F.CommuneName
          AND T1.StartDate < F.StartDate AND T.StartDate <= T1.EndDate ))
  AND NOT EXISTS(
    SELECT *
    FROM TempJoin T2
    WHERE T2.UID = F.UID AND T2.CommuneName = F.CommuneName
      AND (
        (T2.StartDate < F.StartDate AND F.StartDate <= T2.EndDate)
        OR
        (T2.StartDate >= L.EndDate AND L.EndDate < T2.EndDate)
        ))
```

4) Give for each region, the number of users which were living in this region during the complete [01/01/2000,01/01/2001] period. User that moved from one address to another (both addresses in the same region) should be counted. You should assume that the different addresses of any user form a continuous and uninterrupted lifecycle (if a user has a registered address at a time instant A and another or the same registered address at a later time instant B, then he also has a registered address for each time instant C such that $C \in [A, B]$). Furthermore, a user cannot have two different addresses at the same time. We suppose you can compare time points to strings of the form "dd/mm/yyyy" using standard orginal comparators ($>$,$<$,$=$,...).

```
CREATE VIEW Temp_User_Region(UID, RegionName, StartDate, EndDate) AS
  SELECT UA.UID, C.RegionName, UA.StartDate, UA.EndDate
  FROM UserAddress UA, Commune C
  WHERE ST_Intersects(UA.Point, C.Geom)

CREATE VIEW Temp_U_R_Coal(UID, RegionName, StartDate, EndDate) AS
  SELECT DISTINCT F.UID, F.CommuneName, F.StartDate, L.EndDate
  FROM Temp_User_Region F, Temp_User_Region L
  WHERE F.StartDate < L.EndDate AND F.UID = L.UID AND F.CommuneName = L.CommuneName
    AND NOT EXISTS(
      SELECT *
      FROM Temp_User_Region T
      WHERE T.UID = F.UID AND T.CommuneName = F.CommuneName
        AND F.StartDate < T.StartDate AND T.StartDate < L.EndDate
        AND NOT EXISTS(
          SELECT * FROM Temp_User_Region as T1
          WHERE T1.UID = F.UID AND T1.CommuneName = F.CommuneName
            AND T1.StartDate < F.StartDate AND T.StartDate <= T1.EndDate ))
    AND NOT EXISTS(
      SELECT *
      FROM Temp_User_Region T2
      WHERE T2.UID = F.UID AND T2.CommuneName = F.CommuneName
        AND (
          (T2.StartDate < F.StartDate AND F.StartDate <= T2.EndDate)
          OR
          (T2.StartDate >= L.EndDate AND L.EndDate < T2.EndDate)
          ))

SELECT RegionName, COUNT(UID)
FROM Temp_User_Region
WHERE StartDate <= '01/01/2000' AND '01/01/2001' <= EndDate
GROUP BY RegionName
```

5) List the different regions with their total area

```
SELECT RegionName, SUM(ST_AREA(Geom))
FROM Commune
GROUP BY RegionName
```


6) For each commune, provide the distance between its centroid and the centroid of the capital of the region they are in.

```
SELECT C.CommuneName, C.RegionName, ST_Distance(ST_Centroid(C.Geom), ST_Centroid(Cap.Geom)) AS
    DistToCapital
FROM Commune C, Region R, Commune Cap
WHERE C.RegionName = R.RegionName AND R.Capital = Cap.CommuneName
```

7) Give the car trip (CID and StartDate time) which has performed the longest segment in the commune of Ixelles.

```
SELECT CT.CID, CT.StartTime, ST_Length(ST_Intersection(ST.Itinerary, C.Geom) AS Length
FROM CarTrip CT, Commune C
WHERE C.CommuneName = 'Ixelles' AND ST_Intersects(CT.Itinerary, C.Geom)
ORDER BY ST_Length(ST_Intersection(ST.Itinerary, C.Geom)
LIMIT 1
```

8) List for each trip the altitude of the lowest and highest point of the trip.

```
SELECT CT.CID, (stats).max highest, (stats).min lowest
FROM (
  SELECT CT.CID, ST_SummaryStats(ST_Clip(C.altitude, 1, CT.Itinerary, TRUE)) AS stats
  FROM Country C JOIN CarTrip CT ON ST_Intersects(CT.Itinerary, C.altitude)) AS tmp
```

2 Active DB

9) Ensure with a trigger that at any instant a user has a single address

```
CREATE TRIGGER PK_User_Addr ON UserAddress FOR INSERT, UPDATE AS
IF EXISTS(
  SELECT *
  FROM UserAddress UA1
  WHERE 1 < (
    SELECT * COUNT(UA2.UID)
    FROM UserAddress UA2
    WHERE UA1.UID = UA2.UID
      AND UA1.StartDate < UA2.EndDate AND UA1.EndDate > UA2.StartDate)
BEGIN
  RAISERROR('A user can only have one address at a time',1,2)
  ROLLBACK TRANSACTION
END
```

10) Ensure with a trigger that at any instant a user has a single subscription to a commune.

```
-- If a user can only be subscribe to one commune at a time
CREATE TRIGGER PK_User_Subs ON Subscription FOR INSERT, UPDATE AS
IF EXISTS(
  SELECT *
  FROM Subscription S1
  WHERE 1 < (
    SELECT COUNT(S2.UID)
    FROM Subscription S2
    WHERE S1.UID = S2.UID
      AND S1.StartDate < S2.EndDate AND S1.EndDate > S2.StartDate)
BEGIN
  RAISERROR('A user can only have one subscription at a time',1,2)
  ROLLBACK TRANSACTION
END

-- If a user can be subscribed to several communes at a time, but only once at each
CREATE TRIGGER PK_User_Subs ON Subscription FOR INSERT, UPDATE AS
IF EXISTS(
  SELECT *
  FROM Subscription S1
  WHERE 1 < (
    SELECT COUNT(S2.UID)
    FROM Subscription S2
    WHERE S1.UID = S2.UID AND S1.SID = S2.SID
```



```
      AND S1.StartDate < S2.EndDate AND S1.EndDate > S2.StartDate)
BEGIN
  RAISERROR('A user can only have one subscription at a time',1,2)
  ROLLBACK TRANSACTION
END
```

11) Ensure with a trigger that the communes in table Commune do not overlap.

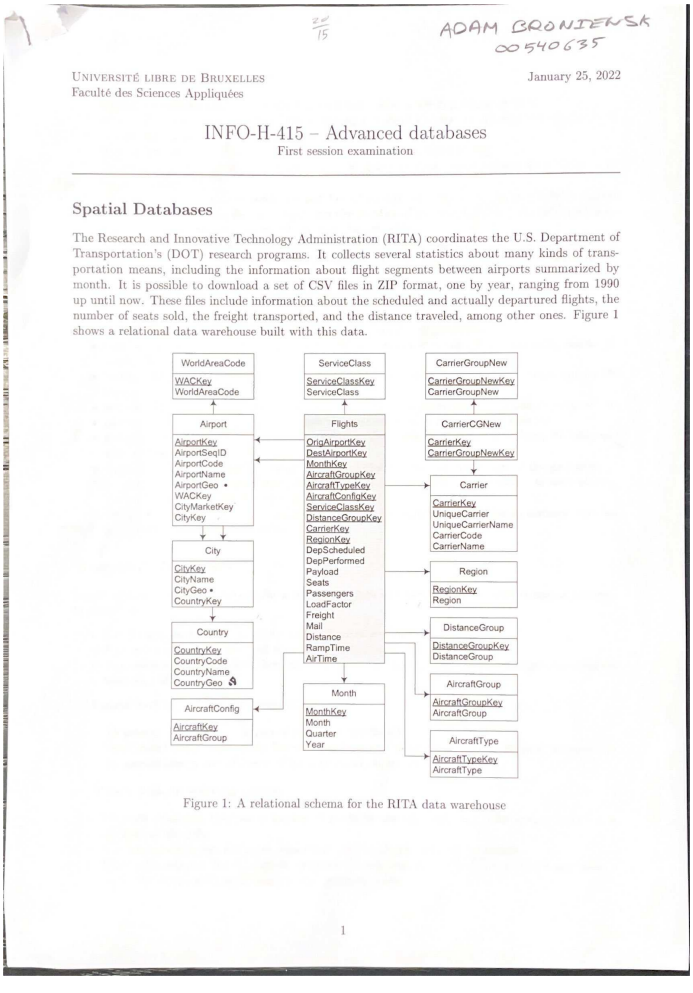
```
CREATE TRIGGER NonOverlap_Comm ON Commune FOR INSERT, UPDATE AS
IF EXISTS(
  SELECT *
  FROM Commune C, Inserted I
  WHERE C.CommuneName <> I.CommuneName
    AND ST_Intersects(ST_Interior(C.geom),ST_Interior(I.Geom))
BEGIN
  RAISERROR('Two communes cannot overlap',1,2)
  ROLLBACK TRANSACTION
END
```

12) Ensure with a trigger that the geometry of a Country is equal to the union of its composing communes.

```
CREATE TRIGGER CountryUnionComm ON Commune FOR INSERT, UPDATE AS
IF EXISTS(
  SELECT ST_DIFFERENCE(Com.Geom, C.Geom)
  FROM Country C
    JOIN Region R ON C.CountryName = R.Country
    JOIN Commune Com ON Com.RegionName = R.RegionName

  UNION

  SELECT ST_DIFFERENCE(C.Geom, ST_UNION(Com.Geom))
  FROM Country C
    JOIN Region R ON C.CountryName = R.Country
    JOIN Commune Com ON Com.RegionName = R.RegionName
  GROUP BY C.CountryName)
BEGIN
  RAISERROR('A country must be equal to the union of its composing communes',1,2)
  ROLLBACK TRANSACTION
END

CREATE TRIGGER CountryUnionComm2 ON Country FOR INSERT, UPDATE AS
IF EXISTS(
  SELECT ST_DIFFERENCE(Com.Geom, C.Geom)
  FROM Country C
    JOIN Region R ON C.CountryName = R.Country
    JOIN Commune Com ON Com.RegionName = R.RegionName

  UNION

  SELECT ST_DIFFERENCE(C.Geom, ST_UNION(Com.Geom))
  FROM Country C
    JOIN Region R ON C.CountryName = R.Country
    JOIN Commune Com ON Com.RegionName = R.RegionName
  GROUP BY C.CountryName)
BEGIN
  RAISERROR('A country must be equal to the union of its composing communes',1,2)
  ROLLBACK TRANSACTION
END
```

```
-- 1.1
-- Case 1: No intersection between UA and S
SELECT UD.StartDate, UD.EndDate
FROM UserAddress UA, Commune C
WHERE UA.UID=1 AND C.CommuneName='Ixelles' AND ST_Intersects(UA.Point, C.Geom)
AND NOT EXISTS (
   SELECT *
   FROM Subscription S, CommuneSubscription CS
   WHERE S.SID=CS.SID AND S.UID=1 AND CS.CommuneName='Ixelles'
   AND UA.StartDate < S.EndDate AND S.StartDate < UA.StartDate
)

UNION

-- Case 2: No R1 covering the start of UL
SELECT UA.StartDate, S.StartDate
FROM UserAdress UA, Commune C, Subscription S, CommuneSubscription CS
WHERE UA.UID=1 AND C.CommuneName='Ixelles' AND S.UID=1 AND
CS.CommuneName='Ixelles'
AND S.SID=CS.SID AND ST_Intersects(UA.Point, C.Geom)
AND UA.StartDate < S.StartDate AND S.StartDate < UA.EndDate
AND NOT EXISTS (
   SELECT *
   FROM Subscription S2, CommuneSubscription CS2
   WHERE S2.UID=1 AND CS2.CommuneName='Ixelles' AND S2.SID=CS2.SID
   AND UA.StartDate < S2.StartDate AND UA.StartDate < S2.EndDate
)

UNION

-- Case 3: No R1 covering the end of UL
SELECT S.EndDate, UA.EndDate
FROM UserAdress UA, Commune C, Subscription S, CommuneSubscription CS
WHERE UA.UID=1 AND C.CommuneName='Ixelles' AND S.UID=1 AND
CS.CommuneName='Ixelles'
AND S.SID=CS.SID AND ST_Intersects(UA.Point, C.Geom)
AND UA.StartDate < S.EndDate AND S.EndDate < UA.EndDate
AND NOT EXISTS (
   SELECT *
   FROM Subscription S2, CommuneSubscription CS2
   WHERE S2.UID=1 AND CS2.CommuneName='Ixelles' AND S2.SID=CS2.SID
   AND S2.StartDate < UA.EndDate AND S.EndDate < S2.EndDate
)

UNION

-- Case 4: There is a hole in UL not covered by any R1
SELECT S1.EndDate, S2.StartDate
FROM UserAdress UA, Commune C, Subscription S1, Subscription S1, CommuneSubscription CS1,
CommuneSubscription CS2
WHERE UA.UID=1 AND C.CommuneName='Ixelles'
AND S1.UID=1 AND CS1.CommuneName='Ixelles' AND S1.SID=CS1.SID
AND S2.UID=1 AND CS2.CommuneName='Ixelles' AND S2.SID=CS2.SID
AND ST_Intersects(UA.Point, C.Geom)
AND UA.StartDate < S1.EndDate AND S1.EndDate < S2.StartDate AND S2.StartDate < UA.End
AND NOT EXISTS (
   SELECT *
   FROM Subscription S3, CommuneSubscription CS3
   WHERE S3.UID=1 AND CS3.CommuneName='Ixelles'
   AND S1.EndDate < S3.EndDate AND S3.StartDate < S2.StartDate
)


-- 1.2
-- First step: Construct intervals during which no subscription change occured
WITH SubsChanges(Day) AS (
   SELECT DISTINCT StartDate FROM Subscription
   UNION
   SELECT DISTINCT EndDate FROM Subscription
),

SubsPeriods (StartDate, EndDate) AS (
   SELECT C1.Day, C2.Day
   FROM SubsChanges C1 , SubsChanges C2
   WHERE C1.Day < C2.Day
   AND NOT EXISTS (
      SELECT * FROM SubsChanges C3
      WHERE C1.Day < C3.Day AND C3.Day < C2.Day
   )
),

-- Second step: Compute the max subscription for these intervals
TempTotal (CommuneName, TotalSubs, StartDate, EndDate) AS (
   SELECT CS.CommuneName, COUNT (DISTINCT UID), P.StartDate, P.EndDate
   FROM Subscription S, CommuneSubscription CS, SubsPeriods P
   WHERE S.SID = CS.SID
   AND S.StartDate <= P.StartDate AND P.EndDate <= S.EndDate
   GROUP BY CS.CommuneName, P.StartDate, P.EndDate
)

SELECT CommuneName, TotalSubs, StartDate, EndDate
FROM TempTotal
WHERE MaxSubs IN (
   SELECT MAX(TotalSub)
   FROM TempTotal TT
   GROUP BY TT.StartDate, TT.EndDate
)


-- 1.3
WITH TempJoin(UID, CommuneName, StartDate, EndDate) AS (
   SELECT UA.UID, CS.CommuneName, maxDate(UA.StartDate, S.StartDate), minDate(UA.EndDate,
S.EndDate)
   FROM UserAddress UA, Subscription S, Commune Subscription CS
   WHERE UA.UID=S.UID AND S.SID=CS.ST_Distance
   AND maxDate(UA.StartDate, S.StartDate) < minDate(UA.EndDate, S.EndDate)
)

SEIECT DISTINCT F.UID, F.CommuneName, F.StartDate, L.EndDate
FROM TempJoin F, TempJoin L
WHERE F.UID = L.UID and F.CommuneName = L.CommuneName AND F.StartDate < L.EndDate
AND NOT EXISTS (
   SELECT * FROM TempJoin M
   WHERE M.UID = F.UID AND M.CommuneName = F.CommuneName
   AND F.ToDate < M.FromDate and M.FromDate <= L.FromDate
   AND NOT EXISTS (
      SELECT FROM TempJoin T1
      WHERE T1.UID = F.UID AND T1.CommuneName = F.CommuneName
      AND T1.StartDate < M.StartDate AND M.StartDate <= T1.EndDate
   )
)
AND NOT EXISTS(
   SELECT * FROM TempCount T2
   WHERE T2.UID = F.UID AND T2.CommuneName = F.CommuneName
   AND ((T2.StartDate < F.StartDate AND F.StartDate <= T2.EndDate)
   OR (T2.StartDate <= L.EndDate AND L.EndDate < T2.EndDate))
)


-- 1.4
WITH TempJoin(UID, RegionName, StartDate, EndDate) AS (
   SELECT UA.UID, C.RegionName, UA.StartDate, UA.EndDate
   FROM UserAddress UA , Commune C
   WHERE ST_Intersects(UA.Point, C.Geom)
),

TempCoalesced(UID, RegionName, StartDate, EndDate) AS (
   SELECT DISTINCT F.UID, F.CommuneName, F.StartDate, L.EndDate
   FROM TempJoin F, TempJoin L
   WHERE F.StartDate < L.EndDate AND F.UID=L.UID AND F.CommuneName=L.CommuneName
   AND NOT EXISTS (
      SELECT * FROM TempJoin M
      WHERE M.UID=F.UID AND M.CommuneName=F.CommuneName
      AND F.EndDate < T.StartDate AND M.StartDate < L.StartDate
      AND NOT EXISTS (
         SELECT * FROM TempJoin as T1
         WHERE T1.UID=F.UID AND T1.CommuneName=F.CommuneName
         AND T1.StartDate < M.StartDate AND M.StartDate <= T1.EndDate))
   AND NOT EXISTS (
      SELECT *
      FROM TempJoin T2
      WHERE T2.UID = F.UID AND T2.CommuneName = F.CommuneName
      AND ((T2.StartDate < F.StartDate AND F.StartDate <= T2.EndDate)
      OR (T2.StartDate <= L.EndDate AND L.EndDate < T2.EndDate )))
)

SELECT RegionName, COUNT (UID)
FROM TempCoalesced
WHERE StartDate <= '01-01-2000' AND '01-01-2001' <= EndDate
GROUP BY RegionName


-- 1.5
SELECT RegionName, SUM(ST_AREA(Geom))
FROM Commune
GROUP BY RegionName


-- 1.6
SELECT C.CommuneName, C.RegionName, ST_Distance(ST_Centroid(C.Geom), ST_Centroid(Cap.
Geom)) AS
DistToCapital
FROM Commune C, Region R, Commune Cap
WHERE C.RegionName = R.RegionName AND R.Capital = Cap.CommuneName
```

```
-- 1.7
SELECT CT.CID, Date(CT.Start), ST_Length(ST_Intersection(ST.Itinerary, C.Geom )
FROM CarTrip CT , Commune C
WHERE C.CommuneName='Ixelles' AND ST_Intersects(CT.Itinerary, C.Geom )
ORDER BY ST_Length(ST_Intersection(ST.Itinerary, C.Geom) DESC
LIMIT 1


-- 1.8
SELECT CT.CID, (stats).max highest, (stats).min lowest
FROM (
   SELECT CT.CID, ST_SummaryStats(ST_Clip(C.altitude, 1, CT.Itinerary, TRUE)) AS stats
   FROM Country C JOIN CarTrip CT ON ST_Intersects(CT.Itinerary, C.altitude)
) AS tmp


-- 2.1
CREATE TRIGGER PK_User_Addr
ON UserAddress
AFTER Insert, Update
AS
IF EXISTS (
   SELECT * FROM Inserted UA1
   WHERE 1 < (
      SELECT COUNT(*)
      FROM UserAddress UA2
      WHERE UA1.UID = UA2.UID
      AND UA1.StartDate < UA2.EndDate AND UA2.StartDate < UA1.EndDate )
)
BEGIN
   RAISERROR ( 'A user can only have one address at a time', 1, 1)
   ROLLBACK TRANSACTION
END


-- 2.2
CREATE TRIGGER PK_User_Subs
ON Subscription
AFTER Insert, Update
AS
IF EXISTS (
   SELECT *
   FROM Subscription S1
   WHERE 1 < (
      SELECT COUNT(*)
      FROM Subscription S2
      WHERE S1.UID = S2.UID
      AND S1.StartDate < S2.EndDate AND S2.StartDate < S1.EndDate)
)
BEGIN
   RAISERROR ('A user can only have one subscription at a time ', 1 , 1)
   ROLLBACK TRANSACTION
END


-- 2.3
CREATE TRIGGER NonOverlap_Comm ON Commune
AFTER Insert, Update
AS
IF EXISTS (
   SELECT *
   FROM Commune C, Inserted I
   WHERE C. CommuneName <> I. CommuneName
   AND ST_Intersects(C.geom, I.Geom))
BEGIN
   RAISERROR ('Two communes cannot overlap ' ,1 , 1)
   ROLLBACK TRANSACTION
END


-- 2.4
CREATE TRIGGER CountryUnionComm1
ON Commune
AFTER Insert, Update
AS
IF EXISTS (
   SELECT *
   FROM Country C
   JOIN Region R ON C. CountryName = R. Country
   JOIN Commune Com ON Com . RegionName = R. RegionName
   WHERE ST_DIFFERENCE (C.Geom, ST_UNION(Com.Geom))
   GROUP BY C. CountryName
   )
BEGIN
   RAISERROR ('A country must be equal to the union of its composing communes ', 1, 1)
   ROLLBACK TRANSACTION
END


CREATE TRIGGER CountryUnionComm2
ON Country
AFTER Insert, Update
AS
IF EXISTS (
   SELECT *
   FROM Country C
   JOIN Region R ON C. CountryName = R. Country
   JOIN Commune Com ON Com . RegionName = R. RegionName
   WHERE ST_DIFFERENCE (C.Geom, ST_UNION(Com.Geom))
   GROUP BY C. CountryName
   )
BEGIN
   RAISERROR ('A country must be equal to the union of its composing communes ', 1, 1)
   ROLLBACK TRANSACTION
END
```



ADAM GRONIENSK
00540635

UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences Appliquées

January 25, 2022

INFO-H-415 – Advanced databases

First session examination

Spatial Databases

The Research and Innovative Technology Administration (RITA) coordinates the U.S. Department of Transportation's (DOT) research programs. It collects several statistics about many kinds of transportation means, including the information about flight segments between airports summarized by month. It is possible to download a set of CSV files in ZIP format, one by year, ranging from 1990 up until now. These files include information about the scheduled and actually departured flights, the number of seats sold, the freight transported, and the distance traveled, among other ones. Figure 1 shows a relational data warehouse built with this data.

Figure 1: A relational schema for the RITA data warehouse

00540635

Write in SQL the following queries.

1. Display the spatial union of all airports with more than 5,000 departures in 2012.
2. Display for each city the three closest airports and their distance to the city independently of the country in which the city and the airport are located.
   N.B. A possible solution for this query uses the window functions in SQL.
3. Give the total number of persons arriving to or departing from airports closer than 15 km from the city center in 2012.
4. Give for 2012 the ratio between the number of persons arriving to or departing from aiports closer than 15 km from the city center and the number of persons arriving to or departing from airports located between 15 and 40 km from the city center.
5. For cities operated by more than one airport, give the total number of arriving and departing passengers at the airport closest to the city center, and the ratio between this value and the city total.

You will find below some of the common functions in PostGIS.

- float ST_Area(geometry): Returns the area of the surface of geometry.
- boolean ST_Within(geometry, geometry): Returns true if geometry A is completely inside geometry B.
- boolean ST_Contains(geometry, geometry): Returns true if geometry B is completely inside geometry A.
- boolean ST_Intersects(geometry, geometry): Returns true if geometry B intersects geometry A.
- geometry ST_Union(geometry): Returns the spatial union from a set of geometries.
- geometry ST_GeomFromText(string): Returns a specified geometry value from Well-Known Text representation (WKT).
- string ST_AsText(geometry): Returns the Well-Known Text representation of the geometry.
- geometry ST_Centroid(geometry): Returns the geometric center of a geometry, or equivalently, the center of mass of the geometry as a POINT.
- integer ST_Distance(geometry, geometry): Returns the minimum 2D Cartesian distance between two geometries in projected units.

Temporal Databases

Figure 2 shows a relational schema for a Northwind data warehouse extended with temporal features. In particular

- The lifespan employees, products, and categories are kept.
- The unit price of products and the description of categories are temporal.
- The supervision relationship between employees and the assignment of employees to cities are temporal relationships.

Enforce with triggers the following constraints.

1. All periods defining the lifespan of a product are disjoint.
2. The lifespan of the relationship between products and categories (ProductCategory) is covered by the intersection of the lifespans of the concerned categories.

Write in SQL the following queries.

3. For each employee, total sales amount of products she sold with unit price greater than $30 at the time of the sale.
4. For each product, list the name, unit price, and total sales amount by month.
5. Total sales amount for employees assigned to only one city. Notice that the assignment of employees to cities is represented in the Territories table.

00540635

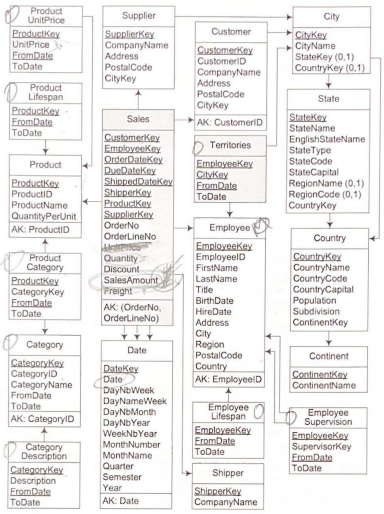


Figure 2: A relational schema for a temporal extension of the Northwind data warehouse

Graph Databases

Consider the relational schema of the Foodmart data warehouse given in Fig. 3 and its implementation in Neo4j given in Fig. 4. Write in Cypher the following queries.

1. All measures for stores in the states of California and Washington summarized at the state level.
2. All measures for the top-five store cities based on sales count.
3. Unit sales and number of customers by product subcategory.
4. Unit sales by customer city and percentage of the unit sales of the city with respect to its state.
5. Sales profit in 2017 by store type and store city, for cities whose unit sales in 2017 exceeded 25,000.

UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences Appliquées

January 24, 2023

INFO-H-415 – Advanced databases

First session examination

Consider a social platform where users can post messages. This platform has the particularity of allowing users to post messages with an expiration time, that is, the messages are not shown anymore after this time. To see the message of other users, a user should first subscribe to these users. Messages are stored on servers. Each server stores the messages posted in some regions during some particular time periods.

The social platform is based on the following relational model:

- **User** (ID, Name, Start, End)
  - Start is the moment where the user registers on the platform
  - End is the moment when the user unregisters
- **UserLocation** (UID, Start, point, End)
  - UID references User.ID
  - point is a geometrical POINT. It is the location of the user. To simplify, the platform will suppose that the user is located at "point" from Start to End.
- **UserSubscription** (SID, ID, Start, End)
  - SID and ID both reference User.ID. SID is the identification of the user who subscribes to the messages of the user identified with ID.
  - UID references User.ID
- **Message** (ID, text, UID, Start, End, topic)
  - UID references User.ID
  - Start is the moment where the message is posted
  - End is the expiry date of the message
- **Server** (ID, point)
  - point is a geometrical point. It is the location of the server.
- **ServerStore** (SID, RID)
  - RID references Region.ID.
  - SID references Server.ID
- **Region** (ID, region)
  - region is a geometrical multipolygon.

"Start" and "End" are always timestamps. A same server might store the messages of different regions (leading to several rows with the same SID in the ServerStore table).

1 Temporal and Spatial Databases

For the following questions, suppose you are using a Microsoft SQL Server database.

(1) Give the periods during which two users with ID "1" and "2" are located in the same region.
(2) 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.
(3) Give the number of subscriptions each user has. Do not coalesce the results.
(4) With table "R3(Name, Count, Start, End)" containing the result of the answer of Question 3, coalesce R3.
(5) 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.
(6) Give, for each user, the server on which is stored the last active message posted by the user.
(7) 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.
(8) Give, for each server, its distance with the centroid of all the regions for which it is hosting messages.

You can access the current time by using the function "getTime()". 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 if these are points
ST_DumpPoints(geometry) Returns a set of all points that make up a geometry
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.

2 Active Databases

(1) The location of users is only kept during the time they are registered on the platform.
(2) At each point in time a user has a single location.
(3) The location of a server is contained in its region.
(4) There are at most 3 servers located in a region.

```
-- 1.1
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 UserLocation UL1, UserLocation UL2, Region R
WHERE UL1.UID=1 AND UL2.UID=2
AND ST_Intersects(R.region, UL2.point)
AND ST_Intersects(R.region, UL2.point)
AND maxDate(UL1.Start, UL2.Start) < minDate(UL1.End, UL2.End)


-- 1.2
-- Case 1: No intersection between US and R1
SELECT US.Start, US.End
FROM UserSubscription US
WHERE US.SID=1 AND US.ID=2
AND NOT EXISTS (
    SELECT *
    FROM R1
    WHERE US.Start < R1.End AND R1.Start < US.End
)

UNION

-- Case 2: No R1 covering the start of US
SELECT US.Start, R11.Start
FROM UserSubscription US, R1 R11
WHERE US.SID=1 AND US.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

-- Case 3: No R1 covering the end of US
SELECT R11.Start, US.End
FROM UserSubscription US, R1 R11
WHERE US.SID=1 AND US.ID=2
AND US.Start < R11.End AND US.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 US not covered by any R1
SELECT R11.End, R12.Start
FROM UserSubscription US, R1 R11, R1 R12
WHERE US.SID=1 AND US.ID=2
AND US.Start < R11.End AND R11.End < R12.Start AND R12.Start < US.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
    )
)

-- Second step: Compute the number of subscriptions for these intervals
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


-- 1.4
-- Third step: Coalescing the results of 1.3
SELECT DISTINCT F.ID, F.Count, F.Start, L.End
FROM R3 F, R3 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))
)


-- 1.5
-- Join User and UserLocation
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

--
SELECT SID, MAX(mCount)
FROM (
    SELECT SID, COUNT(M.ID) mCount
    FROM ServerStore SS, UserLocation UL, Message M, Region R
    WHERE SS.RID = R.ID AND UL.UID = M.UID AND ST_Intersects(R.region, L.point)
    AND DATE(M.Start) <= '01-01-2023' AND DATE(M.End) >= '01-01-2023'
    GROUP BY SID
```

```
)
GROUP BY SID


-- 1.6
WITH tempt(UID, MID, SID, Start, End) AS (
SELECT UL.UID, M.ID, SS.SID, M.Start, M. End
FROM ServerStore SS, UserLocation UL, Message M, Region R
WHERE SS.RID = R.ID AND UL.UID = M.UID AND ST_Intersects(R.region, L.point)
)
SELECT t1.UID, t1.SID
FROM tempt t1
WHERE t1.End > getTime()
AND NOT EXIST (
   SELECT *
   FROM tempt t2
   WHERE t1.Start < t2.Start
)

-- 1.7
WITH R5_Distance(UID, SID, distance) AS (
   SELECT R5.UID, R5.SID, ST_Distance(UL.point, S.point)
   FROM R5, UserLocation UL, Server S, Server S
   WHERE R5.UID = UL.UID AND R5.SID = S.ID
)

SELECT UID, SID
FROM R5_Distance
WHERE distance = (
   SELECT max(distance)
   FROM R5_Distance t
)


-- 1.8
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


-- 2.1
CREATE TRIGGER UserLocTimeInsideUserTime_1
ON User
   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


-- 2.3
-- 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
AFTER Update
AS
IF EXISTS (
   SELECT * FROM UserLocation UL
   WHERE UL.UID IN (
      SELECT ID FROM Inserted
   )
   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


-- 2.2
CREATE TRIGGER UserSingleLocation
ON UserLocation
AFTER Insert, Update
AS
IF EXISTS (
   SELECT * FROM Inserted I

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


-- 2.4
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
```