INFOH415 - Advanced Databases

Jose Antonio Lorencio Abril

Fall 2022

Professor: Gilles Dejaegere

Student e-mail: jose.lorencio.abril@ulb.be

Contents

List of Figures

List of Tables

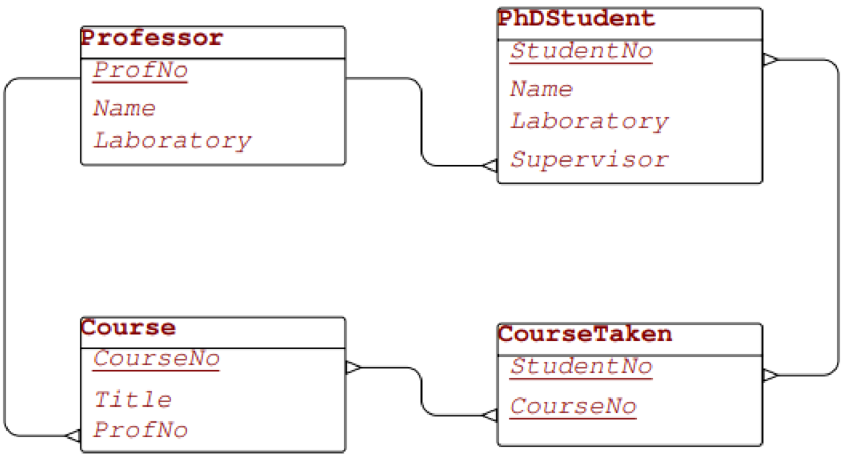
Listings

Part I

Active Databases

1 Session 1: Exercises 'PhD'

Consider the following database schema:



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

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

   This can be violated by:

   (a) Insert into PhDStudent.

   (b) Update of Laboratory or Supervisor in PhDStudent.

```sql
-- For a) and b)
-- Case Abort:
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

-- Case Repair:
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
```

   (c) Update of Laboratory in Professor.

```
-- Abort
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

-- Repair
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
```

(d) Delete from Professor: The professor is deleted and the attributes Laboratory and Supervisor of the PhD students who worked for the deleted professor are set to null.

```
alter table PhDStudent
drop constraint FK_PhDStudent_Professor -- Because SQL server r 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.

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
```

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

This can be violated by:

(a) Insert into PhDStudent

```
-- 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.

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
```

(b) Update of StudentNo in CourseTaken

(c) Delete from CourseTaken

```
-- For 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
```

(d) Delete from Course: 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.

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

   This can violated by:

   (a) Insert into PhDStudent

   (b) Update of Supervisor in PhDStudent

```
-- for a) and b)
-- Abort
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

-- Repair
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
```

   (c) Insert into Course

   (d) Update of ProfNo in Courseç

```sql
-- 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.

-- Repair
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
```

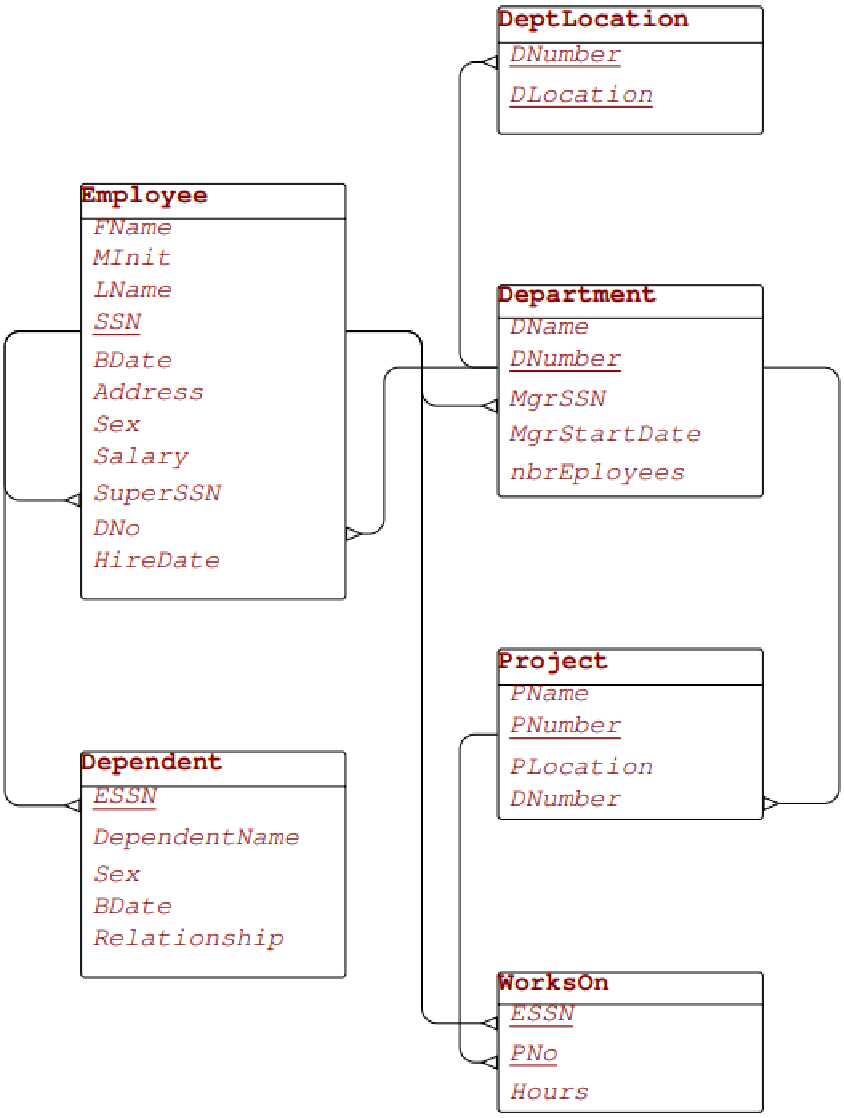
(e) Update of StudentNo or CourseNo in CourseTaken

(f) Delete from CourseTaken

```sql
-- 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
```

2 Sessions 2 and 3: Exercises 'Employees department projects'

We have the 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.

```
--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
```

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
```

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
```

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 )
```

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 )
```

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 )
```

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
```

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
```

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
```

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
```

Exercise 11. Among all employees working on a project, at most 2 can work for 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
```

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
```

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
```

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
```

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
```

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.)

```
-- 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
```

Part II
Graph databases

3 Session 4: Relational graphs

Google released, in 2002, a subset of the structure of the WWW. In this dataset, web pages are represented by graph nodes such that when a web page A contains a hyperlink to web page B, a directed edge is created from node A to node B.

In this activity, we will focus on the performance of different queries. Therefore, we will use three PostgreSQL tables, which are subsets of different sizes of the web structure released by Google:

- webgraph1 table: 605 nodes (web pages) and 1521 edges (hyperlinks)
- webgraph2 table: 1622 nodes (web pages) and 6288 edges (hyperlinks)
- webgraph3 table: 4122 nodes (web pages) and 14356 edges (hyperlinks)

Below, we show the list of different uses cases we want to analyze:

Exercise 3.1. For each pair of connected nodes, find the 1-hop paths. Include four columns in the resultset: fromNode, toNode, length, path, which correspond to the source node, target node, length of the path, and visited nodes, respectively. Exclude repeated nodes in the path. That is, if there is an edges A -> A, do not consider that A, A, 1, A-A is a valid 1-path from A to A.

```sql
SELECT fromNode, toNode, 1 as length, fromNode || '-' || toNode as Path
FROM webgraph1
WHERE fromNode <> toNode;
```

Exercise 3.2. For each pair of connected nodes, find the 2-hop paths. Include four columns in the resultset: fromNode, toNode, length, path, which correspond to the source node, target node, length of the path and the visited nodes, respectively. Exclude repeated nodes in the path. That is, if A -> B and B->A are edges in the graph, do not consider that A, A, 2, A-B-A is a valid 2- path from A to A.

```sql
SELECT w1.fromNode, w2.toNode, 2 as length, w1.fromNode || '-' || w1.toNode || '-' || w2.
    toNode as Path
FROM webgraph1 w1
JOIN webgraph1 w2 ON w1.toNode = w2.fromNode
WHERE w1.fromNode <> w1.toNode AND w1.fromNode <> w2.toNode AND w2.fromNode <> w2.toNode;
```

Exercise 3.3. For each pair of connected nodes, find the 3-hop paths. Include four columns in the resultset: fromNode, toNode, length, path, which correspond to the source node, target node, length of the path and the visited nodes, respectively. Exclude repeated nodes in the path. That is, if A -> B, A->C and B->A, are edges in the graph, do not consider that A, C, 3, A-B-A-C is a valid 3-path from A to C.

```sql
SELECT w1.fromNode, w2.toNode, 2 as length, w1.fromNode || '-' || w1.toNode || '-' || w2.
    toNode as Path
FROM webgraph1 w1
JOIN webgraph1 w2 ON w1.toNode = w2.fromNode
JOIN webgraph1 w3 ON w2.toNode = w3.fromNode
WHERE w1.fromNode <> w3.toNode AND w1.fromNode <> w2.toNode AND w2.fromNode <> w3.toNode;
```

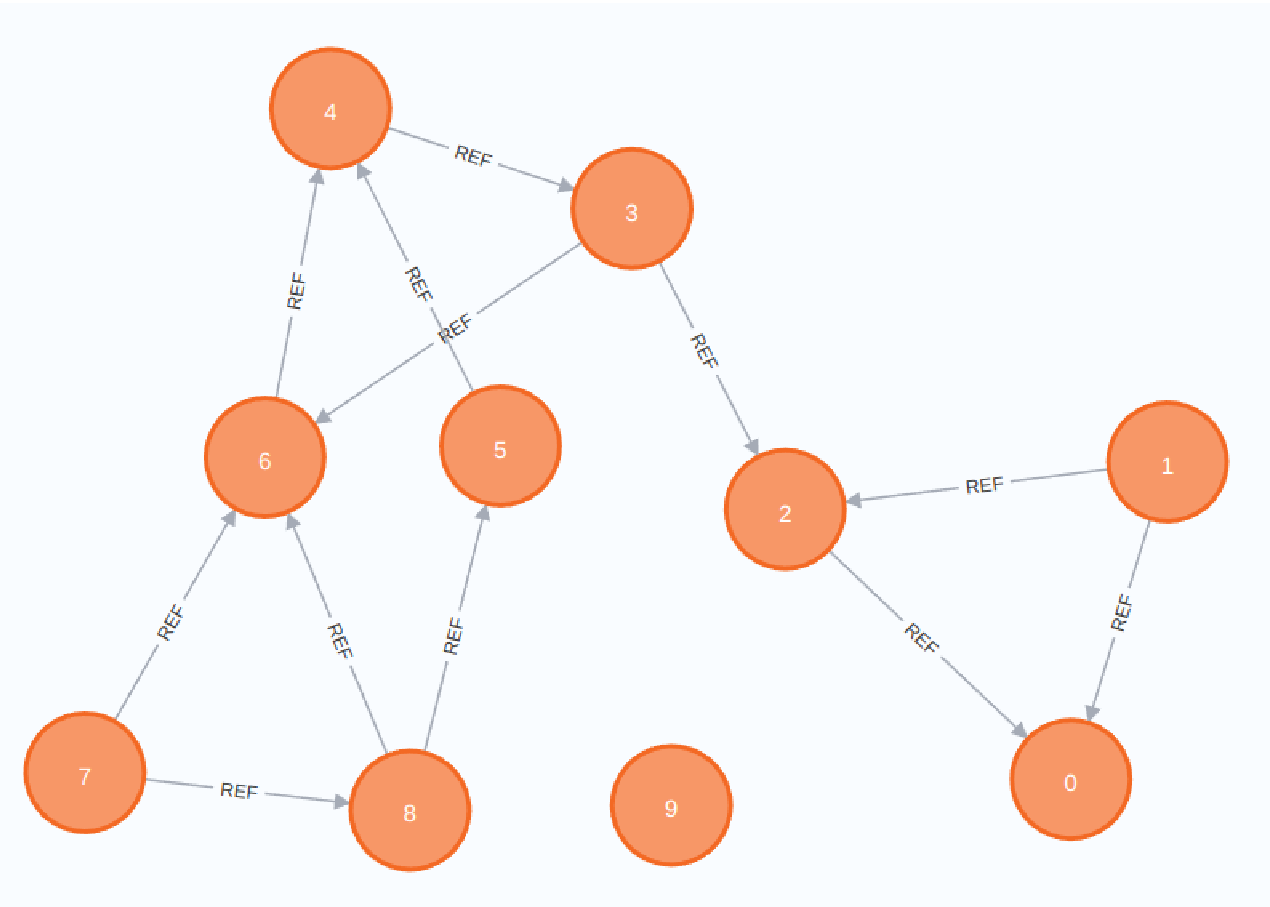
Exercise 3.4. for each pair of connected nodes, find the N-hop paths (the value of N is not known in advance). Include four columns in the result set: fromNode, toNode, length, path, which correspond to the source node, target node, length of the path and the visited nodes, respectively. Exclude repeated nodes in the path like in case “C”.

```
with recursive auxi(fromnode, tonode, long, path) as (
  select wg.fromnode, wg.tonode, 1, '-'|| wg.fromnode ||'-'|| wg.tonode
  from webgraph1 wg
  where wg.fromnode <> wg.tonode

  union

  select auxi.fromnode, wg.tonode, 1 + long , path ||'-'|| wg.tonode
  from auxi, webgraph1 wg
  where auxi.tonode = wg.fromnode
    AND wg.fromnode <> wg.tonode
    AND position('-'|| wg.tonode || '-' in auxi.path ) = 0)

select * from auxi
```

4 Sessions 5 and 6: Neo4j and Cypher

4.1 Session 5: Cypher Part I

We will express queries over graph databases using Cypher, the high-level query language for Neo4J. To be able to easily check the correctness of the results, we will start with a small graph representing a subset of the web structure, as shown below:



Exercise 4.1. Vertex Degree
For each vertex compute its in-degree and out-degree. The result set is a list of vertices and both values. Those vertices that do not have outgoing edges and/or incoming edges, must not appear in the answer. (Note that the result set is not a graph).

I have solved this in two equivalent ways:

```
--1--
MATCH (p:URL)
WHERE size((:URL)-->(p))+size((p)-->(:URL))>0
RETURN p.name as url, size((:URL)-->(p)) as in_degree, size((p)-->(:URL)) as out_degree

--2--
MATCH (p:URL)
WITH p, size((:URL)-->(p)) AS in_degree, size((p)-->(:URL)) AS out_degree
WHERE in_degree+out_degree>0
RETURN p.name as url, in_degree, out_degree
```

Exercise 4.2. Vertex Degree variation

For each vertex calculate its in-degree and out-degree. The result set is a list of all vertices in the graph, together with the two values above, for each vertex. Those vertices that do not have outgoing edges and/or incoming edges, must appear in the answer with value “0”.

The way I solve the first exercise makes this question trivial:

```
MATCH (p:URL)
RETURN p.name as url, size((:URL)-->(p)) as in_degree, size((p)-->(:URL)) as out_degree

-- Another
MATCH (n)
OPTIONAL MATCH ()-[r1]->(n)
OPTIONAL MATCH (n)-[r2]->()
RETURN n.name, COUNT(distinct r1) as indegree, COUNT (distinct r2) as outdegree
ORDER BY n.name
```

Exercise 4.3. Calculating a maximum value

Find the maximum vertex in-degree. (the result set is not a graph).

I have solved this also in two different ways. I prefer the second solution, because it matches all nodes with maximum in-degree and is independent of the order done by the order by.

```
--1--
MATCH (p:URL)
RETURN p.name as url, size((:URL)-->(p)) as in_degree
ORDER BY in_degree DESC LIMIT 1

--2--
MATCH (p:URL)
WITH size((:URL)-->(p)) as in_degree
WITH max(in_degree) as mx
MATCH (q:URL)
WHERE size((:URL)-->(q)) = mx
RETURN q.name as url, mx

--3--
MATCH (n)
OPTIONAL MATCH (src)-->(n)
WITH n, COUNT(distinct src) as indegree
RETURN MAX(indegree)
```

Exercise 4.4. Find influencial nodes

Find the subgraph which contains nodes whose in-degree is maximal in the graph (you should obtain only one node). Do the same for the out-degree (you should obtain four nodes).

For the in-degree:

```
MATCH (n:URL)
WITH size((:URL)-->(n)) as in_degree, n
WITH max(in_degree) as mx
MATCH (n2:URL)
WITH n2, size(()-->(n2)) as in_degree
WHERE in_degree = mx
RETURN n2.name
```

And for the out-degree:

```
MATCH (n:URL)
WITH size((n)-->(:URL)) as out_degree, n
WITH max(out_degree) as mx
MATCH (n2:URL)
WITH n2, size((n2)-->(:URL)) as out_degree
WHERE out_degree = mx
RETURN n2.name
```

Exercise 4.5. Distance between nodes
For each pair of vertices, calculate the distance, i.e. the shortest simple path between them (without repeated edges in the path). Do not show the distance between two disconnected nodes (infinite distance). Exclude paths when source and target are the same node.

Although there is a specific function for this in Neo4j[1], we can manually do this:

```
MATCH path=(a:URL)-[*]->(b:URL)
WHERE a.name <> b.name
RETURN a.name, b.name, min(length(path)) as dist
```

Exercise 4.6. Distance between nodes using Cypher function
Solve the query in Use Case 5, but using the shortestPath built-in Cypher function. It has one parameter that represents a pattern path and returns the shortest path that matches this pattern. If there exists more than one shortest path, it returns any of them.

This one is an easier version of the previous exercise. Also, we see another way of computing the length of a path as *size(relationships(path))*.

```
MATCH (a:URL), (b:URL), sp=shortestpath((a)-[*]->(b))
WHERE a.name <> b.name
RETURN a.name, b.name, size(relationships(sp)) as dist

-- Another
MATCH path= shortestPath((n)-[*]->(p) )
WHERE n <> p
WITH nodes(path) as listanodos
RETURN [n in listanodos|n.name]
```

Exercise 4.7. Diameter
Compute the diameter of the graph, i.e. the longest distance between two nodes in the graph (excluding disconnected pairs of nodes).

This one's easy:

```
MATCH path=(:URL)-[*]->(:URL)
RETURN max(length(path)) as diameter

-- Another
MATCH p=(n1)-[*1..]->(n2)
WHERE n1 <> n2
WITH n1, n2, MIN(length(p)) as distance
RETURN MAX(distance)

-- Another
MATCH p=(n1)-[*1..]->(n2)
WHERE n1 <> n2
WITH n1.name, n2.name, MIN(length(p)) as distance
RETURN distance
ORDER BY distance DESC
LIMIT 1
```

[1] See next exercise.

Exercise 4.8. webgraph3

Repeat use cases 1 to 7 using the webgraph3 database, which represents the same information as the corresponding relational database in Activity 1 (Session 4).

We just have to execute the same queries in the webgraph3 database.

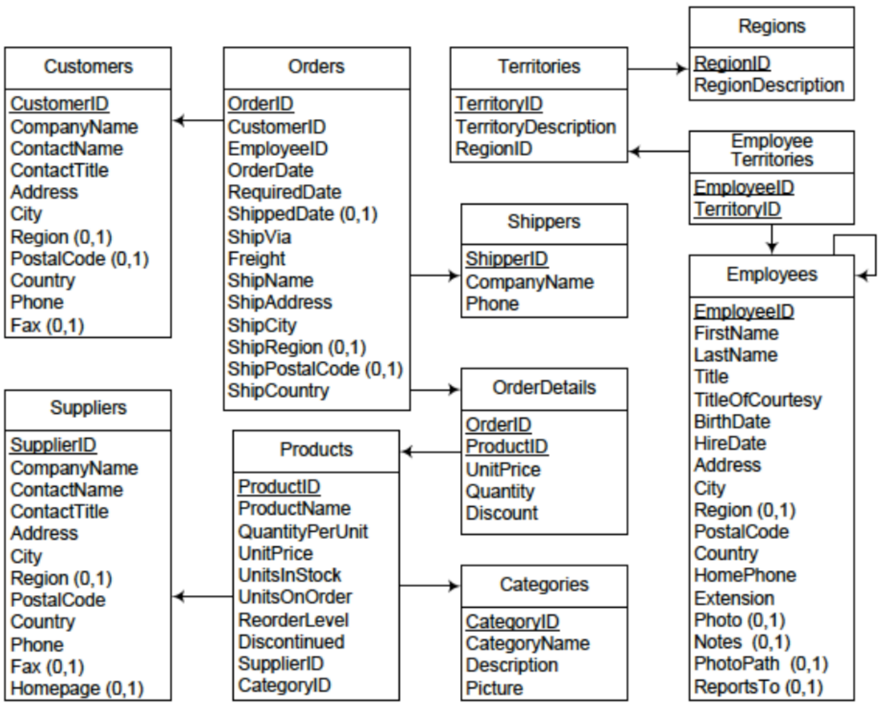
Exercise 4.9. Paths

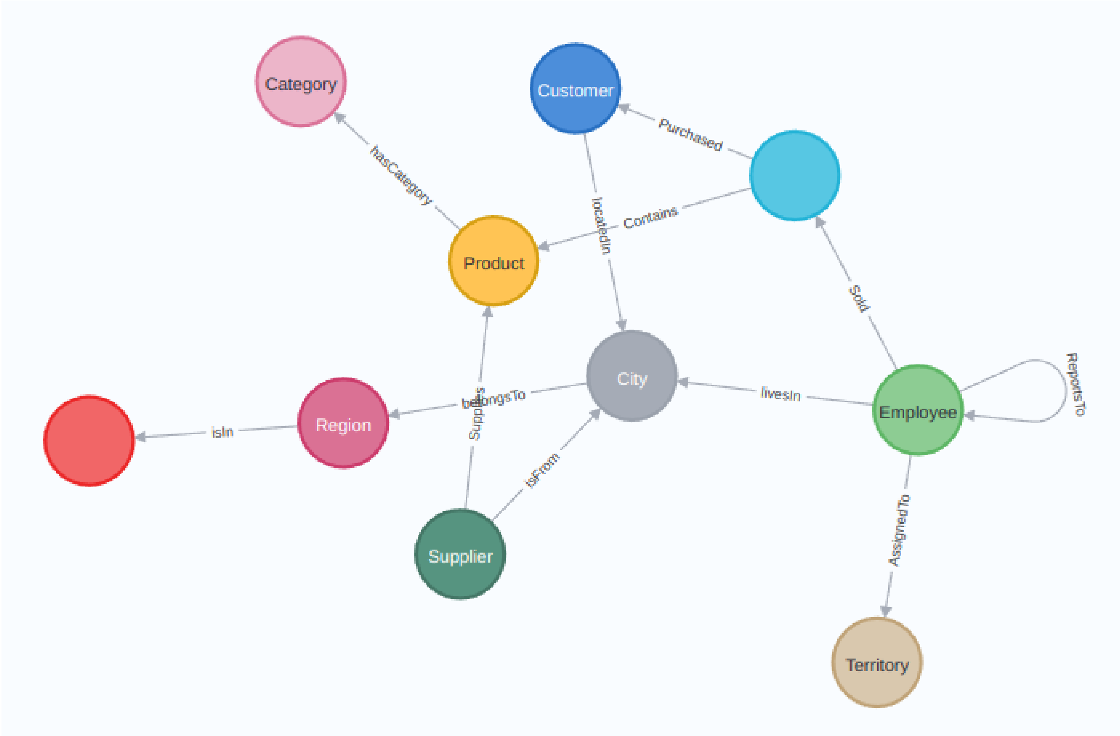
Compute all the 1, 2, 3, and n-hops in the graph, and compare against the results obtained using PostgreSQL in Activity 1. Note: start with "limit X", increasing "X" to prevent that the algorithm runs indefinitely.

```
MATCH path = (end)<-[*1..]-(start)
WHERE ALL (n in nodes(path) where 1 = size([m in nodes(path) where m=n]))
RETURN start.name, LENGTH(path) AS length, [p in NODES(path) | p.name], end.name
ORDER BY length
MATCH path = (end)<-[*1..]-(start:URL{name:745315})
WHERE ALL (n in nodes(path) where
1 = size([m in nodes(path) where m=n]))
RETURN LENGTH(path) AS length, [p in NODES(path) | p.name]
ORDER BY length desc
```

4.2 Session 6: Cypher Part II

Exercise 4.10. Consider the Northwind database, whose schema is:





Write in Cypher the following queries over the northwindhg.db database:

1. List products and their unit price.

```
MATCH(v:Product)
RETURN v.productID, v.unitPrice
```

2. List information about products 'Chocolade' & 'Pavlova'.

```
MATCH (p:Product)
WHERE p.productName IN ['Chocolade','Pavlova']
RETURN p
```

3. List information about products with names starting with a "C", whose unit price is greater than 50.

```
MATCH (p:Product)
WHERE p.productName STARTS WITH "C" AND tofloat(p.unitPrice) > 50
RETURN p.productName, p.unitPrice
```

4. Same as 3, but considering the sales price, not the product's price.

```
MATCH (p:Product) <- [c:Contains] - (o:Order)
WHERE p.productName STARTS WITH "C" AND tofloat(c.unitPrice) > 50
RETURN distinct p.productName, p.unitPrice, c.unitPrice
```

5. Total amount purchased by customer and product.

```
MATCH (c:Customer)
OPTIONAL MATCH (p:Product)<-[pu:Contains]-(:Order)-[:Purchased]->(c)
RETURN c.customerName, p.productName,tofloat(sum(pu.unitPrice) * pu.quantity) as volume
ORDER BY volume desc
```

6. Top ten employees, considering the number of orders sold.

```
MATCH (:Order)<-[:Sold]-(e:Employee)
RETURN e.firstName,e.lastName, count(*) AS Ordenes
ORDER BY Ordenes DESC LIMIT 10
```

7. For each employee, list the assigned territories.

```
MATCH (t:Territory)<-[:AssignedTo]-(e:Employee)
RETURN e.lastName, COLLECT(t.name)
```

8. For each city, list the companies settled in that city.

```
MATCH (c:City)<-[:locatedIn]-(c1:Customer)
RETURN c.cityname, COLLECT(c1.customerName)
```

9. How many persons an employee reports to, either directly or transitively?

```
MATCH (report:Employee)
OPTIONAL MATCH (e)<-[rel:ReportsTo*]-(report)
RETURN report.lastName AS e1, COUNT(rel) AS reports
```

10. To whom do persons called "Robert" report to?

```
MATCH (e:Employee)<-[:ReportsTo*]-(sub:Employee)
WHERE sub.firstName = 'Robert'
RETURN e.firstName,e.lastName,sub.lastName
```

11. Who does not report to anybody?

```
MATCH (e:Employee)
WHERE NOT (e)-[:ReportsTo]->()
RETURN e.firstName as TopBossFirst, e.lastName as TopBossLast
```

12. Suppliers, number of categories they supply, and a list of such categories

```
MATCH (s:Supplier)-->(:Product)-->(c:Category)
WITH s.supplierName as Supplier, collect(distinct c.categoryName) as Categories
WITH Supplier, Categories, size(Categories) AS Cantidad ORDER BY Cantidad DESC
RETURN Supplier, Cantidad, Categories
```

13. Suppliers who supply beverages

```
MATCH (c:Category {categoryName:"Beverages"})<--(:Product)<--(s:Supplier)
RETURN DISTINCT s.supplierName as ProduceSuppliers
```

14. Customer who purchases the largest amount of beverages

```
MATCH (cust:Customer)<-[:Purchased]-(:Order)-[o:Contains]->(p:Product), (p)-[:hasCategory
    ]->
(c:Category{categoryName:"Beverages"})
RETURN cust.customerName as CustomerName, SUM(o.quantity)
LIMIT 1
```

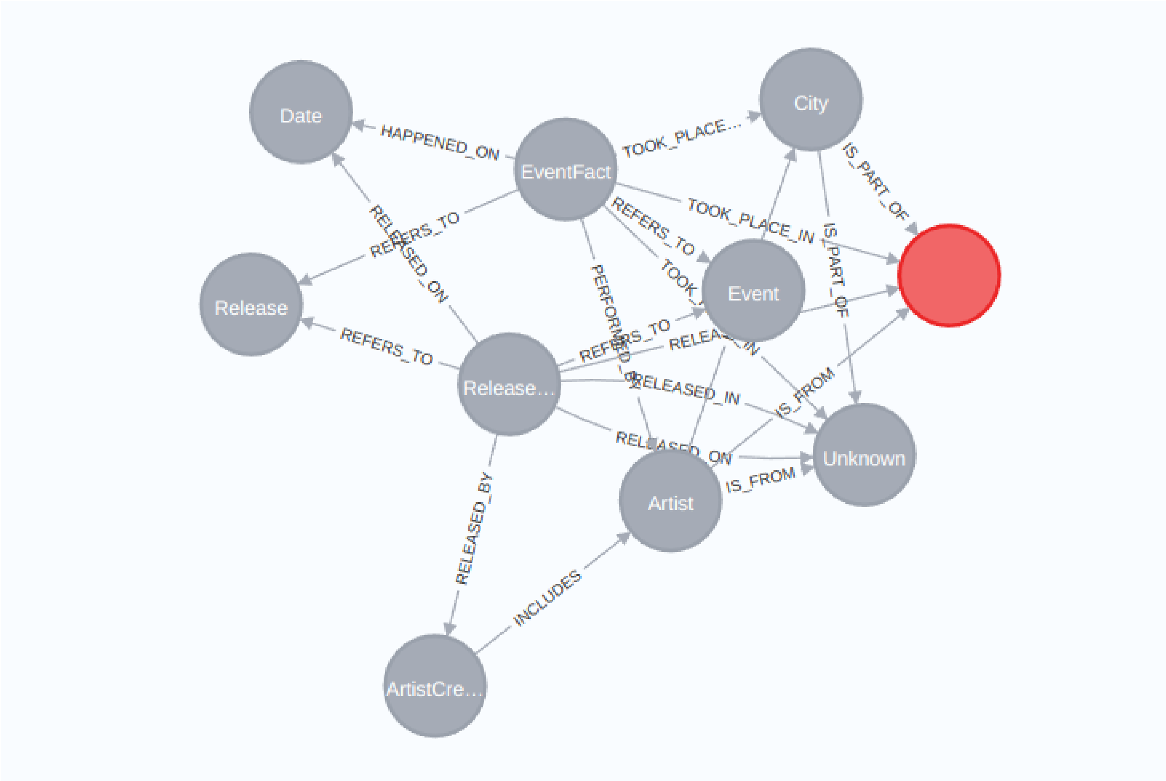
15. List the five most popular products (considering number of orders)

```
MATCH (c:Customer)<-[:Purchased]-(o:Order)-[o1:Contains]->(p:Product)
RETURN c.customerName, p.productName, count(o1) as orders
ORDER BY orders desc LIMIT 5
```

16. Products ordered by customers from the same country than their suppliers

```
MATCH (c:Customer) -[r:locatedIn]->(cy:City)-[:belongsTo]->(:Region)-[:isIn]->(co:Country
    )
WITH co, c MATCH (s:Supplier) WHERE co.countryname = s.country
WITH s, co, c MATCH(s)-[su:Supplies]-(p:Product)<-[:Contains]-(o:Order)-[:Purchased]->(c)
RETURN c.customerName,s.supplierName,co.countryname,p.productName
```

Exercise 4.11. Switch to the MusicBrainz database, doing the same steps as in Assignment 2. Now, the database is musicbrainz. The schema is:



1. Compute the total number of releases per artist.

```
MATCH (r:ReleaseFact)-[rb:RELEASED_BY]->(ac:ArtistCredit)-[inc:INCLUDES]->(a:Artist)
RETURN a.name, COUNT(r)
```

2. Compute the total number of releases per artist and per year.

```
MATCH (d:Date)<-[ro:RELEASED_ON]-(r:ReleaseFact)-[rb:RELEASED_BY]->(ac:ArtistCredit)-[inc:INCLUDES]->(a:Artist)
RETURN a.name, d.year, COUNT(r)
ORDER BY a.name, d.year
```

3. Compute the total number of events per artist.

```
MATCH (e:Event)<-[rt:REFERS_TO]-(ef:EventFact)-[pb:PERFORMED_BY]->(a:Artist)
RETURN a.name, COUNT(e)
```

4. Compute the number of times the artist performed in each event.

```
MATCH (e:Event)<-[rt:REFERS_TO]-(ef:EventFact)-[pb:PERFORMED_BY]->(a:Artist)
RETURN a.name, e.name, COUNT(ef)
```

5. For each (event, artist, year) triple, compute the number of times the artist performed in an event on an year.

```
MATCH (e:Event)<-[rt:REFERS_TO]-(ef:EventFact)-[pb:PERFORMED_BY]->(a:Artist)
MATCH (d:Date)<-[ho:HAPPENED_ON]-(ef)
RETURN a.name, e.name, d.year, COUNT(ef)
```

6. Same as Query 5, for artists in the United Kingdom and events happened after year 2006.

```
MATCH (d:Date)<-[ho:HAPPENED_ON]-(ef)
WHERE d.year > 2006
MATCH (e:Event)<-[rt:REFERS_TO]-(ef:EventFact)-[pb:PERFORMED_BY]->(a:Artist)-[isf:IS_FROM
    ]->(c:Country)
WHERE c.name = 'United Kingdom'
RETURN a.name, e.name, d.year, COUNT(ef)
```

7. Compute the number of releases, per language, in the UK.

```
MATCH (r:Release)<-[rt:REFERS_TO]-(rf:ReleaseFact)-[ri:RELEASED_IN]->(c:Country)
WHERE c.name='United Kingdom'
RETURN r.language, count(r)
```

8. Compute, for each pair of artists, the number of times they have performed together at least twice in an event.

```
MATCH (a1:Artist)<-[pb1:PERFORMED_BY]-(ef:EventFact)-[pb2:PERFORMED_BY]->(a2:Artist)
WHERE a1.name < a2.name
MATCH (e:Event)<-[rt:REFERS_TO]-(ef)
WITH a1.name as artist1, a2.name as artist2, COUNT(e) as times
WHERE times>=2
RETURN artist1, artist2, times
```

9. Compute the triples of artists, and the number of times they have performed together in an event, if this number is at least 3.

```
MATCH (a1:Artist)<-[pb1:PERFORMED_BY]-(ef:EventFact)-[pb2:PERFORMED_BY]->(a2:Artist)
MATCH (ef)-[pb3:PERFORMED_BY]->(a3:Artist)
WHERE a1.name < a2.name < a3.name
MATCH (e:Event)<-[rt:REFERS_TO]-(ef)
WITH a1.name as artist1, a2.name as artist2, a3.name as artist3, COUNT(e) as times
WHERE times>=3
RETURN artist1, artist2, artist3, times
```

10. Compute the quadruples of artists, and the number of times they have performed together in an event, if this number is at least 3.

```
MATCH (a1:Artist)<-[pb1:PERFORMED_BY]-(ef:EventFact)-[pb2:PERFORMED_BY]->(a2:Artist)
MATCH (a4:Artist)<-[pb4:PERFORMED_BY]-(ef)-[pb3:PERFORMED_BY]->(a3:Artist)
WHERE a1.name < a2.name < a3.name < a4.name
MATCH (e:Event)<-[rt:REFERS_TO]-(ef)
WITH a1.name as artist1, a2.name as artist2, a3.name as artist3, a4.name as artist4,
    COUNT(e) as times
WHERE times>=3
RETURN artist1, artist2, artist3, artist4, times
```

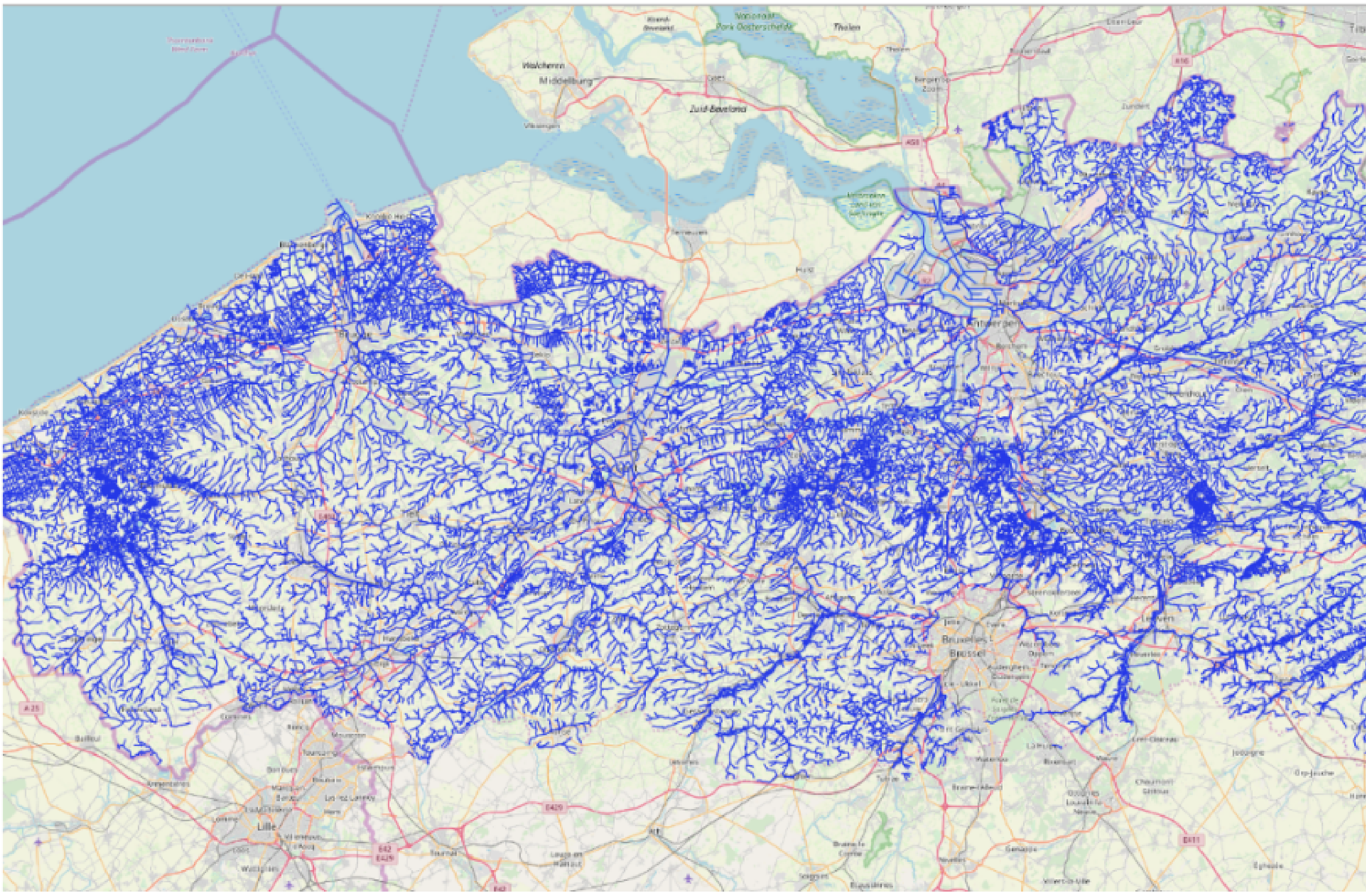
11. Compute the pairs of artists that have performed together in at least two events and that have worked together in at least one release, returning the number of events and releases together.

```
MATCH (a1:Artist)<-[pb1:PERFORMED_BY]-(ef:EventFact)-[pb2:PERFORMED_BY]->(a2:Artist)
WHERE a1.name < a2.name
MATCH (e:Event)<-[rte:REFERS_TO]-(ef)
WITH a1, a2, COUNT(e) as events
MATCH (a1)<-[in1:INCLUDES]-(ac:ArtistCredit)-[in2:INCLUDES]->(a2)
MATCH (ac)<-[rb:RELEASED_BY]-(rf:ReleaseFact)-[rtr:REFERS_TO]->(r:Release)
WITH a1.name as artist1, a2.name as artist2, events, COUNT(r) as releases
WHERE events>=2 AND releases>=1
RETURN artist1, artist2, events, releases
```

12. Compute the number of artists who released a record and performed in at least an event, and the year(s) this happened.

```
MATCH (a1:Artist)<-[pb1:PERFORMED_BY]-(ef:EventFact)-[ho:HAPPENED_ON]->(de:Date)
MATCH (e:Event)<-[rte:REFERS_TO]-(ef)
WITH a1, COUNT(e) as events, COLLECT(DISTINCT(de.year)) as eventyears
MATCH (a1)<-[in1:INCLUDES]-(ac:ArtistCredit)<-[rb:RELEASED_BY]-(rf:ReleaseFact)
MATCH (dr:Date)<-[ro:RELEASED_ON]-(rf)-[rtr:REFERS_TO]->(r:Release)
WITH a1.name as artist1, events, eventyears, COUNT(r) as releases, COLLECT(DISTINCT(dr.
    year)) as releaseyears
WHERE events>=1 AND releases>=1
RETURN artist1, events, releases, eventyears, releaseyears
```

Exercise 4.12. We will query the Flanders river system depicted in Figure 1. The schema and properties are shown in the next figures. Segments are represented as nodes, with label :Segment (and their corresponding properties), and the relation between the nodes is called :flowsTo, defined as follows: there is a relation :flowsTo from node A to node B if the water flows to segment B from segment A. This is stored in the rivers database.



1. Query 1. Compute the average segment length.

```
MATCH (n:Segment)
RETURN avg(n.lengte) AS avglength
```

2. Query 2. Compute the average segment length by segment category

```
MATCH (n:Segment)
RETURN n.catc as category, avg(n.lengte)
AS avglength order by category asc
```

3. Query 3. Find all segments that have a length within a 10% margin of the length of segment with ID 6020612.

```
MATCH (n:Segment {vhas:6020612})
WITH n.lengte as length
MATCH (m:Segment)
WHERE m.lengte < length*1.1 and m.lengte > length*0.9
RETURN m.vhas, m.lengte;
```

4. Query 4. For each segment find the number of incoming and outgoing segments.

```
MATCH (src:Segment)-[:flowsTo]->(n:Segment)-[:flowsTo]->(target:Segment)
RETURN n.vhas as nodenbr, COUNT(DISTINCT src) as segIn,
COUNT (DISTINCT target) as segOut
```

5. Query 5. Find the segments with the maximum number of incoming segments.

```
MATCH (n:Segment)
OPTIONAL MATCH (src:Segment)-[:flowsTo]->(n)
WITH n, COUNT(distinct src) as indegree
WITH COLLECT ([n, indegree]) as tuples, MAX(indegree) as max
RETURN [t in tuples WHERE t[1] = max |t]
```

6. Query 6. Find the number of splits in the downstream path of segment 6020612.

```
MATCH (n:Segment {vhas:6020612})
CALL apoc.path.spanningTree(n,{relationshipFilter:"flowsTo>", minLevel: 1}) YIELD
path AS pp
UNWIND NODES(pp) as p
MATCH (p)-[:flowsTo]->(r:Segment)WITH p, count(DISTINCT r) as co WHERE co > 1
RETURN count(p)
```

7. Query 7: Find the number of in-flowing segments in the downstream path of segment 6020612.

```
MATCH (n:Segment {vhas:6020612})
CALL apoc.path.spanningTree(n,{relationshipFilter:"flowsTo>", minLevel: 1}) YIELD path AS
    pp
WITH [p in NODES(pp) | p.vhas] as ids
UNWIND ids as id
WITH collect(DISTINCT id) as ids
MATCH (s:Segment)-[:flowsTo]->(p)
WHERE NOT s.vhas in ids AND p.vhas <> 6020612 AND p.vhas in ids
RETURN count(DISTINCT s) as inflows
```

8. Query 8. Determine if there is a loop in the downstream path of segment 6031518.

```
MATCH (n:Segment {vhas:6031518})
CALL apoc.path.spanningTree(n, {relationshipFilter:"flowsTo>", minLevel: 1}) YIELD path
    AS pp
WITH [p in NODES(pp) | p] as nodelist
UNWIND nodelist as p
CALL apoc.path.expandConfig(p, {relationshipFilter:"flowsTo>", minLevel: 1,
    terminatorNodes:[p], whitelistNodes:nodelist}) yield path as loop
RETURN count(loop) >0 as loops
```

9. Query 9. Find the length, the # of segments, and the IDs of the segments, of the longest branch of upstream flow starting from a given segment.

```
MATCH (n:Segment {vhas:6020612})
CALL apoc.path.expandConfig(n,{relationshipFilter:"<flowsTo", minLevel: 1}) YIELD path AS
    pp
WITH reduce(longi= tofloat(0), n IN nodes(pp)| longi+ tofloat(n.lengte)) AS blength,
Length(pp) as alength, [p in NODES(pp) |p.vhas] AS nodelist
WITH blength, alength, nodelist[size(nodelist)-1] as id
WITH id, max(blength) as ml, collect([id,blength,alength]) as coll
WITH id, ml, [p in coll WHERE p[0]= id AND p[1]=ml|p[2]] AS lhops
UNWIND lhops as hops
RETURN id,ml,hops order by id desc;
```

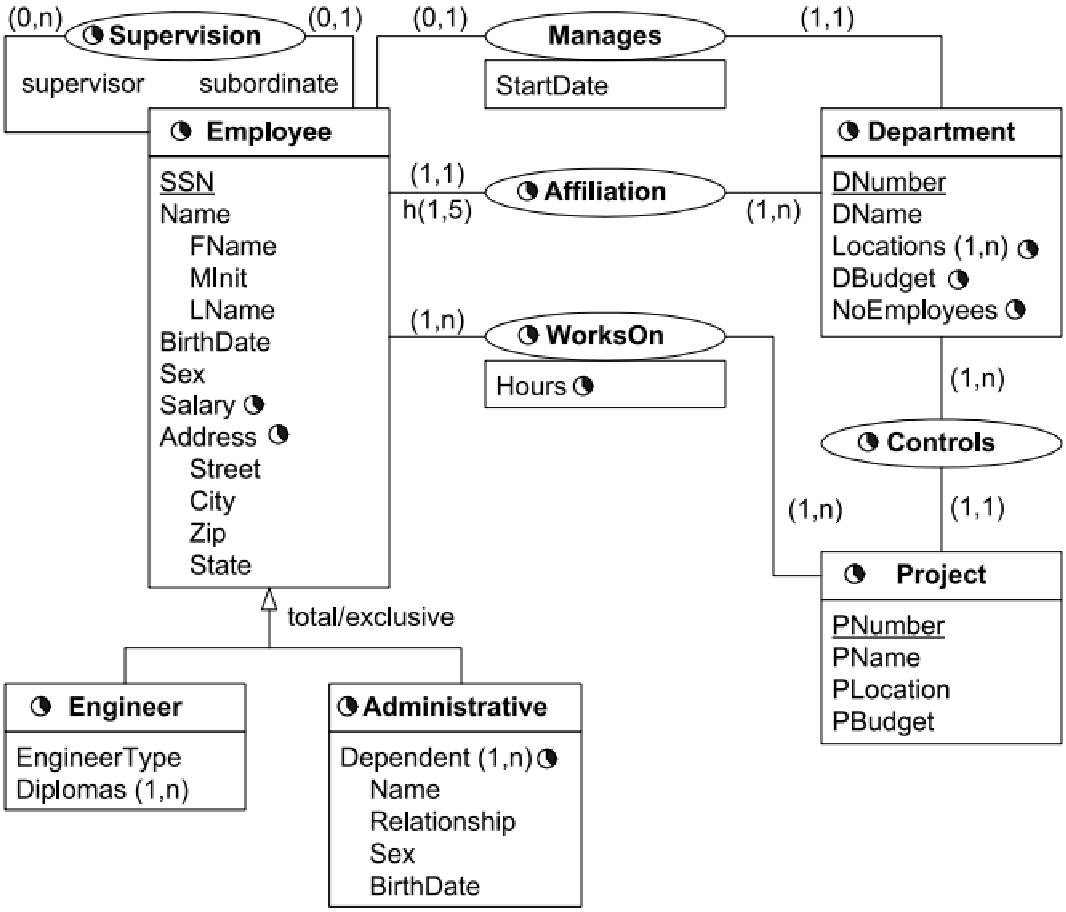
10. Query 10. How many paths exist between two given segments X and Y?

```
MATCH (n:Segment {vhas:6020612}),(m:Segment {vhas: 7036554})
CALL apoc.path.expandConfig(n,{relationshipFilter:"<flowsTo", minLevel: 1,
    terminatorNodes:[m]}) yield path as pp
RETURN count(pp) as paths
```

11. Query 11. Find all segments reachable from the segment closest to Antwerpen's Groenplaats

```
CALL apoc.spatial.geocodeOnce('Groenplaats Antwerpen Flanders Belgium') YIELD location as
     ini
MATCH (n:Segment)
WITH n, ini,distance(point({longitude:n.source_long, latitude:n.source_lat}), point({
    longitude:ini.longitude, latitude:ini.latitude}) ) as d
WITH n, d order by d asc limit 1
CALL apoc.path.spanningTree(n,{relationshipFilter:"flowsTo>", minLevel: 1}) YIELD path as
     pp
UNWIND NODES(pp) as p
RETURN p.vhas;
```

Part III
Temporal databases



Part IV
Spatial databases

5 Lab 10

In the directory **tp10**, perform:

```
// Initialization
$ createdb infoh415
$ psql infoh415 -c "CREATE EXTENSION postgis;"

// Tables creation
$ psql infoh415
infoh415$ \i create_tables.sql

// Tables population
infoh415$ \i insertion.sql
infoh415$ \i create_index.sql

// Importing and exporting
infoh415$ quit
$ cd shapefiles/
$ pgsql2shp infoh415 regions
$ pgsql2shp infoh415 cities
$ shp2pgsql -W "latin1" bel_city.shp >  ../shp_insert.sql
$ shp2pgsql -W "latin1" bel_dist.shp >> ../shp_insert.sql
$ shp2pgsql -W "latin1" bel_prov.shp >> ../shp_insert.sql
$ shp2pgsql -W "latin1" bel_regn.shp >> ../shp_insert.sql
$ shp2pgsql -W "latin1" belriver.shp >> ../shp_insert.sql
$ cd ..
$ psql infoh415
infoh415$ \i shp_insert.sql

// Specify the SRID to use WGS84
infoh415$ \i update_srid.sql
```

Write down and execute the following queries:

1. Get the SRID from table cities.

```
select st_srid(geom) as SRID
from bel_city
limit 5;
```

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.

```
select distinct st_dimension(geom) as dim, st_coorddim(geom) as coorddim
from bel_city;
```

4. Get the geometry type of these objects.

```
select distinct st_geometrytype(geom) as type
from bel_city;
```

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

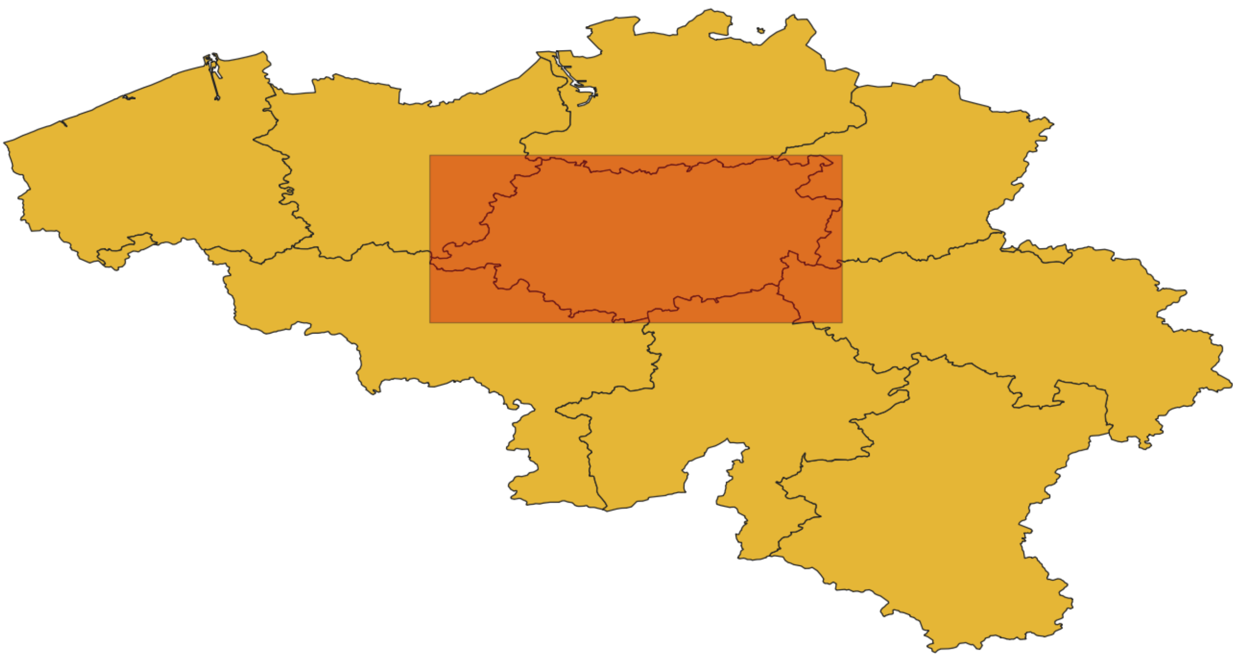
```
select st_distance(
  st_transform((select geom from bel_city where name = 'Ixelles'),3812),
  st_transform((select geom from bel_city where name = 'Brugge'),3812)) as dist,
st_distancesphere(
  (select geom from bel_city where name = 'Ixelles'),
  (select geom from bel_city where name = 'Brugge')) as dist_sphere
```

The number 3812 in the transform function refers to Belgian Lambert 2008, which is a conic representation adapted for Belgium. If we don't apply this transformation, we get a distance measured in degrees, which has little meaning for us.

6. Compute the bounding rectangle for the BRABANT province.

```
create table boundbox as
  select st_envelope(geom)
  from bel_prov
  where name = 'Brabant';
```

We create the table to be able to visualize the result in QGis:



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

```
create table geom_union as
  select st_union(geom)
  from (select geom from bel_regn
      union
      select geom from bel_prov)
      as tmp;
```

8. Compute the length of each river.

```
select st_length2dspheroid(geom, 'SPHEROID["GRS_1980",6378137,298.257222101]')
from belriver;
```

The second argument is the spheroid that the function should use to compute the lengths, as its definition is $ST\_Length2DSpheroid(geometry\ geom, spheroid\ sp)$.

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

```
select name, dist
from
  (select c.name as name, min(st_distancesphere(c.geom, r.geom)) as dist
   from bel_city c, belriver r
   group by c.name) as tmp
where dist < 1000;

-- or

select c.name as name, min(st_distancesphere(c.geom, r.geom)) as dist
from bel_city c
   join belriver r on st_distancesphere(c.geom, r.geom) < 1000
group by c.name;
```

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

```
select r.name as river, p.name as province,
  st_length2dspheroid(st_intersection(r.geom, p.geom),
          'SPHEROID["GRS_1980",6378137,298.257222101]') as inside_len
from belriver r
join bel_prov p on st_intersects(r.geom, p.geom);
```

6 Lab 11

Download http://biogeo.ucdavis.edu/data/climate/worldclim/1_4/grid/cur/alt_10m_bil.zip and extract it to **tp11**.

In the directory **tp11** perform:

```
createdb tp11
psql tp11 -c "CREATE EXTENSION postgis;"
psql tp11 -c "CREATE EXTENSION postgis_raster;"
psql tp11 -f generate.sql

cd bel_alt/
raster2pgsql BEL_alt.vrt > insert_bel_alt.sql
psql tp11 -f insert_bel_alt.sql
cd ../alt_10m_bil/
raster2pgsql alt.bil > insert_alt.sql
psql tp11 -f insert_alt.sql
cd ..


# 3.2 Update rasters
psql tp11
SELECT UpdateRasterSRID('alt', 'rast', 4326);
SELECT UpdateRasterSRID('bel_alt', 'rast', 4326);
```

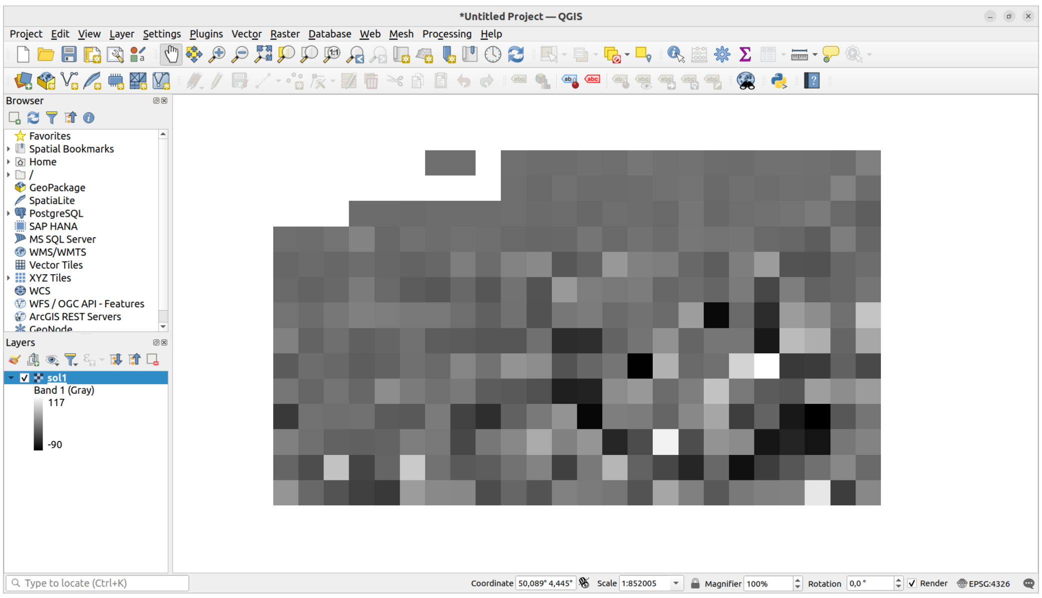
Write down and execute the following queries:

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
```

```
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;
```

   Once we have the table, we execute the command to export the raster and then we import it to QGIS. The result is (not very awesome):



2. Compute the maximum altitude in Belgium.

```
SELECT (ST_SummaryStats(rast)).max as max
FROM bel_alt;
```

3. Get the altitudes of all cities in Belgium.

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

4. Compute the maximum and minimum altitudes for each province.

```sql
SELECT name, (stats).max, (stats).min
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. Create a new raster table restraining the alt_16 raster to Belgium. (Hint: use ST_Intersection.)

```sql
CREATE TABLE alt_16_belgium AS
SELECT rid, ST_Clip(rast, 1, geom, TRUE) AS rast
FROM (SELECT ST_UNION(ARRAY(SELECT geom FROM bel_regn)) AS geom) AS belgium
JOIN alt ON ST_Intersects(geom, rast);
```

6. Compute the altitude along each river.

```sql
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;
```

Spatial functions reference

- **ST_CLIP**: Returns a raster that is clipped by the input geometry geom. If band index is not specified, all bands are processed.

  Rasters resulting from ST_Clip must have a nodata value assigned for areas clipped, one for each band. If none are provided and the input raster do not have a nodata value defined, nodata values of the resulting raster are set to ST_MinPossibleValue(ST_BandPixelType(rast, band)). When the number of nodata value in the array is smaller than the number of band, the last one in the array is used for the remaining bands. If the number of nodata value is greater than the number of band, the extra nodata values are ignored. All variants accepting an array of nodata values also accept a single value which will be assigned to each band.

  If crop is not specified, true is assumed meaning the output raster is cropped to the intersection of the geomand rast extents. If crop is set to false, the new raster gets the same extent as rast.

```sql
-- Clip the first band of an aerial tile by a 20 meter buffer.
SELECT ST_Clip(rast, 1,
        ST_Buffer(ST_Centroid(ST_Envelope(rast)),20)
    ) from aerials.boston
WHERE rid = 4;

-- Demonstrate effect of crop on final dimensions of raster
-- Note how final extent is clipped to that of the geometry
-- if crop = true
SELECT ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, true))) As xmax_w_trim,
    ST_XMax(clipper) As xmax_clipper,
    ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, false))) As xmax_wo_trim,
    ST_XMax(ST_Envelope(rast)) As xmax_rast_orig
FROM (SELECT rast, ST_Buffer(ST_Centroid(ST_Envelope(rast)),6) As clipper
    FROM aerials.boston
WHERE rid = 6) As foo;

   xmax_w_trim     |   xmax_clipper    |   xmax_wo_trim    |  xmax_rast_orig
------------------+------------------+------------------+------------------
 230657.436173996 | 230657.436173996 | 230666.436173996 | 230666.436173996
```

	
Full raster tile before clipping	After Clipping

- **ST_DUMPPOINTS**: A set-returning function (SRF) that extracts the coordinates (vertices) of a geometry. It returns a set of geometry_dump rows, each containing a geometry (geom field) and an array of integers (path field).
  - the geom field POINTs represent the coordinates of the supplied geometry.
  - the path field (an integer[]) is an index enumerating the coordinate positions in the elements of the supplied geometry. The indices are 1-based. For example, for a LINESTRING the paths are {i} where i is the nth coordinate in the LINESTRING. For a POLYGON the paths are {i,j} where i is the ring number (1 is outer; inner rings follow) and j is the coordinate position in the ring.

```
SELECT edge_id, (dp).path[1] As index, ST_AsText((dp).geom) As wktnode
FROM (SELECT 1 As edge_id
  , ST_DumpPoints(ST_GeomFromText('LINESTRING(1 2, 3 4, 10 10)')) AS dp
     UNION ALL
     SELECT 2 As edge_id
  , ST_DumpPoints(ST_GeomFromText('LINESTRING(3 5, 5 6, 9 10)')) AS dp
   ) As foo;
 edge_id | index |     wktnode
---------+-------+--------------
       1 |     1 | POINT(1 2)
       1 |     2 | POINT(3 4)
       1 |     3 | POINT(10 10)
       2 |     1 | POINT(3 5)
       2 |     2 | POINT(5 6)
       2 |     3 | POINT(9 10)
```

- **ST_INTERSECTS**: Compares two geometries and returns true if they intersect. Geometries intersect if they have any point in common.

```
SELECT ST_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 2 0, 0 2 )'::geometry);
 st_intersects
---------------
 f
(1 row)
SELECT ST_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 0 0, 0 2 )'::geometry);
 st_intersects
---------------
 t
(1 row)

-- Look up in table. Make sure table has a GiST index on geometry column for faster
    lookup.
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707
    52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53))');
 id | name
----+-------
  2 | Minsk
(1 row)
```

- **ST_MAPALGEBRA**: Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.
  - With one ruster: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation defined by the expression on the input raster (rast). If nband is not provided, band 1 is

assumed. The new raster will have the same georeference, width, and height as the original raster but will only have one band.

– With two rusters: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation to the two bands defined by the expression on the two input raster bands (rast1, rast2). If no band1, band2 is specified band 1 is assumed. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster. The resulting raster will have the extent defined by the extenttype parameter.

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1,
    -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast
    UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1,
    -1, 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast, 2,
        t2.rast, 1,
        '([rast2] + [rast1.val]) / 2'
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2;
```

- **ST_RESAMPLE**: Resample a raster using a specified resampling algorithm, new dimensions (width & height), a grid corner (gridx & gridy) and a set of raster georeferencing attributes (scalex, scaley, skewx & skewy) defined or borrowed from another raster. If using a reference raster, the two rasters must have the same SRID. New pixel values are computed using the NearestNeighbor (English or American spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor which is the fastest but produce the worst interpolation.

```
SELECT
    ST_Width(orig) AS orig_width,
    ST_Width(reduce_100) AS new_width
FROM (
    SELECT
        rast AS orig,
        ST_Resample(rast,100,100) AS reduce_100
    FROM aerials.boston
    WHERE ST_Intersects(rast,
        ST_Transform(
            ST_MakeEnvelope(-71.128, 42.2392,-71.1277, 42.2397, 4326),26986)
    )
    LIMIT 1
) AS foo;

 orig_width | new_width
------------+-------------
        200 |         100
```

- **ST_SEGMENTIZE**: Returns a modified geometry having no segment longer than the given max_segment_length. Distance computation is performed in 2d only. For geometry, length units are in units of spatial reference. For geography, units are in meters.

```
SELECT ST_AsText(ST_Segmentize(
ST_GeomFromText('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45 -33,-46 -32))')
    ,5)
);
st_astext
--
    ------------------------------------------------------------------------------------------------
MULTILINESTRING((-29 -27,-30 -29.7,-34.886615700134 -30.758766735029,-36 -31,
```

```
-40.8809353009198 -32.0846522890933,-45 -33),
(-45 -33,-46 -32))
(1 row)

SELECT ST_AsText(ST_Segmentize(ST_GeomFromText('POLYGON((-29 28, -30 40, -29 28))'),10));
st_astext
-----------------------
POLYGON((-29 28,-29.8304547985374 37.9654575824488,-30 40,-29.1695452014626
    30.0345424175512,-29 28))
(1 row)
```

- **ST_SUMMARYSTATS**: Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. If no band is specified nband defaults to 1.

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
    FROM dummy_rast CROSS JOIN generate_series(1,3) As band
     WHERE rid=2) As foo;

 rid | band | count | sum  |    mean    |  stddev   | min | max
-----+------+-------+------+------------+-----------+-----+-----
   2 |    1 |    23 | 5821 | 253.086957 |  1.248061 | 250 | 254
   2 |    2 |    25 | 3682 |     147.28 | 59.862188 |  78 | 254
   2 |    3 |    25 | 3290 |      131.6 | 61.647384 |  62 | 254
```

- **ST_UNION**: Unions the input geometries, merging geometry to produce a result geometry with no overlaps. The output may be an atomic geometry, a MultiGeometry, or a Geometry Collection.

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

- **ST_VALUE**: Returns the value of a given band in a given columnx, rowy pixel or at a given geometry point. Band numbers start at 1 and band is assumed to be 1 if not specified.

```
-- get raster values at particular postgis geometry points
-- the srid of your geometry should be same as for your raster
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As
    b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As
    pt_geom) As foo
WHERE rid=2;

 rid | b1pval | b2pval
-----+--------+--------
   2 |    252 |     79


-- general fictitious example using a real table
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast,sometable.geom);
```