Ressources

- Algebre Spark ou Dataset (Nov16)
- Spark - KV Store (Fev15)
- Spark (Fev15)

SQL

Soit table R:

A	B	C	D
x	y	1	2
y	y	1	5
x	z	2	3
y	y	1	4
z	y	4	1
y	y	2	2

Cube

> Ecrivez la requete qui renvoie le resultat suivant

A	B	N
x	y	1
x	z	1
x		2
y	y	3

y		3
z	y	1
z		1
	y	5
	z	1
		6

```
select a, b, count(D) as n
from R
group by cube(a, b)
```

> GROUP BY CUBE(A,B), CUBE(B,C) retourne combien de group?

16 = (2^2) * (2^2)

```
((AB), A), (B), ()) x ((BC), (C), ())
(ABC), (AB), (ABC), (AB), (ABC), (AB), (AC), (A)
(BC), (B), (BC), (B), (BC), (B), (C), ()
```

Rollup

```
SELECT A, B, C, sum(D) as n FROM R
GROUP BY ROLLUP (A, B, C);
```

retourne 13 n-uplets:

A	**B**	**C**	**N**

x	y	1	2
x	y		2
x	z	2	3
x	z		3
x			5
y	y	1	9
y	y	2	2
y	y		11
y			11
z	y	4	1
z	y		1
z			1
			17

Rankings

> Liste des 5 produits les plus vendus en France. On affiche le nom du produit, la quantité vendue et top 5 produits plus vendus

```
with T as (
    Select p.nom, sum(v.quantité), rank over
(order by sum(v.quantite) desc as rang )
    From Ventes v, Prod p, Magasin m, Zone z
    Where v.prod = p.prod and v.magasin =
m.magasin
        And m.zone = z.zone and z.pays =’France’
    Group by p.nom)
Select * from T where rang <=5;
```

> Pour chacun des 10 premiers jours du mois de mai 2015, donnez la quantité moyenne de produits de catégorie alimentaire vendus les 7 jours précédents.

```
Select
    num_date,
    avg(v.quantite) over
        (order by v.num_date
        range between interval '7' Day
        preceding and current row) as qte-moyenne
From Vente v, Prod p, Classe c
Where
    v .prod = p. prod and p.Classe = c.classe
    and c.categorie = 'alimentaire'
    and v.num_date > '01-05-2015'
    and v.num_date < '10-05-2015'
Order by v.num_date desc;
```

> Pour chaque client, donner le montant de ses achats par trimestre en 1993.

```
Select v.client, d.trimestre, sum(v.prix *
v.quantite) over (partition by c.client,
order by sum(v.prix*v.quantite) as
montant_par_trimestre
From Ventes v, Date d
Where v.num_date =d.num_date and d.annee = 1993
Group by v.client, d.trimestre
Order by v.client, montant_par_trimestre;
```

> On veut calculer le montant total des ventes selon les 3 dimensions suivantes : date, produit, magasin. On veut afficher

> la famille du produit et la superficie du magasin, et on veut pouvoir naviguer au travers des différents niveaux de date (année, trimestre, mois, jour).

```
Select c.famille, m.superficie, d.annee,
d.trimestre, d.num_mois, d.num_jour, sum
(v.prix * v.quantite) as montant-total
From Vente v, prod p, classe c, date d, magasin m
Where v.prod=p.prod
And p.classe = c.classe
And v.num_date = d.num_date
And v.magasin = m.magasin
Group by c.famille, m.superficie, rollup(d.annee,
d.trimestre, d.num_mois,
d.num_jour)
```

Scala

Chains

ExoZero

```
scala> R // liste des tuples  (A, B, C)
scala> S // liste des tuples (C, D, E)

// calculer R join S on R.C=S.C
scala> R.map{case(a,b,c)=>(c,
(a,b))}.join(S.map{case(c,d,e)=>(c,(d,e))})
//porduit une RDD de la forme (c, ((a,b), (d,e)))

x(0) // acces a l'element 0 de l'array x
x._1 // acces a l'element 1 du tuple x
```

Ex1

```
scala> val triples =
sc.textFile("...").map(x=>x.split(",")).map(x=>
(x(0),x(1),x(2)))
scala> val p0 = triples.filter{case (s,p,o) =>
p.contains("p0")}
scala> val p1 = triples.filter{case (s,p,o) =>
p.contains("p1")}
scala> val p2 = triples.filter{case (s,p,o) =>
p.contains("p2")}
scala> val p3 = triples.filter{case (s,p,o) =>
p.contains("p3")}

> (q1) select ?x ?z where { ?x p0 ?y . ?y p1 ?w. ?
w p2 ?z}
>>> x -> y
>>> y -> w
>>> w -> z

val q1 = p0.
    map{ case(x, p0, y)=>(y,x) }.
    join(p1.map{ case(y, p1, w) => (y,w) }). //
J'ai (y, (x, w))
    map{ case(y,(x,w)) => (w, (x, y)) }.     //
J'ai (w, (x, y))
    join(p2.map{ case(w, p, z) => (w, z) }). //
J'ai (w, (x, y), z)
    map{ case(w, (x, y), z) => (x, z) }


> (q2) select ?x ?z where { ?x p2 ?z.  ?y p1 ?x. ?
x p3 ?w}
>>> x -> z
>>> y -> x
```

```
>>> x -> w


val q2 = p2.
    map{ case(x, p1, z) => (x, z) }.
    join(p1.map{ case(y, p1, x) => (x, y) } ). //
J'ai (x, (z, y)).
    join(p3.map{ case(x, p3, w) =>(x, w) }).   //
J'ai (x, (z, y), w)
    map( case (x, (z,y), w) => (x, z))
```

Ex2

```
select ?p ?s
where {?p studiedAt ?u. ?p supervisedBy ?s. ?s
studiedAt ?u}

val q2 = studiedAt.
    map{case(p,sat,u)=>(p,u)}.
    join(supervisedBy.map{case(p,sby,s)=>(p,s)}).
    map{case(p,(u,s))=>(s,(p,u))}.
    join(studiedAt.map{case(ps,sat,us)=>(ps,us)}).
    map{case(s,((p,u),us))=>(p,u,s,us)}.

filter{case(p,u,s,us)=>u==us}.map{case(p,u,s,us)=>
(p,s)}
```

Ex3

```
select ?p ?s
where ?p studiedAt ?u. ?p supervisedBy ?s. ?s
studiedAt ?u

val q2 = studiedAt.
```

```
    map{case(p,sat,u)=>(p,u)}.
    join(supervisedBy.map{case(p,sby,s)=>(p,s)}).
    map{case(p,(u,s))=>(s,(p,u))}.
    join(studiedAt.map{case(ps,sat,us)=>(ps,us)}).
    map{case(s,((p,u),us))=>(p,u,s,us)}.

filter{case(p,u,s,us)=>u==us}.map{case(p,u,s,us)=>
(p,s)}
```

Requêtes analytiques

Le top 10 des clients ayant dépensé le plus.

Afficher la liste des clients avec le montant total de leurs commandes. Pour chaque client, donner son n° et le montant total de ses commandes. Trier le résultat par ordre décroissant du montant. Afficher seulement les 10 premiers clients.

```
with R1 as(
select c_custkey, sum(o_totalprice) as TOTAL,
       rank() over(order by sum(o_totalprice)) as
RANG
from Customer, Orders
where c_custkey = o_custkey
group by c_custkey) select * from R1 where rang <=
10;
```

Le top 5 des pays avec le plus grand nombre de clients

Afficher la liste des pays, référencés dans Nation, avec leur nombre de clients. Donner le n° du pays, son nom et le nb de clients. Classer le résultat par ordre décroissant du nombre de clients.

```
with R2 as(
select n_nationkey, n_name, count(*) as
NB_DE_CLIENTS,
       rank() over(order by count(*) desc) as rang
from Nation, Customer
where n_nationkey = c_nationkey
group by n_nationkey, n_name
) select * from R2 where rang <= 5;
```

Le top 20% des pays avec le plus grand nombre de clients.

Afficher seulement les pays classés parmi les 20% meilleurs.

```
with R3 as(
select n_nationkey, n_name, count(*) as
NB_DE_CLIENTS,
       percent_rank() over(order by count(*) desc)
       as rang_pourcent
from Nation, Customer
where n_nationkey = c_nationkey
group by n_nationkey, n_name
) select * from R3 where rang_pourcent <= 0.2;
```

Classement national des produits vendus en plus grande quantité.

Pour chaque pays, donner le classement national des produits les plus achetés par des clients de ce pays. Afficher les attributs nom du pays, numéro du produit, quantité_achetée et rang.

```
with R5 as(
```

```
select n_name as NATION, ps_partkey as PRODUCT,
       count(*) as QUANTITE,
  rank() over(partition by n_name order by
count(*) desc)
  as RANG_PAYS
from Partsupp, Nation, Customer, Lineitem, Orders
where n_nationkey = c_nationkey and c_custkey =
o_custkey
  and l_orderkey = o_orderkey and ps_partkey =
l_partkey
group by n_name, ps_partkey
) select * from R5 where QUANTITE > 180;
```

Fenêtre temporelle glissante

Tenir compte seulement des commandes effectuées après le 01/04/1998. Pour chaque jour donner le prix moyen des commandes effectuées les 90 jours précédents. La moyenne est calculée sur l'ensemble des commandes enregistrées pendant les 90 jours précédents.

```
with R6 as(
select o_orderdate, avg(sum(o_totalprice))
       over (order by o_orderdate
       range between interval '90' day preceding
       and current row) as MOYENNE_TRIMESTRE
from Orders
where o_orderdate >= '01/06/1998'
group by o_orderdate
) select * from R6 where rownum <= 20;
```

Pour chaque jour, quel est le chiffre d'affaire des 30 derniers jours

```
with R7 as (
```

```
select o_orderdate, sum(sum(o_totalprice))
       over (order by o_orderdate
       range between interval '30' day preceding
       and current row) as CA_TRIMESTRE
from Orders
where o_orderdate >= '01/06/1998'
group by o_orderdate
) select * from R7 where rownum <= 10;
```

TME 2 : Cube : manipulation de données multi-dimensionnelles

Exercice 1

Question 1 : Opérations algébriques

- Quelles sont, d'après le cours, les opérations à appliquer sur C1 pour obtenir C2 ?
    - Donner l'ordre dans lequel les opérations sont appliquées.

step 1 :remonter dans la division produit => Roll up de nom_produit à type de produit

step 2 : slice sur la dimension produit(type = cuivre)

step 3 : slice sur la dimension jour()jour<6-01-98

step 4 : enlever une dimension => projection aggregative sur(produit,client)

On peut faire aussi 3,1,2,4

roll up(a,b,c): a,b et c sont 3 niveau de la mm dimension (a est le niveau le plus hau et c est le niveau le plus bas) Cube(a,b,c): a,b,c 3 dimension différent

- Donner la requête SQL calculant toute les cellules de C2 (264

cellules).

```
with C2 as(
select  p_partkey,c_custkey,o_orderdate,
        sum(o_totalprice)
from part,customer,orders,lineitem
where c_custkey = o_orderkey
      and l_orderkey = o_orderkey
      and l_partkey = p_partkey
group by p_partkey,c_custkey,o_orderdate
order by p_partkey,c_custkey)
select *
from C2
where rownum<=10;
```

Résultat :

P_PARTKEY	C_CUSTKEY	O_ORDERDAT	SUM(O_TOTALPRICE)
1	1281	11/12/1994	247632,44
3	129	19/11/1992	261013,14
4	548	21/09/1994	165456,46
4	678	27/02/1993	185606,97
4	930	17/12/1994	277890,79
5	261	29/06/1993	319306,86
5	513	01/05/1995	105559,7
9	102	09/05/1997	164529,1
10	999	05/09/1993	222435,59
11	453	26/05/1997	329149,3

Question 2 : Agrégation sur les niveaux d'une dimension On veut calculer, en une seule requête T2, tous les cubes obtenus à partir de C2 par agrégation sur la dimension client (ayant 3 niveaux). Le résultat de la requête contient toutes les données des cubes à tous les niveaux d'agrégation sur la dimension client.

- T2R: Ecrire T1 en utilisant le mot-clé ROLLUP

```
with T2 as(
          select p_type, c_name, n_name, r_name,
sum(o_totalprice) as TOTAL
          from Part, PartSupp, Lineitem, Orders,
Customer, Region, Nation
          where  p_partkey = ps_partkey and
ps_partkey = l_partkey
                 and l_orderkey = o_orderkey and
c_custkey = o_custkey
                 and n_nationkey = c_nationkey and
r_regionkey = n_regionkey
                 and p_type LIKE '%COPPER' and
o_orderdate >= '01/06/1998'
          group by rollup(p_type, c_name, n_name,
r_name)
)select *
from T2
where rownum <= 30;
```

Question 3 a : Projection agrégative sur une ou plusieurs dimensions

- En utilisant le mot clé CUBE écrire la requête T3 calculant tous les cubes obtenus à partir de C1 par projection agrégative sur toutes les combinaisons des dimensions client (au niveau region r_name), produits (au niveau du type p_type) et date au niveau de l'année qu'on obtient avec extract(year from o_orderdate). Le

schéma de T3 est:

```
select r_name, sum(o_totalprice) as TOTAL
from Part, Customer, Lineitem, Orders, Region,
Nation
where p_partkey = l_partkey and c_custkey =
o_custkey and o_orderkey = l_orderkey
      and n_nationkey = c_nationkey and
r_regionkey = n_regionkey
      and p_type LIKE '%COPPER' and o_orderdate >=
'01/06/1998'
      and r_name = 'AFRICA'
group by r_name;
```

Question 3 b : Opérations algébriques

- R1: Exprimer cette requête sur le schéma TPC-H directement.
- R2: Exprimer une requête équivalente, posée sur la vue T3.

```
with R2 as
(select p_type, r_name, extract(year from
o_orderdate)as ANNEE, sum(o_totalprice) as TOTAL
from Part, Customer, Lineitem, Orders, Region,
Nation
where p_partkey = l_partkey and c_custkey =
o_custkey and o_orderkey = l_orderkey
      and n_nationkey = c_nationkey and
r_regionkey = n_regionkey
      and p_type LIKE '%COPPER' and o_orderdate >=
'01/06/1998'
group by cube (p_type, r_name, extract(year from
o_orderdate))
) select r_name, TOTAL from R2 where r_name =
'AFRICA' and annee is null and p_type is null;
```

Question 4 : Agrégation sur trois dimensions

- On veut calculer, en une seule requête T4, tous les cubes obtenus à partir de C1 par navigation sur les 3 dimensions : produit, client, date. Pour cela on définit la vue :

''Date_commande (orderdate, numMois, numAnnée). L'attribut orderdate correspond au jour de la commande.' a) Définir en SQL la vue Date_commande

```
with Date_commande as
(select o_orderdate, extract(month from
o_orderdate) as numMois, extract(year from
o_orderdate)
```

from Orders)

TME 3 : Introduction à Scala

Exercice 1

Question 1

- Définir la fonction maxEntiers qui retourne le plus grand des entiers d'une liste fournie en entrée.
- Définir la fonction scEntiers qui calcule la somme des carrés des entiers d'une liste fournie en entrée.
- Définir la fonction moyEntiers qui calcule la moyenne des entiers d'une liste fournie en entrée.

```
val listeEntiers = List.range(1,11)

def maxEntiers(in: List[Int])= in.reduce((a,b)=>
(if(a>b)a else b))
```

```
def scEntiers(in: List[Int])=
in.map(e=>e*e).reduce((a,b)=>(a+b))

def moyEntiers(in: List[Int])={val p = in.map(e=>
(1,e)).reduce((a,b)=>(a._1+b._1, a._2+b._2));
p._2/p._1}
```

Question 2

- Calculer pour l'année 2009 le maximum de ses températures.

```
val listeTemp = List("7,2010,04,27,75",
"12,2009,01,31,78", "41,2009,03,25,95",
"2,2008,04,28,76", "7,2010,02,32,91")
val temp2009 =
listeTemp.map(x=>x.split(",")).filter(_(1).toInt==
2009).map(x=>x(3).toInt)

maxEntiers(temp2009)

moyEntiers(temp2009)
```

Question 3

- Il est demandé de construire à partir de melange deux listes distinctes :
    - notes contenant les éléments de la forme (userID, movieID, rating, timestamp) et dont le type est (Int, String, Int, Int) ;
    - films contenant les éléments de la forme (movieID, title, genre) et dont le type est (Int, String, String).

```
val melange = List("1233,100,3,20171010",
```

```
"1224,22,4,20171009", "100,lala,comedie",
"22,loup,documentaire")
val notes =
melange.map(_.split(",")).filter(_(0).toInt>=1000)
.map(x=>(x(0).toInt,x(1), x(2).toInt, x(3).toInt))

val films =
melange.map(_.split(",")).filter(_(0).toInt<=100).
map(x=>(x(0).toInt,x(1), x(2)))
```

Question 4

- Définir une classe Etu(nom:String, annee:Int) et une classe Ens(nom:String, annee:Int)
- Transformer personnes en une liste d'objets de la classe Etu ou Ens encapsulant les information des tuples

```
val personnes = List(("Joe", "etu", 3), ("Lee",
"etu", 4), ("Sara", "ens", 10), ("John", "ens",
5), ("Bill", "eng",20))

class Etu(nom:String, annee:Int){override def
toString()="Etudiant en " +annee+" annee"}

class Ens(nom:String, annee:Int){override def
toString()="Enseignant avec " +annee+" annee
d'experience"}

val classes_personnes = personnes.map(x=> x match
{ case(a,"ens",b) =>new Ens(a,b); case(a, "etu",
b) =>new Etu(a,b); case _=>None}).filter(_!=None)
```

Manipulation de RDD simple

Fichier wordcount: lignes de la forme En.d updates 3 24145.

```
val data = sc.textFile("wordcount.txt")
```

Extraire le 4e champ de chaque élément de data et le convertir en type double

```
val q1 = data.map(x => x.split(" ")).map(x =>
x(3).toDouble)
```

Construire à partir de q1 une liste contenant les nombres compris strictement entre 1000 et 1300 puis convertir en type entier.

```
val q2 = q1.filter(x => x >= 1000 && x <= 1300).
           map(x => x.toInt)
```

Construire à partir de q2 une liste contenant les multiples de 3 et l'appeler q33. Faire de même pour les multiples de 4 et l'appeler q34.

```
val q33 = q2.filter(x => x % 3 == 0)
val q34 = q2.filter(x => x % 4 == 0)
```

Construire une liste obtenue en divisant par 10 chaque élément de q33.

```
val q4 = q33.map(x => x / 10)
```

Construire à partir de q4 un ensemble d'éléments (liste sans doublons).

```
val q51 = q4.distinct
```

Construire une liste contenant les éléments de q2 qui sont multiples de 3 et de 4 à la fois. Utiliser impérativement q33 et q34.

```
val q52 = q33.intersection(q34)
```

Construire une liste contenant les éléments de q2 qui sont multiples de 3 mais pas de 4. Utiliser impérativement q33 et q34.

```
val q6 = q33.subtract(q34)
```

Construire à partir de q2 une liste contenant les éléments de q2 multiples de 3 ou de 10.

```
val q7 = q33.union(q2.filter(x => x % 10 == 0))
```

Convertir les éléments de q8 en type Double puis calculer la somme (q9sum), le minium (q9min) ainsi que le maximum (q9max) des éléments de q8. Calculer la moyenne (q9avg) en utilisant un operateur parmi reduce, fold et aggregate.

```
def min(a : Double, b : Double):Double = if (a <
b) a else b
def max(a : Double, b : Double):Double = if (a >
b) a else b
val q9sum = q8.sum
val q9min = q8.reduce(min)
val q9max = q8.reduce(max)
```

TME 4 : MapReduce en Spark - Algèbre RDD

Exercice 1

Manipulation de RDDs sous forme de paires clé-valeur

- Structurer le contenu de data de sorte à obtenir un tableau de tableaux de chaines de caractères. Ce dernier devra être stocké dans une nouvelle variable nommée list.

```
val list = data.map(_.split(" "))
```

- Afficher ensuite les 100 premiers éléments de la 3e colonne de list.

```
val q12 = list.map(x=>x(2))
```

- Transformer le contenu de list en une liste de paires (‘mot’, nb) où mot correspond à la première colonne de list et nb sa

troisième colonne.

```
val q13 = list.map(x=>(x(0),x(2).toInt))
```

- Grouper les paires par ‘mot’ et additionner leur nombre nb.

```
val q14 = q13.reduceByKey((x,y)=>x+y)
//ou bien q13.reduceByKey(_+_)
```

- Reprendre les questions 3 et 4 en calculant ‘mot’ différemment : désormais, ‘mot’ doit correspondre au préfixe du premier sous-élément de chaque élément de list, çad, pour en.d, mot doit être en, pour fr.d, mot doit être fr, etc.

```
val q13bis = list.map(x=>
(x(0),x(2).toInt)).map(x=>(x._1.split("\\.")(0),
x._2))
```

Exercice 2

- le nombre de notes (ratings) réalisées par chaque utilisateur identifié par son UserID

```
  val q2a  =
```

```
notes.map{case(userId,movieId,rating,ts)=>
(userId,1)}.reduceByKey(_+_)
```

- le nombre de notes (ratings) réalisées par chaque localisation donnée par le Zip-code

```
  val q2b =
utilis.map{case(userId,gender,age,occup,zipcode)=>
(userId,zipcode)}

.join(notes.map{case(userId,movieId,rating,ts)=>
(userId,1)})
                  .map{case (userId,(zipcode,
nb))=>(zipcode, 1)}
                  .reduceByKey(_+_)
```

- le nombre de notes (ratings) réalisées par chaque genre de film //jointure en notes et films

```
val q2c = notes.map(x=>
(x(1),1)).join(films.map(x=>(x(0),x(2))))
                .map{case (movieID, (genre,nb))=>
(genre, 1)}
                .reduceByKey(_+_)
```

- les 10 utilisateurs ayant noté le plus de films.

```
val q2d = notes.map(x=>
(x(0),1)).reduceByKey(_+_).takeOrdered(10)
```

- Les films ayant reçu le moins de notes
- Les utilisateurs n’ayant noté aucun film
- genre de films

```
val films_bis = films.map(x=>
(x._1,x._2,x._3.split("\\|"))).flatMap{case(a,b,l)
=>l.map(x=>(a,b,x))}
```

TME 5 : Dataset

Exercice 1

R1: Les régions dont la capitale (régionale) est Nantes

```
 ?x <hasCapital> <Nantes>
```

La requête s'écrit ainsi avec une opération de sélection where, un renommage withColumnRenamed et une projection select :

```
val r1 = yago.where("prop = '<hasCapital>' and
objet = '<Nantes>'")
             .withColumnRenamed("sujet","x")
             .
select("x")
```

```
r1.show(5)
```

Le résultat de la requête est :

x
<Duchy_of_Brittany>
<Pays_de_la_Loire >

R2: La fiche de Barack Obama

Afficher toutes les paires (propriété,objet) au sujet de <\Barack_Obama>

REPONSE

```
   val t1 = yago.where("sujet =
'<Barack_Obama>'").
     withColumnRenamed("prop","p").
     withColumnRenamed("objet","o").
     select("p", "o")
   t1.show(20)
```

Le résultat doit être :

P	O
<graduatedFrom>	<Columbia_University>
<graduatedFrom>	<Harvard_Law_School>
<hasGender>	<male>
<hasWonPrize>	<Grammy_Award>

<hasWonPrize>	<Nobel_Peace_Prize>
<isAffiliatedTo>	<Democratic_Party>
<isCitizenOf>	<Germany>
<isCitizenOf>	<Wales>
<isCitizenOf>	<United_States>
<isCitizenOf>	<Scotland>
<isCitizenOf>	<Kenya>
<isCitizenOf>	<Israel>
<isPoliticianOf>	<United_States>
<livesIn>	<Illinois>
<wasBornIn>	<Honolulu>

R3: Les leaders parisiens

La requête est une étoile formée de deux triplets

```
    ?x <livesIn> <Paris>
    ?x <isLeaderOf> ?z
```

REPONSE

```
val t1 = yago.where("prop = '<livesIn>' and objet
= '<Paris>'").
    withColumnRenamed("sujet","x").
    select("x")

val t2 = yago.where("prop = '<isLeaderOf>' ").
    withColumnRenamed("sujet","x").
```

```
    withColumnRenamed("objet","z").
    select("x","z")

val t1t2 = t1.join(t2, "x")
t1t2.show(5)
```

Le résultat doit être:

X	Z
<Louis_Blanc>	<Republican_Union...>
<Stefano_Zacchiroli>	<Software_Heritage>

R4: Les joueurs du royaume uni.

La requête est un chemin formé de deux triplets

```
      ?x <playsFor> ?y .

                    ?y <isLocatedIn>
<United_Kingdom>
```

REPONSE

```
val t1 = yago.where("prop = '<playsFor>'").
    withColumnRenamed("sujet","x").
    withColumnRenamed("objet","y").
    select("x","y")

val t2 = yago.where("prop = '<isLocatedIn>' and
objet = '<United_Kingdom>'").
    withColumnRenamed("sujet","y").
    select("y")
```

```
val t1t2 = t1.join(t2, "y")
t1t2.show(5)
```

Les 5 premiers éléments du résultat sont :

Y	X
<Royal_Air_Force>	<George_Ayres>
<Royal_Air_Force>	<Jack_Jones_(foot...
<Royal_Air_Force>	<John_Hinton_(foo...
<University_of_Ed...	<James_Craigen_(f...
<University_of_Ed...	<Ronald_Brebner>

R5: Les acteurs qui influencent des créateurs

La requête est un flocon (ou snowflake) formé de 5 triplets :

```
    ?x <isCitizenOf> ?y .

    ?x <actedIn> ?z .

    ?x <influences> ?t .

                    ?t <livesIn> ?u .

                    ?t <created> ?v
```

REPONSE

```
val t1 = yago.where("prop = '<isCitizenOf>'").
    withColumnRenamed("sujet","x").
```

```
    withColumnRenamed("objet","y").
    select("x","y")

val t2 = yago.where("prop = '<actedIn>'").
    withColumnRenamed("sujet","x").
    withColumnRenamed("objet","z").
    select("x","z")

val t3 = yago.where("prop = '<influences>'").
    withColumnRenamed("sujet","x").
    withColumnRenamed("objet","t").
    select("x", "t")

val t4 = yago.where("prop = '<livesIn>'").
    withColumnRenamed("sujet","t").
    withColumnRenamed("objet","u").
    select("t","u")

val t5 = yago.where("prop = '<created>'").
    withColumnRenamed("sujet","t").
    withColumnRenamed("objet","v").
    select("t", "v")

val r = t1.join(t2,
"x").join(t3,"x").join(t4,"t").join(t5,"t")
r.show(10)
```

Le résultat, limité à 10 lignes, doit être :

T	X	Y	Z	U
<Irvine_Welsh>	<Iggy_Pop>	<Germany>	<Cry-Baby>	<
<Irvine_Welsh>	<Iggy_Pop>	<Germany>	<Cry-Baby>	<
<Irvine_Welsh>	<Iggy_Pop>	<Germany>	<Suck_(film)>	<
<Irvine_Welsh>	<Iggy_Pop>	<Germany>	<Suck_(film)>	<

<Irvine_Welsh>	<Iggy_Pop>	<Germany>	<Hardware_(film)>	<
<Irvine_Welsh>	<Iggy_Pop>	<Germany>	<Hardware_(film)>	<
<Irvine_Welsh>	<Iggy_Pop>	<Germany>	<Dead_Man>	<
<Irvine_Welsh>	<Iggy_Pop>	<Germany>	<Dead_Man>	<
<Irvine_Welsh>	<Iggy_Pop>	<Germany>	<Gimme_Danger>	<
<Irvine_Welsh>	<Iggy_Pop>	<Germany>	<Gimme_Danger>	<