Requêtes hierachiques

E.Coquery

emmanuel.coquery@liris.cnrs.fr

Données organisées de manière hierarchique

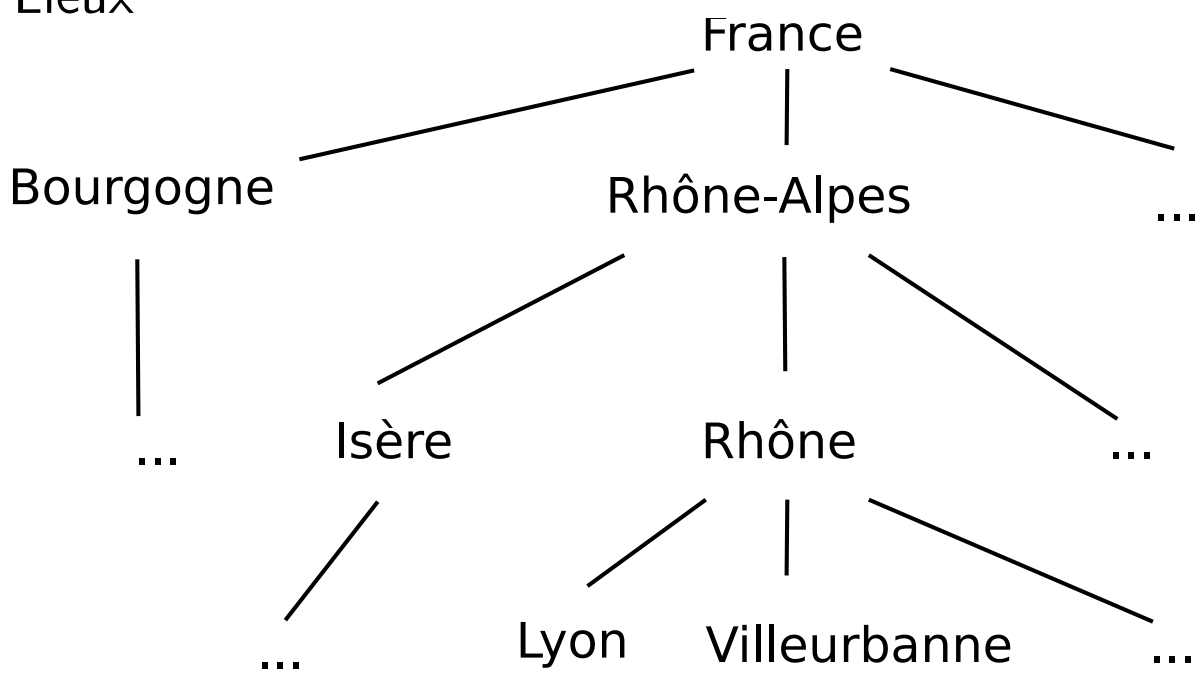
Données pour lesquelles on a un souhaite représenter des notions :

- d'inclusion
- parent-enfant
- de composition
- ...

D'une manière générale, des notions pouvant être représentées de manière arborescente.

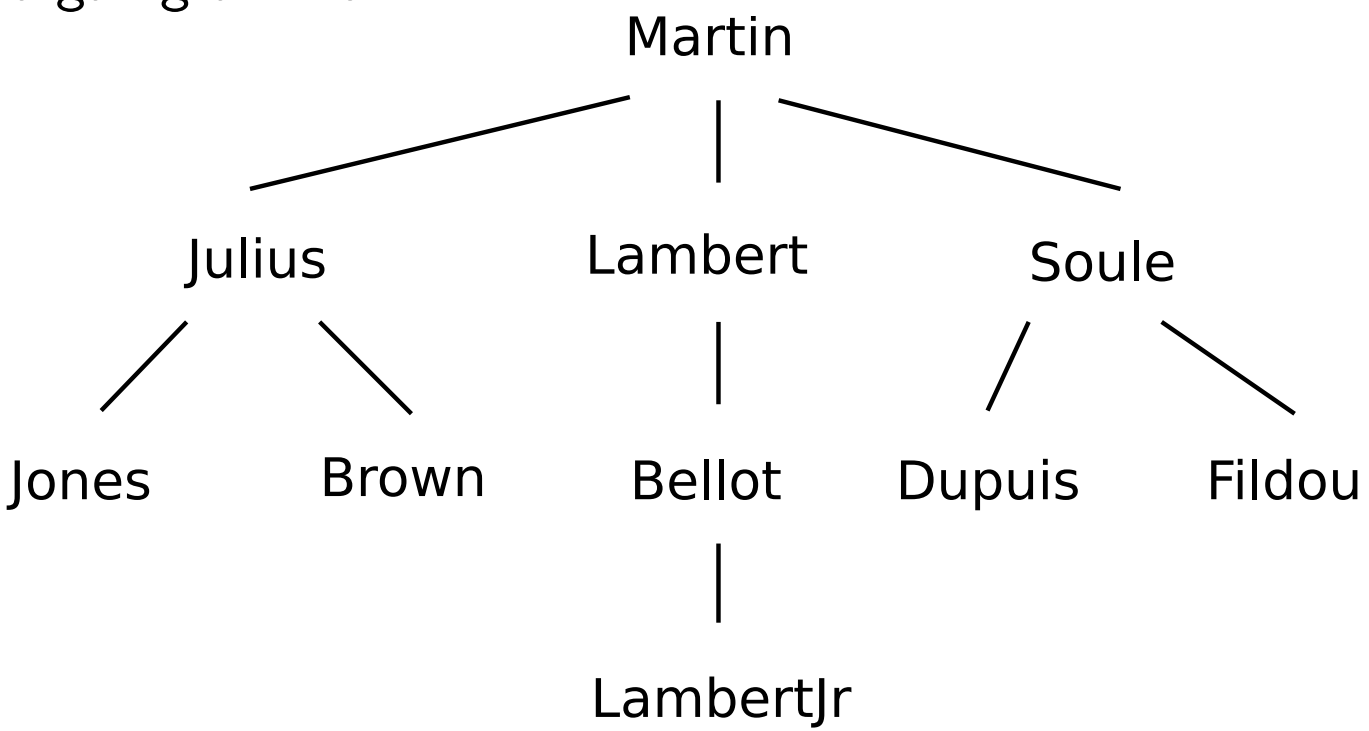
Exemple - 1

- Organiser des concepts par inclusion :
  - Lieux



Exemple - 2

- Organiser le personnel d'une entreprise suivant un organigramme :



Exemple - 3

Organiser les message dans un forum suivant les fils de discussion :

- Aujourd'hui il fait beau
  - Je suis d'accord
  - Je ne touve pas qu'il fasse beau
    - C'est parce que tu n'es pas sorti
    - C'est vrai il y a plein de nuages
  - Demain il pleuvera
- J'ai bien aimé le film d'hier
  - Je n'ai pas du tout aimé
    - Qu'est ce qui t'a déplu ?
  - J'ai bien aimé aussi

Modélisation dans le modèle relationnel

Comment représenter ces notions dans un SGBD ?

En relationnel : ajouter un attribut (ou une combinaison d'attributs) indiquant le parent d'un noeud dans l'arbre

- clé étrangère de la table sur elle-même

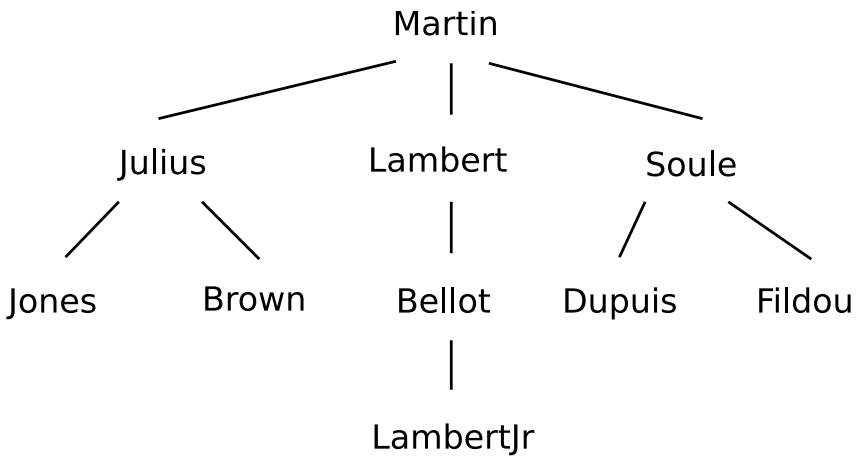
Exemples :

- Lieu(Nom, Description, Lieu_englobant)
- Employe(Nom, Num, Fonction, NSup, Embauche, Num_Dept)
- Message(Id, Titre, Contenu, Date, Auteur, Id_parent)

Exemple : employés

Employe

Nom	Num	NSup	...
Bellot	13021	25012	...
Brown	20663	12569	...
Dupuis	14028	28963	...
Fildou	25631	28963	...
Jones	19563	12569	...
Julius	12569	16712	...
Lambert	25012	16712	...
LambertJr	15630	13021	...
Martin	16712	NULL	...
Soule	28963	16712	...



Requêtes

Quels types de requêtes sur un arbre ?

- Parcours en profondeur
- Parcours en largeur
- Liste des ancêtres
  - Ancêtre racine
- Liste des descendants

Approche naïve - 1

Pour un parcours en profondeur, limité à une profondeur de 2 :

```
SELECT R2.id
FROM R R1, R R2
WHERE (R1.id = R2.id OR R1.id = R2.parent)
  AND R1.parent IS NULL
ORDER BY R1.id, MIN(ABS(R2.id - R1.id), 1), R2.id
```

Approche naïve - 2

Profondeur limitée à 3 :

```
SELECT R3.id
FROM R R1, R R2, R R3
WHERE ((R1.id = R2.id AND R2.id = R3.id)
    OR (R1.id = R2.parent AND R2.id = R3.id)
    OR (R1.id = R2.parent AND R2.id = R3.parent)
  AND R1.parent IS NULL
ORDER BY
  R1.id, MIN(ABS(R1.id - R2.id),1),
  R2.id, MIN(ABS(R2.id - R3.id), 1),
  R3.id
```

$\Rightarrow$ trop complexe et limité à une profondeur fixée

Profondeur arbitraire

Programmer le parcours de manière récursive en PHP :

```
function parcours($id_noeud) {
  print $id_noeud;
  $req = "SELECT ID
          FROM R
          WHERE PARENT=$id_noeud";
  $res = mysql_query($req);
  while ($ligne = mysql_fetch_array()) {
    parcours($ligne["ID"]);
  }
}
```

Récursion et requêtes simultanées

parcours(16712)

SELECT NUM FROM EMP WHERE NSUP=16712

12569

25012

28963

parcours(25012)

SELECT NUM FROM EMP WHERE NSUP=25012

13021

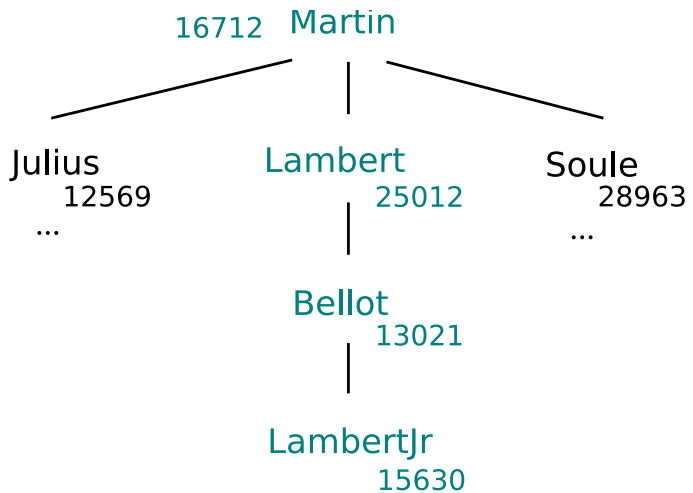
parcours(13021)

SELECT NUM FROM EMP WHERE NSUP=13021

15630

parcours(15630)

SELECT NUM FROM EMP WHERE NSUP=15630



Solution Oracle : CONNECT BY

SELECT $expr_1$, ..., LEVEL
FROM R_1, ...
WHERE Condition
CONNECT BY PRIOR e = e'

- Les n-uplets du select sont retournés en utilisant un parcours en profondeur de l'arbre défini par le lien parent-enfant suivant :
  - e est l'identifiant du père du n-uplet courant, identifiant de la valeur est e'.
  - La valeur précédente pour e (PRIOR e) est e'.

CONNECT BY - 2

SELECT $expr_1$, ...
FROM R_1, ...
WHERE Condition
START WITH $Condition_2$
CONNECT BY PRIOR e = e'
ORDER SIBLINGS BY a_1, ...

- La condition donnée par le START WITH permet de spécifier des noeuds de départ.
- Le ORDER SIBLINGS BY permet spécifier l'ordre des frères dans l'arbre

Parcours en profondeur

Employe(Nom, Num, Fonction, NSup, Embauche, Num_Dept)

```
SELECT Nom, Num, NSup, LEVEL
FROM Employe
START WITH NSup IS NULL
CONNECT BY PRIOR Num = NSup
```

Nom	Num	NSup	LEVEL
Martin	16712		1
Julius	12569	16712	2
Jones	19563	12569	3
Brown	20663	12569	3
Lambert	25012	16712	2
Bellot	13021	25012	3
LambertJr	15630	13021	4
Soule	28963	16712	2
Dupuis	14028	28963	3
Fildou	25631	28963	3

Liste des ancêtres

Employe(Nom, Num, Fonction, NSup, Embauche, Num_Dept)

Les supérieurs de LambertJr (y compris LambertJr lui-même) :

```
SELECT
FROM Employe
START WITH Nom = 'LambertJr'
CONNECT BY PRIOR NSup = Num
```

Nom	Num	NSup
LambertJr	15630	13021
Bellot	13021	25012
Lambert	25012	16712
Martin	16712	

Plus vieil ancêtre

Employe(Nom, Num, Fonction, NSup, Embauche, Num_Dept)

Le patron de LambertJr :

```
SELECT Nom,Num
FROM Employe
WHERE Num IN
          (SELECT Num
           FROM Employe
           START WITH Nom = 'LambertJr'
           CONNECT BY PRIOR NSup = Num)
AND NSup IS NULL
```

Fermeture transitive

Employe(Nom, Num, Fonction, NSup, Embauche, Num_Dept)

```
SELECT E.Nom, Sup.Nom as NomSup
FROM Employe E, Employe Sup
WHERE Sup.Num IN
        (SELECT Num
         FROM Employe
         START WITH Num = E.NSup
         CONNECT BY PRIOR NSup = Num)
ORDER BY Nom
```

Représentation à double indices

La clause CONNECT BY est spécifique à Oracle, sur la plupart des autres SGBD, il faut :

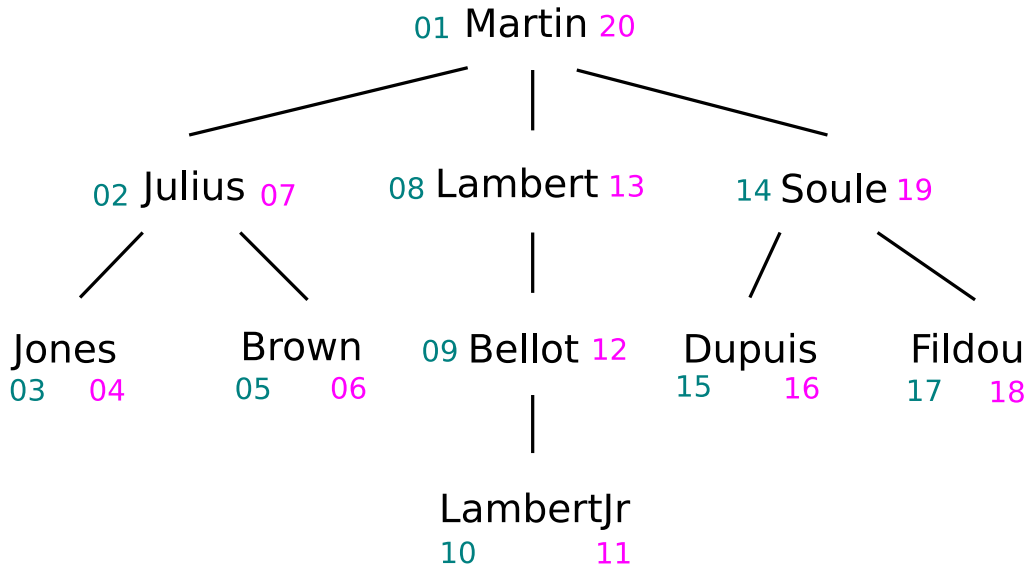
- soit utiliser des procédures stockées (voir l'an prochain)
- soit représenter les arbres différement

Une manière de représenter les arbres est d'ajouter deux indices (d et f) à chaque noeud tels que :

- toutes les valeurs de d et f sont distinctes
- pour chaque noeud, $d < f$
- si un noeud A est un ancêtre de B, alors : $d_A < d_B < f_B < f_A$

Exemple

Nom	Num	NSup	d	f
Martin	16712		1	20
Julius	12569	16712	2	7
Jones	19563	12569	3	4
Brown	20663	12569	5	6
Lambert	25012	16712	8	13
Bellot	13021	25012	9	12
LambertJr	15630	13021	10	11
Soule	28963	16712	14	19
Dupuis	14028	28963	15	16
Fildou	25631	28963	17	18



Liste des ancêtres

Les ancêtres A d'un noeud B sont tels que $d_B \in [d_A, f_A]$.

Ancêtres de LambertJr :

```
SELECT Nom,Num
FROM Employe
WHERE
    (SELECT Emp.D
     FROM Employe Emp
     WHERE Emp.Nom='LambertJr')
    BETWEEN D AND F
ORDER BY D DESC
```

Parcours en profondeur

Il suffit de remarquer que l'attribut d correspond à l'ordre de parcours en profondeur.

```
SELECT Nom,Num
FROM Employe
ORDER BY D
```

Parcours en profondeur - 2

Il faut calculer la valeur de LEVEL, qui est le nombre d'ancêtres du noeud, y compris le noeud lui-même

```
SELECT Nom,Num,
        (SELECT count(*)
         FROM Employe E1
         WHERE Employe.D BETWEEN E1.D AND E1.F)
        as LEVEL
FROM Employe
ORDER BY D
```

Plus vieil ancêtre

Le patron de LambertJr :

```
SELECT Nom,Num
FROM Employe
WHERE D =
            (SELECT MIN(Emp.D)
             FROM Employe Emp
             WHERE (SELECT Lamb.D
                    FROM Employe Lamb
                    WHERE Lamb.Nom='LambertJr')
             BETWEEN D AND F)
```

Fermeture transitive

Plus simple qu'avec le CONNECT BY PRIOR

```
SELECT Emp.Nom, Sup.Nom AS NomSup
FROM Employe Emp, Employe Sup
WHERE Emp.Num <> Sup.Num
AND Emp.D BETWEEN Sup.D AND Sup.F
ORDER BY Emp.Nom
```

Insertion

Pour insérer un noeud B sous un noeud A :

- Faire de la place pour B, en décalant tous les indices plus grands que f_A
- Insérer B avec les bonnes valeurs pour d et f (la valeur de d_B est l'ancienne valeur de f_A)

Les insertions peuvent être très coûteuses

Insertion - 2

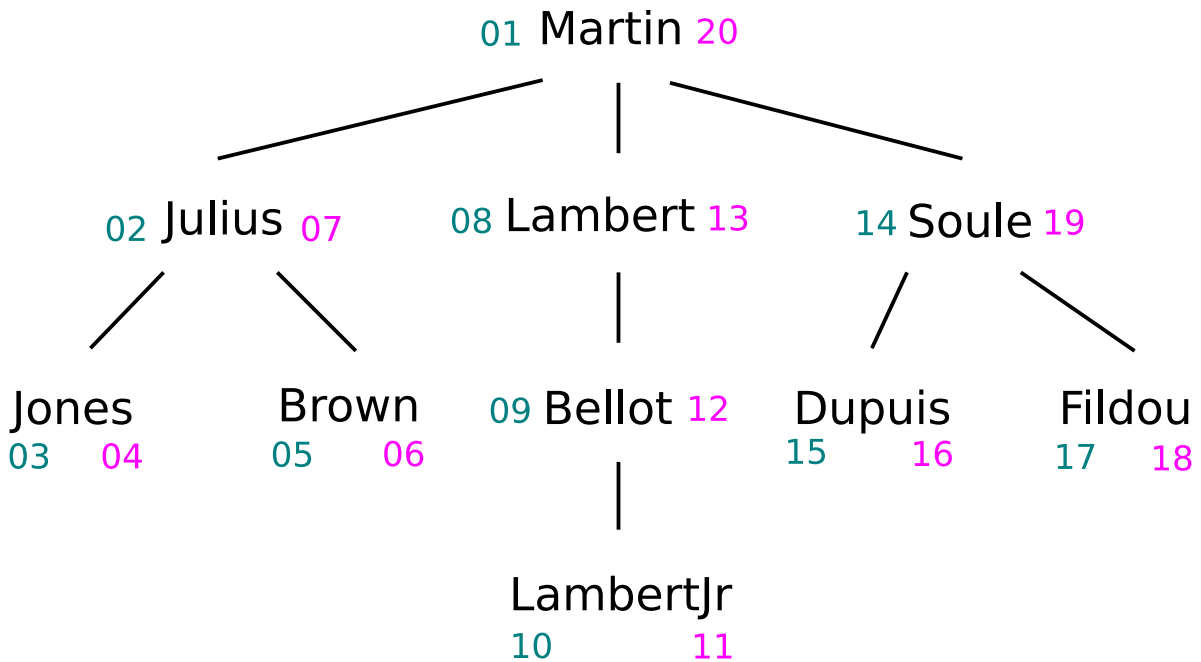
UPDATE Employe
SET D = D+2
WHERE D >= f_A

UPDATE Employe
SET F= F+2
WHERE F >= f_A

INSERT INTO Employe(...,d,f)
VALUES(...,f_A,f_A+1)

Dans la dernière requête, on utilise l'ancienne valeur de f_A.

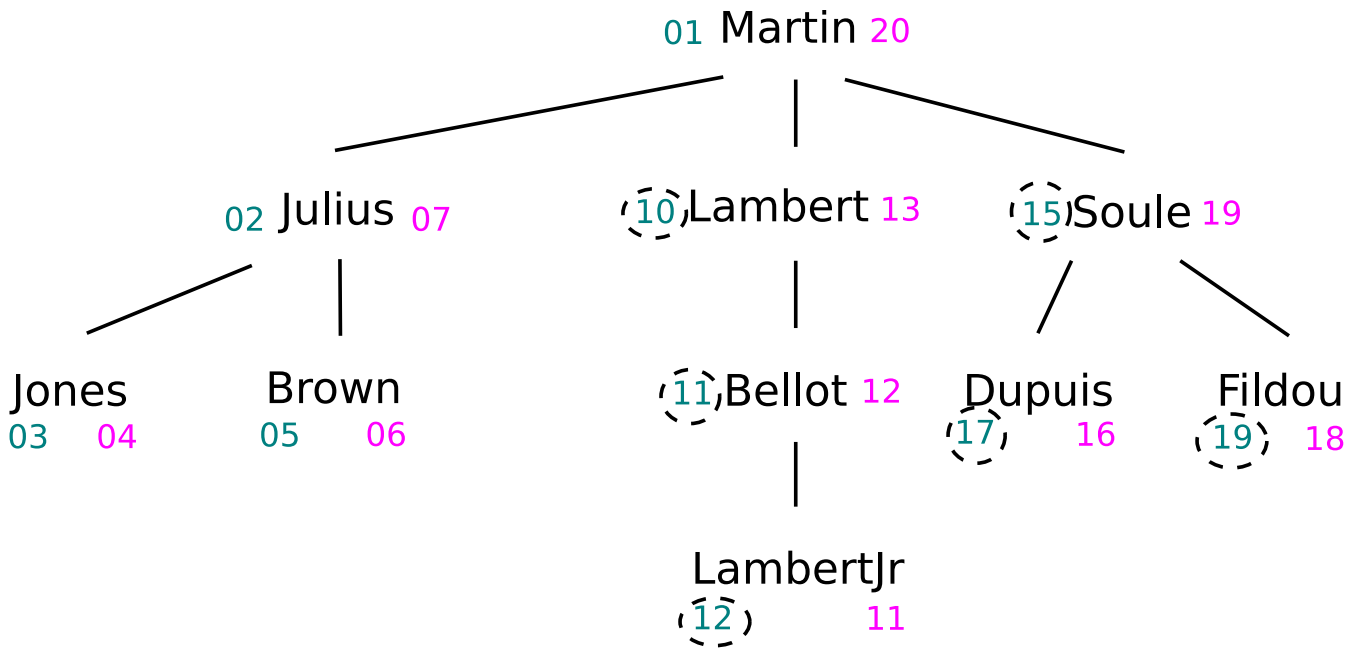
Insertion : exemple



Insertion de 'Toto' comme subordonné de 'Julius'

$f_A = 7$

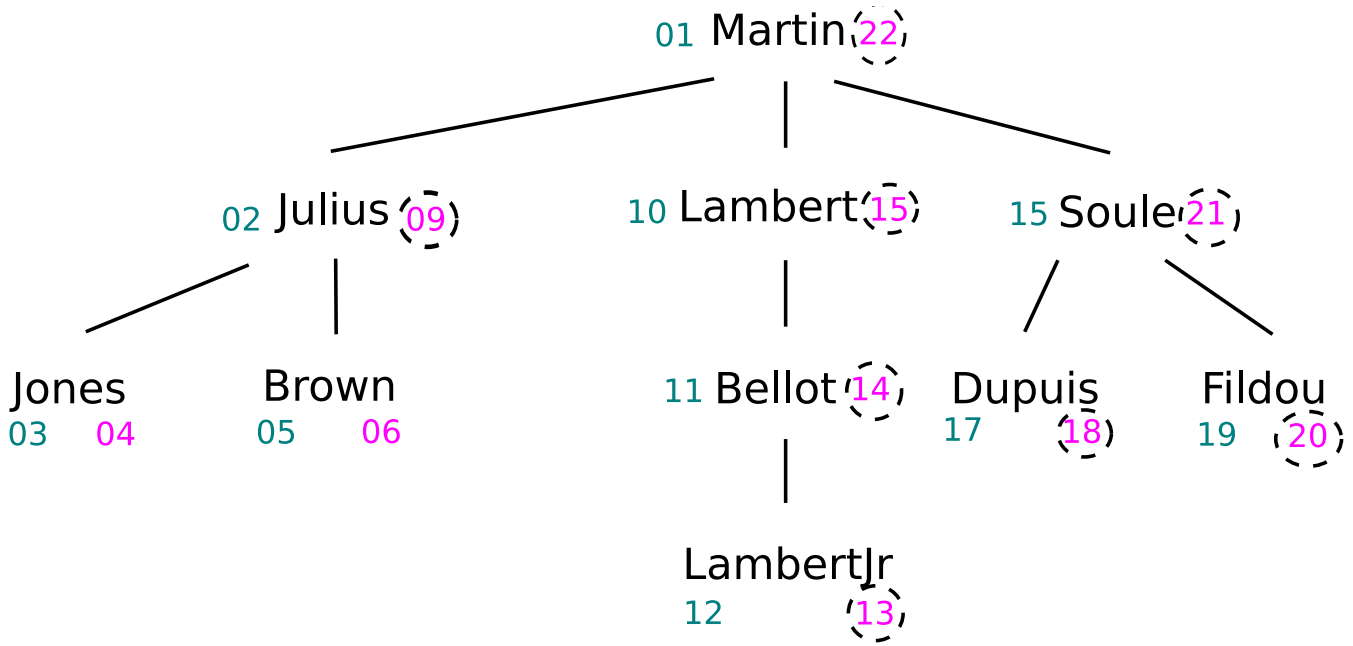
Insertion : exemple - décalage d



Insertion de 'Toto' comme subordonné de 'Julius'

$f_A = 7$

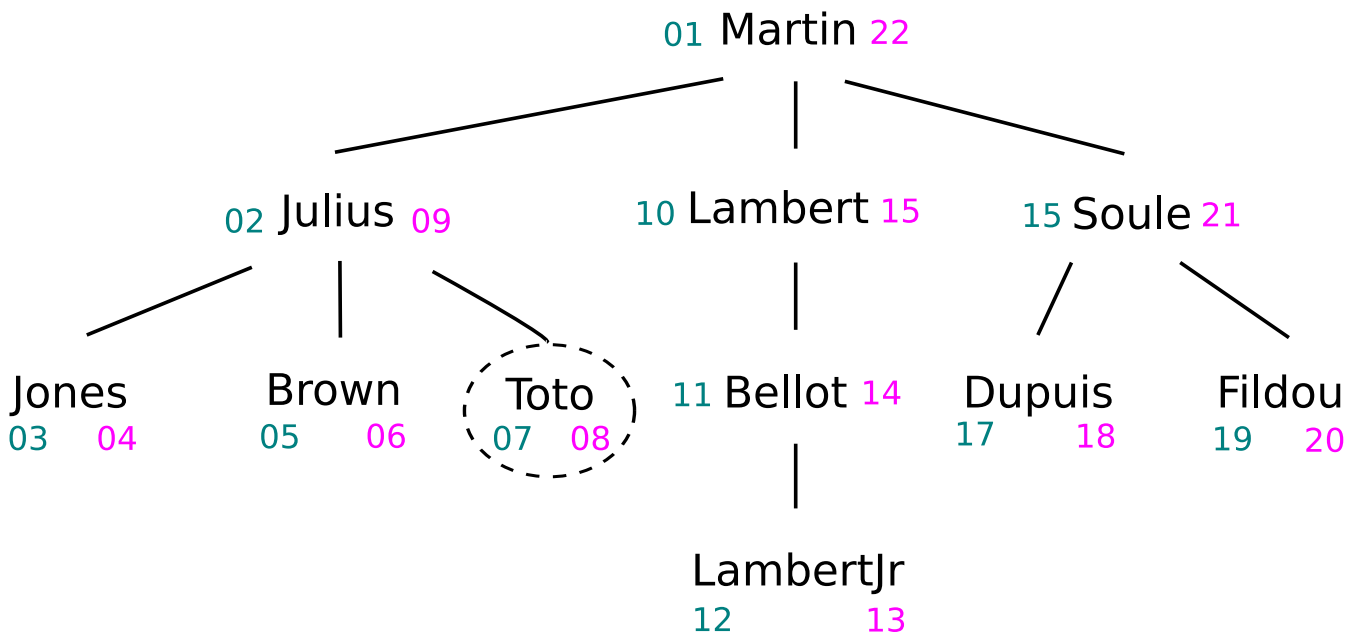
Insertion : exemple - décalage f



Insertion de 'Toto' comme subordonné de 'Julius'

$f_A = 7$

Insertion : exemple - ajout



Insertion de 'Toto' comme subordonné de 'Julius'

$f_A = 7$