Bases de Données

C2 - Conception et Modélisation - Requêtes hiérarchiques

Lina Soualmia

Université de Rouen
LITIS - Équipe TIBS-CISMeF
lina.soualmia@chu-rouen.fr

11 septembre 2015

Conception et Modélisation

- 1. Étape conceptuelle : Conception et Modélisation de bases de données
  Utilisation de :
  - Méthodes, Modèles, Formalismes
  - Modèle Entité-Association E/A, Modèle Entité-Association étendu
  - Modèles Objet, Formalisme UML
    Power AMC, Power Designer WinDev, Oracle Designer Rational Rose, ...
- 2. Étape logique : Implantation d'une base de données
  - Modèle Relationnel / Modèle Objet-Relationnel / Modèle Objet
  - Optimisation du schéma (Normalisation, Dénormalisation ...)

- 3. Étape physique :
  - SGBD Relationnel / SGBD Objet-Relationnel / SGBD Orienté Objet
  - Langages (SQL, PL/SQL, PRO*C, JDBC, Java, ...)
  - Optimisations (Groupement, Index, ...)
  - Administration
    Oracle, DB2, MySQL
- 4. Logiciels (SGBD, Interfaces, ...) & Matériels

Conception du schéma des bases

- C'est une des tâches essentielles des développeurs de bases de données
- Objectif : structuration du domaine d'application afin
  - de le représenter sous forme de types et de tables
  - d'accompagner ces structures de contraintes sur les données et d'en tirer plus de sémantique

La représentation

Elle doit être :
- **juste** pour éviter les abérations sémantiques, notamment dans les résultats des requêtes ;
- **complète** pour permettre le développement des programmes d'application souhaitées ;
- **évolutive** pour supporter la prise en compte rapide de nouvelles demandes.

La démarche

Démarche de conception traditionnelle :
- par abstractions successives
- en descendant depuis les problèmes de l'utilisateur vers le Système de Gestion de Bases de Données.

Cinq étapes :

1. Perception du monde réel et capture des besoins
2. Élaboration du schéma conceptuel
3. Conception du schéma logique
4. Affinement du schéma logique
5. Élaboration du schéma physique

Remarque

- Étape 1 : plutôt relative au domaine du génie logiciel
- Étapes 2, 3, 4 et 5 : relatives au domaine des bases de données

Étape 1 : Perception du monde réel et capture des besoins

- Étude des problèmes des utilisateurs
- Compréhension de leurs besoins
  $\longrightarrow$ Mise en place d'entretiens, d'analyses des flux d'information et des processus métier

Difficulté : Compréhension du problème dans son ensemble
$\longrightarrow$ Réalisation des études de cas partiels par les concepteurs
Résultat : ensemble de vues ou schémas externes devant être intégrés dans l'étape suivante
Vues exprimées dans un modèle de de données : de type éntite-association ou objet, selon la méthode choisie.

Étape 2 : Élaboration du schéma conceptuel

- Intégration des schémas externes obtenus a l'étape précédente
- Chaque composant est un schéma conceptuel : diagramme entité-association ou diagramme de classes
- Résultat : modèle de problème représentant une partie de l'application
- Difficulté : intégration de toutes les parties dans un schema conceptuel global complet, non redondant et cohérent

NB : des allers et retours avec l'étape précédente sont souvent nécessaires.

Étape 3 : Conception du schéma logique

Transformation du schéma conceptuel en structures de données supportées par le système choisi : le schéma logique.

- Avec un SGBD relationnel : passage à des tables.
- Avec un SGBD relationnel-objet : génération de types et de tables. (les types sont réutilisables)
- Avec un SGBD objet : génération de classes et d'associations

NB : Cette étape peut être complètement automatisée.

Étape 4 : Affinement du schéma logique

- Vérification : le schéma logique est-il un "bon" schéma ?
- Définition en première approximation : *un bon schéma est un schéma sans oublis ni redondances d'informations*
- Plus précisément : *un schéma est bon si le modèle relationnel associé respecte au moins la troisième forme normale* et la forme normale de Boyce-Codd (à voir plus loin)
- Objectif en relationnel : regrouper ou décomposer les tables de manière a représenter fidèlement le monde réel modélisé

Étape 5 : Élaboration du schéma physique

- Prise en compte de toutes les transactions concernant les applications traitées
- Permet de déterminer les accès fréquents
- Choix des bonnes structures physiques :
  - groupement ou partitionnement de tables
  - choix des index, etc.
- C'est le point essentiel pour obtenir de bonnes performances.

Schéma Conceptuel

- Modélisation du problème en utilisant les spécifications des besoins obtenues a l'étape 1 (capture des besoins)
- Deux possibilités :
  - utilisation du formalisme Entité Relation (ou Entité Association)
    $\longrightarrow$ production d'un diagramme ER/EA
  - utilisation du formalisme UML
    $\longrightarrow$ production d'un digramme de classes

Indépendance du modèle conceptuel par rapport au schéma physique

Phases d'élaboration du schéma conceptuel

- Identification des entités ou classes
- Identification des associations
- Identification des attributs pour chacune des entités ou classes
- Définition des identifiants

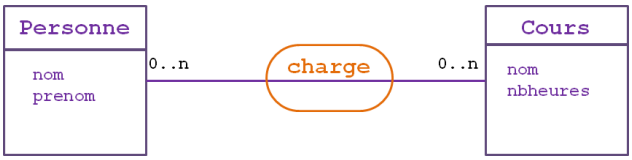
Identification des entités ou classes

- Entités : élement abstrait ou concret (objet, évènement, etc.) reconnu distinctement
  Exemples : Jean Dupont, Michel Durant
- Type-entités : ensemble des entités ayant les mêmes caractéristiques
  Ex. : `Personne (nom, prenom)`
- Par abus de langage, on parle souvent d'entités à la place de type-entités

Dans l'étape 1, il s'agit de la description des éléments.

Identification des associations

- Association : Lien logique entre deux entités
- Type-Association : Ensemble d'associations ou de relations possédant les mêmes caractéristiques.
- Association/type-association : même abus de langage
- À l'étape 1 : une phrase simple reliant deux entités
  Ex. : `un professeur est en charge de cours` (lien entre les entités `professeur` et `cours`)
- Plusieurs types d'association existent



Types d'associations

- **unaire** : relation au sein d'une même entité
  Ex. : `un employé supervise un employé`
- **binaire** : relation entre deux entités (différentes)
  Ex. : `un client passe plusieurs commandes`
- **ternaire** : relation entre trois entités (différentes)
  Ex. : `un internaute note un film à différentes dates` (on veut conserver l'historique des notes).

Cardinalité d'un type-association

- Cardinalité : nombre minimal et maximal de fois qu'une entité peut intervenir dans une association de ce type
  Ex. : `un client peut commander 1 à n produits`
- Remarques :
  - la cardinalité minimale doit être inférieure à la cardinalité maximale
  - la cardinalité doit être associée à chaque patte de la relation

Cardinalité minimale/maximale

- Cardinalité minimale :
  - 0 : une entité peut exister tout en n'étant impliquée dans aucune association
  - 1 : une entité ne peut exister que si elle est impliquée dans au moins une association
  - n : une entité ne peut exister que si elle est impliquée dans plusieurs associations (cas rare, à eviter car cela pose des problèmes)
- Cardinalité maximale :
  - 0 : une entité ne peut pas être impliquée dans une association (normalement inexistant sinon problème de conception)
  - 1 : une entité peut être impliquée dans au maximum une association
  - n : une entité peut être impliquée dans plusieurs associations

Identification des attributs

- Attribut : caractéristique associée a une entité
  Ex. : *`nom, prenom, age`*
- Domaine associé à un attribut : ensemble des valeurs possibles
- Chaque attribut doit posséder une valeur compatible avec son domaine
- Remarque : Éviter absolument les attributs calculés. Toujours utiliser des données primaires - les attributs qui servent à les calculer.

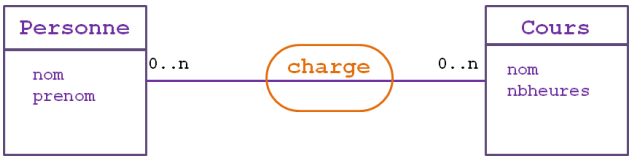
Définition de l'identifiant

- Identifiant : liste des attributs devant avoir une valeur unique dans chaque entité
  Ex. : *numéro d'immatriculation d'une voiture, numéro de sécurité sociale*
- Remarques :
  - On utilise plutôt le terme **clé** que identifiant
  - Chaque type doit posséder un identifiant (formé d'un ou de plusieurs attributs)
  - L'identifiant d'une association est la concaténation des identifiants des entités liées
    NB : on peut définir un identifiant plus naturel

Remarques sur la conception

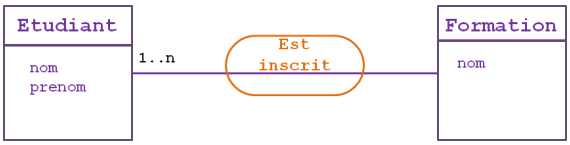
- Un attribut ne peut être partagé entre deux entités ou associations
  - problème de redondance
- En cas de difficulté à choisir entre entité et association :
  - utiliser le contexte pour y répondre
- En cas de difficulté à trouver un identifiant pour un type-entité :
  - ne s'agirait-il pas d'une association ?
- Association dont toutes les pattes ont une cardinalité 1,1 :
  - l'association et les entités liées ne correspondraient-ils pas a une seule entité ?

- Formalisme ER :



- Formalisme UML :

(une des cardinalités est volontairement absente)



Tout `etudiant` participe au moins une fois à l'association `est inscrit`.

Tout étudiant est inscrit dans au moins une formation

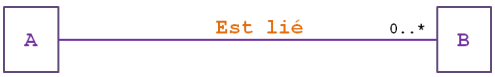
Autrement dit : une instance d'étudiant peut être associée à plusieurs formations.

Formalisme ER :

Interprétations :

- A est lié 0 à n fois à B
- La connaissance de B permet de définir A
- La clé de B définit l'instance de A

Formalisme UML :



ER ou UML ?

- Si conception de bases de données :
  - utilisation du modèle entité/relation
  - On met l'accent sur le système d'information (stockage, traitement, réception, diffusion de l'information)
- Si conception objet et programmation :
  - utilisation de UML (2 incluant l'héritage).
  - On met l'accent sur les structures de données et la programmation

Élaboration du schéma logique

Transformation du modèle conceptuel en une structure de données basée sur un modèle de données spécifique (par exemple relationnel)

- Réalisation de la transformation a l'aide de règles formelles
  $\longrightarrow$ Possibilité d'automatisation de cette étape (Objecteering, Rational Rose)
- Indépendant de la couche physique
- Résultat : modèle logique de la base de données

Passage au relationnel

- Implémentation des entités et associations sous forme de tables
- Les attributs correspondent aux colonnes des tables
  - le nom de l'attribut est le nom de la colonne
  - l'ensemble des valeurs possibles est le domaine
- Exemple :
  - *Professeur(numProf, nom, prenom)*
  - *Cours(nomCours, nom, nbheures)*
  - *Charge(numProf, numCours)*

Principe

- Traduction des associations :
  Règle de base : représentation des associations par une table dont
  - le schéma est le nom de l'association
  - la liste des clés des entités participantes suivie des attributs de l'association
- Amélioration :
  - Regrouper les associations 1..n avec la classe cible
- Exemple :
  - *Voiture(numV, Marque, modele)*
  - *Possede(numProp, numV, Date)*
    ⟶ les deux tables peuvent être regroupées si toutes les voitures n'ont qu'un et un seul propriétaire

Formes normales

Respecter les formes normales

Pourquoi normaliser ?

- pour limiter les redondances des données
- pour limiter les pertes de données
- pour limiter les incohérences au sein des données
- pour améliorer les performances des traitements

8 formes normales :

- Formes normales 1 à 3
- Forme normale de Boyce-Codd
- Formes normales 4/5(/6)
- Forme normale de domaine-cle

Objectifs des trois premières formes normales : permettre la décomposition de relations sans perte d'informations

Une relation en forme normale de niveau N est forcément de forme normale de niveau N - 1

Première forme normale (1FN)

Une relation est en première forme normale si tous ses attributs contiennent des valeurs

- simples et non-décomposables (utiliser une liste ou une table externe)
- non-répétitives
- constantes dans le temps (date de naissance plutôt que l'âge)

Exemple

`Vol(NoVol, CodeAeroDep, CodeAeroArr, HeureDep, HeureArr, Jours)`

devient

`Vol(NoVol, CodeAeroDep, CodeAeroArr, HeureDep, HeureArr)`

`Vol(NoVol, Jour)`

Deuxième forme normale (2FN)

Une relation est en deuxième forme normale si et seulement si :

- elle est en première forme normale
- tout attribut non clé est complètement dépendant de toute la clé

Autrement dit, une des trois conditions doit être respectée :

- La clé primaire n'est formée que d'un seul attribut
- La clé primaire contient tous les attributs de la table
- Si la clé a plus d'un attribut, une dépendance fonctionnelle ne doit jamais exister entre une partie seulement de la clé et un autre attribut de la table.

Exemple

`Avion(Constr, Modele, Conso, Capacite, VitesseMax)`
⟶ il y a une dépendance fontionnelle entre *Modèle* et *Capacité*

On divise la table en deux :

`Avion(Constr, Modele)`

`ModeleAvion(Modele, Conso, Capacite, VitesseMax)`

Troisième forme normale (3FN)

Une relation est en troisième forme normale si et seulement si :

- elle est en deuxième forme normale
- tout attribut n'appartenant pas à une clé ne dépend pas d'un attribut non clé

⟶ les dépendances fonctionnelles entre deux attributs ordinaires (ne faisant par partie de la clé) ne sont pas autorisées

Exemple

`Enseignant(Nom, Categorie, Classe, Salaire)`

⟶ Le salaire dépend de la *Catégorie* et de la *Classe*

devient

`Enseignant(Nom, Categorie, Classe)`

`Salaire(Categorie, Classe, Salaire)`

Forme normale de Boyce-Codd (BCNF)

- Extension plus rigide de la troisième forme normale (définie par R.F. Boyce et E.F. Codd - en partant du constat que la 3FN comportait certaines anomalies)
- Une relation est en forme normale de Boyce-Codd si et seulement si :
  - *aucun attribut faisant partie de la clé ne dépend d'un attribut ne faisant pas partie de la clé primaire*

Un modèle relationnel en FNBC est considéré comme étant de qualité suffisante pour une implantation
Les cas de relations en 3FN qui ne sont pas déjà en FNBC sont très rares

Exemple

`R(A, B, C, D)`

Avec les dépendances : B, C → A ; B, C → D ; D → B, (ce qui entraîne de nombreuses redondances)

On propose les relations :

`R(A, B, C)`

`R'(D, B)`

`ReservationCourtTennis(NomCourt, HeureDebut, HeureFin, ClasseTauxHoraire)`

La classe du taux horaire (SILVER, GOLD, PREMIUM) détermine les courts disponibles.

On propose les relations :

`ReservationCourtTennis(NomCourt, HeureDebut, HeureFin)`

`ClasseCourt(ClassTauxHoraire, NomCourt)`

Objectifs :

- Rechercher de bonnes performances
- Prendre en compte les transactions
- Indexer, dénormaliser, grouper, partitionner les tables

Résultat : modèle physique optimisé de la base de données

Schéma relationel :

`COURS(NumCours, NomC, NbHeures, Annee)`

`PROFESSEURS(NumProf, NomP, Specialite, DateEntree, DerPromo, SalaireBase, SalaireActuel)`

`CHARGE(NumProf, NumCours)`


```
create table  COURS
(NumCours NUMBER(2) NOT NULL,
  NomC  VARCHAR2 (20) NOT NULL,
  NbHeures NUMBER(2),
  Annee NUMBER(1),
  constraint  PKCours  primary key (NumCours));
create table  PROFESSEURS
(NumProf NUMBER(4) NOT NULL,
  NomP  VARCHAR2 (25) NOT NULL,
  Specialite  VARCHAR2 (20),
  DateEntree DATE,
  DerPromo DATE,
  SalaireBase NUMBER
  SalaireActuel NUMBER
  constraint  PKProfesseurs  primary key (NumProf));
```

```
create table  CHARGE
(NumProf  NUMBER(4) NOT NULL,
  NumCours NUMBER(4) NOT NULL,
  constraint PKCharge primary key (NumCours, NumProf));

alter table  CHARGE
  add constraint FKChargeCours foreign key (NumCours)
  references COURS(NumCours);

alter table  CHARGE
  add constraint FKChargeProfesseur foreign key  (NumProf)
  references PROFESSEURS (NumProf);
```



```
create table  COURS
(NumCours NUMBER(2),
  NomC VARCHAR2(20),
...
  constraint PKCours primary key (NumCours),
  constraint NNCoursNomC check (NomCours IS NOT NULL));

create table PROFESSEURS
(NumProf NUMBER(4),
  NomP VARCHAR2(25),
...
  constraint  PKProfesseurs primary key (NumProf),
  constraint  NNProfesseursNomP check (NomP IS NOT NULL));
```



```
create table  CHARGE
(NumProf  NUMBER(4),
  NumCours NUMBER(4),
  constraint PKCharge primary key (NumCours, NumProf));

alter table  CHARGE
  add constraint FKChargeCours foreign key (NumCours)
  references COURS(NumCours);

alter table  CHARGE
  add constraint FKChargeProfesseur foreign key  (NumProf)
  references PROFESSEURS (NumProf);
```


Schéma relationnel-objet

COURS(NumCours, NomC, NbHeures, Annee)

PROFESSEURS(NumProf, NomP, Specialite, DateEntree, DerPromo, SalaireBase, SalaireActuel, Ensemble(COURS))


```
create type  CoursType as object
(NumCours NUMBER(2), NomCoursVARCHAR2(20), NbHeures
NUMBER(2), Annee NUMBER(1));
```

Création du type table

```
create type  LesCoursType as table of CoursType;

create type  ProfesseurType as object
(NumProf NUMBER(4), NomPVARCHAR2(25), Specialite
VARCHAR2(20),..., Cours LesCoursType);
```

Création de la table maître

```
create table  Professeur of ProfesseurType
(primary key(NumProf)),
nested table Cours store as TableCoursP;
```

Cours est une colonne de type table qui fait le lien entre la table maître et la table imbriquée ; TableCoursP est le nom de la table qui contient les lignes de la table imbriquée

Requêtes Hiérarchiques

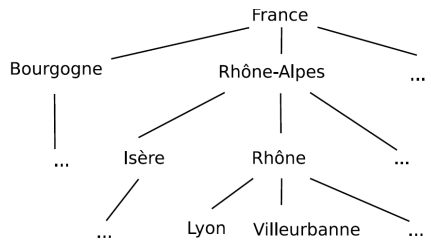
Données hiérarchiques

Les données organisées de manière hiérarchique sont des données pour lesquelles on a un souhaité représenter des notions :

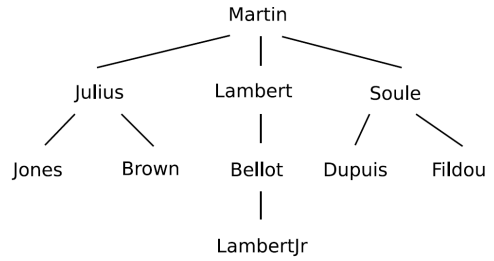
- d'inclusion
- parent-enfant
- de composition

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

Organiser des concepts par inclusion : Lieux



Organiser le personnel d'une entreprise suivant un organigramme :



Organiser les messages 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

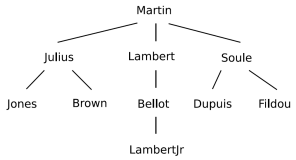
Comment représenter ces notions dans un SGBD ?

- ajouter un attribut (ou une combinaison d'attributs) indiquant le père d'un noeud dans l'arbre
- clé étrangère de la table sur elle-même

Exemples :

- LIEU(Nom,Description,LieuEnglobant)
- EMPLOYE(Nom,Num,Fonction,NumSup,Embauche,Dept)
- MESSAGE(Id,Titre,Contenu,Date,Auteur,IdParent)

Nom	Num	NumSupérieur	..
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	



Quels types de requêtes sur un arbre ?

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

Approche naïve : parcours en profondeur

Exemple de requête pour 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
```


Approche naïve : parcours en profondeur

Exemple de requête pour une profondeur limitée à 3 :

```
select R3.id
from  R R1,R R2,R R3
where ((R1.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
```

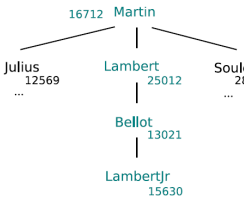
$\Longrightarrow$ trop complexe et limité à une profondeur fixée

Exemple pour une profondeur arbitraire : programmer le parcours de manière récursive

```
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
```




```
SELECT expr1, ..., LEVEL
FROM R1, ...
WHERE Condition
CONNECT BY PRIOR e = e'
```

Solution Oracle : CONNECT BY

- 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
  - La valeur précédente pour e (PRIOR e) est e'.


```
SELECT expr1, ...
FROM R1, ...
WHERE Condition
START WITH Condition2
CONNECT BY PRIOR e = e''
ORDER SIBLINGS BY a1, ...
```

La condition donnée par le START WITH permet de spécifier des nœuds de départ.

Le ORDER SIBLINGS BY permet spécifier l'ordre des frères dans l'arbre

EMPLOYE(Nom,Num,Fonction,NSup,Embauche,Dept)

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

Nom	Num	NumSupérieur	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,Dept)

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

```
SELECT Nom, Num, NSup
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	

Le plus vieil ancêtre (~racine de l'arbre)

EMPLOYE(Nom,Num,Fonction,NSup,Embauche,Dept)

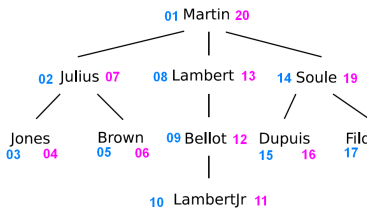
Pour LambertJr :

```
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
```


Représentation à doubles indices

- La clause CONNECT BY est spécifique à Oracle, sur la plupart des autres SGBD, il faut :
  - soit utiliser des procédures stockées
  - soit représenter les arbres différement
- Une manière de représenter les arbres est d'ajouter deux indices (d et f) à chaque nœud tels que :
  - toutes les valeurs de d et f sont distinctes
  - pour chaque nœud, $d < f$
  - si un nœud A est un ancêtre de B, alors : $d_A < d_B < f_B < f_A$

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



Les ancêtres A d'un nœud B sont tels que $d_B \epsilon [d_A, f_A]$

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 nœud, y compris le nœud 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 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)
```


Insertion

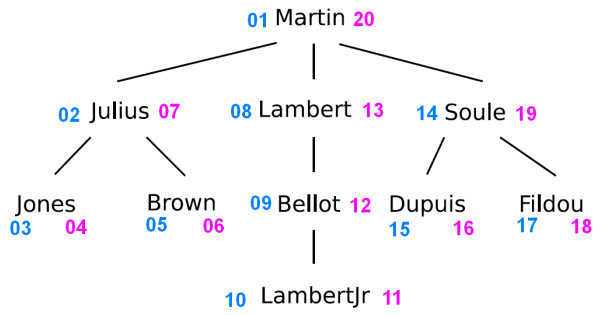
- Pour insérer un nœud B sous un nœud A :
  - Faire de la place pour B, en décalant tous les indices plus grands que f_A
  - Insérer B avec les bonnes valeurs pour d et f (la valeur de d_B est l'ancienne valeur de f_A)
  - Les insertions peuvent être très coûteuses

Insertion - 2

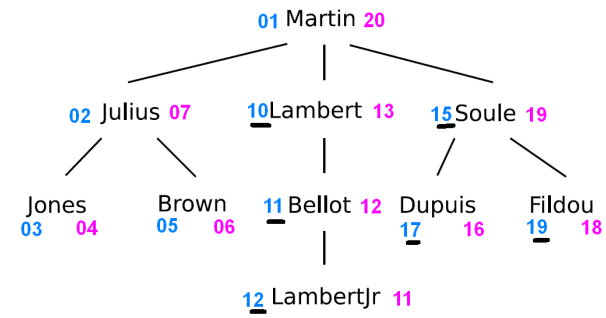
```
UPDATE Employe
SET d = d+2
WHERE d>=fA

UPDATE Employe
SET f= f+2
WHERE f >= fA

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

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

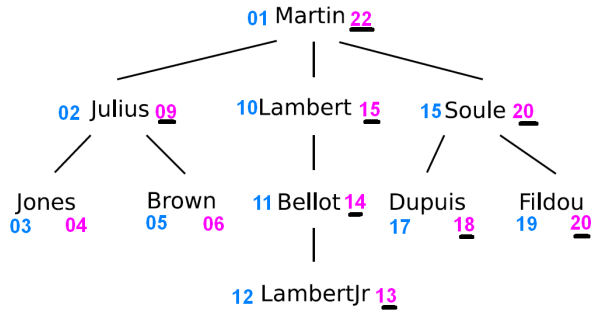
Insertion de 'Toto' comme subordonné de 'Julius' $f_A = 7$



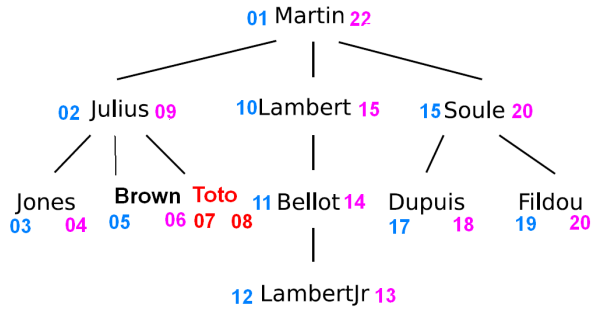
Exemple - décalage *d*



exemple - décalage f



Ajout



Méthode par préfixe

- Méthode alternative de représentation sans `CONNECT BY`.
- Utilisée dans certaines BD natives XML (M1 GIL Langages Web 2 ; M2 GIL BDXML).
- Chaque nœud est representé par une chaîne donnant le chemin d'accès pour arriver au nœud depuis la racine.
- Chemin de la forme "xy..." où :
  - x est le numéro du fils de la racine
  - y est le numéro du petit-fils de la racine etc...
- Si un nœud A est ancêtre d'un nœud B, le chemin d'accès de A est un préfixe de celui de B

Nom	Num	NSup	Chemin
Martin	16712		1
Julius	12569	16712	11
Jones	19563	12569	111
Brown	20663	12569	112
Lambert	25012	16712	12
Bellot	13021	25012	121
LambertJr	15630	13021	1211
Soule	28963	16712	13
Dupuis	14028	28963	131
Fildou	25631	28963	132

Liste des ancêtres

- Les ancêtres A d'un nœud B sonts tels que Chemin(A) est un préfixe de Chemin(B).

Ancêtres de LambertJr :

```
SELECT E.Nom, E.Num
FROM Employe LJ, Employe E
WHERE LJ.Chemin like (E.Chemin||'%') AND LJ.Nom = 'LambertJr'
ORDER BY E.Chemin DESC
```


Parcours en profondeur

L'ordre alphabétique sur les Chemins correspond à un parcours en profondeur.

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


Plus vieil ancêtre

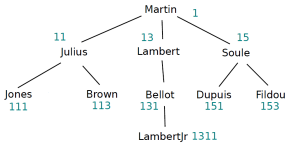
Pour LambertJr :

```
SELECT P.Nom, P.Num
FROM Employe LJ, Employe P
WHERE P.Chemin = substring(LJ.Chemin from 1 for 1)
AND LJ.Nom = 'LambertJr'
```

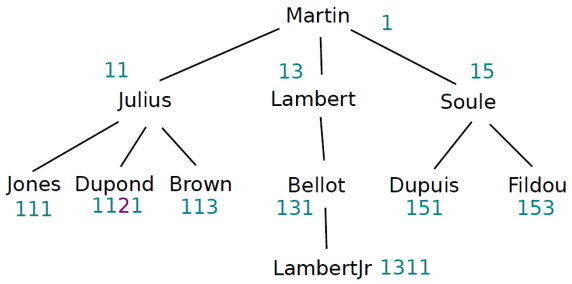

Optimisation de l'insersion

- Insersion en début :
  - Autoriser des indices de fils négatifs.
  - Démarrer l'indiçage au milieu de l'intervalle d'indice.
  - Similaire complément à 2.
- Insersion au milieu :
  - Utiliser uniquement des indices impairs
  - Plutôt que de décaler en cas d'insertion entre deux fils : Insérer un faux nœud d'indice pair
  - Faire la vraie insertion dans ce faux nœud de manière classique

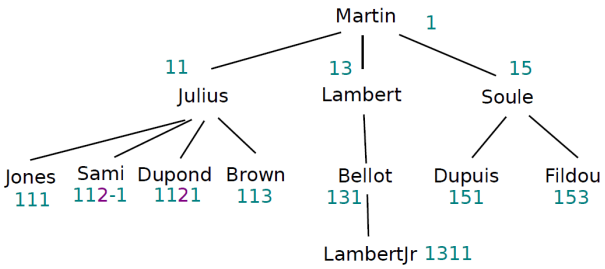
Nom	Num	NSup	Chemin
Martin	16712		1
Julius	12569	16712	11
Jones	19563	12569	111
Brown	20663	12569	112
Lambert	25012	16712	13
Bellot	13021	25012	131
LambertJr	15630	13021	1311
Soule	28963	16712	15
Dupuis	14028	28963	151
Fildou	25631	28963	152



Insertion de 'Dupond' comme subordonné de 'Julius' entre 'Jones' et Brown



Insertion de 'Sami' comme subordonné de 'Julius' entre 'Jones' et 'Dupond'



Insertion de 'Falke' comme subordonné de 'Julius' entre 'Sami' et 'Dupond'