Bases de Données

C3 - Requêtes hiérarchiques - Intro PL/SQL

Lina Soualmia

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

14 septembre 2016

Requêtes Hiérarchiques

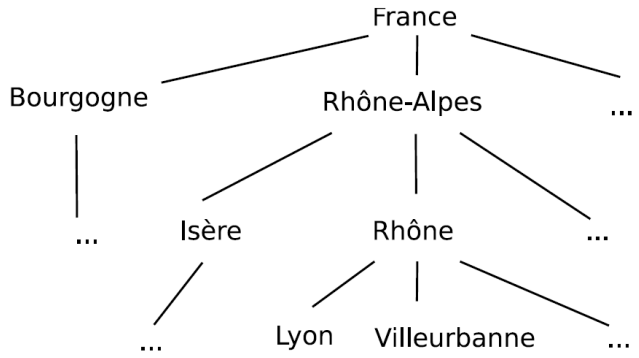
Données hiérarchiques

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

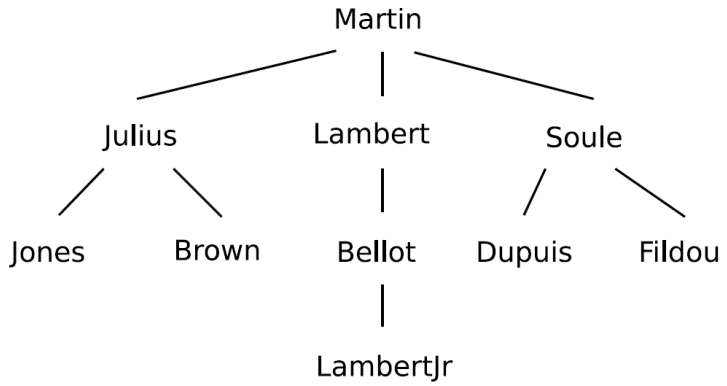
- d'inclusion
- parent-enfant
- de composition

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 trouve 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 pleuvra
- 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

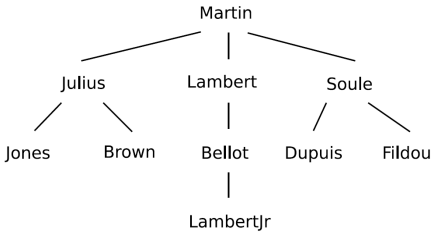
Comment représenter ces notions dans un SGBD ?

- 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
fromR 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
```

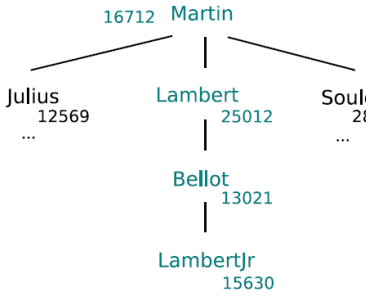
= trop complexe et limité à une profondeur fixée

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)

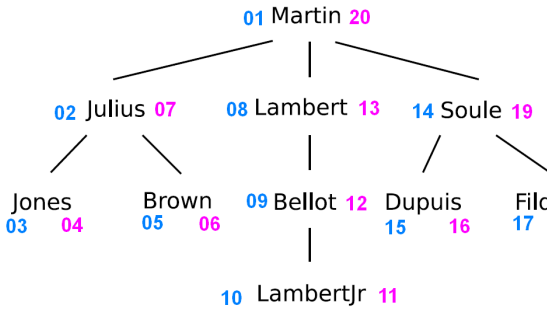
Pour LambertJr :

```
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 différemment
- 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 $d_B \in [d_A, f_A]$

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 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

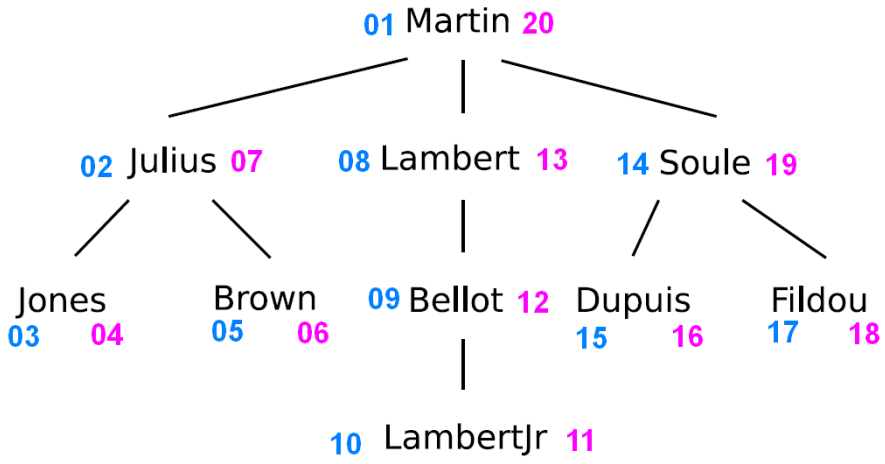
- Pour insérer un nœud B sous un nœud A :
  - Faire de la place pour B, en décalant tous les indices plus grands que f_A
  - Insérer B avec les bonnes valeurs pour d et f (la valeur de d_B est l'ancienne valeur de f_A)
  - Les insertions peuvent être très coûteuses

Insertion - 2

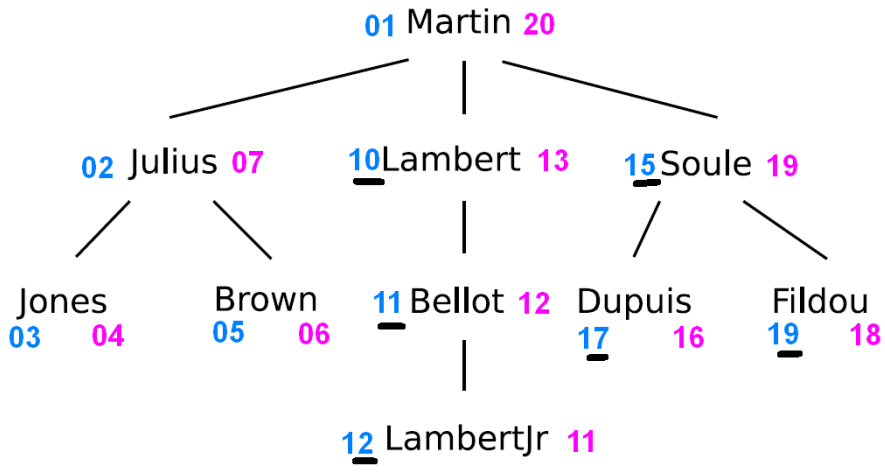
```
UPDATE Employe
SET d = d+2
WHERE d>=fA

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

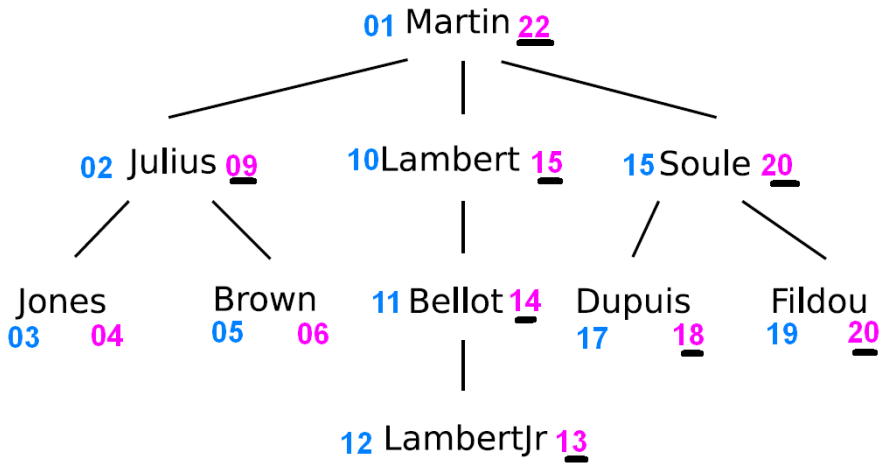
Insertion de 'Toto' comme subordonné de 'Julius' $f_A = 7$



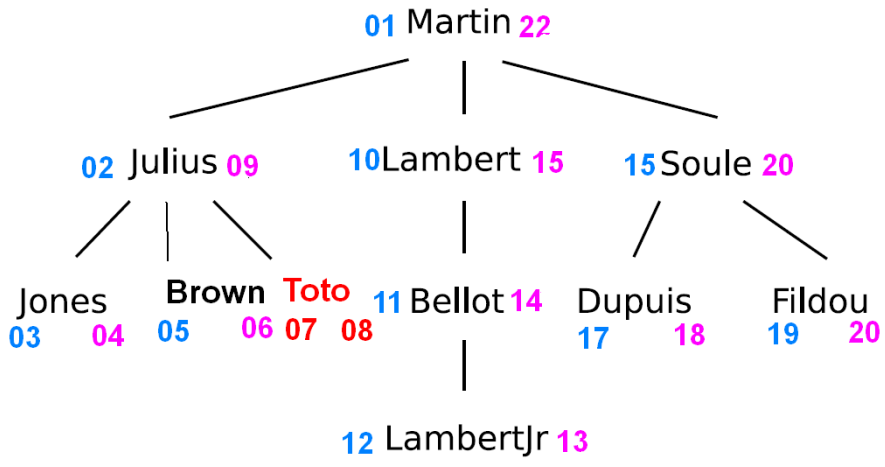
Exemple - décalage *d*



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 représenté 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 sont 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

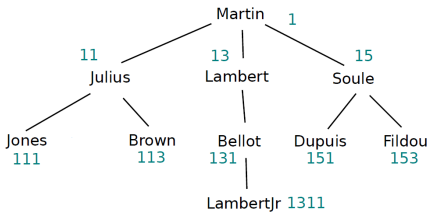
Pour LambertJr :

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

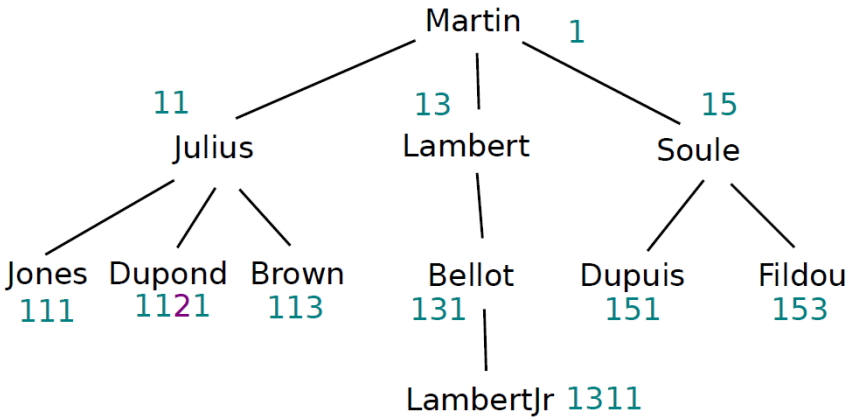

Optimisation de l'insertion

- Insertion en début :
  - Autoriser des indices de fils négatifs.
  - Démarrer l'indiçage au milieu de l'intervalle d'indice.
  - Similaire complément à 2.
- Insertion 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

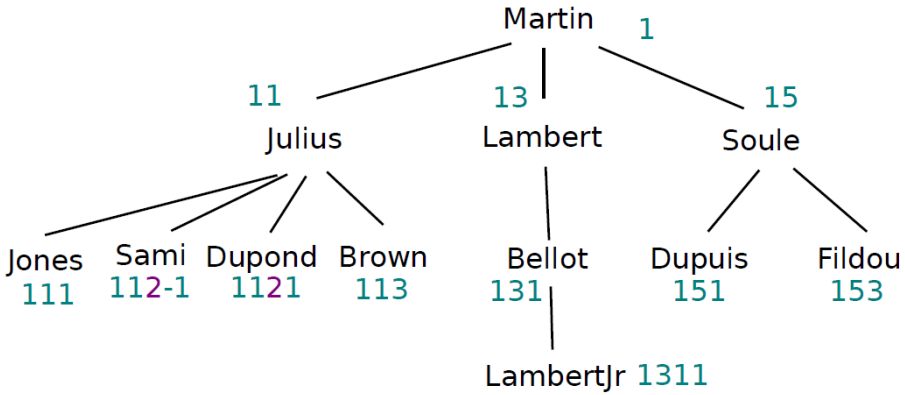
Nom	Num	NSup	Chemin
Martin	16712		1
Julius	12569	16712	11
Jones	19563	12569	111
Brown	20663	12569	112
Lambert	25012	16712	13
Bellot	13021	25012	131
LambertJr	15630	13021	1311
Soule	28963	16712	15
Dupuis	14028	28963	151
Fildou	25631	28963	152



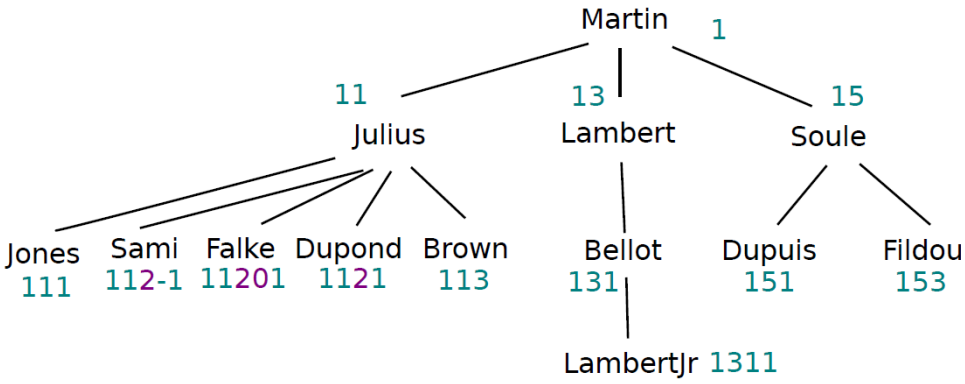
Insertion de 'Dupond' comme subordonné de 'Julius' entre 'Jones' et Brown



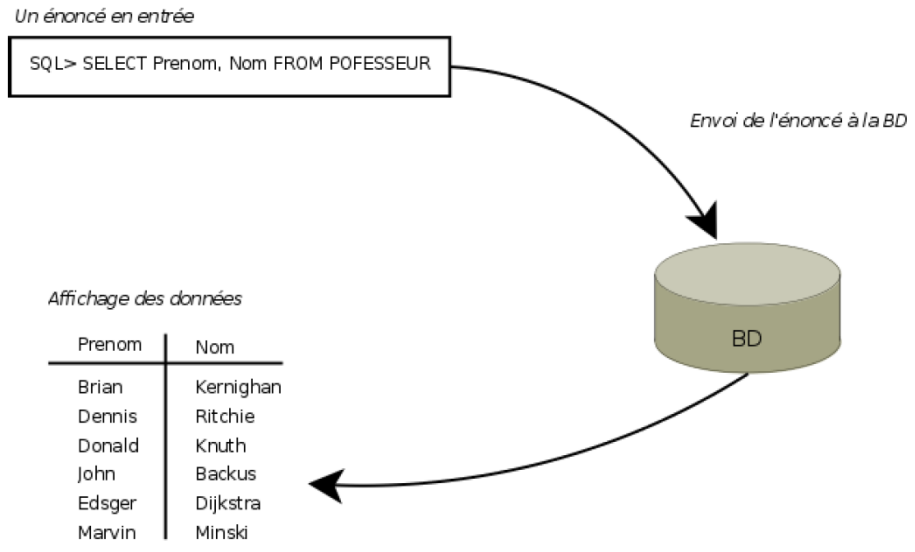
Insertion de 'Sami' comme subordonné de 'Julius' entre 'Jones' et 'Dupond'



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



PL/SQL



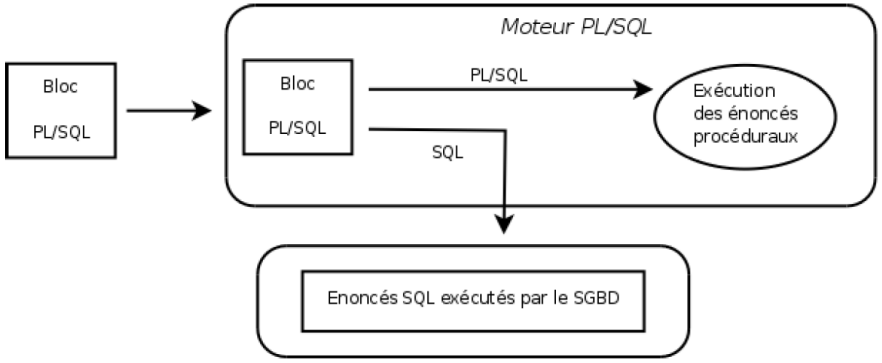
Clauses SQL

SELECT	Interrogation des données
INSERT UPDATE DELETE	Langage de Manipulation de Données (LMD)
CREATE ALTER DROP RENAME TRUNCATE	Langage de Définition de Données (LDD)
GRANT REVOKE	Langage de Contrôle de Données (LCD)
COMMIT ROLLBACK SAVEPOINT	Contrôle de transaction

PL/SQL

- Procedural Language for Structured Query Language
- Langage fournissant une interface procédurale au SGBD Oracle
- Intégration du langage SQL en lui apportant une dimension procédurale
- Réalisation de traitements algorithmiques (ce que ne permet pas SQL)
- Mise à disposition de la plupart des mécanismes classiques de programmation des langages hôtes tels que C, PASCAL, C++, JAVA ...

Environnement PL SQL :

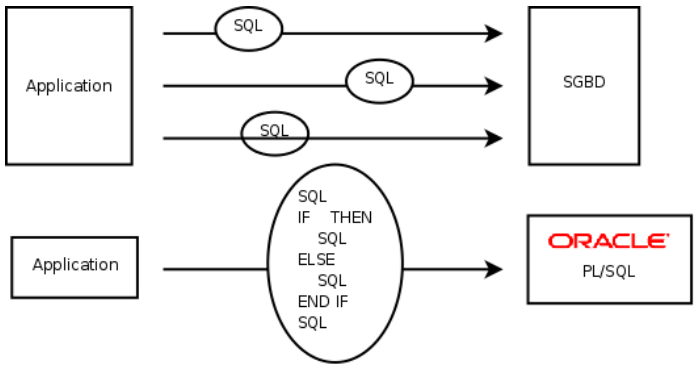


Avantages de PL/SQL

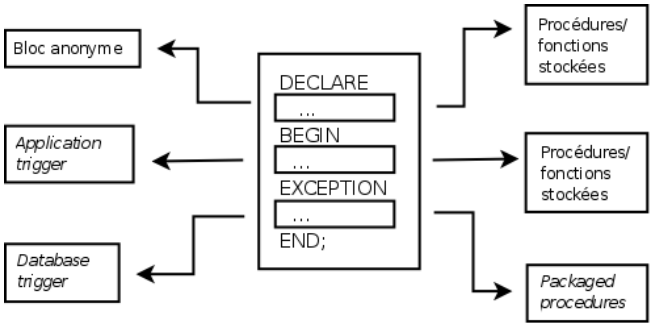
- PL/SQL complément de SQL (qui n'est pas procédural)
- Mécanismes offerts par PL/SQL :
  - Structures itératives : `WHILE *** LOOP`, `FOR *** LOOP`, `LOOP ***`
  - Structures conditionnelles : `IF *** THEN *** ELSE`, `ELSEIF *** ENDIF`, `CASE ***`
  - Déclaration des curseurs et des tableaux
  - Déclaration de variables
  - Affectation de valeurs aux variables
  - Branchements : `GOTO`, `EXIT`
  - Exceptions : `EXCEPTION`

Avantages de PL/SQL

- Intégration : Meilleure cohérence du code avec les données Utilisation de librairies standards prédéfinies
- Blocs : Traitement par/de blocs SQL dans un énoncé PL/SQL (optimisation des transactions réseaux)



- Exécution de programmes modulaires
- Traitements complexes (cas particuliers, erreurs)
- Traitements des exceptions



En résumé :

- Langage portable
- Utilisation de variables de stockage
- Utilisation de type simple ou de type structuré dynamiquement (%TYPE, %ROWTYPE, ...etc.)
- Utilisation des structures de contrôle des langages procéduraux
- Gestion et manipulation des erreurs
- Création d'ordres SQL dynamiques

PL - SQL

- Langage portable
- Utilisation de variables de stockage
- Utilisation de type simple ou de type structuré dynamiquement (%TYPE,% ROWTYPE, ...etc.)
- Utilisation des structures de contrôle des langages procéduraux
- Gestion et manipulation des erreurs
- Création d'ordres SQL dynamiques

Utilisation de `PL/SQL` sous 3 formes

- Bloc de code, exécuté comme une commande SQL
- Fichier de commandes PL SQL
- Programme stocké :
  - procédure,
  - fonction,
  - package
  - ou `trigger` (déclencheur)

Structure d'un bloc PL/ SQL

```
-- Section déclarative, optionnelle
DECLARE
      Variables, curseurs, exceptions, ...

-- Section exécutable, obligatoire
 BEGIN
      Instructions SQL et PL/SQL
      Possibilités de blocs fils (imbrication de blocs)

-- Section de traitement des exceptions, optionnelle
 EXCEPTION optionnelle
       Traitement des exceptions (gestion des erreurs)

-- Terminaison du bloc, obligatoire
 END ;
/
```

Types de blocs :

- Bloc anonyme
- Procédure
- Fonction

Bloc anonyme :

- Structure classique (1 à 3 sections)
- Un bloc ne peut être vide. Il doit contenir une instruction (il peut donc contenir l'instruction NULL).

[DECLARE]
BEGIN
    Instructions
[EXCEPTION]
END

Bloc anonyme : exemple

```
DECLARE x INTEGER;
BEGIN
x := 1 ;
END;
```



```
DECLARE
    varx VARCHAR2(5) .
BEGIN
    SELECT nom_ colonne
    INTO varx
    FROM nom_table
EXCEPTION
    WHEN nom_exception THEN
    ...
END ;
/
```

Procédure :

Bloc PL SQL nommé puis compilé et stocké dans la base

```
PROCEDURE nom IS
BEGIN
    Instructions
[EXCEPTION]
END;
/
```



```
PROCEDURE procedure_exemple IS
    varx VARCHAR2(5)
    BEGIN
        SELECT nom_colonne
        INTO varx
        FROM nom_table
    EXCEPTION
        WHEN nom_exception THEN
        ...
    END ;
/
```

Commande SQL qui crée la procédure PL SQL compile et stocke dans la base le bloc PL SQL compris entre le BEGIN et le END, référencé par 'procedure_exemple'

Procédure

Remarques :

- Exécution (auto) de la procédure :
  - SQL> `EXECUTE proc_exemple`
- Pas d'exécution de procédure (ou d'instructions) en fin de transaction (COMMIT, ROLLBACK, Ordre DDL)
- La décision d'enregistrement ou d'annulation de la transaction en cours est à réaliser par le programme appelant la procédure

Fonction :

Procédure retournant une valeur

```
FUNCTION nom_fonction RETURN type_données IS
    BEGIN
        Instructions
    RETURN valeur;
    [EXCEPTION]
    END;
/
```

```
CREATE OR REPLACE FUNCTION solde (nc INTEGER)
    RETURN REAL IS le_solde REAL;
    BEGIN
        SELECT solde INTO le_solde FROM clients
        WHERE no_cli = nc;
        RETURN le_solde;
    END;
/
```


Exemple : utilisation de fonction au sein d'une requête SQL

```
SQL> SELECT solde(1000) FROM dual ;
Solde(1000)
-----------
12024,50
```

L'appel d'une fonction comme une procédure provoque une erreur. Exemple : fonction mdp(INTEGER)

```
SQL> execute mdp(2);
BEGIN mdp(2); END;
*
ERREUR a la ligne 1 :
ORA-06550: line 1, column 7:
PLS-00221: 'MDP' is not a procedure or is undefined
```


Types de variables :

- Variables locales
  - Constantes
  - Composites
  - Références
- Variables de l'environnement extérieur à PL SQL
  - Attachées (Bind)
  - Hôtes (Host)

Déclaration des variables

Syntaxe :

```
Identificateur [CONSTANT] type_de_données [NOT NULL] [:=expression];
```

Exemple

```
DECLARE
    var_date_naissance  DATE;
    var_departement     NUMBER(2) NOT NULL := 10;
    var_ville           VARCHAR2(13) := 'Rouen';
    c_cumul             CONSTANT NUMBER := 1000;
```

Assignation des variables

Syntaxe :

```
Identificateur := expression ;
```

Exemples :

- Affecter la date de naissance du fils d'un employé

```
var_date_naissance := '23-SEP-2004';
```

- Fixer le nom d'un employé à 'Clémentine'

```
var_nom := 'Clémentine';
```


Exemples

```
v_job              VARCHAR2(9);
v_count            BINARY INTEGER := 0 ;
v_salaire_total    NUMBER(9,2);
v_date             DATE := SYSDATE +7;
v_taux_taxe        CONSTANT NUMBER(3,2) := 8.25 ;
v_valide           BOOLEAN NOT NULL := TRUE;
```


Attribut `TYPE`

- Définition : déclaration d'une variable associée à :
  - une colonne d'une table dans la BD
  - une variable précédemment définie

```
v_nom           emp.nom%TYPE;
v_sal_annuel    NUMBER(7,2);
v_sal_mensuel   v_sal_annuel%TYPE := 2000;
```


Bloc PL/ SQL

Syntaxe et directives :

- Les instructions peuvent être écrites sur plusieurs lignes
- Les unités lexicales peuvent être séparées par des espaces
  - Délimiteurs
  - Identificateurs
  - Littéraux (ou constantes)
  - Commentaires

Les littéraux

- Les dates et les chaînes de caractères délimitées par deux simples côtes

```
v_nom := 'Loulou le pou' ;
```

- Les nombres peuvent être des valeurs simples ou des expressions

```
v_annee := to_number ( to_char ( v_date_fin_periode, 'YY ' ) )
```


Commentaires dans le code :

- Précéder un commentaire écrit sur une seule ligne par '- -'.
- Placer un commentaire écrit sur plusieurs lignes entre les symboles '/*' et '*/'.
- Exemple :

```
v_salaire NUMBER(9, 2);
BEGIN
/* Ceci est un commentaire qui peut
être écrit sur plusieurs lignes (ici 2 lignes)*/
END; --Ceci est un commentaire sur une ligne
```


Les Fonctions SQL en PL/SQL :

- Les fonctions sur les nombres
- Les fonctions sur les chaînes de caractères
- Les fonctions de conversion de type de données
- Les fonctions de dates

Exemples :

- Recomposer l'adresse d'un employé :

```
v_adr_complete := v_rue||CHR(32)||v_ville||CHR(32)
```

- Convertir le nom en majuscule

```
v_nom := UPPER(v_nom) ;
```

- Extraction d'une partie de la chaîne

```
v_chr := Substr('PL/SQL',4,3);
```

- Replacement d'une chaîne par une autre

```
v_chr := Replace('Serv1/Prod/tb_client','Prod','Valid');
```

Blocs imbriqués et portée des variables :

```
x BINARY INTEGER ;
BEGIN --Début de la portée de x
...
DECLARE
    y NUMBER;
    BEGIN --Début de la portée de y
    ...
    END; --Fin de la portée de y
    ...
END; --Fin de la portée de x
```


Opérateurs dans PL/SQL :

- Identiques à SQL :
  - Logique
  - Arithmétique
  - Concaténation
  - Parenthèses pour contrôler l'ordre des opérations
- Opérateur supplémentaire :
  - Opérateur d'exponentiation **

Utilisation des variables liées :

- Pour référencer une variable en PL SQL, on doit préfixer son nom par un ' :'
- Exemple :

```
: code_retour := 0 ;
IF verifier_crédit_ok (compt no) THEN
    : code_retour := 1 ;
END IF ;
```


Instructions SQL dans PL/SQL :

- Extraire une ligne de données à partir de la base de données par la commande SELECT : un seul ensemble de valeurs peut être retourné.
- Effectuer des changements aux lignes dans la base de données par les commandes du LMD
- Contrôler des transactions avec les commandes COMMIT, ROLLBACK, SAVEPOINT
- Déterminer les résultats du LMD avec des curseurs implicites

Instruction SELECT dans PL/ SQL :

- Récupérer une donnée de la base avec SELECT.
- Syntaxe :

```
SELECT liste_sélection
INTO {nom_var [ , nom_var ] ...| nom_record}
FROM nom_table
WHERE condition;
```


Instruction SELECT dans PL/ SQL :

- La clause INTO est obligatoire
- Exemple

```
DECLARE
    v_deptno NUMBER(2);
    v_loc VARCHAR2(15);
BEGIN
    SELECT deptno, loc
    INTO v_deptno, v_loc
    FROM dept
    WHERE nom_d = 'INFORMATIQUE';
    ...
END;
```


Instruction SELECT dans PL/ SQL :

- Retourne la somme des salaires de tous les employés d'un département donné :

```
DECLARE
    v_som_sal   emp.sal%TYPE;
    v_deptno NUMBER NOT NULL := 10;
BEGIN
    SELECT sum (sal) --fonction
    INTO v_som_sal
    FROM emp
    WHERE deptno = v_deptno;
END;
```


Manipulation de données en PL/ SQL

- Effectuer des mises à jour des tables de la base utilisant les commandes du LMD :
  - INSERT
  - UPDATE
  - DELETE

Insertion de données

- Ajouter les informations d'un nouvel employé dans la table emp

```
DECLARE
    v_empno NUMBER NOT NULL := 105 ;
BEGIN
    INSERT INTO emp(empno, emp_nom, poste, deptno)
    VALUES (v_empno,' Clémentine','Directrice',10)
END;
```


Mise à jour de données

- Augmenter le salaire de tous les employés dans la table emp qui ont le poste d'Enseignant.

```
DECLARE
    v_augm_sal  emp.sal%TYPE := 2000;
BEGIN
    UPDATE emp
    SET sal := sal + v_augm_sal
    WHERE job = 'Enseignant';
END;
```


Suppression de données

- Suppression des lignes appartenant au département 10 de la table emp

```
DECLARE
    v_deptno    emp.deptno%TYPE := 10;
BEGIN
    DELETE FROM emp
    WHERE deptno = v deptno ;
END;
```