Vendredi 16 mars 2012

Rapport TP1 & TP2 de IF4-BDC

Thibault JACQUEMIN

Mathieu MARLEIX

Sommaire

TP1 3

Problème 1 3

A. Création des tables 3

B. Insertion des éléments 5

C. Requêtes SQL 6

Problème 2 12

A. Création des tables et de la contrainte a. 12

B. Création du trigger pour la contrainte b. 13

C. Insertion d'éléments 15

TP2 16

O. Création de la table *madeof* 16

A. Calcul de la fermeture transitive 17

B. Vérification de la présence d'un cycle 18

C. Création de la nomenclature 19

1. Décomposition en niveaux 19

2. Assemblage de la nomenclature 21

TP1

Problème 1

A. Création des tables

Les tables à créer sont les suivantes :

Table	Propriétaire	Tablespace	Nombre d'enregistrements estimés
f	jacquemt		0
fpj	jacquemt		0
j	jacquemt		0
p	jacquemt		0

Par précaution, les tables à créer sont effacées si elles existent avec l'instruction :

DROP TABLE IF EXISTS nom_table;

Elles sont ensuite créées avec l'instruction :

CREATE TABLE nom_table (…paramètres de la table…) ;

Un paramètre se précise suivant la forme suivante :

Nom_paramètre type_paramètre [NOT NULL]

Dans notre cas, le type du paramètre est soit **`text`**, soit **`integer`**.

L'attribut NOT NULL est utilisé pour spécifier que l'attribut en question doit avoir une valeur. Nous l'avons utilisé sur les paramètres étant des clés.

Préciser qu'un paramètre est une clé primaire se fait de la manière suivante :

PRIMARY KEY (nom_paramètre)

Enfin, quand un paramètre fait référence à un présent dans une autre table, il est possible de le préciser ainsi :

FOREIGN KEY (nom_paramètre) REFERENCES nom_autre_table
ON DELETE [RESTRICT | SET NULL | CASCADE]

La partie ON DELETE sert à préciser le comportement à adopter en cas de suppression d'une clé dans la table référencée :

- RESTRICT précise que la clé n'a pas à être effacée dans la table ;
- SET NULL passera le paramètre de la table à NULL ;
- CASCADE provoquera l'effacement de la ligne contenant ce paramètre.

```sql
DROP TABLE IF EXISTS f CASCADE;
CREATE TABLE f (
    nf text NOT NULL,
    nomf text,
    categorie integer,
    ville text,
    PRIMARY KEY (nf)
);
DROP TABLE IF EXISTS p CASCADE;
CREATE TABLE p (
    np text NOT NULL,
    nomp text,
    couleur text,
    poids integer,
    ville text,
    PRIMARY KEY (np)
);
DROP TABLE IF EXISTS j CASCADE;
CREATE TABLE j (
    nj text NOT NULL,
    nomj text,
    ville text,
    PRIMARY KEY (nj)
);
DROP TABLE IF EXISTS fpj;
CREATE TABLE fpj (
    nf text,
    np text,
    nj text,
    qte integer,
    PRIMARY KEY (nf, np, nj),
    FOREIGN KEY (nf) REFERENCES f ON DELETE SET NULL,
    FOREIGN KEY (np) REFERENCES p ON DELETE RESTRICT,
    FOREIGN KEY (nj) REFERENCES j ON DELETE CASCADE
);
```

Pas de résultats.
Temps d'exécution total : 732.415 ms
Requête SQL exécutée.

En cas de suppression d'un fabricant, on le passe à NULL dans la table des commandes car celui-ci est remplaçable.

En cas de suppression d'une pièce, on garde sa référence dans la table des commandes pour en commander une qui lui ressemble.

En cas de suppression d'un projet, on supprime toutes les commandes qui lui sont relatives.

Table *f*

Colonne	Type	NOT NULL	Défaut	Contraintes
nf	text	NOT NULL		
nomf	text			
categorie	integer			
ville	text			

Table *p*

Colonne	Type	NOT NULL	Défaut	Contraintes
np	text	NOT NULL		
nomp	text			
couleur	text			
poids	integer			
ville	text			

Table *j*

Colonne	Type	NOT NULL	Défaut	Contraintes
nj	text	NOT NULL		
nomj	text			
ville	text			

Table *fpj*

Colonne	Type	NOT NULL	Défaut	Contraintes
nf	text	NOT NULL		
np	text	NOT NULL		
nj	text	NOT NULL		
qte	integer			

B. Insertion des éléments

L'insertion d'éléments se fait avec cette instruction :

INSERT INTO nom_table VALUES(…parameters à insérer…)

```
INSERT INTO f   VALUES('F1',    'SMITH',    10, 'LONDRES');
INSERT INTO f   VALUES('F2',    'DURAND',   10, 'PARIS');
INSERT INTO f   VALUES('F3',    'DUPONT',   30, 'PARIS');
INSERT INTO f   VALUES('F4',    'CLARK',    20, 'LONDRES');
INSERT INTO f   VALUES('F5',    'KURT',     30, 'COLOGNE');

INSERT INTO p   VALUES('P1',    'ECROU',    'ROUGE',    12, 'LONDRES');
INSERT INTO p   VALUES('P2',    'BOULON',   'VERT',     17, 'PARIS');
INSERT INTO p   VALUES('P3',    'VIS',      'BLEU',     17, 'ROME');
INSERT INTO p   VALUES('P4',    'VIS',      'ROUGE',    14, 'LONDRES');
INSERT INTO p   VALUES('P5',    'CAME',     'BLEU',     12, 'PARIS');
INSERT INTO p   VALUES('P6',    'BAGUE',    'ROUGE',    19, 'LONDRES');

INSERT INTO j   VALUES('J1',    'SPEED',        'PARIS');
INSERT INTO j   VALUES('J2',    'PUNCH',        'ROME');
INSERT INTO j   VALUES('J3',    'LECTEUR',      'COLOGNE');
INSERT INTO j   VALUES('J4',    'CONSOLE',      'COLOGNE');
INSERT INTO j   VALUES('J5',    'COLLECTEUR',   'LONDRES');
INSERT INTO j   VALUES('J6',    'TERMINAL',     'OSLO');
INSERT INTO j   VALUES('J7',    'PREMS',        'LONDRES');

INSERT INTO fpj VALUES('F1',    'P1',   'J1',   200);
INSERT INTO fpj VALUES('F1',    'P1',   'J4',   700);
INSERT INTO fpj VALUES('F2',    'P3',   'J1',   400);
INSERT INTO fpj VALUES('F2',    'P3',   'J2',   200);
INSERT INTO fpj VALUES('F2',    'P3',   'J3',   200);
INSERT INTO fpj VALUES('F2',    'P3',   'J4',   500);
INSERT INTO fpj VALUES('F2',    'P3',   'J5',   600);
INSERT INTO fpj VALUES('F2',    'P3',   'J6',   400);
INSERT INTO fpj VALUES('F2',    'P3',   'J7',   800);
INSERT INTO fpj VALUES('F2',    'P5',   'J2',   100);
INSERT INTO fpj VALUES('F3',    'P3',   'J1',   200);
INSERT INTO fpj VALUES('F3',    'P4',   'J2',   500);
INSERT INTO fpj VALUES('F4',    'P6',   'J3',   300);
INSERT INTO fpj VALUES('F4',    'P6',   'J7',   300);
INSERT INTO fpj VALUES('F5',    'P1',   'J4',   1000);
INSERT INTO fpj VALUES('F5',    'P2',   'J2',   200);
INSERT INTO fpj VALUES('F5',    'P2',   'J4',   100);
INSERT INTO fpj VALUES('F5',    'P3',   'J4',   1200);
INSERT INTO fpj VALUES('F5',    'P4',   'J4',   800);
INSERT INTO fpj VALUES('F5',    'P5',   'J4',   400);
INSERT INTO fpj VALUES('F5',    'P5',   'J5',   500);
INSERT INTO fpj VALUES('F5',    'P5',   'J7',   100);
INSERT INTO fpj VALUES('F5',    'P6',   'J2',   200);
INSERT INTO fpj VALUES('F5',    'P6',   'J4',   500);
```

42 ligne(s) affectée(s). Temps d'exécution total : 80.780 ms Requête SQL exécutée.

Table *f*

nf	nomf	categorie	ville
F1	SMITH	10	LONDRES
F2	DURAND	10	PARIS
F3	DUPONT	30	PARIS
F4	CLARK	20	LONDRES
F5	KURT	30	COLOGNE

Table *p*

np	nomp	couleur	poids	ville
P1	ECROU	ROUGE	12	LONDRES
P2	BOULON	VERT	17	PARIS
P3	VIS	BLEU	17	ROME
P4	VIS	ROUGE	14	LONDRES
P5	CAME	BLEU	12	PARIS
P6	BAGUE	ROUGE	19	LONDRES

Table *j*

nj	nomj	ville
J1	SPEED	PARIS
J2	PUNCH	ROME
J3	LECTEUR	COLOGNE
J4	CONSOLE	COLOGNE
J5	COLLECTEUR	LONDRES
J6	TERMINAL	OSLO
J7	PREMS	LONDRES

Table *fpj*

nf	np	nj	qte
F1	P1	J1	200
F1	P1	J4	700
F2	P3	J1	400
F2	P3	J2	200
F2	P3	J3	200
F2	P3	J4	500
F2	P3	J5	600
F2	P3	J6	400
F2	P3	J7	800
F2	P5	J2	100
F3	P3	J1	200
F3	P4	J2	500
F4	P6	J3	300
F4	P6	J7	300
F5	P1	J4	1000
F5	P2	J2	200
F5	P2	J4	100
F5	P3	J4	1200
F5	P4	J4	800
F5	P5	J4	400
F5	P5	J5	500
F5	P5	J7	100
F5	P6	J2	200
F5	P6	J4	500

C. Requêtes SQL

Requête 1

```
select *
from j;
```

nj	nomj	ville
J1	SPEED	PARIS
J2	PUNCH	ROME
J3	LECTEUR	COLOGNE
J4	CONSOLE	COLOGNE
J5	COLLECTEUR	LONDRES
J6	TERMINAL	OSLO
J7	PREMS	LONDRES

7 ligne(s)

Temps d'exécution total : 1.029 ms

Requête SQL exécutée.

Requête 2

```
select *
from j
where ville = 'LONDRES';
```

nj	nomj	ville
J5	COLLECTEUR	LONDRES
J7	PREMS	LONDRES

2 ligne(s)

Temps d'exécution total : 0.746 ms

Requête SQL exécutée.

Requête 3

```
select distinct nf
from fpj
where nj = 'J1';
```

nf
F2
F3
F1

3 ligne(s)

Temps d'exécution total : 1.788 ms

Requête SQL exécutée.

Requête 4

```
select nf, np, nj
from fpj
where qte between 300 and 750;
```

nf	np	nj
F1	P1	J4
F2	P3	J1
F2	P3	J4
F2	P3	J5
F2	P3	J6
F3	P4	J2
F4	P6	J3
F4	P6	J7
F5	P5	J4
F5	P5	J5
F5	P6	J4

11 ligne(s)

Temps d'exécution total : 1.310 ms

Requête SQL exécutée.

Requête 5

```
select distinct nomj
from j
where nj in
  (select distinct nj
   from fpj
   where nf = 'F1');
```

nomj
CONSOLE
SPEED

2 ligne(s)

Temps d'exécution total : 1.366 ms

Requête SQL exécutée.

Requête 6

```
select distinct couleur
from p
where np in
  (select distinct np
   from fpj
   where nf = 'F1');
```

couleur
ROUGE

1 ligne(s)

Temps d'exécution total : 1.405 ms

Requête SQL exécutée.

Requête 7

```
(select distinct nf
 from fpj
 where nj = 'J1')
  intersect
(select distinct nf
 from fpj
 where nj = 'J2');
```

nf
F2
F3

2 ligne(s)

Temps d'exécution total : 1.436 ms

Requête SQL exécutée.

Requête 8

```
select distinct nf
from fpj
where nj = 'J1' and np in
  (select np
   from p
   where couleur = 'ROUGE');
```

nf
F1

1 ligne(s)

Temps d'exécution total : 1.569 ms

Requête SQL exécutée.

Requête 9	Requête 10

`select distinct np`
`from fpj`
`where nj in`
`  (select nj`
`   from j`
`   where ville = 'LONDRES');`
	`select distinct nf`
	`from fpj`
	`where nj in`
	`  (select distinct nj`
	`   from j`
	`   where ville in`
	`    ('PARIS','LONDRES'))`
	`   and np in`
	`    (select np`
	`     from p`
	`     where couleur = 'ROUGE');`

np P6 P3 P5 3 ligne(s) Temps d'exécution total : 1.372 ms Requête SQL exécutée.	nf F4 F1 2 ligne(s) Temps d'exécution total : 1.755 ms Requête SQL exécutée.
Requête 11	Requête 12

`select distinct np`
`from fpj`
`where nf in`
`  (select distinct nf`
`   from`
`    (f join j on f.ville = j.ville))`
`    as fjLocal;`
	`select distinct nj`
	`from (fpj join f on fpj.nf = f.nf)`
	`     as fVillepj`
	`group by nj`
	`having count(distinct ville) > 1;`

np P2 P6 P4 P3 P5 P1 6 ligne(s) Temps d'exécution total : 1.643 ms Requête SQL exécutée.	nj J1 J2 J3 J4 J5 J7 6 ligne(s) Temps d'exécution total : 1.920 ms Requête SQL exécutée.

Requête 13

```
select distinct nj
from fpj
where nf in
  (select distinct nf
   from f
   where ville <> 'LONDRES')
   and np in
    (select np
     from p
     where couleur = 'ROUGE');
```

nj
J4
J2

2 ligne(s)

Temps d'exécution total : 1.730 ms

Requête SQL exécutée.

Requête 14

```
select distinct nf
from fpj
where np in
  (select np
   from fpj
   where nf in
    (select distinct nf
     from fpj
     where np in
      (select np
       from p
       where couleur = 'ROUGE')));
```

nf
F4
F1
F3
F2
F5

5 ligne(s)

Temps d'exécution total : 1.856 ms

Requête SQL exécutée.

Requête 15

```
select distinct f.ville
          as ville_de_f,
                np,
                j.ville
          as ville_de_j
from j,f,fpj
where f.ville <> j.ville
  and fpj.nf = f.nf
  and fpj.nj = j.nj;
```

ville_de_f	np	ville_de_j
PARIS	P3	LONDRES
COLOGNE	P2	ROME
COLOGNE	P6	ROME
PARIS	P3	COLOGNE
PARIS	P5	ROME
LONDRES	P1	PARIS
PARIS	P3	OSLO
LONDRES	P6	COLOGNE
LONDRES	P1	COLOGNE
PARIS	P3	ROME
PARIS	P4	ROME
COLOGNE	P5	LONDRES

12 ligne(s)

Temps d'exécution total : 2.294 ms

Requête SQL exécutée.

Requête 16

```
select distinct nf
from fpj
where np in
  (select distinct np
   from fpj
   where nj = all
    (select distinct nj
     from fpj))
group by nf,np
having count(distinct nj) =
  (select count(distinct nj)
   from fpj);
```

Pas de résultats.

Temps d'exécution total : 1.521 ms

Requête SQL exécutée.

Requête 17	Requête 18

`select nj`
`from fpj`
`where not exists`
`  (select np`
`   from fpj`
`   where nf <> 'F1');`
	`select distinct np`
	`from fpj`
	`where nj = all`
	`   (select nj`
	`    from j`
	`    where ville = 'LONDRES');`

Pas de résultats. Temps d'exécution total : 0.672 ms Requête SQL exécutée.	Pas de résultats. Temps d'exécution total : 1.160 ms Requête SQL exécutée.
Requête 19	**Requête 20**

`select distinct nj`
`from fpj`
`where np = all`
`  (select np`
`   from fpj`
`   where nf = 'F1');`
	`select distinct nj`
	`from fpj`
	`where not exists`
	`  (select np`
	`   from fpj`
	`   where nf <> 'F2');`

nj J4 J1 2 ligne(s) Temps d'exécution total : 1.678 ms Requête SQL exécutée.	Pas de résultats. Temps d'exécution total : 1.017 ms Requête SQL exécutée.
Requête 21	**Requête 22**

`select count(nj)`
`from fpj`
`where nf = 'F1';`
	`select sum(qte)`
	`from fpj`
	`where nf = 'F1' and np = 'P1';`

count 2 1 ligne(s) Temps d'exécution total : 0.878 ms Requête SQL exécutée.	sum 900 1 ligne(s) Temps d'exécution total : 0.868 ms Requête SQL exécutée.

Requête 23

```
select np,nj,sum(qte)
from fpj
group by nj,np;
```

np	nj	sum
P3	J4	1700
P5	J4	400
P3	J1	600
P3	J5	600
P3	J7	800
P3	J3	200
P5	J5	500
P5	J7	100
P4	J4	800
P6	J3	300
P6	J7	300
P2	J4	100
P6	J4	500
P3	J6	400
P2	J2	200
P6	J2	200
P4	J2	500
P1	J1	200
P1	J4	1700
P5	J2	100
P3	J2	200

21 ligne(s)

Temps d'exécution total : 2.173 ms

Requête SQL exécutée.

Requête 24

```
select distinct np
from fpj
group by nj,np
having avg(qte) > 350;
```

np
P6
P4
P3
P5
P1

5 ligne(s)

Temps d'exécution total : 1.642 ms

Requête SQL exécutée.

Requête 25

```
select distinct nf
from fpj join
  (select nj,avg(qte) as average
   from fpj
   where np = 'P1' group by nj)
as avgP1byJ on fpj.nj = avgP1byJ.nj
where qte > average;
```

nf
F2
F5

2 ligne(s)

Temps d'exécution total : 1.889 ms

Requête SQL exécutée.

Problème 2

Les tables de ce problème sont les suivantes :

Table	Propriétaire	Tablespace	Nombre d'enregistrements estimés
affectation	jacquemt		0
enseignement	jacquemt		0

Contraintes sur la table *affectation* :

a. Interdire l'insertion d'une ligne comportant un professeur qui n'existe pas dans la table *enseignement* ;
b. Interdire l'insertion d'un professeur si la matière qu'il enseigne est déjà enseignée à la classe visée.

A. Création des tables et de la contrainte a.

La contrainte a. peut se mettre en place en précisant que professeur est une clé étrangère déjà présente dans la table *enseignement*.

```
DROP TABLE IF EXISTS enseignement CASCADE;
CREATE TABLE enseignement(
professeur text NOT NULL,
matiere text,
PRIMARY KEY (professeur)
);

DROP TABLE IF EXISTS affectation;
CREATE TABLE affectation(
classe text NOT NULL,
professeur text NOT NULL,
PRIMARY KEY (classe, professeur),
-- Contrainte a)
FOREIGN KEY (professeur)
REFERENCES enseignement (professeur)
ON DELETE RESTRICT
);
```

Pas de résultats.
Temps d'exécution total : 345.225 ms
Requête SQL exécutée.

Table *enseignement*

Colonne	Type	NOT NULL	Défaut	Contraintes
professeur	text	NOT NULL		
matiere	text			

Table *affectation*

Colonne	Type	NOT NULL	Défaut	Contraintes
classe	text	NOT NULL		
professeur	text	NOT NULL		

B. Création du trigger pour la contrainte b.

La mise en place de la contrainte b. se fera à l'aide d'un trigger qui surveille un évènement particulier et déclenche une fonction quand celui-ci se produit.

Il faut donc d'abord créer la fonction de la manière suivante… :

```
CREATE FUNCTION nom_fonction()
RETURNS TRIGGER AS $$
        DECLARE
                nom_variable type_variable
        BEGIN
                … Corps de la function…
        END;
$$ LANGUAGE 'plpgsql';
```

```
DROP FUNCTION IF EXISTS verify_classe () CASCADE;
CREATE FUNCTION verify_classe ()
RETURNS TRIGGER AS $$
    DECLARE
        mat text;
    BEGIN
        select into mat matiere
        from (
            select distinct matiere
            from enseignement
            where professeur in
                (select professeur
                 from affectation
                 where classe = NEW.classe)
        ) as matClasse
        where matiere =
            (select matiere
             from enseignement
             where professeur = NEW.professeur)
        ;
        IF FOUND THEN
            RAISE EXCEPTION
 'Un professeur enseigne déjà la matière % à la classe %',
mat, NEW.classe;
        END IF;
        RETURN NEW;
    END;
$$ LANGUAGE 'plpgsql';
```

Pas de résultats.
Temps d'exécution total : 48.742 ms
Requête SQL exécutée.

Cette fonction vérifie si la matière qu'enseigne le professeur que l'on veut affecter à une classe est déjà enseignée à cette même classe.
Si c'est le cas, alors une exception est levée et la ligne n'est pas insérée.

Fonction	Arguments	Valeur de sortie	Langage
verify_classe		trigger	plpgsql

Définition

```
1       DECLARE
2           mat text;
3       BEGIN
4           select into mat matiere
5           from (
6               select distinct matiere
7               from enseignement
8               where professeur in
9                   (select professeur
10                   from affectation
11                   where classe = NEW.classe)
12          ) as matClasse
13          where matiere =
14              (select matiere
15               from enseignement
16               where professeur = NEW.professeur)
17          ;
18          IF FOUND THEN
19              RAISE EXCEPTION 'Un professeur enseigne déjà la matière % à la classe %', mat, NEW.classe;
20          END IF;
21          RETURN NEW;
22      END;
23
24
```

Coût de la function

Coût d'exécution: 100	Lignes de résultat: 0

Propriétés

VOLATILE
CALLED ON NULL INPUT
SECURITY INVOKER

Propriétaire: jacquemt

… Puis créer le trigger appelant cette fonction :

CREATE TRIGGER nom_trigger BEFORE INSERT ON table_à_surveiller
FOR EACH ROW
EXECUTE PROCEDURE nom_fonction();

```
DROP TRIGGER IF EXISTS trig_bef_ins_affectation
ON affectation RESTRICT;
CREATE TRIGGER trig_bef_ins_affectation
BEFORE INSERT ON affectation
    FOR EACH ROW
    EXECUTE PROCEDURE verify_classe();
```

Pas de résultats.
Temps d'exécution total : 71.541 ms
Requête SQL exécutée.

Nom	Définition	Fonction
trig_bef_ins_affectation	CREATE TRIGGER trig_bef_ins_affectation BEFORE INSERT ON affectation FOR EACH ROW EXECUTE PROCEDURE verify_classe()	verify_classe ()

C. Insertion d'éléments

Tout d'abord, quelques insertions pour alimenter la table *enseignement*… :

```
INSERT INTO enseignement (professeur, matiere) VALUES ('tata', 'maths');
INSERT INTO enseignement (professeur, matiere) VALUES ('titi', 'bio');
INSERT INTO enseignement (professeur, matiere) VALUES ('toto', 'info');
INSERT INTO enseignement (professeur, matiere) VALUES ('tutu', 'info');
```

4 ligne(s) affectée(s). Temps d'exécution total : 23.687 ms Requête SQL exécutée.

Table *enseignement*

professeur	matiere
tata	maths
titi	bio
toto	info
tutu	info

…Puis des affectations valides,… :

```
INSERT INTO affectation (classe, professeur) VALUES ('PC', 'toto');
INSERT INTO affectation (classe, professeur) VALUES ('PC', 'tata');
INSERT INTO affectation (classe, professeur) VALUES ('NIX', 'tata');
```

3 ligne(s) affectée(s). Temps d'exécution total : 36.734 ms Requête SQL exécutée.

…Une affectation contredisant la contrainte a., :

```
INSERT INTO affectation (classe, professeur) VALUES ('NIX', 'trax');
```

Temps d'exécution total : 3.425 ms Requête SQL exécutée.

Erreur SQL :

ERREUR: une instruction insert ou update sur la table « affectation » viole la contrainte de clé étrangère « affectation_professeur_fkey »
DETAIL: La clé (professeur)=(trax) n'est pas présente dans la table « enseignement ».

Dans l'instruction :
INSERT INTO affectation (classe, professeur) VALUES ('NIX', 'trax');

…Et une affectation contredisant la contrainte b.

```
INSERT INTO affectation (classe, professeur) VALUES ('PC', 'tutu');
```

Temps d'exécution total : 2.978 ms Requête SQL exécutée.

Erreur SQL :

ERREUR: Un professeur enseigne déjà la matière info à la classe PC

Dans l'instruction :
INSERT INTO affectation (classe, professeur) VALUES ('PC', 'tutu');

Table *affectation* après toutes les insertions

classe	professeur
PC	toto
PC	tata
NIX	tata

TP2

Les tables nécessaires à ce TP sont :

Table	Propriétaire	Tablespace	Nombre d'enregistrements estimés
closure	jacquemt		0
madeof	jacquemt		0
niveaux	jacquemt		0
nomenclature	jacquemt		0

0. Création de la table *madeof*

```
DROP TABLE IF EXISTS madeof;
CREATE TABLE madeof(
parent integer NOT NULL,
child integer,
qte integer
);

INSERT INTO madeof (parent, child, qte) VALUES (1, 2, 1);
INSERT INTO madeof (parent, child, qte) VALUES (1, 3, 2);
INSERT INTO madeof (parent, child, qte) VALUES (2, 5, 1);
INSERT INTO madeof (parent, child, qte) VALUES (2, 6, 2);
INSERT INTO madeof (parent, child, qte) VALUES (3, 2, 10);
INSERT INTO madeof (parent, child, qte) VALUES (3, 5, 3);
INSERT INTO madeof (parent, child, qte) VALUES (4, 2, 1);
INSERT INTO madeof (parent, child, qte) VALUES (4, 7, 5);
INSERT INTO madeof (parent, child, qte) VALUES (5, 6, 4);
INSERT INTO madeof (parent, child, qte) VALUES (7, 8, 1);
INSERT INTO madeof (parent, child, qte) VALUES (8, 9, 1);
INSERT INTO madeof (parent, child, qte) VALUES (8, 10, 2);
```

12 ligne(s) affectée(s).
Temps d'exécution total : 28.823 ms
Requête SQL exécutée.

Table *madeof*

Colonne	Type	NOT NULL	Défaut	Contraintes
parent	integer	NOT NULL		
child	integer			
qte	integer			

parent	child	qte
1	2	1
1	3	2
2	5	1
2	6	2
3	2	10
3	5	3
4	2	1
4	7	5
5	6	4
7	8	1
8	9	1
8	10	2

A. Calcul de la fermeture transitive

La table *closure* est une table à compléter de manière récursive.

```
DROP TABLE IF EXISTS closure;
CREATE TABLE closure(
parent integer NOT NULL,
descendant integer
);
```

Pas de résultats.
Temps d'exécution total : 20.941 ms
Requête SQL exécutée.

Il faut commencer par y copier la table *madeof* pour avoir les descendants immédiats des parents :

```
INSERT INTO closure (parent, descendant)
select parent, child from madeof;
```

Ensuite sont ajoutés, toujours selon la table *madeof*, les descendants plus 'profonds' qui ne sont pas encore dans la table :

```
INSERT INTO closure (parent, descendant)
((select c.parent, m.child from closure c, madeof m
    where c.descendant = m.parent)
except (select * from closure));
```

Cette étape est à répéter un nombre de fois égale à la profondeur de l'arbre -1 (l'étape initiale étant déjà faite), soit 4 fois dans le cas présent.

Toutefois ces requêtes peuvent être résumées dans une seule requête récursive, qui s'arrêtera quand l requête à répéter ne génèrera plus de nouveaux éléments, s'écrivant de la façon suivante :

INSERT INTO nom_table
WITH RECURSIVE nom_table AS (
    …Requête initiale…
        UNION ALL
    …Requête recursive…
)
…Requête finale…

```
INSERT INTO closure(parent,descendant)
WITH RECURSIVE closure(parent,descendant) AS (
    (select parent,child from madeof)
        UNION ALL
    (select c.parent, m.child from closure c, madeof m
        where c.descendant = m.parent)
)
select distinct * from closure;
```

22 ligne(s) affectée(s).
Temps d'exécution total : 28.076 ms
Requête SQL exécutée.

A noter toutefois qu'il est interdit de faire référence à la table dans laquelle on veut insérer les éléments dans plus d'une requête (que ce soit une requête avec except() ou une sous-requête) ; on utilisera donc le mot-clé distinct dans la requête finale pour obtenir le résultat voulu.

On peut également remplacer le mot UNION ALL par UNION pour que la requête récursive ne s'exécute que sur les lignes de l'itération précédente (et non sur la table complète).
Par ailleurs, il est possible de constater que cette deuxième façon est plus rapide que la précédente puisque les itérations se font sur moins de lignes.

```
INSERT INTO closure(parent,descendant)
WITH RECURSIVE closure(parent,descendant) AS (
    (select parent,child from madeof)
        UNION
    ((select c.parent, m.child from closure c, madeof m
        where c.descendant = m.parent))
)
select * from closure;
```

22 ligne(s) affectée(s).
Temps d'exécution total : 21.522 ms
Requête SQL exécutée.

Table *closure*

Colonne	Type	NOT NULL	Défaut	Contraintes
parent	integer	NOT NULL		
descendant	integer			

parent	descendant
1	2
1	3
2	5
2	6
3	2
3	5
4	2
4	7
5	6
7	8
8	9
8	10
1	5
4	5
1	6
3	6
4	6
4	8
7	9
7	10
4	9
4	10

B. Vérification de la présence d'un cycle

Celle-ci se fait en vérifiant la présence de couples (x,y) et (y,x) dans la fermeture transitive.

```
select c1.parent, c1.descendant from closure c1 where
exists (
    select c2.parent, c2.descendant from closure c2
    where   c1.parent = c2.descendant
    and c1.descendant = c2.parent);
```

Pas de résultats.
Temps d'exécution total : 0.823 ms
Requête SQL exécutée.

Pas de résultat, donc pas de cycle.

Cette vérification a pour but, avant tout, de vérifier si le projet ne bloquera pas à une étape (une pièce qui en a besoin d'une autre, mais qui elle-même a besoin de la précédente).

C. Création de la nomenclature

Le but de cette nomenclature est d'obtenir la quantité de pièces nécessaires en fonction de chaque couple (parent, descendant) de l'arbre et de la profondeur à laquelle se situe le descendant dans l'arbre.

1. Décomposition en niveaux

La première étape consiste donc à isoler pour chaque couple (parent, descendant), et ce à chaque niveau de l'arbre, le nombre de descendants nécessaires à la réalisation du parent.

Il s'agit, comme pour la table *closure* d'une réquête récursive :

- Création de la table *niveaux*

```
DROP TABLE IF EXISTS niveaux;
CREATE TABLE niveaux(
parent integer NOT NULL,
descendant integer,
qte integer,
niveau integer
);
```

Pas de résultats.
Temps d'exécution total : 22.106 ms
Requête SQL exécutée.

- Requête initiale

```
INSERT INTO niveaux(parent, descendant, qte, niveau)
select *,1 as niveau from madeof;
```

- Requête à répéter 4 fois

```
INSERT INTO niveaux(parent, descendant, qte, niveau)
select n.parent, m.child, m.qte*n.qte, n.niveau+1 from niveaux n, madeof m
where n.descendant = m.parent and n.niveau =
    (select max(niveau) from niveaux);
```

- Requête récursive globale, remplaçant les deux précédentes

```
INSERT INTO niveaux(parent, descendant, qte, niveau)
WITH RECURSIVE niveaux(parent, descendant, qte, niveau) AS
(
    (select *,1 as niveau from madeof)
        UNION
    (select n.parent, m.child, m.qte*n.qte as qte,
n.niveau+1 as niveau
        from niveaux n, madeof m
        where n.descendant = m.parent)
)
select * from niveaux;
```

34 ligne(s) affectée(s).
Temps d'exécution total : 23.776 ms
Requête SQL exécutée.

De même que précédemment, la référence à *niveaux* est interdite dans une sous-requête ; on remplacera donc le mot UNION ALL par UNION pour que la requête récursive ne s'exécute que sur les lignes de l'itération précédente (et non sur la table complète).

Table *niveaux*

Colonne	Type	NOT NULL	Défaut	Contraintes
parent	integer	NOT NULL		
descendant	integer			
qte	integer			
niveau	integer			

parent	descendant	qte	niveau
1	2	1	1
1	3	2	1
2	5	1	1
2	6	2	1
3	2	10	1
3	5	3	1
4	2	1	1
4	7	5	1
5	6	4	1
7	8	1	1
8	9	1	1
8	10	2	1
1	5	1	2
3	5	10	2
4	5	1	2
1	6	2	2
3	6	20	2
4	6	2	2
1	2	20	2
1	5	6	2
2	6	4	2
3	6	12	2
4	8	5	2
7	9	1	2
7	10	2	2
1	5	20	3
1	6	40	3
3	6	40	3
4	6	4	3
1	6	4	3
1	6	24	3
4	9	5	3
4	10	10	3
1	6	80	4

2. Assemblage de la nomenclature

Pour obtenir la nomenclature finale, il n'y a plus qu'à sommer les quantités de pièces nécessaires en fonction des triplets (parent, descendant, niveau)

```
DROP TABLE IF EXISTS nomenclature;
CREATE TABLE nomenclature(
parent integer NOT NULL,
descendant integer,
niveau integer,
qte integer
);

INSERT INTO nomenclature(parent, descendant, niveau, qte)
select parent,descendant,niveau,sum(qte) from niveaux
group by parent,descendant,niveau;
```

30 ligne(s) affectée(s).
Temps d'exécution total : 41.682 ms
Requête SQL exécutée.

A noter : l'ordre des colonnes niveau et qte a été inversé pour plus de lisibilité dans la nomenclature.

Table *nomenclature*

Colonne	Type	NOT NULL	Défaut	Contraintes
parent	integer	NOT NULL		
descendant	integer			
niveau	integer			
qte	integer			

parent	descendant	niveau	qte
1	2	2	20
4	2	1	1
1	6	2	2
1	5	2	7
4	7	1	5
3	6	3	40
3	6	2	32
3	5	2	10
4	8	2	5
5	6	1	4
2	5	1	1
1	6	3	68
1	3	1	2
1	5	3	20
2	6	1	2
4	9	3	5
7	10	2	2
4	5	2	1
1	2	1	1
7	8	1	1
7	9	2	1
4	10	3	10
4	6	2	2
2	6	2	4
1	6	4	80
4	6	3	4
8	10	1	2
3	5	1	3
8	9	1	1
3	2	1	10