Dans l'ancienne functionTable, les fonctions de comparaisons parcourait la table pour trouver une valeur à renvoyer en faisant de nombreux calculs, et cela, à chaque appel de fonctions. Les performances étaient catastrophiques.

J'ai donc changé de stratégie : j'ai créé une procédure « createfunctions() » qui parcours la functionTable pour créer les fonctions de comparaisons. Les fonctions de comparaisons créées de la sorte seront semblables aux fonctions de comparaisons créées « à la main » et n'auront pas à parcourir la table à chaque appel.

Voilà la nouvelle functionTable :

```
create table functionTable (name text, condition text, result bit varying);
insert into functionTable values ('level', 'x numeric, y numeric');
insert into functionTable values ('level', 'x=y or ((0<=x and x<2) and (0<=y and y<2))', '11');
insert into functionTable values ('level', '2<=x and x<5 and 2<=y and y<5 and x!=y', '10');
insert into functionTable values ('level', '0<=x and x<2 and 2<=y and y<5', '01');
insert into functionTable values ('level', '2<=x and x<5 and 0<=y and y<2', '01');
insert into functionTable values ('level', '', '00');
insert into functionTable values ('gender', 'x char(1), y char(1)');
insert into functionTable values ('gender', 'x=y', '1');
insert into functionTable values ('gender', 'x!=y', '0');
```

La première colonne « name » correspond au nom de l'attribut. La seconde colonne « condition » correspond aux conditions que doivent respecter les paramètres x et y de la fonction. La troisième colonne « result » correspond au résultat attendu par la fonction avec les conditions sur x et y respectées.

La procédure « createfunctions() » créer une table « clone » qui est une copie de la functionTable, elle va ensuite supprimer un à un les tuples de clone pour créer les fonctions de comparaisons morceaux par morceaux. Quand la table clone est vide, on la supprime et la procédure s'arrête.

Pour chaque name, la procédure va d'abord chercher l'unique tuple avec un result null. La condition de ce tuple correspond aux paramètres de la fonction de comparaisons.

Ensuite on cherche l'unique tuple de l'attribut avec une condition '', ce tuple est facultatif. Il correspond au cas où aucune autres conditions n'est respectée, c'est donc le result de ce tuple qui sera renvoyé avec le ELSE à la fin de la fonction.

Puis pour tous les autres tuples de l'attributs, on remplie notre CASE de la manière suivante : « when condition then result ».

Voilà par exemple les fonctions flevel et fgender créées par la procédure avec la fonctionTable ci-dessus :

```
SELECT CASE
    WHEN x=y OR ((0<=x AND x<2) AND (0<=y AND y<2)) THEN b'11'
    WHEN 2<=x AND x<5 AND 2<=y AND y<5 AND x!=y THEN b'10'
    WHEN 0<=x AND x<2 AND 2<=y AND y<5 THEN b'01'
    WHEN 2<=x AND x<5 AND 0<=y AND y<2 THEN b'01'
    ELSE b'00'
END
```

```
SELECT CASE
    WHEN x=y THEN b'1'
    WHEN x!=y THEN b'0'
END
```

Si on charge la functionTable, nos tables, nos fonctions d'interprétations, nos fichiers functionTable.sql et createtable_comp.sql, on peut enchaîner les 2 procédures pour créer les opérateurs :

```
CALL createfunctions();
CALL createtable_comp('r');

SELECT * FROM r_comp WHERE LEVEL!=1;
```

level	gender	size
(3.2)	(M)	(72)
(3.5)	(F)	(76)
(40)	(F)	(100)