SQL

Division

Syntaxe

```
 select arg1
from table1
except
select arg1
from
    (
        select arg1, arg2
        from table1, table2
        except
        select arg1, arg2
        from table1
            join table2 using(arg2)
    ) as tab;
```

Division de arg1 par arg2

Grouping sets

Ca sert à avoir plusieurs `group by` pour une unique requête

Syntaxe

```
 select
      arg1, arg2, arg3, count(distinct arg4)
from table1
group by
  grouping sets (
      (arg1, arg2, arg3),
      (arg1, arg2),
      (arg1),
      ()
);
```

Cube

Fait la même chose que `grouping sets` mais des doublons s'ajoute

Syntaxe

```
 select
      arg1, arg2, arg3, count(distinct arg4)
from table1
group by
  cube (arg1, arg2, arg3);
```

Coalesce

Coalesce n'est pas comptée comme fonction d'aggréga donc pas de `group by` .

Syntaxe

```
select
    coalesce(arg1, 'Val affichée si arg1 = null')
from tab;
```

Si le premier argument vaut null alors coalesce fait afficher le deuxième.

String agg

Il me semble que cette fonction marche comme une fonction d'aggréga donc a besoin d'un `group by` (comme count, max, min...)

Syntaxe

```
select string_agg(distinct arg1, ' - ')
from table1;
```

String_agg récupère tout les arg1 et au lieu de les afficher sur plusieurs lignes, elle les affiche sur une seule ligne (dans la colonne correpondante) en les séparant avec la chaine de caractère passer en deuxième paramètre.

Case

C'est un équivalent à switch

Syntaxe

```
select case
          when count(distinct arg1) = 1 then 'L argument'
          when count(distinct arg1) > 1 then 'Les arguments'
          else 'Rien'
       end
from tab1;
```

case remplace le *switch* du java. Les différents cas sont ensuite détaillé avec la syntaxe `WHEN` [condition] `THEN` [Résultat] sauf le `ELSE` qui se passe de condition et du mot clé `THEN`.

Max Over

Syntaxe

```
select type
       max(arg1) over(partition by type)
from tab1;
```

Récupérer le maximum par type (et non le maximum de la requête uniquement)

Lag Over

Syntaxe

```
select type
       lag(salaire) over(partition by type order by salaire)
from tab1;
```

Récupérer la valeur qui vient juste avant la valeur courante dans la table par type

Lead Over

Syntaxe

```
select type
       lead(salaire) over(partition by type order by salaire)
from tab1;
```

Récupérer la valeur qui vient juste après la valeur courante dans la table par type

Row number Over

Syntaxe

```
select type
       row_number() over(partition by type order by salaire desc)
from tab1;
```

Attribuer un numéro (un rank comme dans les jeux video de style ow, fortnite ou lol...) à chaque ligne suivant les critère de tri fourni en paramètre du `over`.

Récursivité

Je vais vulgariser un max.

Syntaxe

```
with recursive nom_cte (arg1, arg2, arg3)
as
    (
        select arg1, arg2, arg3
        from table1
        where condition_d_arret(ref Pallez)
        union
        select arg1, t.arg2
        from nom_cte
                join table1 t using(arg3)
    )

select arg1, arg2, arg3
from nom_cte;
```

Explications

En gros, on a un premier `select` qui permet deux choses.
- D'abord, il sélectionne les données initiales.
- Puis il défini la condition qui va arreter la boucle recursive.
Enfin on a un deuxième `select` qui lui va décrire l'appel récursif en utilisant un `join`. (Le nombre de select peux varier en fonction du nombre de condtions d'arret...)
Tout les `select` sont séparé par `union`

Exemple

TP2 question 17 Affichez tous les personnages ayant des descendants avec leur nombre de descendants. Les afficher par ordre décroissant de fécondité....départagez les ex aequo par ordre alphabètique

```
 with recursive descendants (ancetre, descendant)
as (
  select pere, numpers from personne where pere is not null
  union
  select mere, numpers from personne where mere is not null
  union
  select d.ancetre, p.numpers
  from descendants d join personne p
    on p.pere=d.descendant or p.mere=d.descendant
  )

select p1.nom, p1.prenom , count(descendant) as "count(descendant)"
from descendants d join  personne p1
  on p1.numpers =d.ancetre
group by p1.nom, p1.prenom
order by "count(descendant)" desc, p1.nom, p1.prenom;
```

Dans cette exmple, on peut pas partir de l'ancêtre le plus grand pour retrouver ses descendants donc on fait l'inverse. Ici, il y a deux conditions d'arrêt possbile pour déterminer si une personne est le plus grand ancetre. Il faut que sont pere soit null et que ça mere soit null. Cependant, on veut gérer les cas où ils ne sont pas null en même temps donc on fait 2 requetes 'select' pour définir les conditions d'arret. L'appel récursif, ici c'est : si la personne a un pere ou une mere alors je recupere ce parent.