##Introduction.

La clause '**CASE WHEN ELSE END**' permet d'**attribuer une valeur en fonction de la réalisation d'une condition**. À chaque entrée d'une table on attribut une valeur conditionnellement à sa valeur inititale pour une colonne donnée. Cette clause peut être utilisée avec des clauses telles que 'SELECT', 'GROUP BY' et 'ORDER BY'.

> Cette clause est similaire à la méthode 'ifelse' de *R* ou la méthode 'np.where' de *python*.

Dans ce billet nous allons détailler la façon de mettre en place une clause 'CASE WHEN' puis son implémentation avec les clauses *SELECT*, *WHERE*, *GROUP BY*, *HAVING* et *ORDER BY*. Pour illustrer ces implémentations nous allons utiliser la table *employees* de l'ensemble de données *employees_department*. Ce dernier contient 3 tables : la première nous renseigne sur les employées de l'entreprise, la seconde sur les départements de l'entreprise et la troisième sur les ventes de chaque département.

Voici la base de données :

```
SELECT * FROM employees ;
 employee_id |  employee_name  | department_id | manager_id | salary
-------------+-----------------+---------------+------------+--------
           1 | John Dupont     |             1 |          3 |   5000
           2 | Anthony Smith   |             2 |          3 |   6000
           3 | Michelle Paris  |             1 |          4 |   4500
           4 | Emily Blanc     |             3 |            |   5500
           5 | Camille Johnson |             2 |          4 |   5200
(5 lignes)

SELECT * FROM departments ;
 department_id | department_name | location
---------------+-----------------+----------
             1 | Sales           | New York
             2 | Marketing       | Paris
             3 | Finance         | Tokyo
(3 lignes)
```

Définition.

La clause 'CASE' est similaire à une condition "**if... (elif...) else...**" des langages de programmation usuels. En d'autres termes, elle permet simplement de retourner (générer) un résultat en fonction d'une ou plusieurs conditions.

La clause 'CASE WHEN' se structure comme suit :

```
CASE
    WHEN colonne1 = 1 THEN 'option_A'
    WHEN colonne1 = 2 THEN 'option_B'
    ELSE 'option_C'
END
```

Dans le code précédent on souhaite comparer les entrées de l'attribut 'colonne1' à une condition. Si 'colonne1'

- vaut 1, la clause 'CASE' va retourner la valeur 'option_A' ;
- vaut 2, la clause 'CASE' va retourner la valeur 'option_B' ;
- sinon elle retournera la valeur 'option_C'.

> Notez que la condition 'ELSE' est facultative. Si vous ne la préciser pas, les entrées qui ne répondent à aucune condition se veront attribuer la valeur 'NULL'.

Maintenant que nous avons vu comment structurer la clause 'CASE', voyons comment l'utiliser avec la clause 'SELECT'. Notamment, nous allons attribuer à chaque employée la spécialité disciplinaire du département dans lequel il exerce.

Avec la clause 'SELECT'.

Associer la clause 'CASE' à la clause 'SELECT' va nous permettre de créer une colonne à partir d'une autre. Vous savez que la département 1 est dédié aux activités marketing, le département 2 à la finance et le département 3 aux ressources humaines.

```
SELECT
    employee_name
    , CASE
        WHEN department_id = 1 THEN 'marketing'
        WHEN department_id = 2 THEN 'finance'
        ELSE 'autres départements'
    END AS dpt_activity
FROM employees ;
```

```
  employee_name  |    dpt_activity
-----------------+---------------------
 John Dupont     | marketing
 Anthony Smith   | finance
 Michelle Paris  | marketing
 Emily Blanc     | autres départements
 Camille Johnson | finance
(5 lignes)
```

Vous remqruez la présence de l'alias "AS dpt_activity" à la fin de la clause *CASE WHEN* : cela permet de nommer l'attribut qu'on vient de définir.

Et si on souhaite ajouter de façon définitive cette nouvelle colonne à notre table 'employees' ? Il nous suffit d'utiliser la commande 'CASE' avec la commade 'UPDATE' et 'SET'.

Avec la clause UPDATE.

Voyons comment intégrer notre nouvelle colonne à la table 'employees'. On doit d'abord créer notre nouvelle colonne.

```
ALTER TABLE employees
ADD COLUMN dpt_activity VARCHAR(25);
```

Maintenant, on peut implémenter les valeurs de notre colonne via 'CASE WHEN'.

```
UPDATE employees
SET dpt_activity = (
    CASE
        WHEN department_id = 1 THEN 'marketing'
        WHEN department_id = 2 THEN 'finance'
        ELSE 'autres départements'
    END
);
```

Et voilà, notre nouvel attribut est en place !

Avec une fonction d'aggrégation.

Supposons désormais qu'on souhaite connaitre le nombre de salaires élevés, moyens ou faibles. Pour cela, rien de plus simple, on peut utiliser conjointement la commande 'COUNT' et 'CASE'.

```
SELECT
    COUNT(CASE WHEN salary <= 5000 THEN 1 END) AS faible,
    COUNT(CASE WHEN salary > 5000 AND salary <= 5800 THEN 1 END) AS moyen,
    COUNT(CASE WHEN salary > 5800 THEN 1 END) AS élevé
FROM employees;
```

```
 faible | moyen | élevé
--------+-------+-------
      2 |     2 |     1
(1 ligne)
```

Deux employés ont un salaire faible, deux un salaire moyen. Finalement on dénombre un salaire élevé. Cependant, si on souhaite obtenir ces informations sous une forme, disons, plus conventionnelle : l'attribut 'salary_level' et le nombre d'individu par catégorie de revenu, il serait plus pertinent d'utiliser la clause *CASE* conjointement à *GROUP BY*. Voyons comment obtenir un résultat plu sympa.

Take home messages.

- associé à *SELECT*, *CASE* permet de créer un nouvel attribut temporaire.
- un alias permet de nommer cet attribut temporaire.

- *CASE* peut être utilisé avec *COUNT*.

Avec la clause 'GROUP BY'.

L'instruction suivante nous permet d'arriver à un résultat presque identique. On va utiliser la clause **CASE WHEN avec la clause GROUP BY**.

```
SELECT
    COUNT(salary) AS count_per_salary_level
FROM employees
GROUP BY CASE
            WHEN salary <= 5000 THEN 'faible'
            WHEN salary > 5000 AND salary <= 5800 THEN 'moyen'
            ELSE 'élevé'
        END;
```

```
 count_per_salary_level
------------------------
                      2
                      2
                      1
(3 lignes)
```

Il semblerait que ce résultat ne soit pas tès intéressant : on ne sait pas à quel niveau de revenu correspond chaque comptage. Une solution serait d'utiliser un alias pour nommer le résultat du *CASE WHEN*. Cependant, il n'est **pas possible d'utiliser un alias au sein d'un GROUP BY**. Aussi, on ne peut pas nomer notre attribut. Si vous souhaitez obtenir d'une part, les valeurs de l'attribut 'salary_level', et, d'autre part, le nombre d'employés concernés par chaque catégorie de revenu, vous devez procédez de la façon suivante :

```
SELECT
    CASE
        WHEN salary <= 5000 THEN 'faible'
        WHEN salary > 5000 AND salary <= 5800 THEN 'moyen'
        ELSE 'élevé'
    END AS salary_level,
    COUNT(salary) AS count_per_salary_level
FROM employees
GROUP BY
    CASE
        WHEN salary <= 5000 THEN 'faible'
        WHEN salary > 5000 AND salary <= 5800 THEN 'moyen'
        ELSE 'élevé'
    END;
```

```
 salary_level | count_per_salary_level
--------------+------------------------
 faible       |                      2
 moyen        |                      2
 élevé        |                      1
(3 lignes)
```

Super! On a bien à la fois notre attribut 'salary_level' et le nombre d'employé pour chaque niveau de revenu.

Avec la clause HAVING.

La clause *HAVING* est utile si on veut filtrer les regroupements engendrés par *GROUP BY*. Par exemple, si on veut le nombre d'employé par département qui ont plus de deux employés :

```sql
SELECT
    department_id,
    COUNT(employee_id) AS num_employees
FROM employees
GROUP BY department_id
HAVING
    CASE
        WHEN COUNT(employee_id) >= 2 THEN 1
        ELSE 0
    END ;
```

Oops ! Vous recevez une erreur : *l'argument de HAVING doit être de type boolean, et non du type integer*.

En effet, la clause *having* doit recevoir une valeur de type booléen. Autrement dit, on doit préciser à la fin de la clause *CASE* la valeur suivante : **=1** ; Cela équivaut à écrire une condition booléenne. Précisément, il s'agit d'expliciter qu'on souhaite conserver uniquement la condition 1.

```sql
SELECT
    department_id,
    COUNT(employee_id) AS num_employees
FROM employees
GROUP BY department_id
HAVING
    CASE
        WHEN COUNT(employee_id) >= 2 THEN 1
        ELSE 0
    END = 1 ;
```

```
 department_id | num_employees
---------------+---------------
             2 |             2
```

```
              1 |             2
(2 lignes)
```

Si on explicite **END = 0** on recevra uniquement le groupe qui satisfait à la condition 2, soit les départements avec un unique employé. Observez le résultat :

```
SELECT
    department_id,
    COUNT(employee_id) AS num_employees
FROM employees
GROUP BY department_id
HAVING
    CASE
        WHEN COUNT(employee_id) >= 2 THEN 1
        ELSE 0
    END = 0 ;
```

```
 department_id | num_employees
---------------+---------------
             3 |             1
(1 ligne)
```

Voyons comment générer une condition booléenne avec une chaîne de caractère.

```
SELECT
    department_id,
    COUNT(employee_id) AS num_employees
FROM employees
GROUP BY department_id
HAVING
    CASE
        WHEN COUNT(employee_id) >= 2 THEN 'ok'
        ELSE 'pas ok'
    END = 'ok' ;
```

```
 department_id | num_employees
---------------+---------------
             2 |             2
             1 |             2
(2 lignes)
```

Si vous optez pour "END=1" vous obteindrez l'erreur suivante : Erreur : Aucun opérateur ne correspond au nom donné et aux types d'arguments.

Take home messages.

- *CASE* peut être utilisé avec *GROUP BY*.
- la clause *GROUP BY* ne permet pas d'utiliser un alias.
- *CASE* peut être utilisé avec *HAVING*.
- l'argument de *HAVING* doit être de type booléen.

Avec la clause 'WHERE'.

La clause *WHERE* permet de filtrer une table en fonction d'une ou plusieurs condition. Pour mettre en place cette condition on peut utiliser la valeur d'un attribut. Il est donc possible d'utiliser un *CASE WHEN* pour créer cette condition. Supposons qu'on veuille filtrer notre table en fonction du salaire des employés. Précisément, on souhaite conserver les employés dont le salaire est égal à 5 000€, 5500€ ou 6000€ pour les départements 1 et 2. Pour ce faire, on peut mettre en place un **CASE WHEN** conjointement à l'instruction **IN** associée à **WHERE**.

```
SELECT
    employee_name
    , department_id
    , salary
FROM employees
WHERE
    CASE
        WHEN department_id IN (1, 2) THEN salary
        ELSE NULL
    END IN (5000, 5500, 6000);
```

Comme vous pouvez le voir, on doit définir notre *CASE WHEN* immédiatemment après la déclaration *WHERE*. Finalement, une fois les départements filtrés, on garde uniquement le salaire des employés correspondant aux montants indiqués.

> Notez que la clause *CASE WHEN* peut également retourner la valeur d'un attribut présent dans la table. Dans notre cas, elle renvoi le salaire seulement si l'employé fait partie du département 1 ou 2.

Voici le résultat de la requête.

```
 employee_name | department_id | salary
---------------+---------------+--------
 John Dupont   |             1 |   5000
 Anthony Smith |             2 |   6000
(2 lignes)
```

Ce résultat peut être retrouvé comme suit :

```
SELECT
    employee_name
```

```
    , department_id
    , salary
FROM employees
WHERE salary IN (5000, 5500, 6000) AND department_id IN (1, 2) ;

-- ou
SELECT
    employee_name
    , department_id
    , salary
FROM employees
WHERE salary IN (5000, 5500, 6000)
GROUP BY department_id, employee_name, salary
HAVING department_id IN (1, 2) ;
```

```
 employee_name | department_id | salary
---------------+---------------+--------
 John Dupont   |             1 |   5000
 Anthony Smith |             2 |   6000
(2 lignes)
```

Imaginons qu'on veuille sélectionner les employés, mais avec une logique différente selon leur department_id. Par exemple :

1. Si le department_id est 1, on veut tous les employés dont le salaire est supérieur à 4000.
2. Si le department_id est 2, on veut ceux dont le salaire est inférieur à 5500.
3. Pour tous les autres départements, on veut les employés dont le salaire est exactement égal à 6000.

Voici comment on peut utiliser *CASE WHEN* dans la clause *WHERE* pour implémenter cette logique :

```
SELECT
    employee_name,
    salary,
    department_id
FROM employees
WHERE
    CASE
        WHEN department_id = 1 THEN salary > 4000
        WHEN department_id = 2 THEN salary < 5500
        ELSE salary = 6000
    END;

  employee_name  | salary | department_id
-----------------+--------+---------------
 John Dupont     |   5000 |             1
 Michelle Paris  |   4500 |             1
 Camille Johnson |   5200 |             2
(3 lignes)
```

Take home messages.

- *CASE WHEN* peut **renvoyer une valeur d'attribut**.
- *CASE WHEN* peut être utilisé avec la clause *WHERE*.
- Filtrer notre table en fonction de logiques complexes.

Avec la clause 'ORDER BY'.

Associée à la clause 'ORDER BY', on peut générer un ordonnement du résultat compte tenu des valeurs prises pour une colonne spécifique. Par exemple, si on veut ordonner de façon décroissante les employées pour chaque niveau de revenu, on peut procéder comme suit :

```
SELECT
    employee_name
    , salary
FROM employees
ORDER BY
    CASE
        WHEN salary  <= 5000 THEN 'faible'
        ELSE 'élevé'
    END DESC;
```

```
  employee_name   | salary
------------------+--------
 John Dupont      |   5000
 Michelle Paris   |   4500
 Anthony Smith    |   6000
 Emily Blanc      |   5500
 Camille Johnson  |   5200
(5 lignes)
```

On voit que les employés dont le salaire est inférieur à 5000€ par mois sont bien organisés par ordre décroissant ; ceux avec un salaire supérieur également. On peut également utiliser plusieurs variables d'ordonnancement.

```
SELECT
    employee_name
    , department_id
    , salary
FROM employees
ORDER BY
    CASE
        WHEN  department_id = 1 THEN 'marketing'
        WHEN department_id = 2 THEN 'finance'
        ELSE 'autre'
    END ASC
    , salary DESC;
```

On obtient, pour chaque département (du plus grand au plus petit), les salaires ordonnés du plus élevé au plus faible.

```
 employee_name  | department_id | salary
----------------+---------------+--------
 Emily Blanc    |             3 |   5500
 Anthony Smith  |             2 |   6000
 Camille Johnson |            2 |   5200
 John Dupont    |             1 |   5000
 Michelle Paris |             1 |   4500
(5 lignes)
```

Take home messages.

- *CASE* peut être utilisé avec *ORDER BY*.