

DataCratie

Une image contenant symbole, Graphique, logo, Bleu électrique

Description générée automatiquement

**matthieu gaymay**

**lucas pivet**

**muhammad hakeem bhatoo**

**Rapport Base de Données**

**2024 - 2025**

Table des matières

[I – Modèle Conceptuel de Données (MCD) 3](#_Toc189408731)

[II - Explications des choix d’implémentation 4](#_Toc189408732)

[Utilisateur -> Membre 4](#_Toc189408733)

[Rôle -> Membre -> Groupe 5](#_Toc189408734)

[Membre -> Couleur/Image -> Groupe 6](#_Toc189408735)

[Reaction -> Proposition 7](#_Toc189408736)

[Proposition -> Theme/Budget 8](#_Toc189408737)

[Proposition -> Vote 8](#_Toc189408738)

[III - Création des tables dans la base du projet 9](#_Toc189408739)

[IV – Jeu de données 10](#_Toc189408740)

[V – Triggers et Fonction PL/SQL 14](#_Toc189408741)

[V – Création des vues nécessaires à notre base de données 19](#_Toc189408742)

[Création de la vue\_utilisateurs\_membre\_groupes 19](#_Toc189408743)

[Création de la vue\_utilisateurs\_membre\_groupes 19](#_Toc189408744)

[Création de la vue\_utilisateurs\_membre\_groupes 19](#_Toc189408745)

[Création de la vue\_utilisateurs\_membre\_groupes 19](#_Toc189408746)

# I – Modèle Conceptuel de Données (MCD)

Une image contenant texte, diagramme, Plan, capture d’écran

Description générée automatiquement

*Pour télécharger le MCD (.loo) et ainsi mieux le lire :* [*c'est ici !*](https://gitlab.com/HakeemMHB/datacratie/-/blob/061326753a4bd4fe811361a5ed49bb6d17f4da16/PartieBD_GAYMAY-PIVET-BHATOO/MCD_GAYMAY_PIVET_BHATOO.loo)  
***ou en .png :*** [***c'est ici !***](MCD.png)

*Une image contenant diagramme, Plan, texte, Dessin technique

Description générée automatiquement*

*Après avoir télécharger le MCD en version .loo, vous pourrez également accéder au SR depuis Looping.****ou bien le télécharger :*** [***ici !***](schemaRelationnel.png)

# II - Explications des choix d’implémentation

Dans cette partie, nous allons expliquer les grands choix d’implémentation que nous avons pris lors de la construction de notre base de projet.

## Utilisateur -> Membre

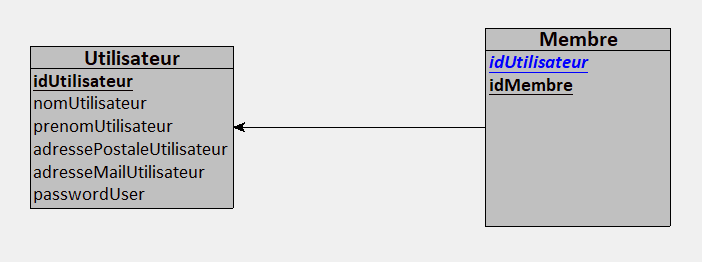
Tout d’abord, les entités Utilisateur et Membre, ainsi que l’association un Utilisateur DEVIENT Membre

Une image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquement

Nous avons fait le choix de distinguer un **UTILISATEUR/internaute** d’un **MEMBRE**. En effet, un UTILISATEUR de l’application n’est pas forcément dans un groupe de discussion. De surcroît, en terme de modélisation, il est plus lisible de les distinguer car, en effet, **un UTILISATEUR peut être MEMBRE de plusieurs groupes**. **Or un MEMBRE, n’est membre que d’un seul groupe**. Au niveau des cardinalités :

* Pour UTILISATEUR : **O,N** car un utilisateur à le choix d’être ou non dans un, ou plusieurs groupes de discussion ;
* Pour MEMBRE : **1,1 (R)** car un MEMBRE **est lié qu’à un seul UTILISATEUR**, il devra donc **récupérer sa clé primaire** ; cependant, comme nous l’avions vu, un UTILISATEUR peut être MEMBRE de plusieurs groupes, il faut donc également **ajouter une clé idMembre** dans l’entité MEMBRE, ce qui fait pour cette entité une paire de **clé (idUtilisateur, idMembre)**. Ensemble, **cette paire de clé garantit l’unicité du MEMBRE**, et nous pouvons le voir sur le schéma relationnel.



## Rôle -> Membre -> Groupe

Ensuite, nous avons les entités MEMBRE, ROLE et GROUPE ainsi que les associations un MEMBRE a pour rôle un/des ROLE et un MEMBRE appartient à un GROUPE :

Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquement

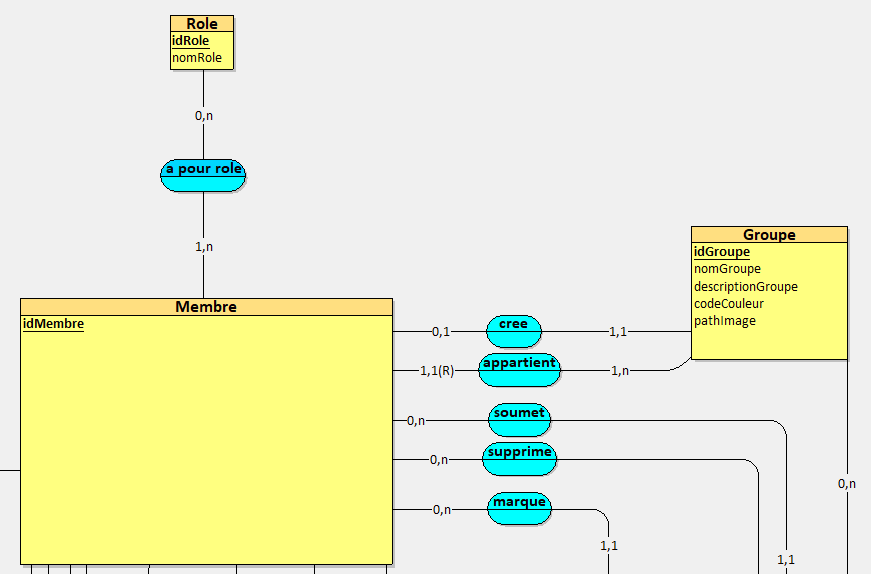
Encore une fois nous avons décidé que idGroupe devait être lié au MEMBRE car, un MEMBRE ne peut être membre que d’un seul GROUPE. Nous aurons donc maintenant pour l’entité MEMBRE une triplé de clé (idMembre, idUtilisateur, idGroupe). Grâce à cette triplé de clé, un MEMBRE est unique et nous retrouve toutes les informations le concernant.

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

## Membre -> Couleur/Image -> Groupe

Ensuite, nous allons voir qu’un MEMBRE (en fonction de son ROLE) peut ajouter une IMAGE ou mettre une COULEUR sur un GROUPE :



Ici, nous pouvons penser que tous les MEMBRES peuvent mettre une image et définir une couleur, or ce n’est pas le cas ; seuls des ROLES spécifiques, comme celui d’administrateur, peuvent le faire. Pour ce faire, nous avons résolu le problème ainsi : **un MEMBRE possède un ou plusieurs ROLES**. Cela est **stocké dans la base** et nous pouvons le voir sur le schéma relationnel :

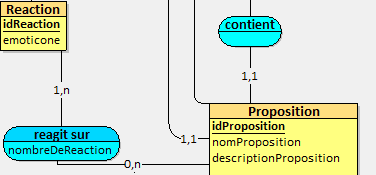
Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquement

**L’association A\_POUR\_ROLE conserve bien ses données**, notamment celles qui indiquent quelles rôles possède un MEMBRE du GROUPE. Ainsi, lorsqu’un MEMBRE souhaitera ajouter une IMAGE ou une COULEUR, s’il ne possède pas le ROLE adéquate, il ne pourra pas le faire (*voir TRIGGER plus bas*). Cette méthodologie s’appliquera pour le reste des associations du MCD. Par exemple, un seul un MEMBRE ayant le ROLE d’administrateur ou modérateur pourra supprimer un COMMENTAIRE.

## Reaction -> Proposition

Ensuite, concernant les REACTIONS que les MEMBRES peuvent mettre sur un COMMENTAIRE ou une PROPOSITION, nous avons décidé que créer une entité REACTION.



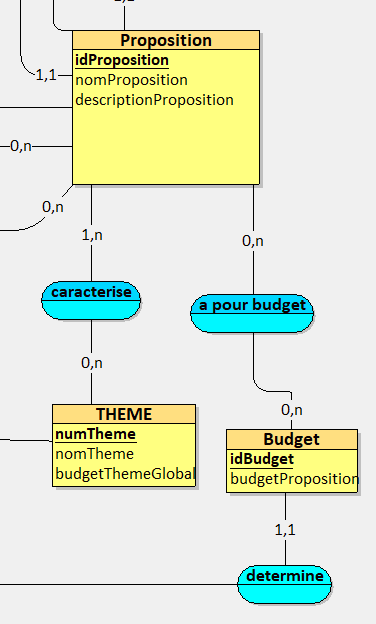
Ce qui est important ici est l’association entre une REACTION et une PROPOSITION. En effet, à partir d’un certain nombre de réactions des MEMBRES, un vote se déclenche automatiquement. Or, il faut stocker le nombre de REACTIONS ; nous avons ainsi décider de faire une association (reagit\_sur) qui générera une **table de correspondance**, qui **contiendra la donnée du nombre de REACTION** d’une PROPOSITION.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

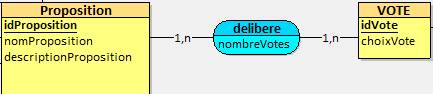
## Proposition -> Theme/Budget

Puis, nous avons décidé que créer une table BUDGET, que le décideur remplira pour chaque PROPOSITION. Le budget de chaque THEME se trouve comme attribut de la table THEME. Et un trigger peut aider à vérifier qu’un BUDGET d’une proposition ne dépasse pas le budget global d’un THEME.



## Proposition -> Vote

Et enfin, pour stocker le nombre de VOTE pour une PROPOSITION, nous avons également opter pour une **table de correspondance**, qui contiendra **le nombre de VOTES pour une PROPOSITION**.



Une image contenant texte, Police, capture d’écran, conception

Description générée automatiquement

# III - Création des tables dans la base du projet

Nous avons créer nos tables sur PhPMyAdmin, en MySQL. Le script de création est fourni dans un dossier .zip avec ce rapport-ci. Vous pouvez également retrouver ce fichier ici.

Veuillez trouver néanmoins ci-contre **la bonne exécution** du script de création des tables.



Et voici les tables qui forment désormais notre base de données :

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

# IV – Jeu de données

Également, pour nos jeux de données, vous retrouver dans ce dossier .zip un fichier contenant l’insertion de notre jeu de données. Vous retrouvez également ce fichier ici.

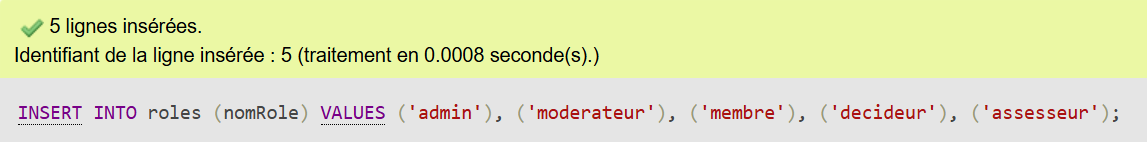
Voici cependant **la bonne exécution** de nos insertions de données :

* **Utilisateur** :

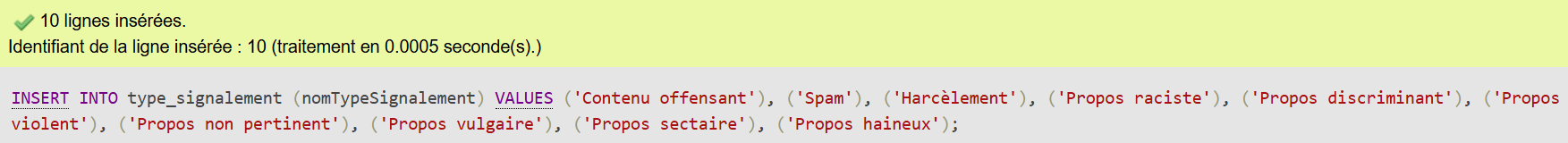
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

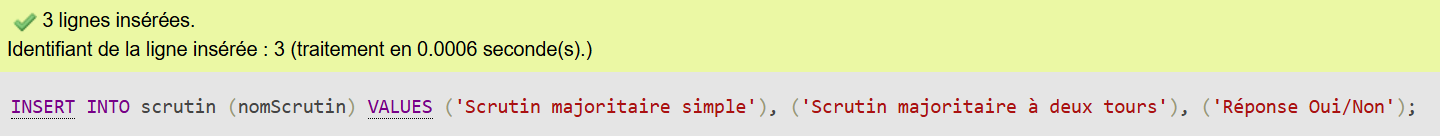
* **Roles** :



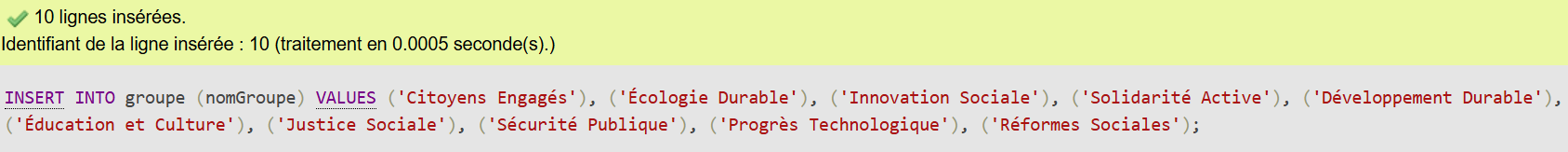
* **Type\_Signalement** :



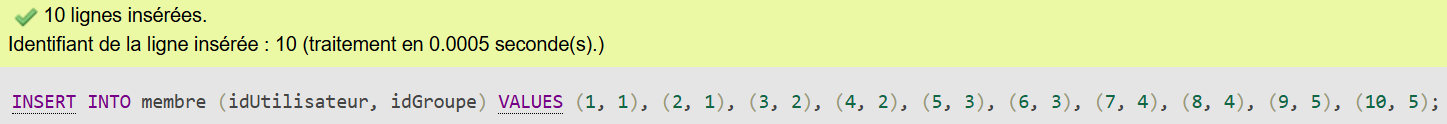
* **Scrutin** :



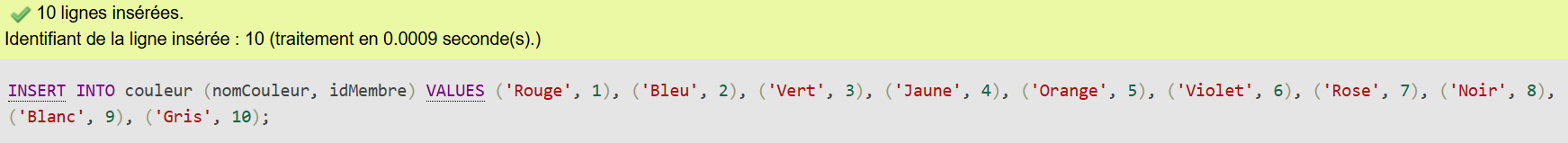
* **Groupe** :



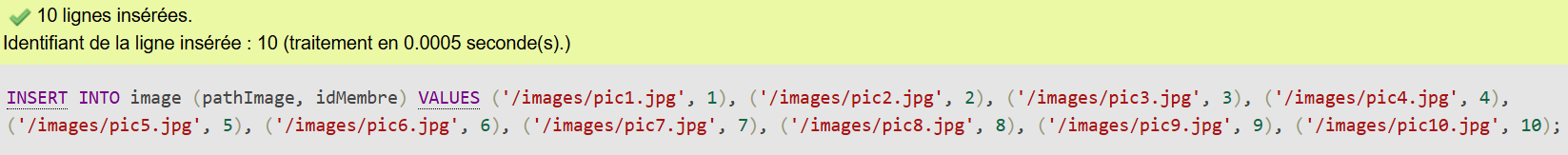
* **Membre** :



* **Couleur** :



* **Image** :

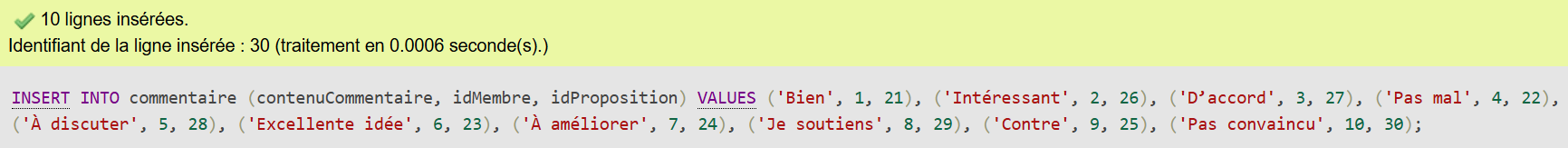


* **Proposition** :

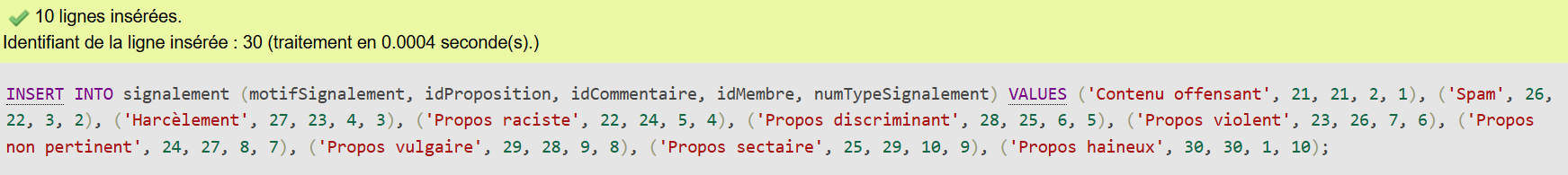
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

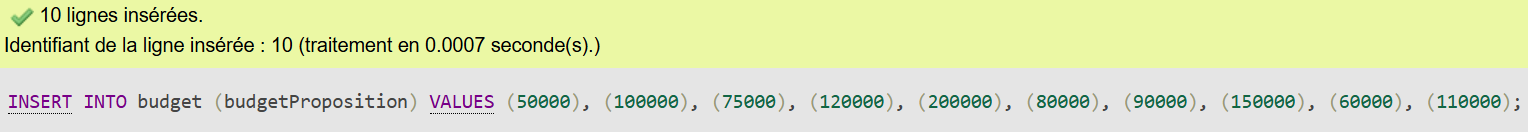
* **Commentaire** :



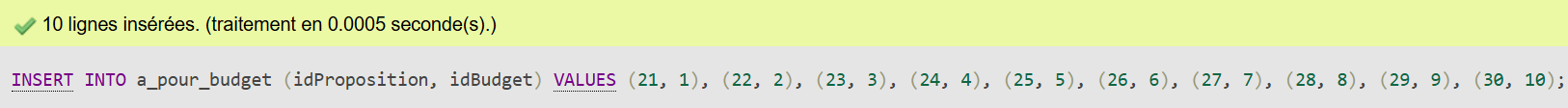
* **Signalement** :



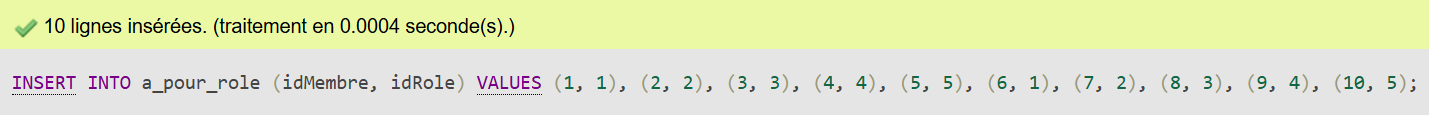
* **Budget** :



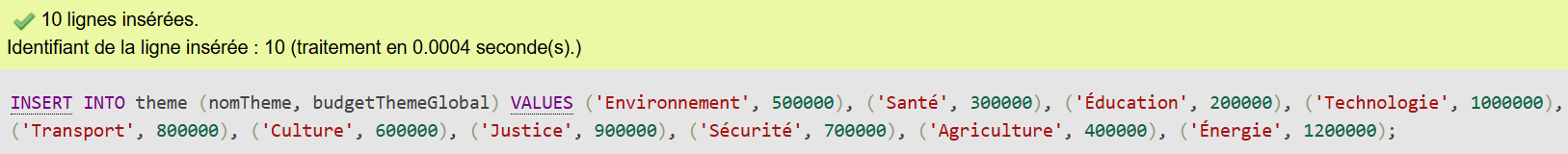
* **A\_Pour\_Budget** :



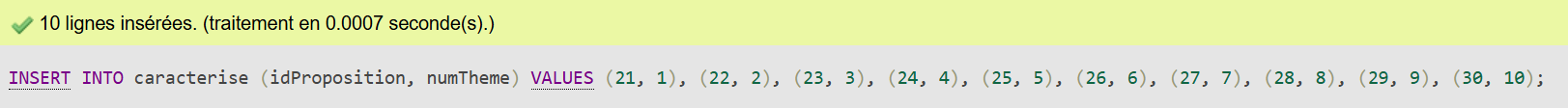
* **A\_Pour\_Role** :



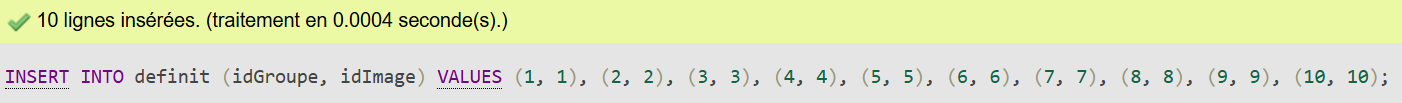
* **Theme** :



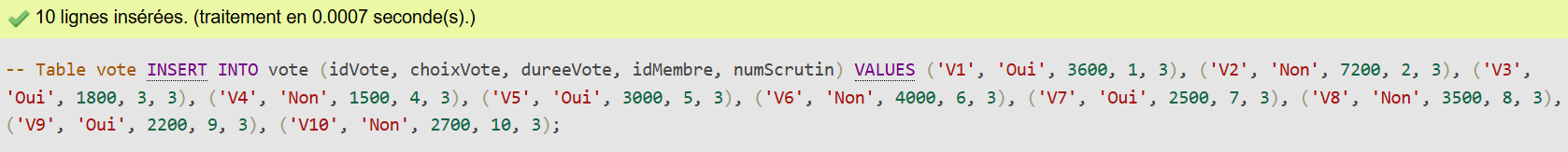
* **Caracterise** :



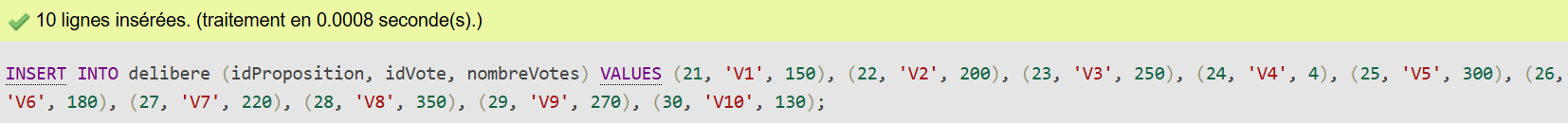
* **Definit** :



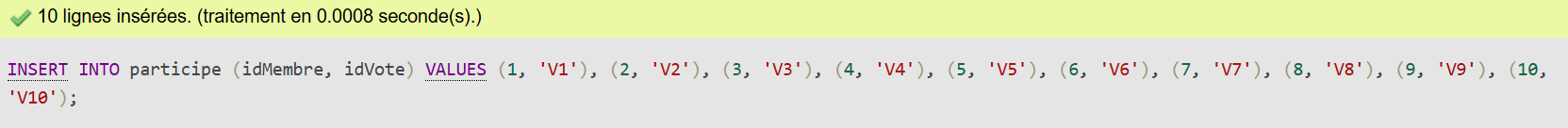
* **Vote** :



* **Delibere** :



* **Participe** :



* **Presente** :

Une image contenant texte, Police, capture d’écran

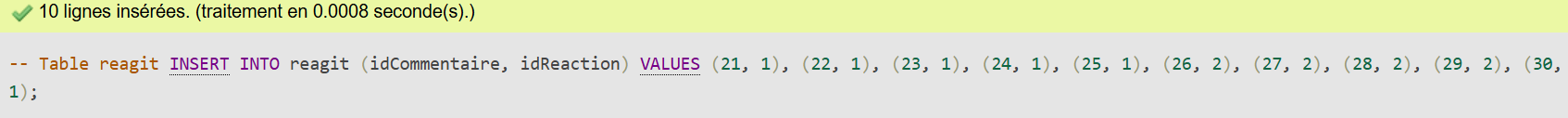
Description générée automatiquement

* **Reaction** :

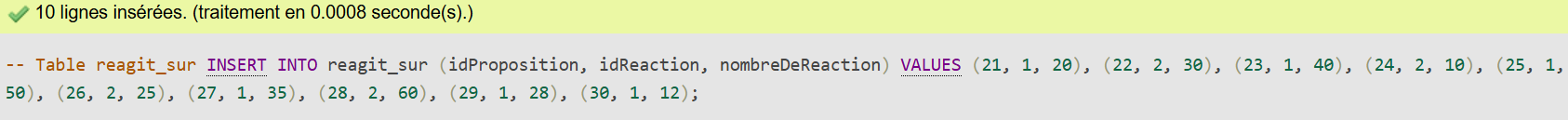
Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

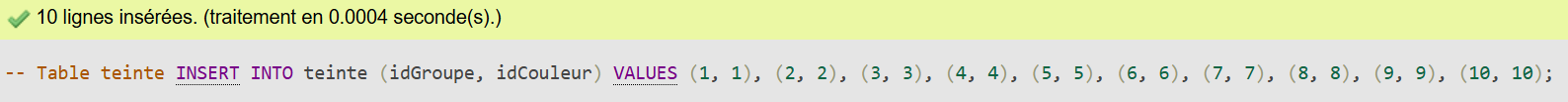
* **Reagit** :



* **Reagit\_sur** :

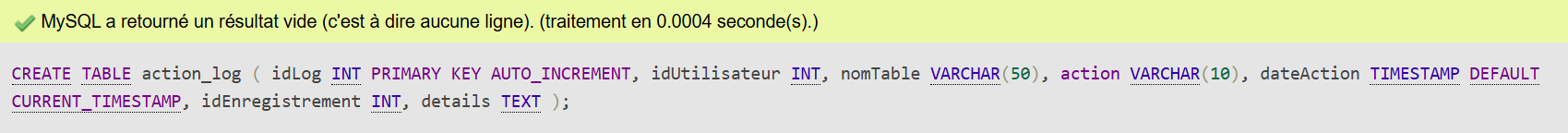


* **Teinte** :



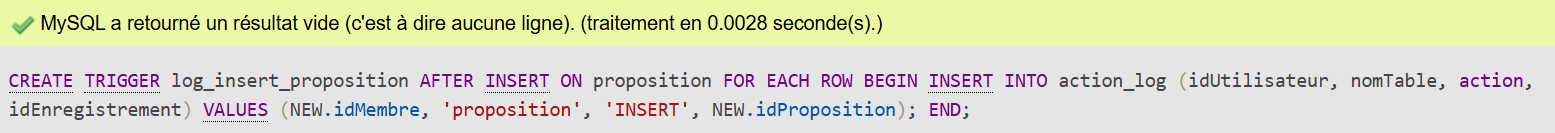
# V – Triggers et Fonction PL/SQL

Nous avons décidé de créer une table action\_log, qui répertorie les actions qui sont effectuées sur les tables de la base de données (INSERT, UPDATE, DELETE).

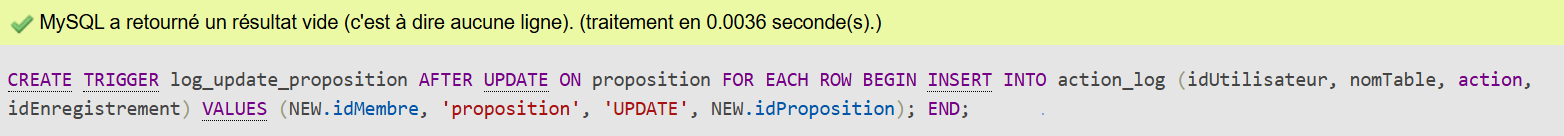


Après avoir créer cette table, nous avons décider de créer trois triggers qui surveilleront la table **PROPOSITION**. En effet, c’est **l’une des tables de données** **les plus importantes** de notre base étant donnée que beaucoup **d’autres tables dépendent d’elle**, et que de surcroît, le but de notre application est de proposer des PROPOSITIONS.

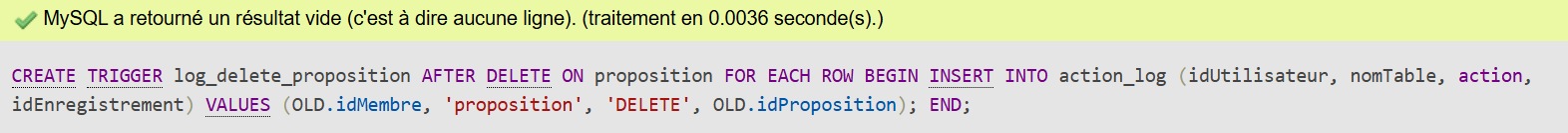
**Un trigger qui surveille l’insertion de données dans la table PROPOSITION :**



**Un trigger qui surveille la mise à jour de données dans la table PROPOSITION :**

****

**Et un trigger qui surveille la supression de données dans la table PROPOSITION :**

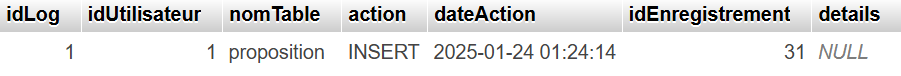
****

Nos triggers crées, nous devons à présent les tester. Nous allons pour celui insérer une PROPOSITION, puis la modifier, et enfin la supprimer et observer si les données sont bien enregistrer dans la table action\_log.

**Insertion d’une PROPOSITION :**

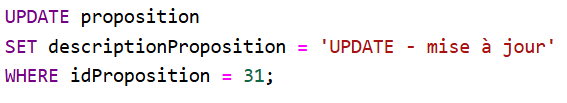






✅

**Et maintenant passons à la modification d’une PROPOSITION :**





Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

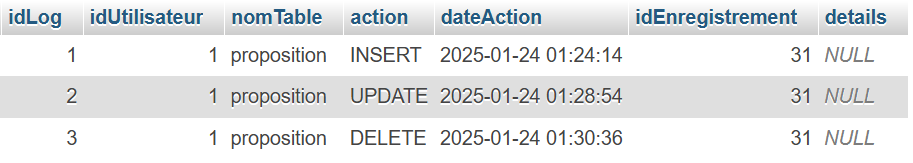
✅

**Et enfin testons la suppression d’une PROPOSITION :**

Une image contenant texte, Police, capture d’écran, blanc

Description générée automatiquement





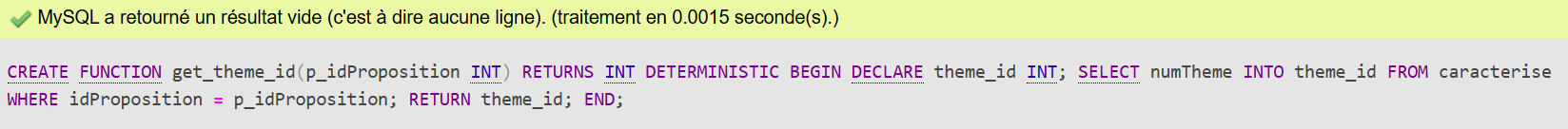
✅

Ce qui est intéressant avec cette table action\_log, c’est que l’on peut créer des triggers pour chaque table, à l’instar de PROPOSITION, pour conserver une trace des actions subit par chaque table.

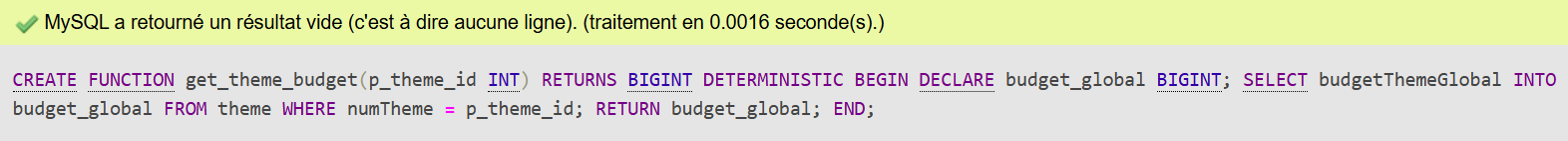
Nous avons cependant eu le besoin de créer un autre trigger concernant la gestion des budgets. En effet, le décideur octroie ou non un BUDGET à une PROPOSITION, et celui-ci ne doit pas excéder le BUDGET total du THEME dans laquelle se trouve la PROPOSITION.

Pour ce faire, nous avons créé plusieurs fonctions PL/SQL annexes que nous utiliserons dans notre trigger.

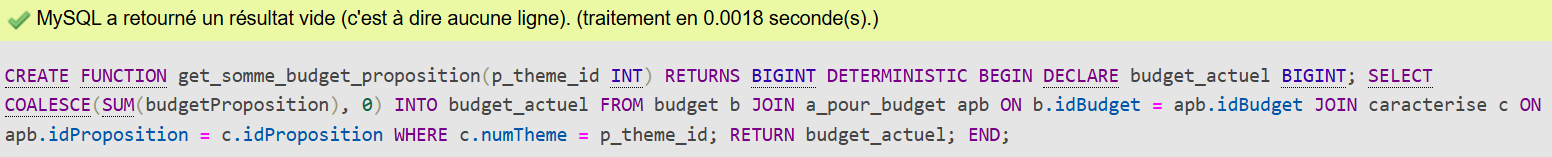
**Tout d’abord, il nous faut une fonction pour récupérer l’identifiant d’un THEME :**



**Également, il nous faudra une fonction pour récupérer le budget d’un THEME :**



**Et enfin, il nous faudra une fonction pour calculer la somme des budget de toutes les propositions d’un THEME :**

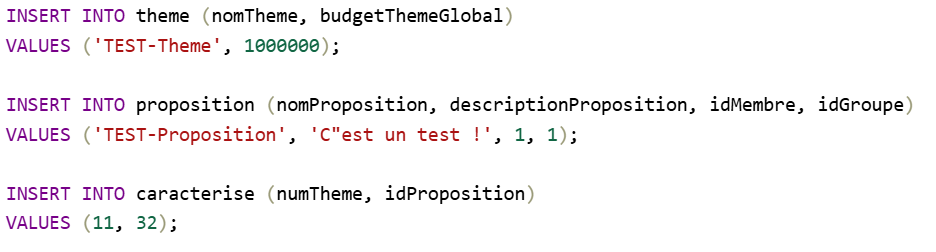


Maintenant, nous pouvons construire notre trigger. Celui-ci fera deux choses : il surveillera que les PROPOSITIONS ne dépassent pas le budget global du THEME ; et qu’une PROPOSITION possède bien un THEME. Si l’un des deux cas n’est pas respecté, le BUDGET ne sera pas accordé.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Maintenant que tout est créés, nous devons tester les comportements attendus.  
  
**Test 1 : Le trigger ne se déclenche pas car le BUDGET de la proposition ne dépasse pas celui du THEME :**



Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, blanc

Description générée automatiquement

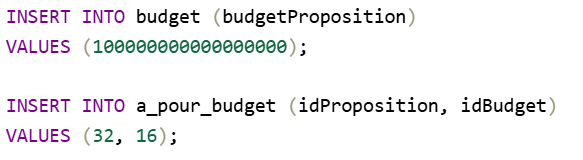
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

**L’insertion s’est produit et il n’y a pas eu de problème. En effet, le BUDGET de la PROPOSITION est de 30 000$, ce qui est inférieur au BUDGET de 1 000 000$ du THEME.**

✅

**Test 2 : Le trigger se déclenche car le BUDGET de la proposition dépasse celui du THEME :**



Une image contenant texte, Police, capture d’écran

Description générée automatiquement

**Le comportement attendu est correct ; le trigger empêche l’insertion du BUDGET.**

✅

**Test 3 : Le trigger se déclenche car la proposition n’appartient à aucun THEME :**

Une image contenant texte, capture d’écran, Police, algèbre

Description générée automatiquement

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

**Le comportement attendu est correct ; le trigger empêche l’insertion du BUDGET car la PROPOSITION ne possède pas de THEME.**

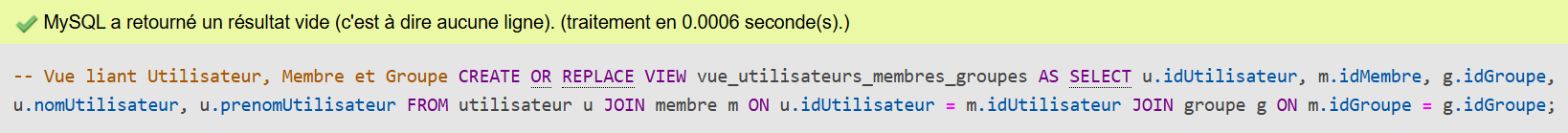
✅

**Ces fonctions et triggers serviront et aideront en particulier le Décideur, lorsque celui-ci sera amener à distribuer les BUDGETS à chaque PROPOSITION.**

# V – Création des vues nécessaires à notre base de données

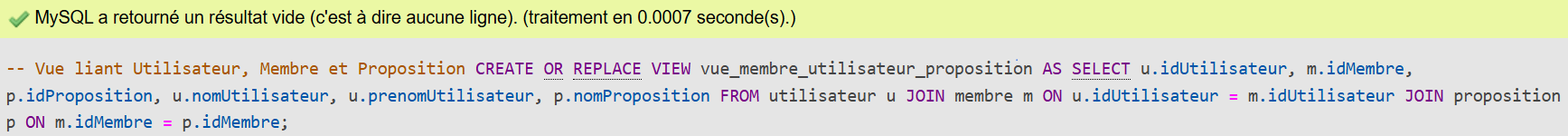
Nous avons décidé de créer des vues qui seront **nécessaire** lors de notre projet. En effet, les vues **nous éviterons de faire beaucoup de jointures dans nos requêtes** MySQL. De surcroît, elles agiront comme des tables de données, et auront l’avantage de **conserver toutes les données** qui nous sont nécessaires **au même endroit** (c’est-à-dire dans le vue elle-même).

## Création de la vue\_utilisateurs\_membre\_groupes



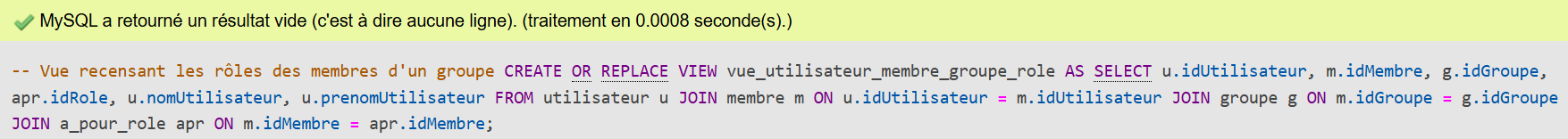
Cette vue regroupe toutes les données qui nous sont nécessaire pour **identifier un Membre** et/ou **un Utilisateur**. En effet, elle va regrouper les données qui nous sont indispensables mais qui se trouvent dans **différentes table**, en **une seule et unique vue**. Cela va nous éviter de faire des requêtes trop longue avec beaucoup de jointures.

## Création de la vue\_utilisateurs\_membre\_groupes



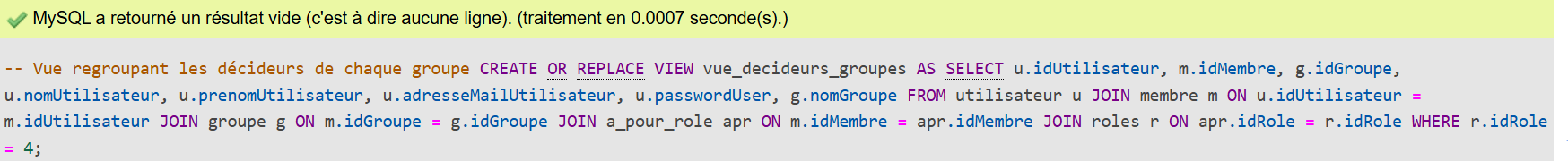
Cette vue est intéressante et utile car elle regroupe comme données les Propositions de chaque Membre. En bref, elle lorsqu’on l’interroge, elle nous renvoie **chaque Proposition, ainsi que l’auteur de celle-ci, ainsi que le Groupe dans lequel il appartient**. Si cette vue n’existait pas, nous devrions faire des jointures entre Utilisateur, Membre, Groupe et Proposition, à chaque requête, ce qui est **fastidieux** et **redondant**.

## Création de la vue\_utilisateurs\_membre\_groupes



Cette simple vue nous permet simplement de savoir quelle est le rôle de chaque Membre dans son Groupe

## Création de la vue\_utilisateurs\_membre\_groupes



Cette vue est **indispensable** pour l’application Java du Décideur, notamment lors de l’authentification de celui-ci. Si cette vue n’existait pas, nous devrions interroger la table Utilisateur, Membre, a\_pour\_role et Groupe pour savoir si la personne qui tente de se loguer est un Décideur. Grâce à cette vue, on a simplement besoin de **l’interroger** et de voir si la personne est dedans, et si tel est le cas, c’est qu’elle est un Décideur et peut accéder à l’application Java.