

Exemple de correction – Activité de la partie 6

Nombre de commentaires par article

```
1
2
3
4
5
6
7
8
9 ALTER TABLE Article
10 ADD nb_commentaires INT NOT NULL DEFAULT 0;
11
12 DELIMITER |
13 CREATE TRIGGER after_insert_commentaire AFTER INSERT
14 ON Commentaire FOR EACH ROW
15 BEGIN
16     UPDATE Article SET nb_commentaires = nb_commentaires + 1 WHERE id = NEW.article_id;
17 END |
18
19 CREATE TRIGGER after_delete_commentaire AFTER DELETE
20 ON Commentaire FOR EACH ROW
21 BEGIN
22     UPDATE Article SET nb_commentaires = nb_commentaires - 1 WHERE id = OLD.article_id;
23 END |
24
25 CREATE TRIGGER after_update_commentaire AFTER UPDATE
26 ON Commentaire FOR EACH ROW
27 BEGIN
28     UPDATE Article SET nb_commentaires = nb_commentaires - 1 WHERE id = OLD.article_id;
29     UPDATE Article SET nb_commentaires = nb_commentaires + 1 WHERE id = NEW.article_id;
30 END |
31 DELIMITER ;
32
33
```

Excellent choix cet ajout d'une colonne à la table Article pour stocker le nombre de commentaires

Bonne idée d'ajouter la valeur par défaut

Il manque quelque chose là... Rien n'initialise la valeur de nb_commentaires, pourtant il y a déjà des commentaires dans la table. Toujours faire attention lorsqu'on modifie des tables alors que des données ont déjà été insérées !

C'est plus logique d'utiliser AFTER plutôt que BEFORE : le nombre de commentaires ne change qu'après l'insertion (/modification/suppression) d'un commentaire

Le BEGIN ... END n'était pas nécessaire, puisqu'il n'y a qu'une requête dans le corps de ces deux triggers

Bien vu, il ne fallait pas oublier l'UPDATE. Ce sera sans doute beaucoup moins courant que INSERT ou DELETE, mais ça peut toujours arriver !

Dans tous les cas, ces deux updates sont exécutés si l'on modifie un commentaire, pourtant ce n'est utile que si l'article lié au commentaire est modifié. Une condition n'aurait donc pas été de trop

L'énoncé demandait :

- 1/ de stocker le nombre de commentaire de façon durable ;
- 2/ de faire en sorte que ce soit toujours à jour.

Il fallait donc stocker le nombre de commentaires dans une table. L'idéal étant d'utiliser la table *Article*. Pas besoin de créer une nouvelle table pour stocker uniquement cette donnée.

Pour la mise à jour, il fallait que ce fût immédiat. Pas de procédure stockée donc, puisqu'il faut l'exécuter manuellement. Par contre, un trigger est parfait !

Résumé par défaut

```
1
2
3
4
5
6 CREATE VIEW V_Article
7 AS SELECT id, titre, COALESCE(resume, SUBSTRING(contenu, 1, 150)), contenu, auteur_id, date_publication, nb_commentaires
8 FROM Article;
9
10
11
12
13
14
15
16
```

Utiliser une vue n'est pas une mauvaise idée. Ça permet effectivement de récupérer un résumé même si l'auteur n'en a pas mis. Deux remarques cependant :
1/ Ca oblige à toujours utiliser la vue. Ce n'est pas un problème en soit, mais il ne faudrait pas oublier de modifier les requêtes de sélection
2/ Ca recalcule la colonne resume à chaque fois. Ce serait plus optimal de stocker ça directement à l'insertion, avec un trigger par exemple

Il aurait fallu utiliser un alias pour récupérer facilement le résumé

Excellente utilisation de deux fonctions imbriquées pour récupérer
1/ resume s'il contient une valeur
2/ les 150 premiers caractères de contenu si resume est NULL

Ici, l'idéal aurait été de créer le résumé lors de l'insertion si l'auteur n'en a pas écrit, via un trigger.

Créer une vue qui remplace le résumé par les 150 premiers caractères du contenu si le résumé est NULL est intéressant mais ça oblige à recalculer à chaque fois le résumé.

Ca ne sert donc finalement qu'à simplifier la requête, ça n'optimise rien.

Statistique utilisateur

```
1
2
3
4
5
6
7
8 CREATE TABLE VM_Stat (
9   id INT UNSIGNED,
10  pseudo VARCHAR(100) NOT NULL,
11  nb_articles INT NOT NULL DEFAULT 0,
12  dernier_article DATETIME,
13  nb_commentaires INT NOT NULL DEFAULT 0,
14  dernier_commentaire DATETIME,
15  PRIMARY KEY(id)
16 );
17
18 INSERT INTO VM_Stat
19 SELECT u.id, u.pseudo, COUNT(DISTINCT a.id), MAX(a.date_publication), COUNT(DISTINCT c.id), MAX(c.date_commentaire)
20 FROM Utilisateur AS u
21 LEFT JOIN Article AS a ON a.auteur_id = u.id
22 LEFT JOIN Commentaire AS c ON c.auteur_id = u.id
23 GROUP BY u.id, u.pseudo;
24
25
26
27
28
29 DELIMITER |
30 CREATE PROCEDURE maj_vm_stat()
31 BEGIN
32   DELETE FROM VM_Stat;
33
34   INSERT INTO VM_Stat
35   SELECT u.id, u.pseudo, COUNT(DISTINCT a.id), MAX(a.date_publication), COUNT(DISTINCT c.id), MAX(c.date_commentaire)
36   FROM Utilisateur AS u
37   LEFT JOIN Article AS a ON a.auteur_id = u.id
38   LEFT JOIN Commentaire AS c ON c.auteur_id = u.id
39   GROUP BY u.id, u.pseudo;
40 END |
41 DELIMITER ;
```

Il fallait bien utiliser une vue matérialisée puisqu'on ne voulait pas recalculer les données chaque fois (ce qui aurait été le cas avec une vue)

Bonne utilisation de INSERT-INTO... SELECT
Il aurait été possible également de combiner CREATE TABLE et SELECT pour créer la table et en initialiser les valeurs en une seule requête avec la syntaxe CREATE TABLE... SELECT, vue en même temps que les tables temporaires (mais pas réservée à celles-ci)

Ne pas oublier le DISTINCT

Pour mettre à jour les vues matérialisées, on a vu deux techniques : les triggers et les procédures stockées.
L'énoncé précisait qu'il n'était pas nécessaire que les valeurs soient tout le temps à jour, mais qu'il fallait pouvoir les recalculer.
La procédure stockée était donc le bon choix !

Et pour terminer, il était nécessaire d'utiliser une vue matérialisée pour les statistiques des utilisateurs.

Une vue ne stocke pas les données et une table temporaire disparaît avec la session en cours.

Il était possible d'utiliser les triggers pour mettre les données à jour mais :

1/ l'énoncé précisait que les données ne devaient pas être à jour en permanence ;

2/ vu la nature des données, cela aurait nécessité des triggers sur plusieurs tables, après insertion, modification et suppression, donc six triggers en tout. Pourquoi faire compliqué si l'on peut faire simple ?

Notes

Implémentation des comportements : 3/3

Le résumé par défaut n'est pas vraiment créé comme il était demandé.

Cependant l'intention est bonne et la vue permet effectivement de récupérer le résumé avec une valeur par défaut quand nécessaire.

Les deux autres comportements ont été correctement implémentés. Ca vaut donc 2,5 qu'on arrondit au supérieur.

Premier comportement (nombre de commentaires) : 1/2

Le nombre de commentaires est correctement stocké dans la table Article, mais les données ne sont pas initialisées. Le trigger sur modification peut être largement optimisé avec une simple condition en plus.

Deuxième comportement (résumé) : 1/2

Comme cela a déjà été dit, il aurait été mieux d'utiliser un trigger à l'insertion pour stocker directement le résumé par défaut en cas de besoin.

Troisième comportement (statistiques) : 2/2

Très bonne utilisation du concept de vue matérialisée et d'une procédure pour la mettre à jour. Les commandes pourraient être simplifiées, notamment en utilisant la syntaxe CREATE TABLE... SELECT, mais c'est parfaitement fonctionnel

Lisibilité du code : 1/1

Le code est bien indenté.

Total : 8/10