

Programmation de bases de données: Transactions

Plan de la séance:

- Retour sur la dernière séance:
 - Point de vue des enseignants.
 - Point de vue des étudiants.
- Introduction
- La propriété ACID d'une transaction
- COMMIT et ROLLBACK
- TRY ...CATCH

Retour sur la dernière séance : Point de vue des enseignants:

- › Bon point pour les étudiants: continuez d'être attentifs.
- › Transact-SQL est l'extension du SQL pour SQL Server. il permet d'améliorer la performance du SGBD
- › Transact-SQL c'est essentiellement du SQL auquel on a ajouté des déclarations et manipulation de variables, des structures de contrôles.
- › Les variables sont précédées du symbole @
- › BEGIN –END permettent de délimiter un BLOC.

Introduction, Transactions

- › **Notions de Transactions :**
- › Une transaction est un bloc d'instructions DML exécuté et qui laisse la base de données dans un état cohérent.
- › Si une seule instruction dans le bloc n'est pas cohérente alors la transaction est annulée, toutes les opérations DML sont annulées. Le principe de transaction est implémenté dans tous les SGBDs.
- › Pour tous les SGBD certaines transactions sont atomiques et donc auto-commit, instruction individuelle qui n'ont pas de BEGIN Transaction (pour SQL Server)

- › Une transaction a la propriété ACID

La propriété ACID

A: Atomicité

Une transaction doit être une unité de travail indivisible ; soit toutes les modifications de données sont effectuées, soit aucune ne l'est.

C: Cohérent

Lorsqu'elle est terminée, une transaction doit laisser les données dans un état cohérent

Lorsqu'une transaction a débuté, elle doit se dérouler correctement jusqu'à la fin (validée), sans quoi l'instance du moteur de base de données annule toutes les modifications effectuées sur les données depuis le début de la transaction

La propriété ACID

I: Isolement

Les modifications effectuées par des transactions concurrentes doivent être isolées transaction par transaction → système de verrous.

Une transaction reconnaît les données dans l'état où elles se trouvaient avant d'être modifiées par une transaction simultanée, ou les reconnaît une fois que la deuxième transaction est terminée, mais ne reconnaît jamais un état intermédiaire.

D: Durabilité

Lorsqu'une transaction durable est terminée, ses effets sur le système sont permanents. Les modifications sont conservées même en cas de défaillance du système

Transactions, principe

Une transaction débute par un **begin transaction** et termine par un **commit** ou un **rollback**.

L'opération **commit** détermine le point où la base de données est de nouveau **cohérente**.

L'opération **rollback** annule toutes les opérations et retourne la base de données dans l'état où elle était au moment du **begin transaction**, donc du dernier **commit**.

Transactions, principe

Définition: la variable @@TRANCOUNT contient le nombre de BEGIN TRANSACTION de la connexion actuelle.

BEGIN TRANSACTION: permet de débiter une transaction. Cette instruction incrémente la variable @@TRANCOUNT de 1

COMMIT, permet d'officialiser la transaction. Ce qui permet de diminuer la variable @@TRANCOUNT de 1 ;

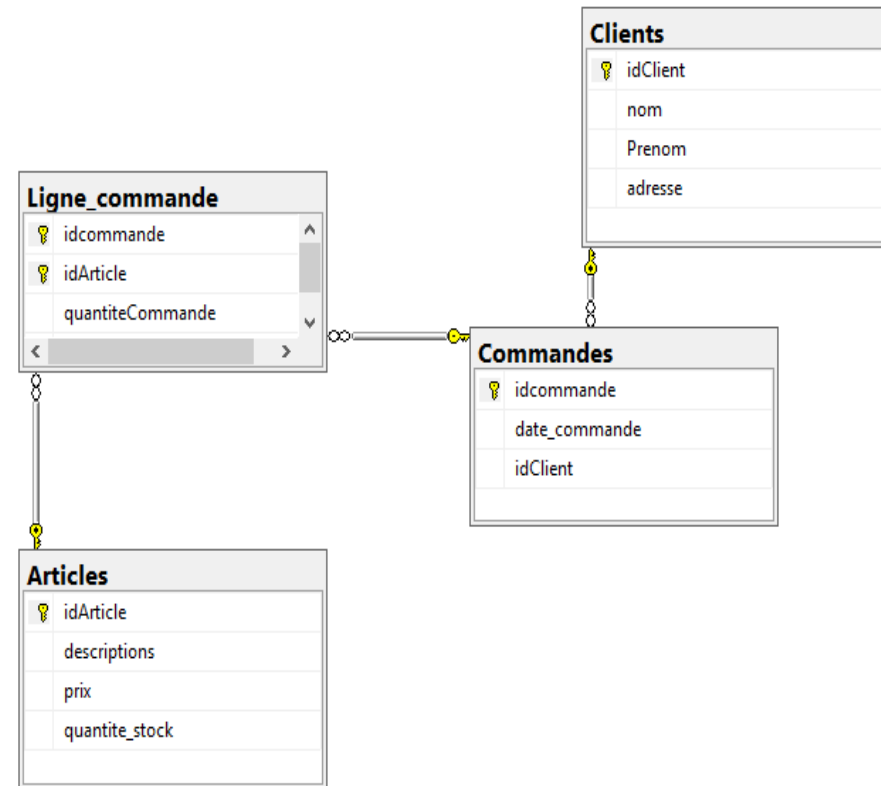
ROLLBACK permet d'annuler une transaction qui n'a pas été COMMITÉ et ramène la variable @@TRANCOUNT à zéro.

Lorsque @@TRANCOUNT >0, cela veut dire qu'il y a encore des transaction en suspend. Il est fort possible que la table sur laquelle porte la transaction soit verrouillée.

Il est très important de ne pas imbriquer des transactions.

Transactions, Exemple 1

1. Écrire le code Transact-SQL qui permet de faire les opérations suivantes : (comme un TOUT, doit être dans un bloc BEGIN et END)
 - a. On crée une nouvelle commande pour le client numéro 6. (ou un client de votre choix. (Donc on insère dans la table Commandes)
 - b. On insère dans la table Ligne_commande la commande que l'on vient de créer pour l'article numéro 7 (ou un article de votre choix)
 - c. Si la quantité dans la table Articles n'est pas suffisante : ce qui veut dire que la quantité commandée par le client est plus grande ou égale à la quantité en stock, alors on annule la transaction.
 - d. Sinon(Si la quantité dans la table Articles est suffisante) alors
 - i. On met à jour la table Articles par la nouvelle quantité en stock. La nouvelle quantité est égale à la quantité initiale moins la quantité commandée.
 - ii. On met à jour le montant de la commande pour cet article dans la table Ligne_commande.
 - iii. On officialise la transaction



Transactions, Exemple 1

```
begin
```

```
declare @idcommande int,@quantite int,@quaCde int =20,@prix money,@idclient smallint =6,@idArticle int =7;
```

```
select @prix = prix from Articles where idArticle=7;
```

```
    begin transaction
```

```
        INSERT INTO Commandes(datecommande,idClient) values('2024-08-29',@idclient);
```

```
        select @idcommande=@@IDENTITY;
```

```
        insert into Lignecommande (idcommande,idArticle, quantiteCommande) values (@idcommande,@idArticle,@quaCde);
```

```
        select @quantite = quantitestock from Articles where idArticle =@idArticle;
```

```
            if (@quantite <= @quaCde) rollback;
```

```
            else
```

```
                Begin
```

```
                update Articles set quantitestock =quantitestock-@quaCde where idArticle =@idArticle;
```

```
                update Lignecommande set montant =@quaCde* @prix
```

```
                where idArticle =@idArticle and idcommande =@idcommande;
```

```
                commit; end;
```

```
end;
```

Transactions, Exemple 2

Exemple2:

```
begin transaction
```

```
insert into CategorieTrivia values('y', 'physique', 'mauve');
```

Remarquez que la transaction ne se termine pas correctement. Il n'y a ni COMMIT ni ROLLBACK.

Si vous essayez de faire un `SELECT * FROM CategorieTrivia` vous n'aurez pas de résultat. La table est verrouillée. La BD n'est pas dans un état cohérent.

```
@@TRANCOUNT =1
```

Transactions, principe

TRY ... CATCH

Il arrive que nous voulions que toutes les opérations à l'intérieur d'une transaction soient TOUTES exécutées si toutes les opérations DML sont correctes, dans ce cas, il faudra les inclure à l'intérieur d'un block TRY CATCH.

Exemple:

Dans l'exemple qui suit, remarquez le INSERT suivant:

```
INSERT INTO ReponseTrivia (LaReponse, estBonne, idQuestion)
```

```
values('Victor Hugo', 'Z', @idQuestion);
```

La valeur est soit n ou o

Transactions, Exemple 3

```
BEGIN
DECLARE @idQuestion int;
BEGIN TRY
    BEGIN TRANSACTION
    INSERT INTO QuestionTrivia(enonce, difficulte, flag, idCategorie) VALUES
('Qui a écrit: les neiges de kilimandjaro ?', 'm', 'n', 'a');
    SELECT @idQuestion = @@IDENTITY;
    INSERT INTO ReponseTrivia (LaReponse, estBonne, idQuestion) values ('John Steinbeck', 'n', @idQuestion);
    INSERT INTO ReponseTrivia (LaReponse, estBonne, idQuestion) values ('Ernest Hemingway', 'o', @idQuestion);
    INSERT INTO ReponseTrivia (LaReponse, estBonne, idQuestion) values ('Victor Hugo', 'z', @idQuestion);
    INSERT INTO ReponseTrivia (LaReponse, estBonne, idQuestion) values ('Edgar Frank Codd', 'n', @idQuestion);
    COMMIT;
END TRY
BEGIN CATCH
ROLLBACK;
END CATCH; END
```



CONCLUSION



QUESTIONS ??