Les transactions

Définition

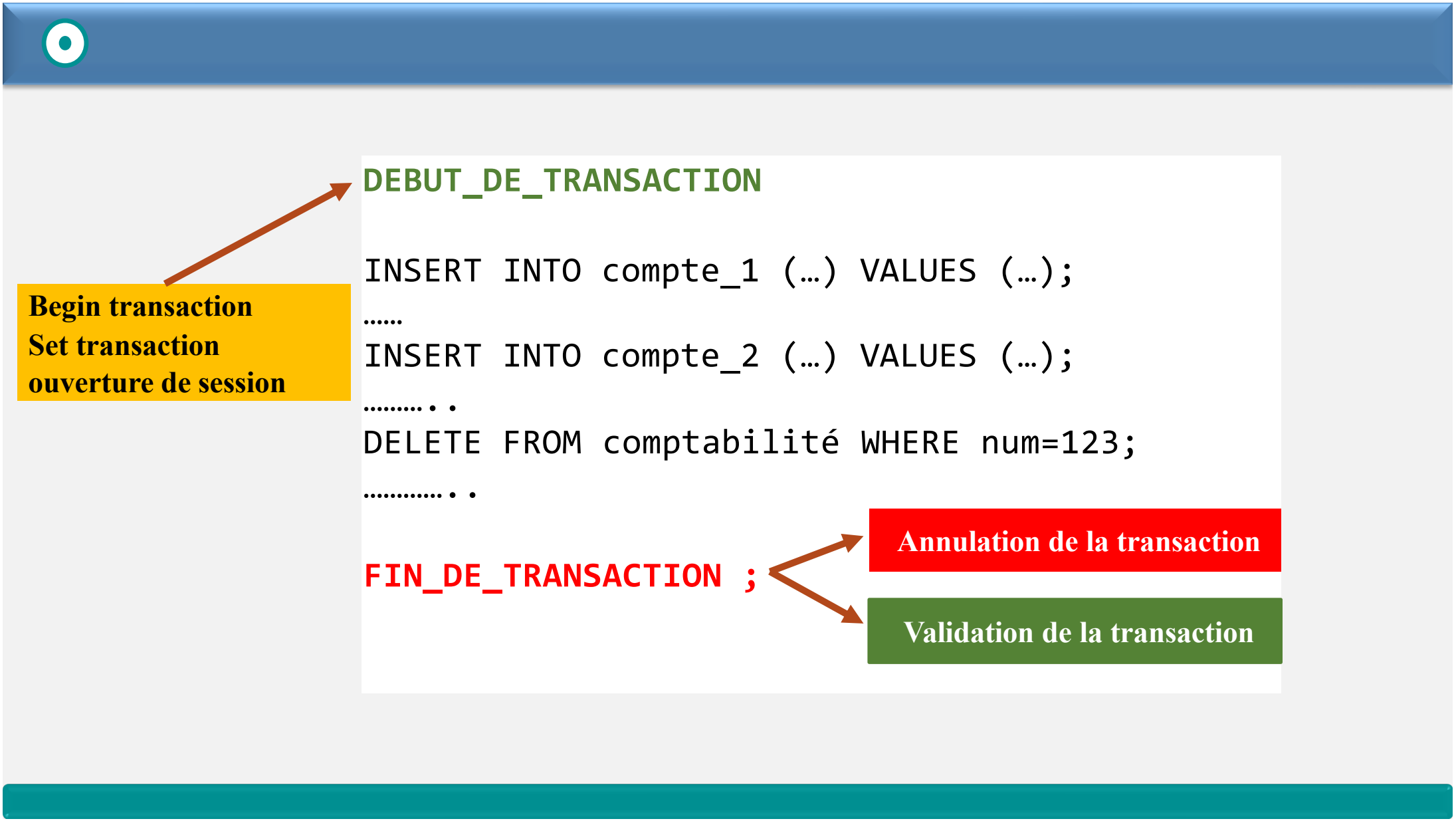
En SQL Oracle, une **transaction** est un ensemble d'opérations SQL exécutées comme une unité logique unique. Toutes les opérations d'une transaction doivent être validées ensemble pour que leurs effets soient appliqués à la base de données. Si l'une des opérations échoue, la transaction peut être annulée pour rétablir la base de données dans son état initial, comme si la transaction n'avait jamais été exécutée.

Une transaction regroupe plusieurs étapes en une seule opération tout ou rien

Intérêt

L'intérêt des transactions réside dans les propriétés **ACID** (Atomicité, Cohérence, Isolation et Durabilité) qui garantissent la fiabilité des opérations dans un environnement multi-utilisateur :

1. **Atomicité** : Une transaction est tout ou rien. Soit toutes les opérations de la transaction sont exécutées avec succès, soit aucune ne l'est. Si une erreur survient, Oracle permet d'annuler toutes les opérations de la transaction.
2. **Cohérence** : La transaction amène la base de données d'un état valide à un autre. Elle garantit que toutes les contraintes de la base de données sont respectées.
3. **Isolation** : Chaque transaction est isolée des autres, évitant que les transactions concurrentes ne se voient les unes les autres en cours d'exécution, ce qui prévient les incohérences.
4. **Durabilité** : Une fois validée (commit), les modifications de la transaction sont permanentes et seront toujours présentes, même en cas de panne système.


DEBUT_DE_TRANSACTION
INSERT INTO compte_1 (…) VALUES (…);
……
INSERT INTO compte_2 (…) VALUES (…);
……..
DELETE FROM comptabilité WHERE num=123;
……..
FIN_DE_TRANSACTION ;
Begin transaction
Set transaction
ouverture de session
Annulation de la transaction
Validation de la transaction

Imaginons un système de transfert bancaire. Lors d'un transfert, une transaction va :

- Débiter le compte de l'expéditeur.
- Créditez le compte du destinataire.

```
-- Débiter le compte de l'expéditeur
UPDATE COMPTE
SET solde = solde - 100
WHERE compte_id = 1;

-- Créditez le compte du destinataire
UPDATE COMPTE
SET solde = solde + 100
WHERE compte_id = 2;
```

Si une erreur survient lors de la deuxième mise à jour (par exemple, une contrainte échoue ou une panne de connexion), le compte 1 sera débité, mais le compte 2 ne sera pas crédité, créant une incohérence dans les données.

```
BEGIN
    -- Débiter le compte de l'expéditeur
    UPDATE COMPTE
    SET solde = solde - 100
    WHERE compte_id = 1;

    -- Créditez le compte du destinataire
    UPDATE COMPTE
    SET solde = solde + 100
    WHERE compte_id = 2;

    -- Valider la transaction
    COMMIT;
END;
```

- **BEGIN** : Démarre la transaction.
- Les deux mises à jour sont exécutées dans une seule transaction.
- **COMMIT** : Valide la transaction, rendant les changements permanents si tout est exécuté avec succès.

Si une erreur se produit, la transaction sera incomplète et il faudra alors relancer la requête.

La commande **COMMIT** valide une transaction en rendant toutes les modifications permanentes dans la base de données. Une fois qu'un **COMMIT** est exécuté, les changements effectués par la transaction deviennent visibles pour tous les utilisateurs, et ils ne peuvent plus être annulés.

Son Rôle est :

- **Finaliser une transaction** : COMMIT marque la fin d'une transaction réussie.
- **Rendre les changements permanents** : Une fois le COMMIT exécuté, les modifications apportées par les requêtes de la transaction sont définitivement enregistrées dans la base de données.
- **Permettre la visibilité des modifications** : Les modifications deviennent accessibles aux autres utilisateurs de la base de données.

```
BEGIN
    -- Débiter le compte de l'expéditeur
    UPDATE COMPTE
    SET solde = solde - 100
    WHERE compte_id = 1;

    -- Créditez le compte du destinataire
    UPDATE COMPTE
    SET solde = solde + 100
    WHERE compte_id = 2;

    -- Annuler la transaction
    ROLLBACK;
END;
```

- **BEGIN** : Démarre la transaction.
- Deux mises à jour sont faites, mais elles ne sont pas encore définitives.
- **ROLLBACK** : Annule la transaction, ce qui ramène la base de données à son état initial avant les mises à jour.

Dans cet exemple, les mises à jour sont annulées par le **ROLLBACK**,

ROLLBACK

La commande **ROLLBACK** annule une transaction en ramenant la base de données à son état initial avant le début de la transaction. Cela signifie que toutes les modifications effectuées dans la transaction sont annulées et ne seront pas enregistrées dans la base de données.

Son Rôle est :

- **Annuler une transaction** : ROLLBACK est utilisé pour revenir en arrière et annuler tous les changements dans une transaction en cours.
- **Préserver l'intégrité des données** : En cas d'erreur ou de problème, ROLLBACK garantit que les données restent dans un état cohérent en annulant les changements incomplets ou incorrects.
- **Réinitialiser les changements temporaires** : Toutes les modifications depuis le dernier COMMIT (ou le début de la transaction si aucun COMMIT n'a été effectué) sont annulées.

```
BEGIN
    -- Débiter le compte de l'expéditeur
    UPDATE COMPTE
    SET solde = solde - 100
    WHERE compte_id = 1;

    -- Créditez le compte du destinataire
    UPDATE COMPTE
    SET solde = solde + 100
    WHERE compte_id = 2;

    -- Simuler une condition d'échec
    IF (SELECT solde FROM COMPTE WHERE compte_id = 1) < 0 THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Erreur : Le solde est insuffisant, la transaction est
annulée.');
        RETURN;
    END IF;

    -- Valider la transaction
    COMMIT;
END;
```

```
BEGIN
    UPDATE COMPTE SET solde = solde - 100 WHERE compte_id = 1;
    SAVEPOINT apres_debit;

    UPDATE COMPTE SET solde = solde + 100 WHERE compte_id = 2;

    -- Si une erreur est détectée
    ROLLBACK TO apres_debit;  -- Annule seulement les changements
depuis apres_debit
    COMMIT;
END;
```

Après le **ROLLBACK TO apres_debit** et le **COMMIT** :

- Le compte de l'expéditeur (compte_id = 1) est débité de 100 unités.
- Le compte du destinataire (compte_id = 2) n'a subi aucun changement.

SAVEPOINT est un point de sauvegarde intermédiaire dans une transaction. Il permet de définir un point spécifique auquel on peut revenir avec **ROLLBACK** en cas d'erreur, sans annuler toute la transaction. Plusieurs **SAVEPOINTS** peuvent être définis dans une seule transaction.

Rôle :

- **Définir des étapes intermédiaires dans une transaction** : SAVEPOINT permet de diviser une transaction en plusieurs étapes.
- **Permettre un retour partiel en arrière** : Avec **ROLLBACK TO SAVEPOINT**, on peut annuler seulement les modifications effectuées après un SAVEPOINT, sans annuler la transaction entière.
- **Contrôler la gestion des erreurs** : Cela permet une gestion d'erreurs plus fine en choisissant de ne pas annuler toute la transaction.