Groupe F - Requêtes SQL

Hudayfa Koujdal - Hugo Ligneres - Soukaina Mourabit - Samantha Ortega

UE L315 - Semaine 2

Table des matières

1 Insérer des données

1.1 Ajouter 5 nouveaux clients dans la table Clients.

```
INSERT INTO clients (nom, prenom, telephone, mail, adresse, date_naissance, password)

VALUES

    ('Dupont', 'Jean', '0612345678', 'jean.dupont@example.com', 1, '1985-05-15', '
       motdepasse1'),

    ('Martin', 'Sophie', '0623456789', 'sophie.martin@example.com', 2, '1990-08-22',
       'motdepasse2'),

    ('Leroy', 'Pierre', '0634567890', 'pierre.leroy@example.com', 3, '1978-03-10', '
       motdepasse3'),

    ('Moreau', 'Alice', '0645678901', 'alice.moreau@example.com', 4, '1995-11-30', '
       motdepasse4'),

    ('Bernard', 'Luc', '0656789012', 'luc.bernard@example.com', 5, '1982-07-25', '
       motdepasse5');
```

Cette requête SQL insère cinq nouvelles lignes dans la table clients en spécifiant les colonnes `nom`, `prenom`, `telephone`, `mail`, `adresse`, `date_naissance`, et `password`. Chaque ligne contient des informations personnelles pour un client, telles que le nom, le numéro de téléphone, l'adresse e-mail, une adresse (référencée par un ID), la date de naissance, et un mot de passe. Cette opération ajoute donc ces clients à la base de données.

1.2 Créer 3 nouveaux comptes pour chaque client ajouté

```
INSERT INTO compte (id_client, numero_compte, type_compte, IBAN, solde,
    decouvert_autorise, montant_decouvert) VALUES (1, 'FR7612345678901234567890111',
    'Courant', 'FR7612345678901234567890111', 1500.000, 0, NULL), (1, '
    FR7612345678901234567890222', 'Epargne', 'FR7612345678901234567890222', 5000.000,
     0, NULL), (1, 'FR7612345678901234567890333', 'Joint', '
    FR7612345678901234567890333', 10000.000, 0, NULL),

(2, 'FR7623456789012345678901444', 'Courant', 'FR7623456789012345678901444',
    2000.000, 0, NULL), (2, 'FR7623456789012345678902555', 'Epargne', '
    FR7623456789012345678902555', 6000.000, 0, NULL), (2, '
    FR7623456789012345678903666', 'Joint', 'FR7623456789012345678903666', 12000.000,
    0, NULL),

(3, 'FR7634567890123456789014777', 'Courant', 'FR7634567890123456789014777',
    2500.000, 0, NULL), (3, 'FR7634567890123456789015888', 'Epargne', '
    FR7634567890123456789015888', 7000.000, 0, NULL), (3, '
    FR7634567890123456789016999', 'Joint', 'FR7634567890123456789016999', 15000.000,
    0, NULL),

(4, 'FR7645678901234567890127111', 'Courant', 'FR7645678901234567890127111',
    3000.000, 0, NULL), (4, 'FR7645678901234567890128222', 'Epargne', '
```

```
    FR7645678901234567890128222', 8000.000, 0, NULL), (4, '
    FR7645678901234567890129333', 'Joint', 'FR7645678901234567890129333', 18000.000,
    0, NULL),

(5, 'FR7656789012345678901230444', 'Courant', 'FR7656789012345678901230444',
    3500.000, 0, NULL), (5, 'FR7656789012345678901231555', 'Epargne', '
    FR7656789012345678901231555', 9000.000, 0, NULL), (5, '
    FR7656789012345678901232666', 'Joint', 'FR7656789012345678901232666', 20000.000,
    0, NULL);
```

Cette requête SQL insère 3 nouveaux comptes pour chaque client ajouté dans la table `compte`. Chaque client a un compte de type `Courant`, un compte `Épargne`, et un compte `Joint`. Les informations insérées incluent l'id_client, le numero_compte, le type_compte, l'IBAN, le solde, le decouvert_autorise (toujours à 0), et le montant_decouvert (toujours NULL). En résumé, la requête ajoute 15 comptes au total (3 comptes $\times$ 5 clients)

1.3 Insérer 10 transactions pour différents comptes

```
INSERT INTO transactions (id_compte, type_transaction, montant, date_transaction,
    description, statut) VALUES (1, 'Depot', 500.00, '2023-10-01 09:15:00', 'Depot
    initial', 'Effectuee'), (1, 'Retrait', 200.00, '2023-10-02 14:30:00', 'Retrait au
     distributeur', 'Effectuee'),

(2, 'Depot', 1000.00, '2023-10-03 10:00:00', 'Depot mensuel', 'Effectuee'), (2, '
    Virement', 300.00, '2023-10-04 16:45:00', 'Virement a un ami', 'Effectuee'),

(3, 'Depot', 1500.00, '2023-10-05 11:20:00', 'Depot pour investissement', 'Effectuee'
    ), (3, 'Retrait', 700.00, '2023-10-06 12:10:00', 'Retrait pour achats', '
    Effectuee'),

(4, 'Depot', 2000.00, '2023-10-07 08:50:00', 'Depot salaire', 'Effectuee'), (4, '
    Virement', 400.00, '2023-10-08 17:30:00', 'Virement pour facture', 'Effectuee'),

(5, 'Depot', 2500.00, '2023-10-09 13:00:00', 'Depot bonus', 'Effectuee'), (5, '
    Retrait', 1000.00, '2023-10-10 18:00:00', 'Retrait pour vacances', 'Effectuee');
```

Cette requête effectue dix insertions dans la table `transactions` pour enregistrer différents types d'opérations financières sur plusieurs comptes.

2 Lire des données

2.1 Sélectionner les clients ayant un solde supérieur à 10 000 €

```
SELECT clients.id_client, compte.numero_compte, clients.nom, clients.prenom, compte.
    solde

FROM compte

JOIN clients ON compte.id_client = clients.id_client

 WHERE solde>10000;
```

Cette requête sélectionne les clients ayant un solde supérieur à 10 000 €, en joignant les tables compte et clients sur l'ID du client.

2.2 Afficher toutes les transactions effectuées le mois dernie

```
SELECT *

FROM transactions

WHERE MONTH(date_transaction) = MONTH(CURRENT_DATE - INTERVAL 1 MONTH);
```

Cette requête affiche toutes les transactions effectuées le mois dernier, en utilisant la fonction `MONTH()` pour comparer le mois de la transaction avec le mois précédent à partir de la date actuelle.

2.3 Lister tous les comptes avec un découvert autorisé

```
SELECT *

FROM `compte`

WHERE decouvert_autorise=1;
```

Cette requête liste tous les comptes ayant un découvert autorisé, en filtrant les résultats avec la condition `decouvert_autorise=1`.
Pour que le `tinyint` de `decouvert_autorise` soit soit comme un boolean et que ce soit que 0 (false) ou 1 (true) accepté, il faut rentrer la requête suivante :

```
ALTER TABLE compte ADD COLUMN decouvert_autorise TINYINT(1) DEFAULT 0, ADD CONSTRAINT
    chk_decouvert_autorise CHECK (decouvert_autorise IN (0, 1));
```

3 Mettre à jour des données

3.1 Mettre à jour le numéro de téléphone d'un client spécifique

```sql
UPDATE clients

SET telephone = '0123456789'

WHERE id_client = 1 ;
```

Cette requête SQL met à jour le numéro de téléphone d'un client spécifique dans la table clients. Elle remplace la valeur du champ telephone par 0123456789 uniquement pour l'enregistrement où `id_client` est égal à 1 (par exemple), grâce à la condition `WHERE`. Ainsi, seule la ligne correspondant à ce client est modifiée, sans affecter les autres enregistrements de la table.

3.2 Augmenter le découvert autorisé pour certains comptes

```sql
UPDATE compte

SET decouvert_autorise = 1, montant_decouvert = 500

WHERE id_compte = 3 ;
```

Cette requête SQL modifie les paramètres de découvert d'un compte spécifique dans la table compte. Elle active l'autorisation de découvert (`decouvert_autorise = 1`) et fixe le montant maximum de découvert à 500 (`montant_decouvert = 500`) uniquement pour le compte ayant l'identifiant `id_compte = 3`, grâce à la condition `WHERE`.

3.3 Modifier le statut des transactions en attente

```sql
UPDATE transactions

SET statut = 'Validee'

WHERE statut = 'En attente' ;
```

Cette requête SQL met à jour toutes les transactions dans la table transaction qui ont actuellement un statut de `'En attente'`. Elle remplace la valeur du champ statut par `'Validée'` pour ces enregistrements. Toutes les transactions qui répondent à la condition `statut = 'En attente'` sont affectées, tandis que les autres restent inchangées.

4 Supprimer des données

4.1 Supprimer les comptes inactifs depuis plus de 2 ans

```
DELETE FROM `compte`
    WHERE `id_compte` NOT IN (
    SELECT DISTINCT `id_compte`
    FROM `transactions`
WHERE `date_transaction` >= DATE_SUB(NOW(), INTERVAL 2 YEAR)
);
```

D'abord, on extrait la liste des comptes ayant des transactions dans les deux dernières années avec `SELECT DISTINCT ‘id_compte‘ FROM ‘transactions‘`, en utilisant le filtre `WHERE ‘date_transaction‘ >= DATE_SUB(NOW(), INTERVAL 2 YEAR)`. Ensuite, on englobe cette sous-requête avec la requête principale `WHERE ‘id_compte‘ NOT IN (...)`, qui identifie les comptes dont les `id_compte` ne correspondent pas au filtre de la sous-requête.

4.2 Effacer les transactions refusées ou annulées

```
DELETE FROM `transactions`
WHERE `statut` IN ('Annulee','Rejetee');
```

On supprime les transactions avec ces statuts grâce à la condition
`WHERE ‘statut‘ IN (’Annulee’,’Rejetee’)`, qui sélectionne les transactions avec un statut spécifique.

4.3 Retirer les clients sans transactions actives

```
DELETE FROM `clients`
WHERE `id_client` NOT IN (
    SELECT DISTINCT `id_client`
    FROM `compte` c
    JOIN `transactions` t ON c.`id_compte` = t.`id_compte`
);
```

Dans un premier temps, au niveau de la ligne `JOIN ‘transactions‘ t ON c.‘id_compte‘ = t.‘id‘_compte‘` on met en place une jointure entre les tables `compte` et `transactions` grâce à la clé étrangère `id_compte`. Ensuite, grâce à cette jointure et à `SELECT DISTINCT ‘id_client‘ FROM ‘compte‘ c` , on extrait une liste des clients qui ont au moins une transaction associée à leur compte. Dans un second temps, on identifie les clients qui ne font pas partie de cette liste extraite, avec `WHERE ‘id_client‘ NOT IN (...)`, puis on supprime ces clients.

5 Requêtes complexes

5.1 Compter le nombre total de transactions par type de compte

```
SELECT c.type_compte, COUNT(t.id_transaction) AS nombre_transactions FROM compte c
    JOIN transactions t ON c.id_compte = t.id_compte GROUP BY c.type_compte;
```

Cette requête compte le nombre total de transactions pour chaque type de compte en utilisant une jointure entre les tables `compte` et `transactions`. La clause `JOIN` relie les deux tables via la clé id_compte commune aux deux tables. La fonction d'agrégation `COUNT(t.id_transaction)` calcule le nombre de transactions associées à chaque type de compte. La clause `GROUP BY c.type_compte` regroupe les résultats par type de compte, permettant de compter les transactions pour chaque type distinct.

5.2 Calculer la moyenne des soldes de tous les comptes épargne

```
SELECT AVG(c.solde) AS moyenne_solde_epargne FROM compte c WHERE c.type_compte = '
    Compte epargne';
```

Cette requête utilise la fonction d'agrégation `AVG` pour calculer la moyenne des soldes des comptes dans la table compte. La clause `WHERE` filtre les enregistrements pour ne retenir que ceux dont le champ type_compte est égal à "Compte épargne". Chaque ligne correspondant à cette condition est utilisée pour calculer la moyenne des valeurs de la colonne solde. Le résultat de la fonction `AVG(c.solde)` est ensuite présenté avec l'alias `moyenne_solde_epargne`.

5.3 Trouver les 5 clients les plus actifs en termes de transactions

```
SELECT id_client, nom, prenom, COUNT(transactions.id_transaction) AS
    nombre_transactions

FROM clients JOIN transactions ON clients.id_client = transactions.id_compte

GROUP BY clients.id_client

ORDER BY nombre_transactions

 DESC LIMIT 5;
```

Cette requête trouve les 5 clients ayant effectué le plus de transactions. Elle fait une jointure entre les tables `clients` et `transactions` en associant l'`id_client` de `clients` avec l'`id_compte` de `transactions`, puis elle compte le nombre de transactions par client. Ensuite, les résultats sont triés par ordre décroissant du nombre de transactions et limités à 5.

5.4 Lister les prêts dont la durée restante est inférieure à un an

```
SELECT id_pret, id_client, montant, taux, date_debut, duree, statut

FROM prets
```

```
WHERE DATE_ADD(date_debut, INTERVAL duree MONTH) > CURDATE()

AND DATE_ADD(date_debut, INTERVAL duree MONTH) <=       DATE_ADD(CURDATE(), INTERVAL
    12 MONTH);
```

Cette requête liste les prêts dont la durée restante est inférieure à un an. Elle utilise la fonction `DATE_ADD` pour calculer la date de fin du prêt en ajoutant la durée (en mois) à la date de début (`date_debut`). Ensuite, elle compare cette date de fin à la date actuelle (`CURDATE()`) et sélectionne les prêts dont la date de fin est supérieure à la date actuelle mais inférieure ou égale à la date actuelle plus 12 mois (c'est-à-dire dans les 12 prochains mois). Cela permet de filtrer les prêts dont il reste moins d'un an à rembourser.

5.5 Afficher le total des prêts accordés par conseiller

```
SELECT con.nom,SUM(p.montant) AS total_pret FROM prets p
JOIN clients c ON p.id_client = c.id_client
JOIN conseillers con ON c.id_client = con.id_client
GROUP BY con.id_conseiller, con.nom;
```

La requête affiche le nom de chaque conseiller et le montant total des prêts accordés par chaque conseiller. Elle utilise une jointure entre les tables prêts, clients, et conseillers pour relier les informations, puis agrège les montants des prêts avec `SUM` et regroupe les résultats par ID et nom de conseiller.

6 Requêtes avancées

6.1 Identifier les clients avec un total d'investissements supérieur à leur solde total

```sql
SELECT c.id_client,

SUM(i.montant) AS total_investissements,

SUM(c.solde) AS total_solde

FROM compte c

JOIN investissements i ON c.id_client = i.id_client

GROUP BY c.id_client

HAVING total_investissements > total_solde ;
```

Cette requête SQL identifie les clients dont le total des investissements dépasse leur solde total. Elle calcule la somme des montants investis et des soldes pour chaque client en reliant les tables `compte` et `investissements`. Ensuite, elle filtre les résultats pour ne conserver que les clients ayant des investissements supérieurs à leur solde.

6.2 Trouver les comptes ayant le plus haut taux de transactions réussie

```sql
SELECT t.id_compte,

COUNT(CASE WHEN t.statut IN ('Validee', 'Effectuee') THEN 1 END) * 100.0 / COUNT(*)
    AS taux_reussite

FROM transactions t

GROUP BY t.id_compte

ORDER BY taux_reussite DESC
```

Cette requête SQL calcule le taux de réussite des transactions pour chaque compte. Elle compte le nombre de transactions avec un statut `'Validée'` ou `'Effectuée'`, le divise par le total des transactions pour ce compte, puis multiplie par 100 pour obtenir un pourcentage. Les résultats sont regroupés par `id_compte`, triés par taux de réussite décroissant (`taux_reussite DESC`) afin de mettre en avant les comptes avec les meilleurs taux.

6.3 Lister les clients qui n'ont pas utilisé de services de prêt ou d'investissement

```
SELECT `id_client`, `nom`, `prenom`
FROM `clients`
WHERE `id_client` NOT IN (
    SELECT `id_client` FROM `prets`
    UNION
    SELECT `id_client` FROM `investissements`
    );
```

On a d'abord une sous-requête qui combine deux listes avec `UNION`, en supprimant les doublons :

- Une liste qui extrait les clients ayant au moins un prêt :
  `SELECT ‘id_client‘ FROM ‘prets‘`
- Une liste qui extrait les clients ayant au moins un investissement :
  `SELECT ‘id_client‘ FROM ‘investissements‘ ;`

Ensuite, on sélectionne les clients qui n'apparaissent dans aucune des deux listes avec `WHERE ‘id_client‘ NOT IN (...)`, puis on retourne une liste de ces clients, en affichant leurs id_client, leurs noms et leurs prénoms.

6.4 Déterminer le montant total des intérêts générés par les prêts

```
SELECT SUM(`montant` * `taux` / 100) AS `total_interets`

FROM `prets`;
```

On calcule d'abord chaque intérêt présent dans la table prêts, en utilisant la formule `‘montant‘ * ‘taux‘ / 100`, avec `/ 100` car le taux est exprimé en pourcentages. Ensuite, on additionne les intérêts calculés, et on place cette somme dans la variable `total_interets`. Cette requête retourne un seul résultat : le montant total des intérêts générés par tous les prêts.

6.5 Calculer la variation mensuelle du nombre de transactions

```
SELECT

    YEAR(date_transaction) AS annee_transaction,

    MONTH(date_transaction) AS mois_transaction,

    COUNT(id_transaction) AS total_transactions,

    COUNT(id_transaction) - (

        SELECT

COUNT(id_transaction)

        FROM

            transactions t2

        WHERE

            YEAR(t2.date_transaction) = YEAR(t1.date_transaction)

            AND MONTH(t2.date_transaction) = MONTH(t1.date_transaction) - 1

    ) AS variation_transactions

FROM

    transactions t1

GROUP BY

    YEAR(date_transaction),

    MONTH(date_transaction)

ORDER BY

    annee_transaction DESC,

    mois_transaction DESC;
```

Cette requête SQL permet de calculer la variation mensuelle du nombre de transactions en comparant le nombre de transactions pour chaque mois avec celui du mois précédent. Elle commence par sélectionner l'année et le mois de chaque transaction, puis elle compte le nombre total de transactions par mois. Ensuite, une sous-requête est utilisée pour récupérer le nombre de transactions du mois précédent et calculer la différence, donnant ainsi la variation mensuelle. Les résultats sont groupés par année et mois, et triés par ordre décroissant de l'année et du mois, ce qui permet de voir l'évolution des transactions mois par mois.