

# Examen SQL (Banque) – Transactions & Résolution de problèmes – Intermédiaire (Questions + Réponses)

## SQL – Transactions & Requêtes (Banque)

**Q1.** Écrire une transaction SQL qui exécute un virement interne (from\_account → to\_account) de :amount en garantissant l'atomicité, et en refusant si balance insuffisante.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
-- Schéma simplifié (banque)
-- accounts(account_id PK, customer_id, balance, currency, status, opened_at)
-- transfers(transfer_id PK, from_account, to_account, amount, currency, status, created_at, idempotency_key)
-- ledger_entries(entry_id PK, account_id, transfer_id, direction('DEBIT'/'CREDIT'), amount, created_at)
-- cards(card_id PK, account_id, status, daily_limit, daily_spent, updated_at)

BEGIN;

-- Verrouiller les 2 comptes (ordre stable pour éviter deadlocks)
SELECT account_id, balance
FROM accounts
WHERE account_id IN (:from_account, :to_account)
FOR UPDATE;

-- Vérifier solde
-- (selon SGBD, faire côté app ou via condition)
UPDATE accounts
SET balance = balance - :amount
WHERE account_id = :from_account AND balance >= :amount;

-- S'assurer qu'une ligne a été modifiée
-- si 0 ligne => ROLLBACK
UPDATE accounts
SET balance = balance + :amount
WHERE account_id = :to_account;

INSERT INTO transfers(transfer_id, from_account, to_account, amount, currency, status, created_at, idempotency_key)
VALUES(:transfer_id, :from_account, :to_account, :amount, :currency, 'POSTED', CURRENT_TIMESTAMP, :idem_key);

INSERT INTO ledger_entries(entry_id, account_id, transfer_id, direction, amount, created_at)
VALUES(:e1, :from_account, :transfer_id, 'DEBIT', :amount, CURRENT_TIMESTAMP),
(:e2, :to_account, :transfer_id, 'CREDIT', :amount, CURRENT_TIMESTAMP);

COMMIT;
```

*Senior/TechLead: discuter isolation, erreurs, idem key, outbox/event.*

**Q2.** Trouver les 10 clients dont la somme des débits sur les 30 derniers jours est la plus élevée (en utilisant ledger\_entries).

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
SELECT a.customer_id,
       SUM(CASE WHEN le.direction='DEBIT' THEN le.amount ELSE 0 END) AS total_debits
FROM accounts a
JOIN ledger_entries le ON le.account_id = a.account_id
WHERE le.created_at >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY a.customer_id
```

```
ORDER BY total_debits DESC
LIMIT 10;
```

Ajouter index: *ledger\_entries(account\_id, created\_at), accounts(customer\_id)*.

**Q3.** Détecter les transferts en doublon via idempotency\_key (et expliquer comment empêcher la double exécution).

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
-- Détection:
SELECT idempotency_key, COUNT(*) c
FROM transfers
GROUP BY idempotency_key
HAVING COUNT(*) > 1;

-- Prévention: contrainte UNIQUE + logique:
-- 1) INSERT transfers(..., idempotency_key) en premier
-- 2) si violation UNIQUE => retourner résultat existant (SELECT ... WHERE idempotency_key = :key)
-- 3) sinon poursuivre la transaction.
```

*TechLead: stratégie idem + retry réseau + exactly-once vs at-least-once.*

**Q4.** Écrire une requête qui retourne, pour chaque compte, le solde calculé à partir du ledger (CREDIT - DEBIT) et comparer au balance stocké.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
SELECT a.account_id,
       a.balance AS stored_balance,
       COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT' THEN -le.amount
                        ELSE 0 END),0) AS ledger_balance,
       (a.balance - COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT' THEN -le.amount
                        ELSE 0 END),0)) AS diff
FROM accounts a
LEFT JOIN ledger_entries le ON le.account_id = a.account_id
GROUP BY a.account_id, a.balance
ORDER BY ABS(a.balance - COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT' THEN -le.amount
                        ELSE 0 END),0)) DESC;
```

*Senior: discussion sur source of truth (ledger) et reconciliation.*

**Q5.** Gérer la limite journalière de carte : dans une transaction, refuser un paiement si daily\_spent + :amount > daily\_limit, sinon incrémenter daily\_spent.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
BEGIN;

SELECT card_id, daily_limit, daily_spent
FROM cards
WHERE card_id = :card_id
FOR UPDATE;

UPDATE cards
SET daily_spent = daily_spent + :amount,
    updated_at = CURRENT_TIMESTAMP
WHERE card_id = :card_id
  AND daily_spent + :amount <= daily_limit;

-- si 0 ligne modifiée => refuser (ROLLBACK)
```

```
COMMIT;
```

*TechLead: reset quotidien (job/cron), fuseaux horaires, audit, concurrence.*

**Q6.** Écrire une transaction SQL qui exécute un virement interne (from\_account → to\_account) de :amount en garantissant l'atomicité, et en refusant si balance insuffisante.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
-- Schéma simplifié (banque)
-- accounts(account_id PK, customer_id, balance, currency, status, opened_at)
-- transfers(transfer_id PK, from_account, to_account, amount, currency, status, created_at, idempotency_key)
-- ledger_entries(entry_id PK, account_id, transfer_id, direction('DEBIT'/'CREDIT'), amount, created_at)
-- cards(card_id PK, account_id, status, daily_limit, daily_spent, updated_at)

BEGIN;

-- Verrouiller les 2 comptes (ordre stable pour éviter deadlocks)
SELECT account_id, balance
FROM accounts
WHERE account_id IN (:from_account, :to_account)
FOR UPDATE;

-- Vérifier solde
-- (selon SGBD, faire côté app ou via condition)
UPDATE accounts
SET balance = balance - :amount
WHERE account_id = :from_account AND balance >= :amount;

-- S'assurer qu'une ligne a été modifiée
-- si 0 ligne => ROLLBACK
UPDATE accounts
SET balance = balance + :amount
WHERE account_id = :to_account;

INSERT INTO transfers(transfer_id, from_account, to_account, amount, currency, status, created_at, idempotency_key)
VALUES(:transfer_id, :from_account, :to_account, :amount, :currency, 'POSTED', CURRENT_TIMESTAMP, :idem_key);

INSERT INTO ledger_entries(entry_id, account_id, transfer_id, direction, amount, created_at)
VALUES(:e1, :from_account, :transfer_id, 'DEBIT', :amount, CURRENT_TIMESTAMP),
      (:e2, :to_account, :transfer_id, 'CREDIT', :amount, CURRENT_TIMESTAMP);

COMMIT;
```

*Senior/TechLead: discuter isolation, erreurs, idem key, outbox/event.*

**Q7.** Trouver les 10 clients dont la somme des débits sur les 30 derniers jours est la plus élevée (en utilisant ledger\_entries).

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
SELECT a.customer_id,
       SUM(CASE WHEN le.direction='DEBIT' THEN le.amount ELSE 0 END) AS total_debits
FROM accounts a
JOIN ledger_entries le ON le.account_id = a.account_id
WHERE le.created_at >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY a.customer_id
ORDER BY total_debits DESC
LIMIT 10;
```

*Ajouter index: ledger\_entries(account\_id, created\_at), accounts(customer\_id).*

**Q8.** Détecter les transferts en doublon via idempotency\_key (et expliquer comment empêcher la double exécution).

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
-- Détection:
SELECT idempotency_key, COUNT(*) c
FROM transfers
GROUP BY idempotency_key
HAVING COUNT(*) > 1;

-- Prévention: contrainte UNIQUE + logique:
-- 1) INSERT transfers(..., idempotency_key) en premier
-- 2) si violation UNIQUE => retourner résultat existant (SELECT ... WHERE idempotency_key = :key)
-- 3) sinon poursuivre la transaction.
```

*TechLead: stratégie idem + retry réseau + exactly-once vs at-least-once.*

**Q9.** Écrire une requête qui retourne, pour chaque compte, le solde calculé à partir du ledger (CREDIT - DEBIT) et comparer au balance stocké.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
SELECT a.account_id,
       a.balance AS stored_balance,
       COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT'   THEN -le.amount
                        ELSE 0 END),0) AS ledger_balance,
       (a.balance - COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT'   THEN -le.amount
                        ELSE 0 END),0)) AS diff
FROM accounts a
LEFT JOIN ledger_entries le ON le.account_id = a.account_id
GROUP BY a.account_id, a.balance
ORDER BY ABS(a.balance - COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT'   THEN -le.amount
                        ELSE 0 END),0)) DESC;
```

*Senior: discussion sur source of truth (ledger) et reconciliation.*

**Q10.** Gérer la limite journalière de carte : dans une transaction, refuser un paiement si daily\_spent + :amount > daily\_limit, sinon incrémenter daily\_spent.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
BEGIN;

SELECT card_id, daily_limit, daily_spent
FROM cards
WHERE card_id = :card_id
FOR UPDATE;

UPDATE cards
SET daily_spent = daily_spent + :amount,
    updated_at = CURRENT_TIMESTAMP
WHERE card_id = :card_id
   AND daily_spent + :amount <= daily_limit;

-- si 0 ligne modifiée => refuser (ROLLBACK)
COMMIT;
```

*TechLead: reset quotidien (job/cron), fuseaux horaires, audit, concurrence.*

**Q11.** Écrire une transaction SQL qui exécute un virement interne (from\_account → to\_account) de :amount en garantissant l'atomicité, et en refusant si balance insuffisante.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
-- Schéma simplifié (banque)
-- accounts(account_id PK, customer_id, balance, currency, status, opened_at)
-- transfers(transfer_id PK, from_account, to_account, amount, currency, status, created_at, idempotency_key)
-- ledger_entries(entry_id PK, account_id, transfer_id, direction('DEBIT'/'CREDIT'), amount, created_at)
-- cards(card_id PK, account_id, status, daily_limit, daily_spent, updated_at)

BEGIN;

-- Verrouiller les 2 comptes (ordre stable pour éviter deadlocks)
SELECT account_id, balance
FROM accounts
WHERE account_id IN (:from_account, :to_account)
FOR UPDATE;

-- Vérifier solde
-- (selon SGBD, faire côté app ou via condition)
UPDATE accounts
SET balance = balance - :amount
WHERE account_id = :from_account AND balance >= :amount;

-- S'assurer qu'une ligne a été modifiée
-- si 0 ligne => ROLLBACK
UPDATE accounts
SET balance = balance + :amount
WHERE account_id = :to_account;

INSERT INTO transfers(transfer_id, from_account, to_account, amount, currency, status, created_at, idempotency_key)
VALUES(:transfer_id, :from_account, :to_account, :amount, :currency, 'POSTED', CURRENT_TIMESTAMP, :idem_key);

INSERT INTO ledger_entries(entry_id, account_id, transfer_id, direction, amount, created_at)
VALUES(:e1, :from_account, :transfer_id, 'DEBIT', :amount, CURRENT_TIMESTAMP),
       (:e2, :to_account, :transfer_id, 'CREDIT', :amount, CURRENT_TIMESTAMP);

COMMIT;
```

*Senior/TechLead: discuter isolation, erreurs, idem key, outbox/event.*

**Q12.** Trouver les 10 clients dont la somme des débits sur les 30 derniers jours est la plus élevée (en utilisant ledger\_entries).

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
SELECT a.customer_id,
       SUM(CASE WHEN le.direction='DEBIT' THEN le.amount ELSE 0 END) AS total_debits
FROM accounts a
JOIN ledger_entries le ON le.account_id = a.account_id
WHERE le.created_at >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY a.customer_id
ORDER BY total_debits DESC
LIMIT 10;
```

*Ajouter index: ledger\_entries(account\_id, created\_at), accounts(customer\_id).*

**Q13.** Détecter les transferts en doublon via idempotency\_key (et expliquer comment empêcher la double exécution).

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
-- Détection:
SELECT idempotency_key, COUNT(*) c
FROM transfers
GROUP BY idempotency_key
```

```
HAVING COUNT(*) > 1;
```

```
-- Prévention: contrainte UNIQUE + logique:  
-- 1) INSERT transfers(..., idempotency_key) en premier  
-- 2) si violation UNIQUE => retourner résultat existant (SELECT ... WHERE idempotency_key = :key)  
-- 3) sinon poursuivre la transaction.
```

*TechLead: stratégie idem + retry réseau + exactly-once vs at-least-once.*

**Q14.** Écrire une requête qui retourne, pour chaque compte, le solde calculé à partir du ledger (CREDIT - DEBIT) et comparer au balance stocké.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
SELECT a.account_id,  
       a.balance AS stored_balance,  
       COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount  
                        WHEN le.direction='DEBIT' THEN -le.amount  
                        ELSE 0 END),0) AS ledger_balance,  
       (a.balance - COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount  
                        WHEN le.direction='DEBIT' THEN -le.amount  
                        ELSE 0 END),0)) AS diff  
FROM accounts a  
LEFT JOIN ledger_entries le ON le.account_id = a.account_id  
GROUP BY a.account_id, a.balance  
ORDER BY ABS(a.balance - COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount  
                        WHEN le.direction='DEBIT' THEN -le.amount  
                        ELSE 0 END),0)) DESC;
```

*Senior: discussion sur source of truth (ledger) et reconciliation.*

**Q15.** Gérer la limite journalière de carte : dans une transaction, refuser un paiement si daily\_spent + :amount > daily\_limit, sinon incrémenter daily\_spent.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
BEGIN;  
  
SELECT card_id, daily_limit, daily_spent  
FROM cards  
WHERE card_id = :card_id  
FOR UPDATE;  
  
UPDATE cards  
SET daily_spent = daily_spent + :amount,  
    updated_at = CURRENT_TIMESTAMP  
WHERE card_id = :card_id  
    AND daily_spent + :amount <= daily_limit;  
  
-- si 0 ligne modifiée => refuser (ROLLBACK)  
COMMIT;
```

*TechLead: reset quotidien (job/cron), fuseaux horaires, audit, concurrence.*

**Q16.** Écrire une transaction SQL qui exécute un virement interne (from\_account → to\_account) de :amount en garantissant l'atomicité, et en refusant si balance insuffisante.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
-- Schéma simplifié (banque)  
-- accounts(account_id PK, customer_id, balance, currency, status, opened_at)  
-- transfers(transfer_id PK, from_account, to_account, amount, currency, status, created_at, idempotency_key)
```

```

-- ledger_entries(entry_id PK, account_id, transfer_id, direction('DEBIT'/'CREDIT'), amount, created_at)
-- cards(card_id PK, account_id, status, daily_limit, daily_spent, updated_at)

BEGIN;

-- Verrouiller les 2 comptes (ordre stable pour éviter deadlocks)
SELECT account_id, balance
FROM accounts
WHERE account_id IN (:from_account, :to_account)
FOR UPDATE;

-- Vérifier solde
-- (selon SGBD, faire côté app ou via condition)
UPDATE accounts
SET balance = balance - :amount
WHERE account_id = :from_account AND balance >= :amount;

-- S'assurer qu'une ligne a été modifiée
-- si 0 ligne => ROLLBACK
UPDATE accounts
SET balance = balance + :amount
WHERE account_id = :to_account;

INSERT INTO transfers(transfer_id, from_account, to_account, amount, currency, status, created_at, idempotency_key)
VALUES(:transfer_id, :from_account, :to_account, :amount, :currency, 'POSTED', CURRENT_TIMESTAMP, :idem_key);

INSERT INTO ledger_entries(entry_id, account_id, transfer_id, direction, amount, created_at)
VALUES(:e1, :from_account, :transfer_id, 'DEBIT', :amount, CURRENT_TIMESTAMP),
       (:e2, :to_account, :transfer_id, 'CREDIT', :amount, CURRENT_TIMESTAMP);

COMMIT;

```

*Senior/TechLead: discuter isolation, erreurs, idem key, outbox/event.*

**Q17.** Trouver les 10 clients dont la somme des débits sur les 30 derniers jours est la plus élevée (en utilisant ledger\_entries).

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```

SELECT a.customer_id,
       SUM(CASE WHEN le.direction='DEBIT' THEN le.amount ELSE 0 END) AS total_debits
FROM accounts a
JOIN ledger_entries le ON le.account_id = a.account_id
WHERE le.created_at >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY a.customer_id
ORDER BY total_debits DESC
LIMIT 10;

```

*Ajouter index: ledger\_entries(account\_id, created\_at), accounts(customer\_id).*

**Q18.** Détecter les transferts en doublon via idempotency\_key (et expliquer comment empêcher la double exécution).

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```

-- Détection:
SELECT idempotency_key, COUNT(*) c
FROM transfers
GROUP BY idempotency_key
HAVING COUNT(*) > 1;

-- Prévention: contrainte UNIQUE + logique:
-- 1) INSERT transfers(..., idempotency_key) en premier
-- 2) si violation UNIQUE => retourner résultat existant (SELECT ... WHERE idempotency_key = :key)
-- 3) sinon poursuivre la transaction.

```

*TechLead: stratégie idem + retry réseau + exactly-once vs at-least-once.*

**Q19.** Écrire une requête qui retourne, pour chaque compte, le solde calculé à partir du ledger (CREDIT - DEBIT) et comparer au balance stocké.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
SELECT a.account_id,
       a.balance AS stored_balance,
       COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT' THEN -le.amount
                        ELSE 0 END),0) AS ledger_balance,
       (a.balance - COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT' THEN -le.amount
                        ELSE 0 END),0)) AS diff
FROM accounts a
LEFT JOIN ledger_entries le ON le.account_id = a.account_id
GROUP BY a.account_id, a.balance
ORDER BY ABS(a.balance - COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                                        WHEN le.direction='DEBIT' THEN -le.amount
                                        ELSE 0 END),0)) DESC;
```

*Senior: discussion sur source of truth (ledger) et reconciliation.*

**Q20.** Gérer la limite journalière de carte : dans une transaction, refuser un paiement si daily\_spent + :amount > daily\_limit, sinon incrémenter daily\_spent.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
BEGIN;

SELECT card_id, daily_limit, daily_spent
FROM cards
WHERE card_id = :card_id
FOR UPDATE;

UPDATE cards
SET daily_spent = daily_spent + :amount,
    updated_at = CURRENT_TIMESTAMP
WHERE card_id = :card_id
  AND daily_spent + :amount <= daily_limit;

-- si 0 ligne modifiée => refuser (ROLLBACK)
COMMIT;
```

*TechLead: reset quotidien (job/cron), fuseaux horaires, audit, concurrence.*

**Q21.** Écrire une transaction SQL qui exécute un virement interne (from\_account → to\_account) de :amount en garantissant l'atomicité, et en refusant si balance insuffisante.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```
-- Schéma simplifié (banque)
-- accounts(account_id PK, customer_id, balance, currency, status, opened_at)
-- transfers(transfer_id PK, from_account, to_account, amount, currency, status, created_at, idempotency_key)
-- ledger_entries(entry_id PK, account_id, transfer_id, direction('DEBIT'/'CREDIT'), amount, created_at)
-- cards(card_id PK, account_id, status, daily_limit, daily_spent, updated_at)

BEGIN;

-- Verrouiller les 2 comptes (ordre stable pour éviter deadlocks)
SELECT account_id, balance
```



```

FROM accounts
WHERE account_id IN (:from_account, :to_account)
FOR UPDATE;

-- Vérifier solde
-- (selon SGBD, faire côté app ou via condition)
UPDATE accounts
SET balance = balance - :amount
WHERE account_id = :from_account AND balance >= :amount;

-- S'assurer qu'une ligne a été modifiée
-- si 0 ligne => ROLLBACK
UPDATE accounts
SET balance = balance + :amount
WHERE account_id = :to_account;

INSERT INTO transfers(transfer_id, from_account, to_account, amount, currency, status, created_at, idempotency_key)
VALUES(:transfer_id, :from_account, :to_account, :amount, :currency, 'POSTED', CURRENT_TIMESTAMP, :idem_key);

INSERT INTO ledger_entries(entry_id, account_id, transfer_id, direction, amount, created_at)
VALUES(:e1, :from_account, :transfer_id, 'DEBIT', :amount, CURRENT_TIMESTAMP),
       (:e2, :to_account, :transfer_id, 'CREDIT', :amount, CURRENT_TIMESTAMP);

COMMIT;

```

*Senior/TechLead: discuter isolation, erreurs, idem key, outbox/event.*

**Q22.** Trouver les 10 clients dont la somme des débits sur les 30 derniers jours est la plus élevée (en utilisant ledger\_entries).

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```

SELECT a.customer_id,
       SUM(CASE WHEN le.direction='DEBIT' THEN le.amount ELSE 0 END) AS total_debits
FROM accounts a
JOIN ledger_entries le ON le.account_id = a.account_id
WHERE le.created_at >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY a.customer_id
ORDER BY total_debits DESC
LIMIT 10;

```

*Ajouter index: ledger\_entries(account\_id, created\_at), accounts(customer\_id).*

**Q23.** Détecter les transferts en doublon via idempotency\_key (et expliquer comment empêcher la double exécution).

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```

-- Détection:
SELECT idempotency_key, COUNT(*) c
FROM transfers
GROUP BY idempotency_key
HAVING COUNT(*) > 1;

-- Prévention: contrainte UNIQUE + logique:
-- 1) INSERT transfers(..., idempotency_key) en premier
-- 2) si violation UNIQUE => retourner résultat existant (SELECT ... WHERE idempotency_key = :key)
-- 3) sinon poursuivre la transaction.

```

*TechLead: stratégie idem + retry réseau + exactly-once vs at-least-once.*

**Q24.** Écrire une requête qui retourne, pour chaque compte, le solde calculé à partir du ledger (CREDIT - DEBIT) et comparer au balance stocké.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```

SELECT a.account_id,
       a.balance AS stored_balance,
       COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT'   THEN -le.amount
                        ELSE 0 END),0) AS ledger_balance,
       (a.balance - COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT'   THEN -le.amount
                        ELSE 0 END),0)) AS diff
FROM accounts a
LEFT JOIN ledger_entries le ON le.account_id = a.account_id
GROUP BY a.account_id, a.balance
ORDER BY ABS(a.balance - COALESCE(SUM(CASE WHEN le.direction='CREDIT' THEN le.amount
                        WHEN le.direction='DEBIT'   THEN -le.amount
                        ELSE 0 END),0)) DESC;

```

*Senior: discussion sur source of truth (ledger) et reconciliation.*

**Q25.** Gérer la limite journalière de carte : dans une transaction, refuser un paiement si `daily_spent + :amount > daily_limit`, sinon incrémenter `daily_spent`.

**Réponse :** Solution SQL fournie ci-dessous.

**SQL :**

```

BEGIN;

SELECT card_id, daily_limit, daily_spent
FROM cards
WHERE card_id = :card_id
FOR UPDATE;

UPDATE cards
SET daily_spent = daily_spent + :amount,
    updated_at = CURRENT_TIMESTAMP
WHERE card_id = :card_id
    AND daily_spent + :amount <= daily_limit;

-- si 0 ligne modifiée => refuser (ROLLBACK)
COMMIT;

```

*TechLead: reset quotidien (job/cron), fuseaux horaires, audit, concurrence.*