Examen SQL (Banque) – Transactions & Résolution de problèmes – Senior (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. (niveau senior: optimiser, indexer, gérer concurrence)

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_ke
-- 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, idempote
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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q2. Trouver les 10 clients dont la somme des débits sur les 30 derniers jours est la plus élevée (en utilisant ledger_entries). (niveau senior: optimiser, indexer, gérer concurrence)

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). Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q3. Détecter les transferts en doublon via idempotency_key (et expliquer comment empêcher la double exécution). (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q4. Écrire une requête qui retourne, pour chaque compte, le solde calculé à partir du ledger (CREDIT - DEBIT) et comparer au balance stocké. (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q5. Gérer la limite journalière de carte : dans une transaction, refuser un paiement si daily_spent + :amount > daily_limit, sinon incrémenter daily_spent. (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

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. (niveau senior: optimiser, indexer, gérer concurrence)

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_ke
-- 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, idempote
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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q7. Trouver les 10 clients dont la somme des débits sur les 30 derniers jours est la plus élevée (en utilisant ledger_entries). (niveau senior: optimiser, indexer, gérer concurrence)

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). Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q8. Détecter les transferts en doublon via idempotency_key (et expliquer comment empêcher la double exécution). (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q9. Écrire une requête qui retourne, pour chaque compte, le solde calculé à partir du ledger (CREDIT - DEBIT) et comparer au balance stocké. (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q10. Gérer la limite journalière de carte : dans une transaction, refuser un paiement si daily_spent + :amount > daily_limit, sinon incrémenter daily_spent. (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

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. (niveau senior: optimiser, indexer, gérer concurrence)

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_ke
-- 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, idempote
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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q12. Trouver les 10 clients dont la somme des débits sur les 30 derniers jours est la plus élevée (en utilisant ledger_entries). (niveau senior: optimiser, indexer, gérer concurrence)

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). Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q13. Détecter les transferts en doublon via idempotency_key (et expliquer comment empêcher la double exécution). (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q14. Écrire une requête qui retourne, pour chaque compte, le solde calculé à partir du ledger (CREDIT - DEBIT) et comparer au balance stocké. (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q15. Gérer la limite journalière de carte : dans une transaction, refuser un paiement si daily_spent + :amount > daily_limit, sinon incrémenter daily_spent. (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

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. (niveau senior: optimiser, indexer, gérer concurrence)

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_ke
-- 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, idempote
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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q17. Trouver les 10 clients dont la somme des débits sur les 30 derniers jours est la plus élevée (en utilisant ledger_entries). (niveau senior: optimiser, indexer, gérer concurrence)

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). Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q18. Détecter les transferts en doublon via idempotency_key (et expliquer comment empêcher la double exécution). (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q19. Écrire une requête qui retourne, pour chaque compte, le solde calculé à partir du ledger (CREDIT - DEBIT) et comparer au balance stocké. (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q20. Gérer la limite journalière de carte : dans une transaction, refuser un paiement si daily_spent + :amount > daily_limit, sinon incrémenter daily_spent. (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

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. (niveau senior: optimiser, indexer, gérer concurrence)

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_ke
-- 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, idempote
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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q22. Trouver les 10 clients dont la somme des débits sur les 30 derniers jours est la plus élevée (en utilisant ledger_entries). (niveau senior: optimiser, indexer, gérer concurrence)

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). Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q23. Détecter les transferts en doublon via idempotency_key (et expliquer comment empêcher la double exécution). (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q24. Écrire une requête qui retourne, pour chaque compte, le solde calculé à partir du ledger (CREDIT - DEBIT) et comparer au balance stocké. (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.

Q25. Gérer la limite journalière de carte : dans une transaction, refuser un paiement si daily_spent + :amount > daily_limit, sinon incrémenter daily_spent. (niveau senior: optimiser, indexer, gérer concurrence)

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. Attendu: isolation READ COMMITTED/REPEATABLE, locks, plan d'exécution.