10 ~ Fiabilité des données (ACID)

? En un mot

ACID est un ensemble de garanties qui rendent tes transactions dans la base de donnée fiables, prévisibles et sûres en production.

? Ce que ACID est ?

Il garantit en pratique :

- □□ Pas d’opérations “à moitié faites”
- □□ Des données toujours valides après une transaction
- □□♂ Un comportement correct même avec plusieurs utilisateurs en même temps
- □□ Des modifications conservées même après un crash serveur

? Ce que ACID n'est pas ?

- Une garantie de performance maximale
- Une protection contre une mauvaise logique métier
- Une solution magique contre tous les problèmes de concurrence
- Un modèle obligatoire pour tous les cas (analytics, cache, etc.)

? Les 4 piliers d’ACID

? A - Atomicité

? Principe

Une transaction est indivisible : si une étape échoue, **tout est annulé**.

? Cas concret

Transfert d’argent entre deux comptes.

On veut :

- Débiter A
- Créditer B
- Ajouter une ligne dans l’historique

Si une seule étape échoue → rien ne doit être appliqué.

? Exemple MySQL

```
START TRANSACTION;

UPDATE accounts
SET balance = balance - 100
WHERE id = 1;

UPDATE accounts
SET balance = balance + 100
WHERE id = 2;

INSERT INTO ledger (from_account_id, to_account_id, amount)
VALUES (1, 2, 100);

COMMIT;
```

? Si problème :

```
ROLLBACK;
```

? Explication

Tant que `COMMIT` n’est pas exécuté :

- Les modifications ne sont pas définitives
- Si une requête échoue (ex: contrainte, crash, validation métier)
- `ROLLBACK` annule TOUT

Résultat : Pas de débit sans crédit. Pas d’état incohérent.

? C - Cohérence

? Principe

Après `COMMIT`, la base respecte toutes ses contraintes.

? Cas concret

Empêcher :

- Un solde négatif
- Une référence vers un compte inexistant

? Définition des contraintes

```
CREATE TABLE accounts (
  id INT PRIMARY KEY,
  balance DECIMAL(10,2) NOT NULL,
  CHECK (balance >= 0)
) ENGINE=InnoDB;

CREATE TABLE ledger (
  id INT AUTO_INCREMENT PRIMARY KEY,
```

```
  from_account_id INT,
  to_account_id INT,
  amount DECIMAL(10,2) CHECK (amount > 0),
  FOREIGN KEY (from_account_id) REFERENCES accounts(id),
  FOREIGN KEY (to_account_id) REFERENCES accounts(id)
) ENGINE=InnoDB;
```

? Tentative invalide

```
START TRANSACTION;

INSERT INTO ledger (from_account_id, to_account_id, amount)
VALUES (1, 9999, 50);

COMMIT;
```

? Explication

- `9999` n’existe pas
- La contrainte FK bloque l’opération
- La transaction échoue

La base reste valide.

ACID empêche un état incohérent d’être validé.

? I - Isolation

? Principe

Les transactions concurrentes ne doivent pas créer d’anomalies.

? Cas concret

Deux utilisateurs tentent de dépenser le même solde en même temps.

? Exemple avec verrouillage

Transaction A :

```
START TRANSACTION;

SELECT balance
FROM accounts
WHERE id = 1
FOR UPDATE;

UPDATE accounts
SET balance = balance - 50
WHERE id = 1;

COMMIT;
```

Transaction B (en parallèle) :

```
START TRANSACTION;

SELECT balance
FROM accounts
WHERE id = 1
FOR UPDATE;
```

? Explication

- `FOR UPDATE` verrouille la ligne
- Transaction B attend la fin de A
- Pas de double débit simultané

Isolation = contrôle des conflits concurrents.

? D - Durabilité

? Principe

Une fois `COMMIT`, les données survivent à un crash.

? Cas concret

Paiement validé → serveur plante immédiatement après.

? Exemple

```
START TRANSACTION;


UPDATE accounts
SET balance = balance + 200
WHERE id = 2;


COMMIT;
```

Si le serveur tombe juste après :

```
SELECT balance FROM accounts WHERE id = 2;
```

La modification est toujours présente.

? Explication

Ici, sous InnoDB des logs (redo log) sont utilisés automatiquement pour garantir que :

- Une transaction validée sera restaurée après redémarrage
- L’état reste cohérent même après crash

? En pratique

ACID est crucial pour :

- □□ Paiements
- □□ Gestion de stock
- □□ Droits utilisateurs
- □□ Facturation
- □□ Systèmes financiers

Il garantit que ta base ne devient pas un chaos concurrent.

Révision #4
Créé 2026-02-13 12:20:28 CET par Larananas

Mis à jour 2026-02-13 12:46:25 CET par Larananas