Bases de données

Transactions

Nadime Francis

Université Gustave Eiffel
LIGM - 4B130 Copernic
nadime.francis@univ-eiffel.fr

Scénario 1 : fonctionnement normal

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

Scénario 1 : fonctionnement normal

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

Scénario 1 : fonctionnement normal

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

Scénario 1 : fonctionnement normal

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

Scénario 1 : fonctionnement normal

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

```
-- retire 81 points de la carte de Natacha au magasin Manut
```

Scénario 1 : fonctionnement normal

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

```
-- retire 81 points de la carte de Natacha au magasin Manut
```

```
UPDATE stocke
SET quantite = quantite -1
WHERE idmag = 43 AND idpro = 96;
```

Scénario 1 : fonctionnement normal

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

```
-- retire 81 points de la carte de Natacha au magasin Manut
```

```
UPDATE stocke
SET quantite = quantite -1
WHERE idmag = 43 AND idpro = 96;
```

```
-- retire un caisson des stocks du magasin
```

Scénario 1 : fonctionnement normal

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

```
-- retire 81 points de la carte de Natacha au magasin Manut
```

```
UPDATE stocke
SET quantite = quantite -1
WHERE idmag = 43 AND idpro = 96;
```

```
-- retire un caisson des stocks du magasin
```

. . .

Scénario 2 : panne

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

Scénario 2 : panne

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

Scénario 2 : panne

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

Scénario 2 : panne

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

Scénario 2 : panne

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

```
-- retire 81 points de la carte de Natacha au magasin Manut
```

Scénario 2 : panne

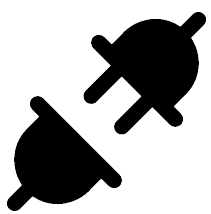
Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

```
-- retire 81 points de la carte de Natacha au magasin Manut
```



Natacha perd son argent et il n'y a pas de traces de son achat...

Scénario 3 : achat concurrent

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

Scénario 3 : achat concurrent

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

Scénario 3 : achat concurrent

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

Scénario 3 : achat concurrent

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

Scénario 3 : achat concurrent

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

```
-- retire 81 points de la carte de Natacha au magasin Manut
```

Scénario 3 : achat concurrent

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

```
-- retire 81 points de la carte de Natacha au magasin Manut
```

Robert Templier achète juste avant Natacha le dernier caisson.

Scénario 3 : achat concurrent

Natacha Joly veut acheter un caisson au magasin *Manut* de Dijon.

```
SELECT idmag, idpro, prixUnit, quantite
FROM magasin NATURAL JOIN stocke NATURAL JOIN produit
WHERE nom = 'Manut' AND libelle = 'caisson' AND ville = 'dijon';
```

```
-- récupère la quantité (1) et le prix (80.34), l'id du magasin (43) et du produit (96)
```

```
UPDATE fidelite
SET points = points - 81
WHERE idmag = 43 AND numcli = 650;
```

```
-- retire 81 points de la carte de Natacha au magasin Manut
```

Robert Templier achète juste avant Natacha le dernier caisson.

```
UPDATE stocke
SET quantite = quantite -1
WHERE idmag = 43 AND idpro = 96;
```

```
ERROR: new row for relation "stocke" violates check constraint "quant0"
```

Natacha perd son argent et il n'y a pas de traces de son achat...

Transactions

Transaction : exécution d'une séquence d'opérations (lectures, écritures) sur la base de données

Transactions

Transaction : exécution d'une séquence d'opérations (lectures, écritures) sur la base de données

Hypothèse : les transactions sont correctes – on ne s'inquiète pas d'éventuelles erreurs de programmation

Transactions

Transaction : exécution d'une séquence d'opérations (lectures, écritures) sur la base de données

Hypothèse : les transactions sont correctes – on ne s'inquiète pas d'éventuelles erreurs de programmation

> Une transaction exécutée en entier et en isolation conduit la base de données d'un état cohérent à un état cohérent.

Transactions

Transaction : exécution d'une séquence d'opérations (lectures, écritures) sur la base de données

Hypothèse : les transactions sont correctes – on ne s'inquiète pas d'éventuelles erreurs de programmation

> Une transaction exécutée en entier et en isolation conduit la base de données d'un état cohérent à un état cohérent.

Pannes et concurrence nuisent à l'exécution correcte des transations

- → état incohérent de la base de données
  Ex : il y a un nombre négatif de produits en stock
- → comportement incorrect de la transaction
  Ex : le compte du client est débité mais la facture n'est pas enregistrée

Propriétés attendues du système : ACID

- Atomicité : chaque transaction est effectuée en entier ou pas du tout ; pas d'exécutions partielles
- Cohérence : tout ensemble de transactions préserve la cohérence des données, peu importe l'ordre d'évaluation des opérations
- Isolation : chaque transaction se comporte comme si elle était seule
- Durabilité : les modifications apportées par une transaction ne peuvent pas être perdue suite à une panne

Propriétés attendues du système : ACID

- Atomicité : chaque transaction est effectuée en entier ou pas du tout ; pas d'exécutions partielles
- Cohérence : tout ensemble de transactions préserve la cohérence des données, peu importe l'ordre d'évaluation des opérations
- Isolation : chaque transaction se comporte comme si elle était seule
- Durabilité : les modifications apportées par une transaction ne peuvent pas être perdue suite à une panne

Question : quelles sont les propriétés enfreintes par les scénarios précédents ? Décrivez des situations enfreignant les autres propriétés.

Gestion des transactions

C'est la responsabilité :

- du SGBD de permettre l'exécution correcte des transactions avec divers niveaux de garantie
- du développeur et de l'administrateur de paramétrer correctement le système selon les cas d'erreurs que l'on veut éviter

Sous PostgreSQL :

- **BEGIN** : initie une transaction
- **COMMIT** : termine une transaction en enregistrant les modifications
- **ROLLBACK** : termine une transaction en annulant les modifications

Une transaction qui lève une exception est toujours annulée

Aperçu des erreurs liées à la concurrence

- Lecture sale (dirty read)
  Lecture de données qui n'ont pas encore été validées (commit)
  Données en cours de modification, amenées à être annulées, etc.

- Lecture non-reproductible (nonrepeatable read)
  Deux lectures du même tuple renvoient des valeurs différentes

- Lecture fantôme (phantom read)
  Deux exécutions de la même requête renvoient des résultats différents

- Anomalie de sérialisation (serialization anomaly)
  L'exécution concurrente de deux transactions T et T' n'est équivalente à aucune exécution en série de T et T' ($T; T'$ ou $T'; T$)

Niveaux d'isolation

Niveaux d'isolation

Les SGBD garantissent atomicité et durabilité pour toutes les transactions

Le standard SQL définit plusieurs niveaux d'isolation en fonction des problèmes de cohérence et d'isolation que l'on veut éviter

Chaque niveau d'isolation correspond à un compromis différent entre performance et fiabilité

Du plus performant au plus sûr :

	dirty read	nonrepeatable read	phantom read	serialization anomaly
read uncommitted	possible	possible	possible	possible
read committed	impossible	possible	possible	possible
repeatable read	impossible	impossible	possible	possible
serializable	impossible	impossible	impossible	impossible

Read uncommitted

Read uncommitted :

- Toutes les données peuvent être lues, même les données en cours de modification par une transaction concurrente
- Aucune garantie sur l'ordre des opérations

Erreurs possibles :

- Dirty read
- Nonrepeatable read
- Phantom read
- Serialization anomaly

Exemple : transaction avec read uncommitted

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

Bob

```
BEGIN;
```

Exemple : transaction avec read uncommitted

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

Bob

```
BEGIN;
```

Exemple : transaction avec read uncommitted

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 1000 (dirty read)
```

Exemple : transaction avec read uncommitted

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

```
ROLLBACK;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 1000 (dirty read)
```

Exemple : transaction avec read uncommitted

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000
1	3	1000

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

```
ROLLBACK;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 1000 (dirty read)
```

```
INSERT INTO soutient
VALUES (1, 3, 1000);
COMMIT;
```

Read committed

Read committed :

- Seules les données validées (par un commit) peuvent être lues
  $\Rightarrow$ pas de dirty read
- Aucune garantie sur l'ordre des opérations

Erreurs possibles :

- Nonrepeatable read
- Phantom read
- Serialization anomaly

Exemple : transaction avec read committed

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

Bob

```
BEGIN;
```

Exemple : transaction avec read committed

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

Bob

```
BEGIN;
```

Exemple : transaction avec read committed

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 500 (pas de dirty read)
```

Exemple : transaction avec read committed

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

```
COMMIT;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 500 (pas de dirty read)
```

Exemple : transaction avec read committed

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000
1	3	500

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

```
COMMIT;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 500 (pas de dirty read)
```

```
INSERT INTO soutient
VALUES (1, 3, 500);
```

Exemple : transaction avec read committed

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000
1	3	500

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

```
COMMIT;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 500 (pas de dirty read)
```

```
INSERT INTO soutient
VALUES (1, 3, 500);
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 1000 (nonrepeatable)
COMMIT;
```

Repeatable read

Repeatable read :

- Seules les données validées (par un commit) peuvent être lues
  $\Rightarrow$ pas de dirty read
- Deux lectures de la même données renvoient la même valeur
  $\Rightarrow$ pas de nonrepeatable read
- La sérialisabilité n'est pas encore garantie

Erreurs possibles :

- Phantom read
- Serialization anomaly

Exemple : transaction avec repeatable read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

Bob

```
BEGIN;
```

Exemple : transaction avec repeatable read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

Bob

```
BEGIN;
```

Exemple : transaction avec repeatable read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 500 (pas de dirty read)
```

Exemple : transaction avec repeatable read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

```
COMMIT;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 500 (pas de dirty read)
```

Exemple : transaction avec repeatable read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000
1	3	500

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

```
COMMIT;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 500 (pas de dirty read)
```

```
INSERT INTO soutient
VALUES (1, 3, 500);
```

Exemple : transaction avec repeatable read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	1000

soutient

uid	pid	montant
2	2	6000
1	3	500

Alice

```
BEGIN;
```

```
UPDATE projet SET requis = 1000
WHERE pid = 3;
```

```
COMMIT;
```

Bob

```
BEGIN;
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 500 (pas de dirty read)
```

```
INSERT INTO soutient
VALUES (1, 3, 500);
```

```
SELECT requis FROM projet
WHERE pid = 3;
-- renvoie requis = 500 (repeatable)
COMMIT;
```

Exemple : erreur de type phantom read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000

Administrateur

```
BEGIN;
```

Bob

```
BEGIN;
```

Exemple : erreur de type phantom read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000

Administrateur

```
BEGIN;
```

```
SELECT sum(montant) FROM soutient
WHERE pid = 2;
-- renvoie 6000
```

Bob

```
BEGIN;
```

Exemple : erreur de type phantom read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	annulé	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000

Administrateur

```
BEGIN;
```

```
SELECT sum(montant) FROM soutient
WHERE pid = 2;
-- renvoie 6000
```

```
UPDATE projet SET statut = 'annulé'
WHERE pid = 2;
```

Bob

```
BEGIN;
```

Exemple : erreur de type phantom read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	annulé	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000
1	2	4000

Administrateur

```
BEGIN;
```

```
SELECT sum(montant) FROM soutient
WHERE pid = 2;
-- renvoie 6000
```

```
UPDATE projet SET statut = 'annulé'
WHERE pid = 2;
```

Bob

```
BEGIN;
```

```
INSERT INTO soutient
VALUES (1, 2, 4000);
COMMIT;
```

Exemple : erreur de type phantom read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	annulé	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000
1	2	4000

Administrateur

```
BEGIN;
```

```
SELECT sum(montant) FROM soutient
WHERE pid = 2;
-- renvoie 6000
```

```
UPDATE projet SET statut = 'annulé'
WHERE pid = 2;
```

```
SELECT sum(montant) FROM soutient
WHERE pid = 2;
-- renvoie 10000 (phantom read)
```

Bob

```
BEGIN;
```

```
INSERT INTO soutient
VALUES (1, 2, 4000);
COMMIT;
```

Exemple : erreur de type phantom read

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	annulé	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000
1	2	4000

Administrateur

```
BEGIN;
```

```
SELECT sum(montant) FROM soutient
WHERE pid = 2;
-- renvoie 6000
```

```
UPDATE projet SET statut = 'annulé'
WHERE pid = 2;
```

```
SELECT sum(montant) FROM soutient
WHERE pid = 2;
-- renvoie 10000 (phantom read)
```

```
COMMIT;
```

Bob

```
BEGIN;
```

```
INSERT INTO soutient
VALUES (1, 2, 4000);
COMMIT;
```

Exemple : erreur de type serialization anomaly

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000
4	3	500

Administrateur

```
BEGIN;
```

Bob

```
BEGIN;
```

Exemple : erreur de type serialization anomaly

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	financé	500

soutient

uid	pid	montant
2	2	6000
4	3	500

Administrateur

```
BEGIN;
```

```
UPDATE projet SET statut = 'financé'
WHERE pid IN (
  SELECT pid
  FROM projet NATURAL JOIN soutient
  GROUP BY pid
  HAVING sum(montant) >= requis
);
```

Bob

```
BEGIN;
```

Exemple : erreur de type serialization anomaly

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	financé	500

soutient

uid	pid	montant
2	2	6000
4	3	500
1	2	5000

Administrateur

```
BEGIN;
```

```
UPDATE projet SET statut = 'financé'
WHERE pid IN (
  SELECT pid
  FROM projet NATURAL JOIN soutient
  GROUP BY pid
  HAVING sum(montant) >= requis
);
```

Bob

```
BEGIN;
```

```
INSERT INTO soutient
SELECT 1, pid, 5000 FROM projet
WHERE statut = 'attente'
AND titre LIKE '%VR%';
```

Exemple : erreur de type serialization anomaly

projet

pid	titre	statut	requis
1	Hoverboard	annulé	50000
2	Full body VR	annulé	10000
3	Perpetual motion	financé	500

soutient

uid	pid	montant
2	2	6000
4	3	500
1	2	5000

Administrateur

```
BEGIN;
```

```
UPDATE projet SET statut = 'financé'
WHERE pid IN (
  SELECT pid
  FROM projet NATURAL JOIN soutient
  GROUP BY pid
  HAVING sum(montant) >= requis
);
```

```
UPDATE projet SET statut = 'annulé'
WHERE statut = 'attente';
```

Bob

```
BEGIN;
```

```
INSERT INTO soutient
SELECT 1, pid, 5000 FROM projet
WHERE statut = 'attente'
AND titre LIKE '%VR%';
```

Exemple : erreur de type serialization anomaly

projet

pid	titre	statut	requis
1	Hoverboard	annulé	50000
2	Full body VR	annulé	10000
3	Perpetual motion	financé	500

soutient

uid	pid	montant
2	2	6000
4	3	500
1	2	5000

Administrateur

```
BEGIN;
```

```
UPDATE projet SET statut = 'financé'
WHERE pid IN (
  SELECT pid
  FROM projet NATURAL JOIN soutient
  GROUP BY pid
  HAVING sum(montant) >= requis
);
```

```
UPDATE projet SET statut = 'annulé'
WHERE statut = 'attente';
```

```
COMMIT;
```

Bob

```
BEGIN;
```

```
INSERT INTO soutient
SELECT 1, pid, 5000 FROM projet
WHERE statut = 'attente'
AND titre LIKE '%VR%';
```

```
COMMIT;
```

Exemple : erreur de type serialization anomaly

projet

pid	titre	statut	requis
1	Hoverboard	annulé	50000
2	Full body VR	annulé	10000
3	Perpetual motion	financé	500

soutient

uid	pid	montant
2	2	6000
4	3	500
1	2	5000

Administrateur

```
BEGIN;
```

```
UPDATE projet SET statut = 'financé'
WHERE pid IN (
  SELECT pid
  FROM projet NATURAL JOIN soutient
  GROUP BY pid
  HAVING sum(montant) >= requis
);
```

```
UPDATE projet SET statut = 'annulé'
WHERE statut = 'attente';
```

```
COMMIT;
```

Bob

```
BEGIN;
```

```
INSERT INTO soutient
SELECT 1, pid, 5000 FROM projet
WHERE statut = 'attente'
AND titre LIKE '%VR%';
```

```
COMMIT;
```

Aucune exécution en série ne peut aboutir à cet état final !

Exemple : erreur de type serialization anomaly

Première exécution en série : Administrateur → Bob

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000
4	3	500

Administrateur

Bob

Exemple : erreur de type serialization anomaly

Première exécution en série : Administrateur → Bob

projet

pid	titre	statut	requis
1	Hoverboard	annulé	50000
2	Full body VR	annulé	10000
3	Perpetual motion	financé	500

soutient

uid	pid	montant
2	2	6000
4	3	500

Administrateur

```
BEGIN;
UPDATE projet SET statut = 'financé'
WHERE pid IN (
  SELECT pid
  FROM projet NATURAL JOIN soutient
  GROUP BY pid
  HAVING sum(montant) >= requis
);
UPDATE projet SET statut = 'annulé'
WHERE statut = 'attente';
COMMIT;
```

Bob

Exemple : erreur de type serialization anomaly

Première exécution en série : Administrateur → Bob

projet

pid	titre	statut	requis
1	Hoverboard	annulé	50000
2	Full body VR	annulé	10000
3	Perpetual motion	financé	500

soutient

uid	pid	montant
2	2	6000
4	3	500

Administrateur

```
BEGIN;
UPDATE projet SET statut = 'financé'
WHERE pid IN (
  SELECT pid
  FROM projet NATURAL JOIN soutient
  GROUP BY pid
  HAVING sum(montant) >= requis
);
UPDATE projet SET statut = 'annulé'
WHERE statut = 'attente';
COMMIT;
```

Bob

```
BEGIN;
INSERT INTO soutient
SELECT 1, pid, 5000 FROM projet
WHERE statut = 'attente'
AND titre LIKE '%VR%';
COMMIT;
```

Exemple : erreur de type serialization anomaly

Première exécution en série : Administrateur → Bob

projet

pid	titre	statut	requis
1	Hoverboard	annulé	50000
2	Full body VR	annulé	10000
3	Perpetual motion	financé	500

soutient

uid	pid	montant
2	2	6000
4	3	500

Administrateur

```
BEGIN;
UPDATE projet SET statut = 'financé'
WHERE pid IN (
  SELECT pid
  FROM projet NATURAL JOIN soutient
  GROUP BY pid
  HAVING sum(montant) >= requis
);
UPDATE projet SET statut = 'annulé'
WHERE statut = 'attente';
COMMIT;
```

Bob

```
BEGIN;
INSERT INTO soutient
SELECT 1, pid, 5000 FROM projet
WHERE statut = 'attente'
AND titre LIKE '%VR%';
COMMIT;
```

Si l'administrateur termine en premier, Bob ne finance pas le projet

Exemple : erreur de type serialization anomaly

Deuxième exécution en série : Bob → Administrateur

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000
4	3	500

Administrateur

Bob

Exemple : erreur de type serialization anomaly

Deuxième exécution en série : Bob → Administrateur

projet

pid	titre	statut	requis
1	Hoverboard	attente	50000
2	Full body VR	attente	10000
3	Perpetual motion	attente	500

soutient

uid	pid	montant
2	2	6000
4	3	500
1	2	5000

Administrateur

Bob

```
BEGIN;
INSERT INTO soutient
SELECT 1, pid, 5000 FROM projet
WHERE statut = 'attente'
AND titre LIKE '%VR%';
COMMIT;
```

Exemple : erreur de type serialization anomaly

Deuxième exécution en série : Bob → Administrateur

projet

pid	titre	statut	requis
1	Hoverboard	annulé	50000
2	Full body VR	financé	10000
3	Perpetual motion	financé	500

soutient

uid	pid	montant
2	2	6000
4	3	500
1	2	5000

Administrateur

```
BEGIN;
UPDATE projet SET statut = 'financé'
WHERE pid IN (
  SELECT pid
  FROM projet NATURAL JOIN soutient
  GROUP BY pid
  HAVING sum(montant) >= requis
);
UPDATE projet SET statut = 'annulé'
WHERE statut = 'attente';
COMMIT;
```

Bob

```
BEGIN;
INSERT INTO soutient
SELECT 1, pid, 5000 FROM projet
WHERE statut = 'attente'
AND titre LIKE '%VR%';
COMMIT;
```

Exemple : erreur de type serialization anomaly

Deuxième exécution en série : Bob → Administrateur

projet

pid	titre	statut	requis
1	Hoverboard	annulé	50000
2	Full body VR	financé	10000
3	Perpetual motion	financé	500

soutient

uid	pid	montant
2	2	6000
4	3	500
1	2	5000

Administrateur

```
BEGIN;
UPDATE projet SET statut = 'financé'
WHERE pid IN (
  SELECT pid
  FROM projet NATURAL JOIN soutient
  GROUP BY pid
  HAVING sum(montant) >= requis
);
UPDATE projet SET statut = 'annulé'
WHERE statut = 'attente';
COMMIT;
```

Bob

```
BEGIN;
INSERT INTO soutient
SELECT 1, pid, 5000 FROM projet
WHERE statut = 'attente'
AND titre LIKE '%VR%';
COMMIT;
```

Si Bob termine en premier, l'administrateur valide le projet

Serializable

Serializable :

- Les transactions sont validées (commit) uniquement si elles sont équivalentes à une exécution en série
- Si une anomalie de sérialisation est détectée au moment du commit, la transaction entière est annulée

Erreurs possibles :

- Aucunes !

Remarques :

- Ce niveau d'isolation simule une exécution en série des transactions
- Il est très probable que les transactions longues échouent et doivent être relancées jusqu'à aboutir

Niveaux d'isolation en pratique

- Les spécifications du standard SQL sont des minima requis
- Les SGBD peuvent imposer des restrictions plus strictes
- Les choix et détails d'implémentation varient d'un système à l'autre

Sous PostgreSQL :

```
BEGIN TRANSACTION ISOLATION LEVEL niveau;
-- initie une transaction avec un niveau d'isolation choisi
```

- **READ UNCOMMITTED** existe pour des raisons de portabilité mais est implémenté comme **READ COMMITTED**
- **REPEATABLE READ** prévient aussi les lectures fantômes
- Implémentation avec locks (verrous) et snapshots (instantanés)

Attention aux spécificités de votre système : deadlocks, transactions abandonnées, erreurs de sérialisation, garanties des différents niveaux, etc.

Niveaux d'isolation sous PostgreSQL

	dirty read	nonrepeatable read	phantom read	serialization anomaly
read uncommitted	pas sous Postgres	possible	possible	possible
read committed	impossible	possible	possible	possible
repeatable read	impossible	impossible	pas sous Postgres	possible
serializable	impossible	impossible	impossible	impossible

En jaune : cas où PostgreSQL est plus strict que le standard SQL