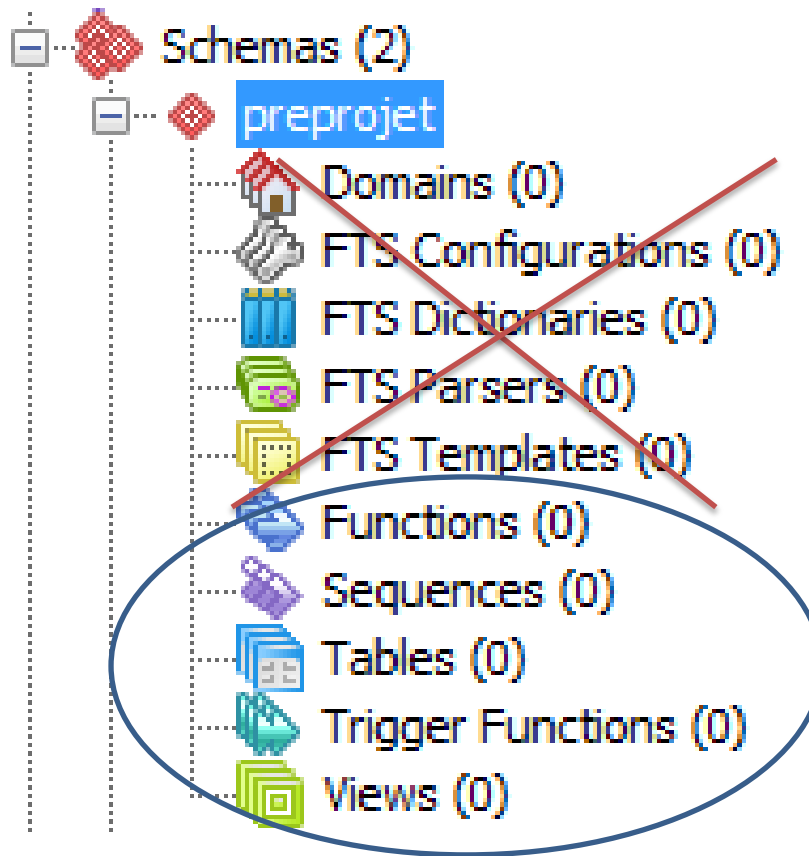


# SCHEMA

```
CREATE SCHEMA preprojet;
```



# PK auto-générée

```
CREATE SEQUENCE preprojet.pk_utilisateurs;  
CREATE SEQUENCE preprojet.pk_operations;
```

- En général

```
CREATE SEQUENCE name [ INCREMENT [ BY ] increment ]  
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE  
    maxvalue | NO MAXVALUE ] [ START [ WITH ] start ]
```

```
CREATE TABLE preprojet.utilisateurs (  
    id_utilisateur INTEGER PRIMARY KEY  
        DEFAULT NEXTVAL ('preprojet.pk_utilisateurs'),  
    nom VARCHAR(100) NOT NULL CHECK (nom<>''),  
    prenom VARCHAR(100) NOT NULL CHECK (prenom<>'')  
);
```

```
CREATE TABLE preprojet.comptes (  
    numero CHARACTER(10) PRIMARY KEY  
        CHECK(numero SIMILAR TO '[0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9][0-9]'),  
    id_utilisateur INTEGER REFERENCES preprojet.utilisateurs  
        (id_utilisateur)  
);
```

```
CREATE TABLE preprojet.operations (  
    id_operation INTEGER PRIMARY KEY DEFAULT NEXTVAL  
        ('preprojet.pk_operations'),  
    compte_source CHARACTER(10) REFERENCES preprojet.comptes (numero),  
    compte_destination CHARACTER(10) REFERENCES preprojet.comptes (numero),  
    montant INTEGER NOT NULL CHECK (montant>0),  
    date_op TIMESTAMP NOT NULL,  
    CHECK (compte_source<>compte_destination)  
);
```

```
INSERT INTO preprojet.utilisateurs VALUES (DEFAULT, 'Grolaux', 'Donatien');
INSERT INTO preprojet.utilisateurs VALUES (DEFAULT, 'Damas', 'Christophe');
INSERT INTO preprojet.utilisateurs VALUES (DEFAULT, 'Ferneeuw', 'Stéphanie');
```

```
INSERT INTO preprojet.comptes VALUES ('5632-12564',1);
INSERT INTO preprojet.comptes VALUES ('1236-02364',1);
INSERT INTO preprojet.comptes VALUES ('1234-56789',2);
INSERT INTO preprojet.comptes VALUES ('9876-87654',2);
INSERT INTO preprojet.comptes VALUES ('7896-23565',3);
```

```
INSERT INTO preprojet.operations VALUES
    (DEFAULT, '1234-56789', '5632-12564', 100, '2006-12-1');
INSERT INTO preprojet.operations VALUES
    (DEFAULT, '5632-12564', '1236-02364', 120, '2006-12-2');
INSERT INTO preprojet.operations VALUES
    (DEFAULT, '9876-87654', '7896-23565', 80, '2006-12-3');
INSERT INTO preprojet.operations VALUES
    (DEFAULT, '7896-23565', '9876-87654', 80, '2006-12-4');
INSERT INTO preprojet.operations VALUES
    (DEFAULT, '1236-02364', '7896-23565', 150, '2006-12-5');
INSERT INTO preprojet.operations VALUES
    (DEFAULT, '5632-12564', '1236-02364', 120, '2006-12-6');
INSERT INTO preprojet.operations VALUES
    (DEFAULT, '1234-56789', '5632-12564', 100, '2006-12-7');
INSERT INTO preprojet.operations VALUES
    (DEFAULT, '9876-87654', '7896-23565', 80, '2006-12-8');
INSERT INTO preprojet.operations VALUES
    (DEFAULT, '7896-23565', '9876-87654', 80, '2006-12-9');
```

```

SELECT u1.nom, u1.prenom, c1.numero,
       u2.nom, u2.prenom, c2.numero,
       o.date_op, o.montant
FROM preprojet.comptes c1, preprojet.comptes c2,
     preprojet.utilisateurs u1, preprojet.utilisateurs u2,
     preprojet.operations o
WHERE o.compte_source=c1.numero
AND c1.id_utilisateur=u1.id_utilisateur
AND o.compte_destination=c2.numero
AND c2.id_utilisateur=u2.id_utilisateur
ORDER BY o.date_op

```


	Data Output	Explain	Messages	History				
	nom character varying(100)	prenom character varying(100)	numero character(10)	nom character varying(100)	prenom character varying(100)	numero character(10)	date_op timestamp without time zone	montant integer
1	Damas	Christophe	1234-56789	Grolaux	Donatien	5632-12564	2006-12-01 00:00:00	100
2	Grolaux	Donatien	5632-12564	Grolaux	Donatien	1236-02364	2006-12-02 00:00:00	120
3	Damas	Christophe	9876-87654	Ferneeuw	Stéphanie	7896-23565	2006-12-03 00:00:00	80
4	Ferneeuw	Stéphanie	7896-23565	Damas	Christophe	9876-87654	2006-12-04 00:00:00	80
5	Grolaux	Donatien	1236-02364	Ferneeuw	Stéphanie	7896-23565	2006-12-05 00:00:00	150
6	Grolaux	Donatien	5632-12564	Grolaux	Donatien	1236-02364	2006-12-06 00:00:00	120
7	Damas	Christophe	1234-56789	Grolaux	Donatien	5632-12564	2006-12-07 00:00:00	100
8	Damas	Christophe	9876-87654	Ferneeuw	Stéphanie	7896-23565	2006-12-08 00:00:00	80
9	Ferneeuw	Stéphanie	7896-23565	Damas	Christophe	9876-87654	2006-12-09 00:00:00	80

```
CREATE OR REPLACE FUNCTION preprojet.insererTransaction (VARCHAR(100), VARCHAR(100), CHARACTER(10),
    VARCHAR(100), VARCHAR(100), CHARACTER(10), TIMESTAMP, INTEGER) RETURNS INTEGER AS $$
DECLARE
    nom_source ALIAS FOR $1;
    prenom_source ALIAS FOR $2;
    compte_source ALIAS FOR $3;
    nom_destination ALIAS FOR $4;
    prenom_destination ALIAS FOR $5;
    compte_destination ALIAS FOR $6;
    date_operation ALIAS FOR $7;
    montant_operation ALIAS FOR $8;
    id INTEGER:=0;
BEGIN
    IF NOT EXISTS (SELECT * FROM preprojet.comptes c, preprojet.utilisateurs u
        WHERE c.numero=compte_source AND c.id_utilisateur=u.id_utilisateur
        AND u.nom=nom_source and u.prenom=prenom_source) THEN
        RETURN -1;
    END IF;
    IF NOT EXISTS (SELECT * FROM preprojet.comptes c, preprojet.utilisateurs u
        WHERE c.numero=compte_destination AND c.id_utilisateur=u.id_utilisateur
        AND u.nom=nom_destination and u.prenom=prenom_destination) THEN
        RETURN -2;
    END IF;
    INSERT INTO preprojet.operations VALUES
        (DEFAULT,compte_source,compte_destination,montant_operation,date_operation)
    RETURNING id_operation INTO id;
    RETURN id;
EXCEPTION
WHEN check_violation THEN RETURN -3;
END;
$$ LANGUAGE plpgsql;
```



Attention aux conflits de nom  
entre les variables locales à la  
procédure et les noms des tables  
et colonnes !

# Automatisation

- Base de données dénormalisée = redondance, risque d'incohérence
- SQL procédural = code exécutable directement sur le serveur
-  Utilisons SQL procédural pour gérer la cohérence des données automatiquement : TRIGGER

# Exemple

9. Pour chaque compte en banque, ajoutez un champ solde. Ce champ contiendra le solde du compte en banque (somme de tous les montants dont ce compte est destinataire moins la somme de tous les montants dont ce compte est l'origine). Créez un trigger pour mettre ce champ à jour automatiquement.

```
ALTER TABLE preprojet.comptes ADD COLUMN solde INTEGER;
```



# TRIGGER

```
CREATE TRIGGER nom
  { BEFORE | AFTER } { evenement [ OR ... ] }
  ON table
  [ FOR [ EACH ] { ROW | STATEMENT } ]      EXECUTE
  PROCEDURE nomfonc ( arguments )
```

- **événement** : INSERT, UPDATE **ou** DELETE
- BEFORE, AFTER : **avant/après** que l'action se soit passé
- FOR EACH ROW | STATEMENT : **trigger appelé pour chaque ligne/une seule fois pour l'opération au complet**

# TRIGGER en PL/pgSQL

- La procédure `RETURNS TRIGGER` obligatoirement
  - Type `TRIGGER` est similaire au type `RECORD`
- Des variables locales sont automatiquement déclarées, typées et initialisées
  - `NEW` : de type `RECORD`
  - `OLD` : de type `RECORD`
  - `TG_NARGS` : de type `INTEGER`, le nombre d'arguments donnés à la procédure trigger
  - `TG_ARGV[]` : de type tableau de `TEXT`, les arguments donnés à la procédure trigger
  - ... confer syllabus

```
CREATE OR REPLACE FUNCTION preprojet.trigger () RETURNS TRIGGER AS $$
DECLARE
    ancien_solde INTEGER;
BEGIN
    SELECT c.solde FROM preprojet.comptes c
        WHERE c.numero=NEW.compte_source INTO ancien_solde;
    UPDATE preprojet.comptes
        SET solde=ancien_solde-NEW.montant
        WHERE numero=NEW.compte_source;
    SELECT c.solde FROM preprojet.comptes c
        WHERE c.numero=NEW.compte_destination INTO ancien_solde;
    UPDATE preprojet.comptes
        SET solde=ancien_solde+NEW.montant
        WHERE numero=NEW.compte_destination;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigger_solde AFTER INSERT ON preprojet.operations  
    FOR EACH ROW EXECUTE PROCEDURE preprojet.trigger();
```

```
DELETE FROM preprojet.operations;  
UPDATE preprojet.comptes SET solde=0;
```

```
INSERT INTO preprojet.operations VALUES  
    (DEFAULT, '1234-56789', '5632-12564', 100, '2006-12-1');  
INSERT INTO preprojet.operations VALUES  
    (DEFAULT, '5632-12564', '1236-02364', 120, '2006-12-2');  
INSERT INTO preprojet.operations VALUES  
    (DEFAULT, '9876-87654', '7896-23565', 80, '2006-12-3');  
INSERT INTO preprojet.operations VALUES  
    (DEFAULT, '7896-23565', '9876-87654', 80, '2006-12-4');  
INSERT INTO preprojet.operations VALUES  
    (DEFAULT, '1236-02364', '7896-23565', 150, '2006-12-5');  
INSERT INTO preprojet.operations VALUES  
    (DEFAULT, '5632-12564', '1236-02364', 120, '2006-12-6');  
INSERT INTO preprojet.operations VALUES  
    (DEFAULT, '1234-56789', '5632-12564', 100, '2006-12-7');  
INSERT INTO preprojet.operations VALUES  
    (DEFAULT, '9876-87654', '7896-23565', 80, '2006-12-8');  
INSERT INTO preprojet.operations VALUES  
    (DEFAULT, '7896-23565', '9876-87654', 80, '2006-12-9');
```

```

SELECT u.nom, u.prenom, c.*
FROM preprojet.utilisateurs u, preprojet.comptes c
WHERE u.id_utilisateur=c.id_utilisateur;

```

Data Output	Explain	Messages	History			
	<b>nom</b> character varying(100)	<b>prenom</b> character varying(100)	<b>numero</b> character(10)	<b>id_utilisateur</b> integer	<b>solde</b> integer	
1	Grolaux	Donatien	1236-02364	1	90	
2	Damas	Christophe	1234-56789	2	-200	
3	Grolaux	Donatien	5632-12564	1	-40	
4	Ferneeuw	Stéphanie	7896-23565	3	150	
5	Damas	Christophe	9876-87654	2	0	

# TRIGGER : précisions supplémentaires

```
CREATE TRIGGER nom
  { BEFORE | AFTER } { evenement [ OR ... ] }
  ON table
  [ FOR [ EACH ] { ROW | STATEMENT } ]      EXECUTE PROCEDURE
  nomfunc ( arguments )
```

- BEFORE = avant l'opération, AFTER = après
- BEFORE **et** niveau ROW :
  - RETURN NULL = ne pas faire l'opération du tout.
  - RETURN unRecord = utiliser le tuple unRecord plutôt que ce qui était prévu.
  - RETURN NEW = effectuer l'opération telle que prévue (en considérant que NEW n'a pas été modifié par la procédure).
- AFTER **ou** niveau STATEMENT :
  - RETURN sans effet, on peut RETURN NULL tout le temps.

Tant que l'on en est à ajouter de  
l'intelligence à notre base de  
données...

- SQL cherche à offrir une solution complète pour la gestion des données
  - Performance, sécurité, concurrence, autres, etc...

# Performances...

```
SELECT * FROM preprojet.comptes WHERE solde>0;
```

```
SELECT * FROM preprojet.comptes NATURAL INNER JOIN  
preprojet.utilisateurs;
```

- Il faut scanner toute les tables pour répondre au SELECT.
- La performance sera proportionnelle à la quantité de données.
  - Truc 1 : serveur avec beaucoup de mémoire vive pour que la BD soit entièrement en mémoire.
  - Truc 2 : optimiser le calcul des conditions : INDEX !



# INDEX

- Souvenons-nous du cours de première année :  
B-Tree et fonction de hashing.
  - Recherche passe de  $O(n)$  à  $O(\log_m(n))$

```
CREATE INDEX nom ON table ( { colonne | ( expression  
    ) } [, ...] )
```

# Exemples

- `CREATE INDEX title_idx ON films (title);`

Crée un index B-tree sur la colonne titre dans la table films

- `CREATE INDEX idx_titre_minuscule ON films ((lower(titre)));`

Crée un index sur l'expression lower(titre), permettant une recherche efficace quelque soit la casse

# Quand créer un INDEX ?

- Accélère les parcours complet des tables.
- La maintenance de l'index ralentit l'insertion/la modification/l'effacement des tuples.
  - Il faut donc déterminer les parcours complets des tables effectués lors des requêtes dont l'application a besoin.

# Sécurité

- Tous les utilisateurs n'ont pas tous les mêmes droits sur la DB.
  - La DB est le dernier rempart pour vérifier ces droits.
  - Gestion par rôles plutôt que par utilisateur individuel.
    - La couche application gère les utilisateurs individuels
    - La base de données gère la distinction entre un utilisateur normal, un administrateur système, un utilisateur spécial, etc.

# CREATE ROLE

```
CREATE ROLE nom [ [ WITH ] option [ ... ] ]
```

où *option* peut être :

```
SUPERUSER | NOSUPERUSER  
| CREATEDB | NOCREATEDB  
| CREATEROLE | NOCREATEROLE  
| CREATEUSER | NOCREATEUSER  
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'motdepasse'  
| VALID UNTIL 'dateheure'
```

```
CREATE USER davide WITH PASSWORD 'jw8s0F4';
```

# GRANT/REVOKE

## Manipulation des droits d'un rôle

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE |  
REFERENCES | TRIGGER } [,...] | ALL [ PRIVILEGES ] } ON  
[ TABLE ] nomtable [, ...] TO {nomrole | PUBLIC } [,  
...] [ WITH GRANT OPTION ]
```

```
REVOKE { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE |  
REFERENCES | TRIGGER } [,...] | ALL [ PRIVILEGES ] } ON  
[ TABLE ] nomtable [, ...] TO {nomrole | PUBLIC } [,  
...] [ WITH GRANT OPTION ]
```

```
GRANT INSERT ON films FROM PUBLIC;
```

```
REVOKE INSERT ON films FROM PUBLIC;
```

# Facilités supplémentaires

- Table normalisée = information partagée entre beaucoup de tables.
- Pour récupérer de l'information utile il faudra fréquemment faire `SELECT` avec jointures.
- Simplification : `VIEW`
  - Table virtuelle = au résultat d'un `SELECT`
  - Lecture seule (limitation de PostgreSQL)
  - Attention les noms des colonnes doivent être différentes

# VIEW

```
CREATE VIEW nom AS requête
```

```
CREATE VIEW preprojet.tout AS
  SELECT u1.nom AS "Nom Source", u1.prenom AS "Prenom Source",
         c1.numero AS "Numero Source", u2.nom AS "Nom Destination",
         u2.prenom AS "Prenom Destination",
         c2.numero AS "Numero Destination",
         o.date_op, o.montant
  FROM preprojet.comptes c1, preprojet.comptes c2,
       preprojet.utilisateurs u1, preprojet.utilisateurs u2,
       preprojet.operations o
  WHERE o.compte_source=c1.numero AND
         c1.id_utilisateur=u1.id_utilisateur
         AND o.compte_destination=c2.numero and
         c2.id_utilisateur=u2.id_utilisateur
  ORDER BY o.date_op;

SELECT * FROM preprojet.tout;
```



```
SELECT * FROM preprojet.tout;
```

	Nom Source character varying(100)	Prenom Source character varying(100)	Nomero Source character(10)	Nom Destination character varying(100)	Prenom Destination character varying(100)	Nomero Destination character(10)	date_op timestamp without time zone	montant integer
1	Damas	Christophe	1234-56789	Grolaux	Donatien	5632-12564	2006-12-01 00:00:00	100
2	Grolaux	Donatien	5632-12564	Grolaux	Donatien	1236-02364	2006-12-02 00:00:00	120
3	Damas	Christophe	9876-87654	Ferneeuw	Stéphanie	7896-23565	2006-12-03 00:00:00	80
4	Ferneeuw	Stéphanie	7896-23565	Damas	Christophe	9876-87654	2006-12-04 00:00:00	80
5	Grolaux	Donatien	1236-02364	Ferneeuw	Stéphanie	7896-23565	2006-12-05 00:00:00	150
6	Grolaux	Donatien	5632-12564	Grolaux	Donatien	1236-02364	2006-12-06 00:00:00	120
7	Damas	Christophe	1234-56789	Grolaux	Donatien	5632-12564	2006-12-07 00:00:00	100
8	Damas	Christophe	9876-87654	Ferneeuw	Stéphanie	7896-23565	2006-12-08 00:00:00	80
9	Ferneeuw	Stéphanie	7896-23565	Damas	Christophe	9876-87654	2006-12-09 00:00:00	80

```
SELECT * FROM preprojet.tout
WHERE "Nom Source"='Grolaux';
```

	Nom Source character varying(100)	Prenom Source character varying(100)	Nomero Source character(10)	Nom Destination character varying(100)	Prenom Destination character varying(100)	Nomero Destination character(10)	date_op timestamp without time zone	montant integer
1	Grolaux	Donatien	5632-12564	Grolaux	Donatien	1236-02364	2006-12-02 00:00:00	120
2	Grolaux	Donatien	1236-02364	Ferneeuw	Stéphanie	7896-23565	2006-12-05 00:00:00	150
3	Grolaux	Donatien	5632-12564	Grolaux	Donatien	1236-02364	2006-12-06 00:00:00	120

# Problématique supplémentaire :

## Gestion de la concurrence

- Généralement les BD sont utilisées par plusieurs personnes simultanément.
  - ex : site web de réservation de place d'avion
- Risque d'erreurs :

George	Système	Jerry
Est-ce que la place 3 est libre ?		
	oui	
		Est-ce que la place 3 est libre ?
	oui	
Place George dans 3		
	3 -> George	
		Place Jerry dans 3
	3 -> Jerry	

# Transaction

- Pour gérer la concurrence on utilise le concept de transaction.
  - Au sein de chaque transaction, tout se déroule comme si on avait l'exclusivité sur la BD
  - Chaque transaction peut soit
    - Réussir (commit) : toutes les modifications de la transaction sont prises en compte, comme si elle avait eu un accès exclusif à la BD.
    - S'annuler (rollback) : toutes les modifications sont annulées, comme si la transaction n'avait jamais eu lieu.

# ACID

- Acronyme des conditions nécessaires au fonctionnement des transactions
  - Atomicité
  - Cohérence
  - Isolation
  - Durable

# Atomicité

- Les transactions se terminent soit par
  - commit : l'intégralité de la transaction est effectuée
  - rollback : l'intégralité de la transaction est annulée, comme si elle n'avait jamais eu lieu

# Cohérence

- Le contenu de la BD à la fin de la transaction doit être cohérent.
  - Pendant la transaction, le contenu peut être incohérent.
  - Si le résultat d'une transaction est incohérent, elle est complètement annulée (rollback)

# Isolation

- Si les deux transactions A et B sont exécutées en même temps
  - Les modifications de A ne sont pas visibles par B
  - Les modifications de B ne sont pas visibles par A
- C'est seulement au commit que les modifications deviennent visibles.

# Durable

- Une fois commité, les modifications sont belles et bien présentes dans la BD.
- Si A et B s'exécutent en même temps, A est commitée, B ne peut pas recouvrir les modifications de A.
  - Si B essaie, il est annulé (rollback).



# En SQL pur (pas pl/pgSQL)

- Jusque là nous n'avons jamais géré les transactions !
- Dans ce cas : chaque instruction fonctionne dans sa propre transaction
  - commit automatique si l'instruction réussit
  - rollback sinon + message d'erreur
  - Ceci s'appelle l'auto-commit

# Bloc de transaction

Commence par

```
START TRANSACTION [ mode_transaction]
```

Et se termine par

```
COMMIT
```

Ou bien

```
ROLLBACK
```

# [mode\_transaction]

- SQL défini 4 niveaux d'isolation
  - Contrôle 3 effets pathologiques
    - **lecture sale** : Une transaction lit des données écrites par une transaction concurrente non validée.
    - **lecture non reproductible** : Une transaction relit des données qu'elle a lu précédemment et trouve que les données ont été modifiées par une autre transaction (validée depuis la lecture initiale).
    - **lecture fantôme** : Une transaction ré-exécute une requête renvoyant un ensemble de lignes satisfaisant une condition de recherche et trouve que l'ensemble des lignes satisfaisant la condition a changé du fait d'une autre transaction récemment validée.

# [mode\_transaction]

Niveau d'isolation	Lecture sale	Lecture non reproductible	Lecture fantôme
<b>Read Uncommitted</b> (en français, « Lecture de données non validées »)	Possible	Possible	Possible
<b>Read Committed</b> (en français, « Lecture de données validées »)	Impossible	Possible	Possible
<b>Repeatable Read</b> (en français, « Lecture répétée »)	Impossible	Impossible	Possible
<b>Serializable</b> (en français, « Sérialisable »)	Impossible	Impossible	Impossible

En PostgreSQL : seulement SERIALIZABLE et READ COMMITTED

# Bloc de transaction

START TRANSACTION [ *mode\_transaction* ]

Où [ *mode\_transaction* ] est

ISOLATION LEVEL { SERIALIZABLE | READ COMMITTED }  
READ WRITE | READ ONLY

# En pl/pgSQL (donc pas en pur SQL)

- Chaque fonction est exécutée dans sa propre transaction.
  - La transaction en cours si elle existe
  - Une nouvelle transaction en mode auto-commit au sinon
- Si il y a une exception non attrapée
  - Il y a un rollback automatique
- Comme PostgreSQL ne supporte pas les transactions imbriquées
  - START TRANSACTION, COMMIT et ROLLBACK sont interdits en PL/pgSQL !

```

CREATE OR REPLACE FUNCTION preprojet.insererTransaction(VARCHAR(100),VARCHAR(100),CHARACTER(10),
                VARCHAR(100),VARCHAR(100),CHARACTER(10),TIMESTAMP,INTEGER) RETURNS INTEGER AS
    $$
DECLARE
    nom_source ALIAS FOR $1;
    prenom_source ALIAS FOR $2;
    compte_source ALIAS FOR $3;
    nom_destination ALIAS FOR $4;
    prenom_destination ALIAS FOR $5;
    compte_destination ALIAS FOR $6;
    date_operation ALIAS FOR $7;
    montant_operation ALIAS FOR $8;
    id INTEGER:=0;
BEGIN
    IF NOT EXISTS(SELECT * FROM preprojet.comptes c, preprojet.utilisateurs u
                    WHERE c.numero=compte_source AND c.id_utilisateur=u.id_utilisateur
                    AND u.nom=nom_source and u.prenom=prenom_source) THEN
        RAISE foreign key violation;
    END IF;
    IF NOT EXISTS(SELECT * FROM preprojet.comptes c, preprojet.utilisateurs u
                    WHERE c.numero=compte_destination AND c.id_utilisateur=u.id_utilisateur
                    AND u.nom=nom_destination and u.prenom=prenom_destination) THEN
        RAISE foreign key violation;
    END IF;
    INSERT INTO preprojet.operations VALUES
        (DEFAULT,compte_source,compte_destination,montant_operation,date_operation)
        RETURNING id_operation INTO id;
    RETURN id;
END;
$$ LANGUAGE plpgsql;

```