```
DROP SCHEMA IF EXISTS preprojet CASCADE;
CREATE SCHEMA preprojet;

CREATE TABLE preprojet.utilisateurs (
    id_utilisateur SERIAL PRIMARY KEY,
    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) NOT NULL
);


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

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

```
CREATE OR REPLACE FUNCTION preprojet.insererTransaction(nom_source VARCHAR(100), prenom_source VARCHAR(100),
    compte_source CHARACTER(10), nom_destination VARCHAR(100), prenom_destination VARCHAR(100), compte_destination
    CHARACTER(10),date_operation DATE, montant_operation INTEGER) RETURNS INTEGER AS $$

DECLARE

    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;
```

AUTOMATISATION

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

EXEMPLE

- **Dans pubs2, le champ total_sales de la table titre doit être mis à jour lors d'un tuple dans sales_detail**
  - -> utilisation de trigger

- **Trigger**
  - Procédure stockée déclenchée avant/après une insertion/ mis-à-jour/suppression

TRIGGER

```
CREATE TRIGGER nom

   {BEFORE|AFTER} {INSERT|UPDATE|DELETE} ON table
   [ FOR EACH { ROW | STATEMENT } ]
   EXECUTE PROCEDURE nomfonc ( arguments )
```

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 DÉCLENCHE UNE PROCÉDURE

- **La procédure `RETURNS TRIGGER` obligatoirement**
  - Type `TRIGGER` est similaire au type `RECORD`
- **Des variables locales sont automatiquement déclarées, typées et initialisées dans la procédure**
  - `OLD`: de type `RECORD`
    - Contient
      - l'ancienne valeur du tuple modifié pour un UPDATE
      - l'ancienne valeur du tuple effacé pour un DELETE
      - NULL pour un INSERT
  - `NEW`: de type `RECORD`
    - Contient
      - La nouvelle valeur du tuple dans la table pour un INSERT/UPDATE
      - NULL pour un DELETE

```
CREATE OR REPLACE FUNCTION total_sales() RETURNS TRIGGER AS $$
   DECLARE
       total INTEGER;
   BEGIN
       SELECT SUM(sd.qty) FROM salesdetail sd WHERE sd.title_id=NEW.title_id
                 INTO total;
       UPDATE titles SET total_sales = total WHERE title_id=NEW.title_id;
       RETURN NEW;
   END;

$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER sales_detail_trigger AFTER INSERT ON SALESDETAIL FOR EACH ROW
   EXECUTE PROCEDURE total_sales();
```

TRIGGER : PRÉCISIONS SUPPLÉMENTAIRES

- **`CREATE TRIGGER nom`**

  `{BEFORE|AFTER} {evenement [OR ...]} ON table`
  `[ FOR [ EACH ] { ROW | STATEMENT } ]`
  `EXECUTE PROCEDURE nomfonc ( 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 NEW` tout le temps.

BONNE PRATIQUE

- **Ne jamais utiliser un trigger BEFORE en renvoyant NULL**
  - Dangereux
    - Un insert peut sembler avoir réussi alors que le trigger l'a silencieusement annulé
    - En cas de problème, il est très difficile de trouver l'origine de ce problème
- **Si on veut empêcher l'opération**
  - Lancer une exception

VIEW

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
- 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 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	Khaddam	Iyad	5632-12564	2006-12-01 00:00:00	100
2	Khaddam	Iyad	5632-12564	Khaddam	Iyad	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	Khaddam	Iyad	1236-02364	Ferneeuw	Stéphanie	7896-23565	2006-12-05 00:00:00	150
6	Khaddam	Iyad	5632-12564	Khaddam	Iyad	1236-02364	2006-12-06 00:00:00	120
7	Damas	Christophe	1234-56789	Khaddam	Iyad	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"='Damas';
```

Data Output | Explain | Messages | History

	Nom Source character varying(100)	Prenom Source character varying(100)	Numero Source character(10)	Nom Destination character varying(100)	Prenom Destination character varying(100)	Numero Destination character(10)	date_op timestamp without time zone	montant integer
1	Damas	Christophe	1234-56789	Khaddam	Iyad	5632-12564	2006-12-01 00:00:00	100
2	Damas	Christophe	9876-87654	Ferneeuw	Stéphanie	7896-23565	2006-12-03 00:00:00	80
3	Damas	Christophe	1234-56789	Khaddam	Iyad	5632-12564	2006-12-07 00:00:00	100
4	Damas	Christophe	9876-87654	Ferneeuw	Stéphanie	7896-23565	2006-12-08 00:00:00	80

SÉCURITÉ

Tous les utilisateurs n’ont pas tous les mêmes droits sur la DB.

- La DB est le dernier rempart pour vérifier ces droits..

CREATE USER

CREATE USER *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 CONNECT ON DATABASE pubs2 TO davide;

( GRANT USAGE ON SCHEMA pubs2 TO davide; )
```

```
GRANT SELECT ON sales, salesdetail, titles, publishers, store,
   titleauthors TO davide ;

GRANT INSERT ON TABLE sales, salesdetail TO davide ;
```