Base de donnees - Achats

Théo Lorio / Gabriel Bailleul / Louis Iwinski

06/10/2021

Base de données - Achats

Théo Lorio / Gabriel Bailleul / Louis Iwinski

Sommaire

Étape 1

Diagramme E/R

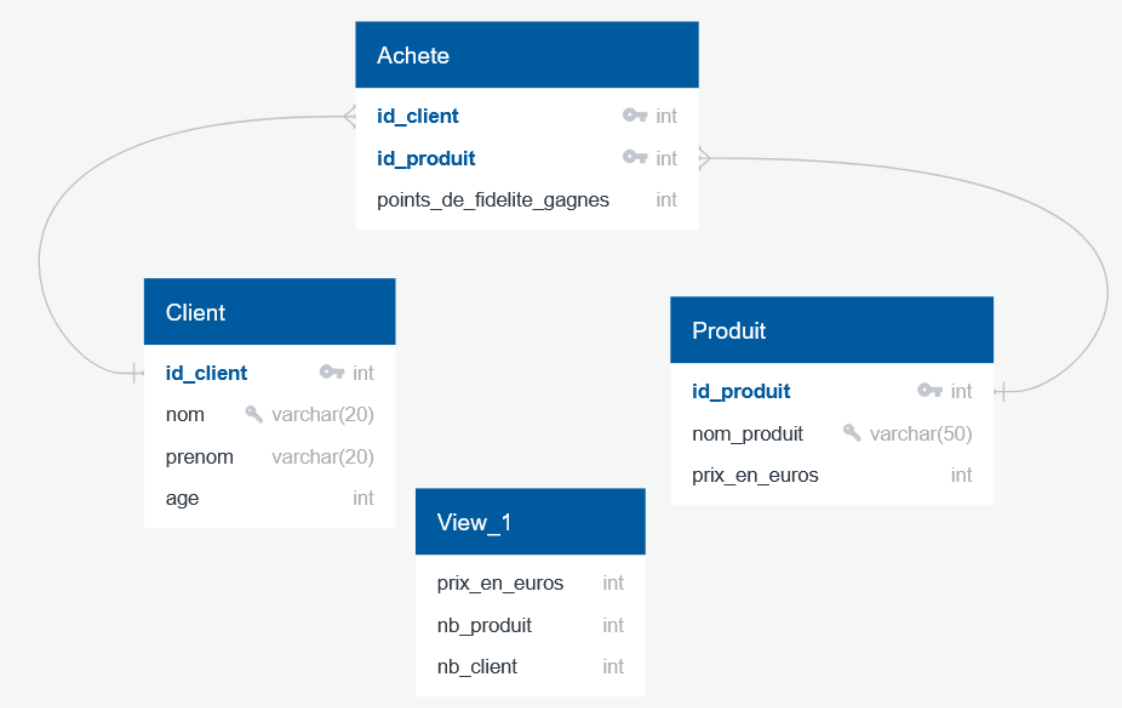


Diagramme E/R de notre base de données Achats réalisé sur QuickDBD

Légende :

Unique Primary Key varchar(50) int Type des données

Description sommaire de la base de données

Durant ce premier semestre, en groupe de 3 (Louis Iwinski, Théo Lorio et Gabriel Bailleul), on nous a demandé de réaliser une activité, dans le cadre du cours Base de données avancées, qui permet de créer une base de données. Nous avons décidé de la nommer « Achats » et elle comporte trois tables (« Client », « Produit » et « Achete ») et une vue (« view_1 »).

```
achats=# \dt
          Liste des relations
 SchÚma |   Nom   | Type  | PropriÉtaire
--------+---------+-------+--------------
 public | achete  | table | postgres
 public | client  | table | postgres
 public | produit | table | postgres
(3 lignes)
```

Tableau des trois tables de la base de données « Achats »

Cette base de données pourrait s'apparenter à une base qui pourrait être utilisé dans un magasin de sport et de vêtements. On peut le voir grâce aux tuples insérés dans les différentes tables de base de données.

Par exemple :

```
achats=# select * from produit;
 id_produit |   nom_produit    | prix_en_euros
------------+------------------+---------------
          1 | Ballon de basket |            15
          2 | T-shirt RC Lens  |            60
          3 | Sac              |            20
          4 | Kayak            |           150
          5 | Chaussettes      |             8
          6 | Protéines        |             4
          7 | Sacoche          |             7
          8 | Bermudas         |            15
          9 | Eau              |             3
         10 | Chasuble         |            10
         11 | Chaussures       |            50
         12 | Drapeau          |            14
         13 | Protege genou    |            40
         14 | Panier basket    |            80
         15 | Velo             |           160
(15 lignes)
```

Tableau de tuples de la table Produit

Les tables de cette base de données sont créées avec plusieurs contraintes. Notamment, avec des « NOT NULL » ou encore des « UNIQUE ». Ce qui

permet de ne pas faire d'erreurs quand on insère de nouvelles valeurs dans les tables en question.

```
create table client(
    id_client int primary key,
    nom varchar(20) not null,
    prenom varchar(20) not null,
    age int,
    check(id_client>0),
    check(age>0),
    unique(nom),
    unique(id_client)
)
```

```
create table produit(
    id_produit int primary key,
    nom_produit varchar(50) not null,
    prix_en_euros int not null,
    check(prix_en_euros>=0),
    check(id_produit>0),
    unique(nom_produit)
)
```

```
create table achete(
    id_client int,
    id_produit int,
    points_de_fidelite_gagnes int,
    primary key(id_produit,id_client),
    foreign key(id_client) references client(id_client) on delete cascade on update cascade,
    foreign key(id_produit) references produit(id_produit) on delete cascade on update cascade
    check(points_de_fidelite_gagnes>=0)
)
```

Capture de la création des tables de la base de données

On a donc trois tables. La première se nomme Client est regroupe l'identifiant (id_client), le nom (nom), le prénom (prenom) ainsi que l'âge (age) du client. La deuxième se nomme Produit et regroupe l'identifiant (id_produit), le nom du produit (nom_produit) et le prix en euros (prix_en_euros) du produit. Pour finir, on a la table Achète qui regroupe les deux clés primaires des deux autres tables (id_client et id_produit).

Chaque table comporte quinze tuples différentes.

Tests des contraintes et affichages des erreurs

- Test numéro 4 Client– sur la contrainte NOT NULL sur nom de Client

```
achats=# insert into client values(16,null,'Marc',17);
ERREUR:  une valeur NULL viole la contrainte NOT NULL de la colonne « nom » dans la relation « client »
DÉTAIL : La ligne en échec contient (16, null, Marc, 17).
```

- Test numéro 6 Client – sur la contrainte de l'age>0 sur age de Client

```
achats=# insert into client values(16,'Marc','Oracle',0);
ERREUR:  la nouvelle ligne de la relation « client » viole la contrainte de vérification « client_age_check »
DÉTAIL : La ligne en échec contient (16, Marc, Oracle, 0).
```

- Test numéro 3 Produit – Contrainte UNIQUE sur nom_produit de Produit

```
achats=# insert into produit values(18,'Velo',3);
ERREUR:  la valeur d'une clé dupliquée rompt la contrainte unique « produit_nom_produit_key »
DÉTAIL : La clé « (nom_produit)=(Velo) » existe déjà.
```

- Test numéro 8 Produit – Contrainte varchar(50) sur nom_produit de Produit

```
achats=# insert into produit values(18,'oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooo',85);
ERREUR:  valeur trop longue pour le type character varying(50)
```

- Test numéro 1 Achete – Contrainte UNIQUE sur (id_client,id_produit) de Achete

```
achats=# insert into achete values(1,6,15);
ERREUR:  la valeur d'une clé dupliquée rompt la contrainte unique « achete_pkey »
DÉTAIL : La clé « (id_produit, id_client)=(6, 1) » existe déjà.
```

Étape 2

Diagramme E/R sans contrainte d'intégrité

Diagramme E/R de notre base de données Achats sans contraintes d'intégrité réalisé sur QuickDBD

Étape 3

Un trigger et une fonction PL/SQL portant sur une table contenant une clé étrangère

```
CREATE OR REPLACE FUNCTION test_achete_after_iut() RETURNS trigger AS
$$
    DECLARE
    nb INTEGER;
    BEGIN
        select count(*) into nb
        from achete
        where id_client=NEW.id_client;
        if nb>1 then
            RAISE EXCEPTION 'deux tuples de achete(id_client) ne peuvent pas avoir la même valeur';
        end if;
        select count(*) into nb
        from achete
        where id_produit=NEW.id_produit;
        if nb>1 then
            RAISE EXCEPTION 'deux tuples de achete(id_produit) ne peuvent pas avoir la même valeur';
        end if;
        PERFORM count(*)
        from achete
        where id_client=NEW.id_client;
        IF NOT FOUND THEN
            RAISE EXCEPTION 'id_client de achete référence id_client de la table client';
        end if;
        PERFORM COUNT(*)
        from achete
        where id_produit=NEW.id_produit;
        IF NOT FOUND THEN
            RAISE EXCEPTION 'id_produit de achete référence id_produi de la table produit';
        end if;
        RETURN NEW;
    END;
$$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigger_achete_after_iu AFTER INSERT OR UPDATE ON ACHETE
    FOR EACH ROW EXECUTE PROCEDURE test_achete_after_iut();
```

Sur la capture d'écran au-dessus, nous pouvons voir une fonction PL/SQL qui se nomme test_achete_after_iut(). La fonction permet de tester une insertion ou une mise à jour d'un tuple de la table ACHETE. Celle-ci retourne un trigger. On commence à coder cette fonction par un DECLARE pour déclarer une variable nb en INTEGER. Par la suite, nous faisons une requête SQL pour prendre le tuple qu'on choisit avec un WHERE. Au passage, nous faisons aussi un COUNT(*) pour savoir le nombre d'id qui sont identiques.

De plus, nous réalisons une boucle « if » qui parcourt la table ACHETE et regarde si le nombre d'id identique est supérieur à 1. Si cela est vrai, alors la fonction retourne un message d'exception.

Par la suite, nous réalisons un PERFORM COUNT(*) pour exécuter la requête SQL sans récupérer le résultat. On regarde si on le trouve avec un NOT FOUND, s'il ne le trouve pas on envoie un message d'exception aussi.

Le trigger trigger_achete_after_iu permet d'exécuter la procédure test_achete_after_iut() après l'insertion ou une mise à jour d'un tuple de la table ACHETE. À l'aide, notamment, d'AFTER et du FOR EACH ROW EXECUTE PROCEDURE (qui permet de parcourir les lignes de la table une seule fois et de lancer la procédure).

Résultats des commandes :

```
achats=# INSERT INTO achete VALUES (25,1,15);
INSERT 0 1
```

```
achats=# INSERT INTO achete VALUES (4,25,10);
INSERT 0 1
```

```
achats=# INSERT INTO achete VALUES (25,1,15);
ERREUR:  deux tuples de achete(id_client) ne peuvent pas avoir la m^me valeur
CONTEXTE : fonction PL/pgSQL test_achete_after_iut(), ligne 9 à RAISE
```

```
achats=# INSERT INTO achete VALUES (8,25,10);
ERREUR:  deux tuples de achete(id_produit) ne peuvent pas avoir la m^me valeur
CONTEXTE : fonction PL/pgSQL test_achete_after_iut(), ligne 15 à RAISE
```

Les deux tuples s'insèrent normalement. Quand on saisit la commande une seconde fois, cela lance l'exception personnalisée grâce au trigger et à la fonction PL/SQL.

Étape 4

Schéma relationnel textuel des tables et de la vue

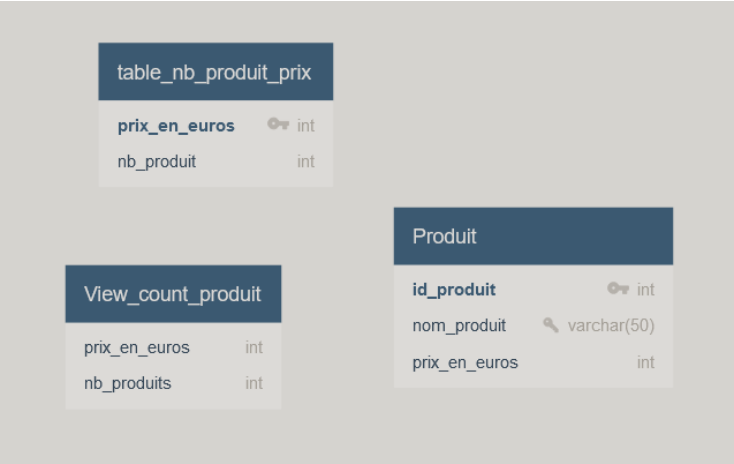


Diagramme E/R

```
create table produit(
    id_produit int PRIMARY KEY,
    nom_produit VARCHAR(50) NOT null,
    prix_en_euros int NOT NULL,
    CHECK(prix_en_euros>=0),
    CHECK(id_produit>0),
    UNIQUE(nom_produit)
);
```

```
create table table_nb_produit_prix (
    prix_en_euros INT PRIMARY KEY,
    nb_produit int,
    CHECK (nb_produit>0)
);
```

Tables produit et table_nb_produit_prix

Pour cette étape 4, on a décidé de reprendre la table produit déjà présente dans les étapes précédentes. Elle a pour attributs id_produit (qui va être la clé primaire), nom_produit (un varchar qui n'est jamais nul) et prix_en_euros (un integer qui n'est pas nul).

Néanmoins, nous avons ajoutés une nouvelle table qui se nomme « table_nb_produit_prix » qui a pour attributs prix_en_euros (un integer qui est la clé primaire de cette table) et nb_produit (qui est un integer, lui aussi, supérieur à 0).

```sql
-- Vue comptabilisant le nombre de produit par leur prix
CREATE VIEW view_count_produit AS
SELECT prix_en_euros,count(*) as "nb_produits"
FROM produit
GROUP BY prix_en_euros
ORDER BY prix_en_euros;
```

Vue view_count_produit

De plus, nous avons créés une nouvelle vue qui se nomme « view_count_produit » qui permet de visualiser le nombre de produits par rapport au prix en euros.

Code source des tests et affichage produit pour chaque requête SQL

```sql
select *
from produit;

select *
from table_nb_produit_prix;
```

```
achats=#
achats=# select *
achats-# from produit;
 id_produit | nom_produit | prix_en_euros
------------+-------------+---------------
(0 ligne)


achats=#
achats=# select *
achats-# from table_nb_produit_prix;
 prix_en_euros | nb_produit
---------------+------------
(0 ligne)
```

Contenus initiaux des tables produit et table_nb_produit_prix

```sql
INSERT INTO produit(id_produit,nom_produit,prix_en_euros) VALUES
(1,'Ballon de basket',15),
(2,'T-shirt RC Lens',50),
(3,'Sac',5),
(4,'Kayak',150),
(5,'Chaussettes',15),
(6,'Protéines',5),
(7,'Sacoche',15),
(8,'Bermudas',15),
(9,'Eau',5),
(10,'Chasuble',15),
(11,'Chaussures',50),
(12,'Drapeau',15),
(13,'Protege genou',50),
(14,'Panier basket',50),
(15,'Velo',150);
```

Un insert sur la table produit

```
\echo '\n***produit***'
select *
from produit;

\echo '\n***table_nb_produit_prix***'
select *
from table_nb_produit_prix
order by prix_en_euros;

\echo '\n***view_count_produit***'
select *
from view_count_produit;
```

```
***produit***
achats=# select *
achats-# from produit;
 id_produit |   nom_produit    | prix_en_euros
------------+------------------+---------------
          1 | Ballon de basket |            15
          2 | T-shirt RC Lens  |            50
          3 | Sac              |             5
          4 | Kayak            |           150
          5 | Chaussettes      |            15
          6 | Protéines        |             5
          7 | Sacoche          |            15
          8 | Bermudas         |            15
          9 | Eau              |             5
         10 | Chasuble         |            15
         11 | Chaussures       |            50
         12 | Drapeau          |            15
         13 | Protege genou    |            50
         14 | Panier basket    |            50
         15 | Velo             |           150
(15 lignes)
```

```
***table_nb_produit_prix***
achats=# select *
achats-# from table_nb_produit_prix
achats-# order by prix_en_euros;
 prix_en_euros | nb_produit
---------------+------------
             5 |          3
            15 |          6
            50 |          4
           150 |          2
(4 lignes)
```

```
***view_count_produit***
achats=# select *
achats-# from view_count_produit;
 prix_en_euros | nb_produits
---------------+-------------
             5 |           3
            15 |           6
            50 |           4
           150 |           2
(4 lignes)
```

Les résultats des requêtes SQL

Un delete sur la table produit

```
delete
from produit
where id_produit in (1,4);
```

```
***produit***
achats=#
achats=# select *
achats-# from produit;
 id_produit |   nom_produit   | prix_en_euros
------------+-----------------+---------------
          2 | T-shirt RC Lens |            50
          3 | Sac             |             5
          5 | Chaussettes     |            15
          6 | Protéines       |             5
          7 | Sacoche         |            15
          8 | Bermudas        |            15
          9 | Eau             |             5
         10 | Chasuble        |            15
         11 | Chaussures      |            50
         12 | Drapeau         |            15
         13 | Protege genou   |            50
         14 | Panier basket   |            50
         15 | Velo            |           150
(13 lignes)
```

```
***view_count_produit***
achats=#
achats=# select *
achats-# from view_count_produit;
 prix_en_euros | nb_produits
---------------+-------------
             5 |           3
            15 |           5
            50 |           4
           150 |           1
(4 lignes)
```

```
***table_nb_produit_prix***
achats=#
achats=# select *
achats-# from table_nb_produit_prix
achats-# order by prix_en_euros;
 prix_en_euros | nb_produit
---------------+------------
             5 |          3
            15 |          5
            50 |          4
           150 |          1
(4 lignes)
```

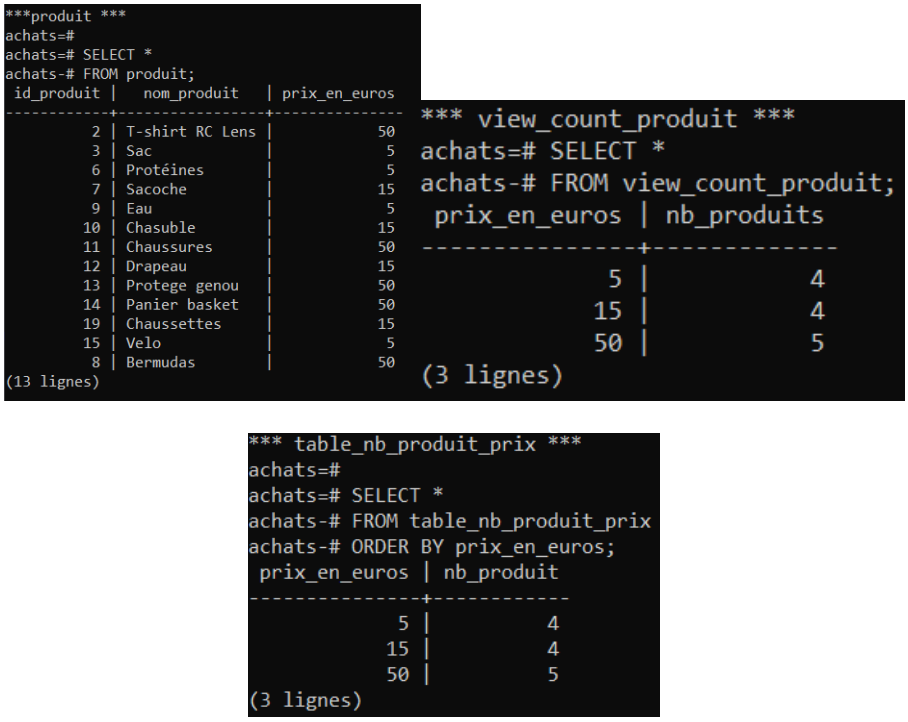
Les résultats des requêtes SQL

Une update sur la table produit

```sql
update produit
set id_produit=19
where id_produit=5;

update produit
set prix_en_euros=5
where id_produit=15;

update produit
set prix_en_euros=50
where id_produit=8;
```



```
***produit ***
achats=#
achats=# SELECT *
achats-# FROM produit;
 id_produit |   nom_produit   | prix_en_euros
------------+-----------------+--------------
          2 | T-shirt RC Lens |           50
          3 | Sac             |            5
          6 | Protéines       |            5
          7 | Sacoche         |           15
          9 | Eau             |            5
         10 | Chasuble        |           15
         11 | Chaussures      |           50
         12 | Drapeau         |           15
         13 | Protege genou   |           50
         14 | Panier basket   |           50
         19 | Chaussettes     |           15
         15 | Velo            |            5
          8 | Bermudas        |           50
(13 lignes)
```

```
*** view_count_produit ***
achats=# SELECT *
achats-# FROM view_count_produit;
 prix_en_euros | nb_produits
---------------+-------------
             5 |           4
            15 |           4
            50 |           5
(3 lignes)
```

```
*** table_nb_produit_prix ***
achats=#
achats=# SELECT *
achats-# FROM table_nb_produit_prix
achats-# ORDER BY prix_en_euros;
 prix_en_euros | nb_produit
---------------+------------
             5 |          4
            15 |          4
            50 |          5
(3 lignes)
```

Les résultats des requêtes SQL

Le principe de cette étape est de réaliser des fonctions qui déclenchent des triggers. Ces derniers copient les valeurs insérées, updatées et supprimées dans la table « table_nb_produit_prix » à partir de la table « produit ». La vue sert d'exemple à ce que doit ressembler à la table « table_nb_produit_prix ».