Livrable semaine 50
Triggers et procédures stockées

ALLEMANE Axel - ROMA Quentin - LE DONNE Antonin - RIBEIRO VAUR Nino - OLIVENCIA Eliot

Sommaire

1) Triggers....3
Trigger update points de fidélité....3
1) Code....3
2) Test et trace d'exécution....4
Trigger update prix total d'une commande....6
1) Code....6
2) Test et trace d'exécution....6
Trigger update quantité en stock d'un produit / vérifiant qu'il reste du stock....8
1) Code....8
2) Test et trace d'exécution....9
Trigger vérifiant si un avis peut être déposé par un client sur un produit....11
1) Code....11
2) Test et trace d'exécution....11
Trigger vérifiant qu'un produit ne peut pas être apparenté à lui-même et qu'un lien entre deux produits n'est pas déjà existant....14
1) Code....14
2) Test et trace d'exécution....15
3) Procédures / fonctions stockées....16
Procédure permettant de retirer les stocks des produits étant dans une Composition....16
1) Code....16
2) Test et trace d'exécution....18
3) Test depuis php....20
Procédure permettant de récupérer toutes les photos d'un produit....21
1) Code....21
2) Test depuis php....22
Fonction permettant de savoir si un utilisateur à le droit de déposer un avis sur un produit....24
1) Code....24
2) Test depuis php....25
Procédure permettant de recalculer les points de fidélité d'un client....28
1) Code....28
2) Test depuis php....29

1)Triggers

Trigger update points de fidélité

1) Code

```
DELIMITER $$
CREATE TRIGGER t_a_u_PtsFidelitea
AFTER UPDATE ON Commande
FOR EACH ROW
BEGIN
 DECLARE etait_payee BOOLEAN DEFAULT (OLD.statutCommande IN ('payee',
'expediee', 'livree'));
 DECLARE est_payee BOOLEAN DEFAULT (NEW.statutCommande IN ('payee',
'expediee', 'livree'));

 -- Commande réglée --> on crédite les points de fidélités
 IF (NOT etait_payee) AND est_payee THEN
   UPDATE Client
   SET ptsFidelite = ptsFidelite + (NEW.prixTotal * 0.10)
   WHERE idClient = NEW.idClient;
 END IF;

 -- Commande annulée : on débite les points de fidélités
 IF etait_payee AND (NOT est_payee) THEN
   UPDATE Client
   SET ptsFidelite = ptsFidelite - (OLD.prixTotal * 0.10)
   WHERE idClient = NEW.idClient;
 END IF;
END$$

DELIMITER ;
```

2) Test et trace d'exécution

```
SELECT * FROM `Client` WHERE idClient=52;
```

idClient	nom	prenom	email	motDePasse	adresse	codePostal	ville	telephone	ptsFidelite
52	Ribeiro	Nino	ninoribeiro@email.fr	$2y$10$VhUO8PV5EuIsUaZeIz6Buud2nzXW0.ELJpSHVJmvICQ...	28 Rue Alfred Rambaud	31400	Toulouse	0603486161	0

Insertion d'une nouvelle commande composée d'un seul produit à 19,99 "en attente" (non-débitée):

```
INSERT INTO Commande (
  idCommande, idClient, numTransaction, dateCommande, dateExpedition,
  dateReception, statutCommande, adresseLivraison, typePaiement, prixTotal
)
VALUES (
  1000,                       -- idCommande
  52,                         -- idClient
  'TX123456',                 -- numTransaction
  '2025-01-15 10:30:00',      -- dateCommande (prédefinie ici)
  NULL,                       -- dateExpedition
  NULL,                       -- dateReception
  'en_attente',               -- statutCommande
  '28 Rue Alfred Rambaud, 31400 Toulouse',  -- adresseLivraison
  'CB',
  19.99
);
INSERT INTO LigneCommande (
  idCommande, idDeclinaison, quantite, prixTotal
)
VALUES (
  1000,
  1,      -- idDeclinaison = potion de mana majeure
  1,      -- quantité
  19.99   -- prixTotal de la ligne
);
```

On vérifie le nombre de points de fidélité.

```
SELECT ptsFidelite FROM `Client` WHERE idClient=52;
```

attendu : 0
obtenu :

ptsFidelite
0

On modifie le statut de la commande à “payée”.

```
UPDATE Commande
SET statutCommande = 'payee'
WHERE idCommande = 1000;
```

```
SELECT ptsFidelite FROM Client WHERE idClient = 52;
```

Attendu : ptsFidelite = 2
Obtenu :

ptsFidelite
2

Le client a maintenant 2 points de fidélité, le test **PASSE**.

On modifie à nouveau le statut de la commande pour le basculer à “annulée”.

```
UPDATE Commande
SET statutCommande = 'annulee'
WHERE idCommande = 1000;
```

```
SELECT ptsFidelite FROM Client WHERE idClient = 52;
```

Attendu : ptsFidelite = 0
Obtenu :

ptsFidelite
0

Les points de fidélité ont bien été retirés au client, il a maintenant 0 points de fidélité, le test **PASSE**.

Trigger update prix total d'une commande

1) Code

```
DELIMITER $$
-- Trigger qui update le prix total d'une commande (dans la table
Commande)
CREATE Trigger t_ai_ligneCommande_prixTotalCommande
   AFTER INSERT ON LigneCommande
   FOR EACH ROW
BEGIN
   DECLARE prixTotCommande DECIMAL(10,2);
   -- Calcul du prix total de la commande
   SELECT SUM(prixTotal) INTO prixTotCommande FROM LigneCommande L
   WHERE NEW.idCommande = L.idCommande;

   -- Mise à jour du prix total de la commande
   UPDATE Commande SET prixTotal = prixTotCommande WHERE idCommande =
NEW.idCommande;

END$$

DELIMITER ;
```

2) Test et trace d'exécution

<u>Création d'une commande :</u>

```
INSERT INTO `commande` (`idClient`, `numTransaction`, `dateCommande`,
`dateExpedition`, `dateReception`, `statutCommande`, `adresseLivraison`,
`typePaiement`) VALUES ('4', '10169999999', '2024-11-10 09:15:22',
'2024-11-11 08:45:00', '2024-11-13 14:00:00', 'livree', 'IUT Blagnac',
'CB');
```

idCommande	idClient	numTransaction	dateCommande	dateExpedition	dateReception	statutCommande	adresseLivraison	typePaiement	prixTotal
103	4	10169999999	2024-11-10 09:15:22	2024-11-11 08:45:00	2024-11-13 14:00:00	livree	IUT Blagnac	CB	*NULL*

Ajout d'une ligne commande :

```
INSERT INTO `lignecommande` (`idCommande`, `idDeclinaison`, `quantite`)
VALUES ('103', '1', '2');
```

Vérification de la maj dans la table LigneCommande ***:***

On a acheté le produit 1 qui coûte 19.99 deux fois. Donc 2 * 19.99 = 39.98.

Contenu de la table LigneCommande :

idCommande	idDeclinaison	quantite	prixTotal
103	1	2	39.98

Vérification dans la table Commande

Contenu de la table Commande :

idCommande	idClient	numTransaction	dateCommande	dateExpedition	dateReception	statutCommande	adresseLivraison	typePaiement	prixTotal
103	4	10169999999	2024-11-10 09:15:22	2024-11-11 08:45:00	2024-11-13 14:00:00	livree	IUT Blagnac	CB	39.98

Le prix total de la commande est bien à jour.

On ajoute un autre produit dans la commande :

```
INSERT INTO `lignecommande` (`idCommande`, `idDeclinaison`, `quantite`)
VALUES ('103', '2', '6');
```

On a acheté le produit 2 qui coûte 7.99€ 6 fois. Donc 6 * 7.99 = 47.94
Le nouveau prix total de la commande est donc : 39.98+47.94 = 87.92

Contenu de la table Commande :

idCommande	idClient	numTransaction	dateCommande	dateExpedition	dateReception	statutCommande	adresseLivraison	typePaiement	prixTotal
103	4	10169999999	2024-11-10 09:15:22	2024-11-11 08:45:00	2024-11-13 14:00:00	livree	IUT Blagnac	CB	87.92

Trigger update quantité en stock d’un produit / vérifiant qu’il reste du stock

1) Code

```
DELIMITER $$

CREATE TRIGGER t_ai_stock_reservation
AFTER INSERT ON LigneCommande
FOR EACH ROW
BEGIN
   -- Déclaration des variables
   DECLARE qteStockProduit INT;
   DECLARE msgErreur VARCHAR(255);
   -- Récupération sécurisée du stock du produit
   SELECT qteStock INTO qteStockProduit
   FROM DeclinaisonProduit
   WHERE idDeclinaison = NEW.idDeclinaison;

   -- Vérification si la quantité demandée dépasse le stock disponible
   IF NEW.quantite > qteStockProduit THEN
       -- Création du message d'erreur
       SET msgErreur = CONCAT(
           'Impossible de retirer la quantité ',
           NEW.quantite,
           ' du produit ',
           NEW.idDeclinaison
       );
       --Exception
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msgErreur;
   END IF;
   -- Mise à jour du stock si la quantité est disponible
   UPDATE DeclinaisonProduit
   SET qteStock = qteStock - NEW.quantite
   WHERE idDeclinaison = NEW.idDeclinaison;
END $$
DELIMITER ;
```

2) Test et trace d'exécution

Création d'une commande :

```
INSERT INTO `commande` (`idClient`, `numTransaction`, `dateCommande`,
`dateExpedition`, `dateReception`, `statutCommande`, `adresseLivraison`,
`typePaiement`) VALUES ('4', '10169999999', '2024-11-10 09:15:22',
'2024-11-11 08:45:00', '2024-11-13 14:00:00', 'livree', 'IUT Blagnac',
'CB');
```

idCommande	idClient	numTransaction	dateCommande	dateExpedition	dateReception	statutCommande	adresseLivraison	typePaiement	prixTotal
1001	4	10169999999	2024-11-10 09:15:22	2024-11-11 08:45:00	2024-11-13 14:00:00	livree	IUT Blagnac	CB	*NULL*

On va commander ce produit n'ayant plus que 10 unités en stock

idDeclinaison	libelleDeclinaison	idProduit	descDeclinaison	ficheTechnique	prix	prixSolde	qteStock	seuilQte	qualite	provenance	moyenneAvis ▲ 1
2	Potion de mana mineure	1	Régénère une petite quantité de mana (20%)	Contenance: 50ml, Effet: Instantané, Poids: 0.1kg	7.99	*NULL*	10	100	Commun	Falconia	*NULL*

Ajout d'une ligne commande qui veut prendre plus de stock que ce qui est disponible réellement (11>10) :

```
INSERT INTO `lignecommande` (`idCommande`, `idDeclinaison`, `quantite`)
VALUES ('1001', '2', '11');
```

Erreur obtenue :

Erreur

Requête SQL : Copier

```
INSERT INTO `lignecommande` (`idCommande`, `idDeclinaison`, `quantite`) VALUES ('1001', '2', '11');
```

MySQL a répondu :

```
#1644 - Impossible de retirer la quantité 11 du produit 2
```

Ajout d'une ligne commande qui veut prendre autant de stock que ce qui est disponible réellement (10 = 10) :

```
INSERT INTO `lignecommande` (`idCommande`, `idDeclinaison`, `quantite`)
VALUES ('1001', '2', '10');
```

La ligne commande est bien insérée sans erreur.

idCommande	idDeclinaison	quantite	prixTotal
1001	2	10	79.90

La quantité en stock du produit a également été mise à jour dans la table DeclinaisonProduit (10-10 = 0)

idDeclinaison	libelleDeclinaison	idProduit	descDeclinaison	ficheTechnique	prix	prixSolde	qteStock
2	Potion de mana mineure	1	Régénère une petite quantité de mana (20%)	Contenance: 50ml, Effet: Instantané, Poids: 0.1kg	7.99	*NULL*	0

Trigger vérifiant si un avis peut être déposé par un client sur un produit

1) Code

```
DELIMITER $$
CREATE TRIGGER t_bi_avis
BEFORE INSERT ON Avis
FOR EACH ROW
BEGIN
    -- On utilise la fonction stockée isProduitAvisPossible()
    IF NOT isProduitAvisPossible(NEW.idClient, NEW.idDeclinaisonProduit) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Avis impossible : produit non acheté ou avis déjà
déposé.';
    END IF;
END$$
DELIMITER ;
```

Voir plus bas la description de la fonction stockée isProduitAvisPossible()

2) Test et trace d'exécution

On insère un client n'ayant acheté aucun produit

```
INSERT INTO `client` (`nom`, `prenom`, `email`, `motDePasse`, `adresse`,
`codePostal`, `ville`, `telephone`)
VALUES ('ROMA', 'Quentin', 'test@mail.fr',
'$2y$10$bcdefghijklmnopqrstuvw', '45 avenue des Champs-Élysées'
, '75008', 'Paris', '0156789012';
```

idClient	nom	prenom	email	motDePasse	adresse	codePostal	ville	telephone	ptsFidelite
54	ROMA	Quentin	test@mail.fr	$2y$10$bcdefghijklmnopqrstuvw	45 avenue des Champs-Élysées	75008	Paris	0156789012	0

On insère un avis pour ce client sur un produit qu'il n'a pas acheté

```
INSERT INTO `avis` (`idClient`, `idDeclinaisonProduit`, `note`, `commentaire`,
`dateAvis`)
 VALUES ('54', '1', '5', 'Voici mon avis interdit !', '2025-12-03 08:55:46')
```

Erreur obtenue :

Erreur

Requête SQL : Copier

```
INSERT INTO `avis` (`idClient`, `idDeclinaisonProduit`, `note`, `commentaire`, `dateAvis`) VALUES ('54', '1', '5', 'Voici mon avis interdit !', '2025-12-03 08:55:46');
```

MySQL a répondu :

#1644 - Avis impossible : produit non acheté ou avis déjà déposé.

On ajoute le produit 1 dans une commande du client

Création de la commande :

```
INSERT INTO `commande` (`idClient`, `numTransaction`, `dateCommande`,
`dateExpedition`,
`dateReception`, `statutCommande`, `adresseLivraison`, `typePaiement`)
VALUES ('54', '101628031772131660', '2024-11-15 11:20:10', '2024-11-16
09:00:00',
'2024-11-18 10:30:00', 'livree', '89 avenue de Versailles, 92130
Issy-les-Moulineaux',
'Virement')
```

idCommande	idClient	numTransaction	dateCommande	dateExpedition	dateReception	statutCommande	adresseLivraison	typePaiement	prixTotal
1002	54	101628031772131660	2024-11-15 11:20:10	2024-11-16 09:00:00	2024-11-18 10:30:00	livree	89 avenue de Versailles, 92130 Issy-les-Moulineaux	Virement	*NULL*

Ajout du produit 1 dans la commande :

```
INSERT INTO `lignecommande`
(`idCommande`, `idDeclinaison`, `quantite`)
VALUES ('1002', '1', '2')
```

idCommande	idDeclinaison	quantite	prixTotal
1002	1	2	39.98

On réinsère l'avis sur le produit 1, maintenant le client l'a acheté

```
INSERT INTO `avis` (`idClient`, `idDeclinaisonProduit`, `note`, `commentaire`,
`dateAvis`)
 VALUES ('54', '1', '5', 'Voici mon avis interdit !', '2025-12-03 08:55:46')
```

idAvis	idClient	note	idDeclinaisonProduit	commentaire	dateAvis
25	54	5	1	Voici mon avis interdit !	2025-12-03 08:55:46

On essaie de déposer l'avis une deuxième fois, cela est impossible

```
INSERT INTO `avis` (`idClient`, `idDeclinaisonProduit`, `note`, `commentaire`,
`dateAvis`)
 VALUES ('54', '1', '5', 'Voici mon avis interdit !', '2025-12-03 08:55:46')
```

Erreur

Requête SQL : Copier

```
INSERT INTO `avis` (`idClient`, `idDeclinaisonProduit`, `note`, `commentaire`, `dateAvis`)
 VALUES ('54', '1', '5', 'Voici mon avis interdit !', '2025-12-03 08:55:46');
```

MySQL a répondu :

```
#1644 - Avis impossible : produit non acheté ou avis déjà déposé.
```

Trigger vérifiant qu’un produit ne peut pas être apparenté à lui-même et qu’un lien entre deux produits n’est pas déjà existant

1) Code

```
DELIMITER $$

CREATE TRIGGER t_bi_apparenter_contrainte
BEFORE INSERT ON Apparenter
FOR EACH ROW
BEGIN
    DECLARE vCountLiensExistant INT;
    DECLARE msgErreur VARCHAR(255);

    -- Auto-lien interdit
    IF NEW.idProduit1 = NEW.idProduit2 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Un produit ne peut pas s''apparenter lui-même';
    END IF;
    -- Vérifie si le lien existe déjà dans un sens OU dans l’autre
    SELECT COUNT(*) INTO vCountLiensExistant
    FROM Apparenter
    WHERE (idProduit1 = NEW.idProduit1 AND idProduit2 = NEW.idProduit2)
       OR (idProduit1 = NEW.idProduit2 AND idProduit2 = NEW.idProduit1);

    -- Si le lien existe déjà
    IF vCountLiensExistant > 0 THEN
        SET msgErreur = CONCAT(
            'Lien entre ',
            NEW.idProduit1,
            ' et ',
            NEW.idProduit2,
            ' déjà existant (direct ou inverse)'
        );
        -- Lancer une exception
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msgErreur;
    END IF;
END $$
DELIMITER ;
```

2) Test et trace d'exécution

Insertion d'un produit s'apparentant à lui-même

```
INSERT INTO `apparenter` (`idProduit1`, `idProduit2`) VALUES ('1', '1') ;
```

Erreur

Requête SQL : Copier

```
INSERT INTO `apparenter` (`idProduit1`, `idProduit2`) VALUES ('1', '1');
```

MySQL a répondu :

```
#1644 - Un produit ne peut pas s'apparenter lui-même
```

Insertion d'un produit déjà apparenté à un autre

	idProduit1	idProduit2
Éditer Copier Supprimer	1	2

```
INSERT INTO `apparenter` (`idProduit1`, `idProduit2`) VALUES ('1', '2') ;
```

MySQL a répondu :

```
#1644 - Lien entre 1 et 2 déjà existant (direct ou inverse)
```

```
INSERT INTO `apparenter` (`idProduit1`, `idProduit2`) VALUES ('2', '1') ;
```

MySQL a répondu :

```
#1644 - Lien entre 2 et 1 déjà existant (direct ou inverse)
```

Insertion d'un produit déjà apparenté à aucun autre produit

1 ligne insérée. (traitement en 0,0003 seconde(s).)

```
INSERT INTO `apparenter` (`idProduit1`, `idProduit2`) VALUES ('1', '50');
```

3)Procédures / fonctions stockées

Procédure permettant de retirer les stocks des produits étant dans une Composition

1) Code

```
DELIMITER $$
-- Permet de mettre à jour les stocks des produits présents dans une
composition
CREATE PROCEDURE majStockCompositionProduit(
    IN pIdCompo INT, -- L'identifiant de la composition en elle meme
    IN pQteDemandee INT -- Le nombre de fois que cette composition doit être
achetée
)
BEGIN
    DECLARE done INT DEFAULT FALSE; -- Savoir si le curseur a fini de
parcourir
    DECLARE vIdDeclinaison INT; -- Id d'un produit dans la composition -- Ex :
l'id d'une chaise
    DECLARE vQteNecessaire INT; -- Quantité du produit dans la composition (ex
: 2 chaises dans la compo table + chaises)
    DECLARE vStockActuel INT; -- Stock actuel du produit
    DECLARE msg VARCHAR(255); -- Pour le message d'erreur si exception

    -- Curseur pour parcourir tous les produits étant présents dans une
composition
    DECLARE comp_cursor CURSOR FOR
        SELECT idDeclinaison, quantiteProduit
        FROM CompositionProduit
        WHERE idCompo = pIdCompo;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;


    -- Transaction pour sécuriser la gestion du stock
    START TRANSACTION;

    OPEN comp_cursor;
```

```
    -- Pour chaque déclinaison produit présent dans la composition, on
effectue les actions
    -- suivantes
    check_loop: LOOP
        FETCH comp_cursor INTO vIdDeclinaison, vQteNecessaire;
        IF done THEN LEAVE check_loop; END IF;

        -- Stock actuel de la déclinaison dans la variable qteStock
        SELECT qteStock INTO vStockActuel
        FROM DeclinaisonProduit
        WHERE idDeclinaison = vIdDeclinaison
        FOR UPDATE; -- Sécurise le stock en concurrentiel

        -- Vérification du stock nécessaire
        -- On regarde si il y a assez de stock pour acheter la qte demandée de
compo
        IF vStockActuel < (vQteNecessaire * pQteDemandee) THEN
            ROLLBACK;
            SET msg = CONCAT(
                'Stock insuffisant pour la déclinaison ',
                vIdDeclinaison
            );
            -- Lancer une nouvelle exception
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = msg;

        END IF;
    END LOOP;

    CLOSE comp_cursor;

    -- RESET du curseur
    SET done = FALSE;
    OPEN comp_cursor;

    -- Décrémentation des stocks maintenant qu’on est sûr qu'il y a la bonne
qte pour tout
    update_loop: LOOP
        FETCH comp_cursor INTO vIdDeclinaison, vQteNecessaire;
        IF done THEN LEAVE update_loop; END IF;

        -- On retire le nb de produits qu'on a besoin pour acheter cette compo
```

```
        UPDATE DeclinaisonProduit
        SET qteStock = qteStock - (vQteNecessaire * pQteDemandee)
        WHERE idDeclinaison = vIdDeclinaison;
    END LOOP;

    CLOSE comp_cursor;

    COMMIT;
END$$

DELIMITER ;
```

2) Test et trace d'exécution

Exemple sur la composition 75

idCompo	idDeclinaison	quantiteProduit
75	25	1
75	26	1
75	60	1
75	64	3

Celle-ci est composée de :

- Produit 25 : x1
- Produit 26 : x1
- Produit 60 : x1
- Produit 64 : x3

Les produits cités sont en stock :

idDeclinaison	libelleDeclinaison	idProduit	descDeclinaison	ficheTechnique	prix	prixSolde	qteStock
25	Boule de feu	15	Le classique : lance une boule de feu	Dégâts: 50 Feu, Zone: 2m, Incantation: 1.5s	19.99	NULL	100
26	Chaos bouillonnant	15	Magie du chaos instable	Dégâts: 120 Feu/Ténèbres, Zone: 4m, Risque: Explos...	99.99	NULL	5

idDeclinaison	libelleDeclinaison	idProduit	descDeclinaison	ficheTechnique	prix	prixSolde	qteStock
60	Torche inépuisable	40	Brule éternellement sans se consumer	Durée: Infinie, Luminosité: Forte, Poids: 1kg	85.00	NULL	20

idDeclinaison	libelleDeclinaison	idProduit	descDeclinaison	ficheTechnique	prix	prixSolde	qteStock
64	Pain elfique	43	Une bouchée rassasie un homme	Valeur nutritionnelle: Haute, Conservation: 1 an, ...	30.00	NULL	50

Donc l'achat de la composition doit être possible.

On tente de simuler l'achat de la composition 75 (une seule)

```
CALL majStockCompositionProduit(75,1);
```

Mise à jour des quantité sur chaque produit étant dans la composition :

- Produit 25 : 100 - 1 = 99 restant
- Produit 26 :5 - 1 = 4 restant
- Produit 60 : 20 - 1 = 19 restant
- Produit 64 : 50 - 3 = 47 restant

idDeclinaison	libelleDeclinaison	idProduit	descDeclinaison	ficheTechnique	prix	prixSolde	qteStock
25	Boule de feu	15	Le classique : lance une boule de feu	Dégâts: 50 Feu, Zone: 2m, Incantation: 1.5s	19.99	NULL	99
26	Chaos bouillonnant	15	Magie du chaos instable	Dégâts: 120 Feu/Ténèbres, Zone: 4m, Risque: Explos...	99.99	NULL	4

idDeclinaison	libelleDeclinaison	idProduit	descDeclinaison	ficheTechnique	prix	prixSolde	qteStock
60	Torche inépuisable	40	Brule éternellement sans se consumer	Durée: Infinie, Luminosité: Forte, Poids: 1kg	85.00	NULL	19

idDeclinaison	libelleDeclinaison	idProduit	descDeclinaison	ficheTechnique	prix	prixSolde	qteStock
64	Pain elfique	43	Une bouchée rassasie un homme	Valeur nutritionnelle: Haute, Conservation: 1 an, ...	30.00	NULL	47

Donc la quantité des stocks de chaque produit contenus dans la composition a bien été mise à jour.

On tente de simuler l'achat de la composition 75 (une seule) en ayant un produit de la compo en rupture de stock
On met le produit 26 à 0 de stock :

idDeclinaison	libelleDeclinaison	idProduit	descDeclinaison	ficheTechnique	prix	prixSolde	qteStock
26	Chaos bouillonnant	15	Magie du chaos instable	Dégâts: 120 Feu/ Ténèbres, Zone: 4m, Risque: Explos...	99.99	NULL	0

Appel de la procédure stockée :

```
CALL majStockCompositionProduit(75,1);
```

Erreur

Requête SQL : Copier

```
CALL majStockCompositionProduit(75,1);
```

MySQL a répondu :

```
#1644 - Stock insuffisant pour la déclinaison 26
```

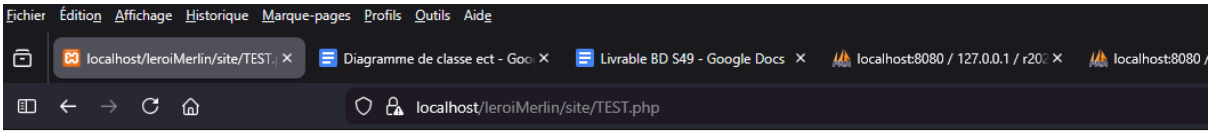
On obtient l'exception nous montrant qu'il est impossible **d'acheter la composition** car le produit 26 qui n'a plus de stock.

3) Test depuis php

Code exemple de mes divers tests suivants

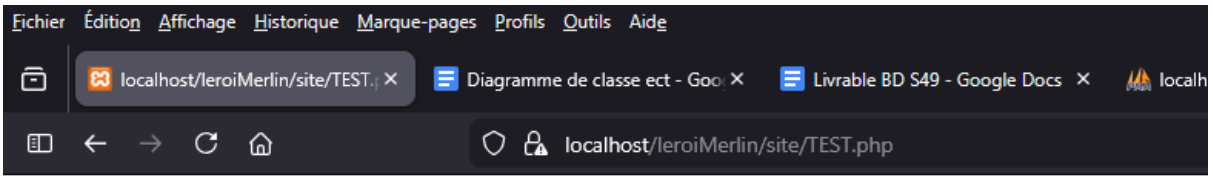
```php
<?php

require "./connec.inc.php";


try {

    $idCompo = 75;
    $qteDemandee = 1;

    // Appel de la procédure stockée
    $stmt = $connection->prepare("CALL majStockCompositionProduit(:idCompo,
:qteDemandee)");
    $stmt->bindParam(':idCompo', $idCompo, PDO::PARAM_INT);
    $stmt->bindParam(':qteDemandee', $qteDemandee, PDO::PARAM_INT);
    $stmt->execute();

    echo "Achat de la composition réussi !";

    $stmt->closeCursor();
} catch (PDOException $e) {
    // Récupère l'erreur levée par SIGNAL dans MySQL
    echo "Impossible d'acheter cette déclinaison, aucun stock dispo. : " .
$e->getMessage();
}

?>
```

Résultat obtenu avec 0 stock sur le produit 26:



Impossible d'acheter cette déclinaison, aucun stock dispo. : SQLSTATE[45000]: <>: 1644 Stock insuffisant pour la déclinaison 26

Résultat obtenu avec 10 de stock sur le produit 26 :



Achat de la composition réussi !

Procédure permettant de récupérer toutes les photos d’un produit

1) Code

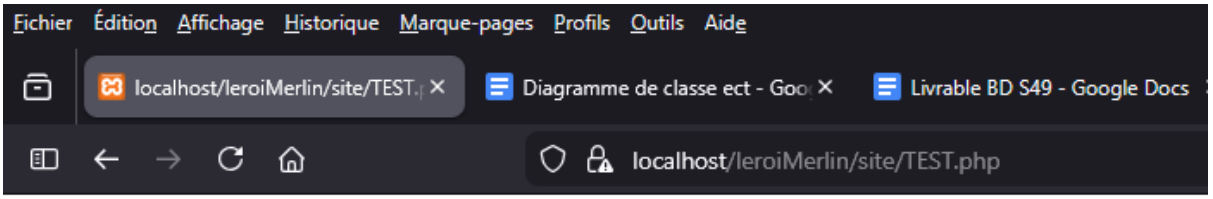
```
DELIMITER $$
CREATE PROCEDURE `getPhotosProduit`(IN `idDeclinaison` INT)
BEGIN
   DECLARE countDeclinaisons INT;
   DECLARE nbPhotos INT;

   SELECT COUNT(*) INTO countDeclinaisons FROM DeclinaisonProduit D
   WHERE idDeclinaison = D.idDeclinaison;

    -- Si la déclinaison n'existe pas, lever une exception
   IF countDeclinaisons = 0 THEN
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = 'Déclinaison innexistante !';
   END IF;

   -- Compter le nombre de photos pour cette déclinaison
   SELECT COUNT(*) INTO nbPhotos
   FROM PhotoProduit P
   WHERE idDeclinaison = P.idDeclinaison;

   -- Si aucune photo trouvée, lever une exception
   IF nbPhotos = 0 THEN
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = 'Aucune photo trouvée pour cette déclinaison';
   END IF;

   -- Récupérer les url des photos
   SELECT url
   FROM PhotoProduit P
   WHERE idDeclinaison = P.idDeclinaison
   ORDER BY idPhoto ASC;
END$$
DELIMITER ;
```

2) Test depuis php

Code exemple de mes divers tests suivants

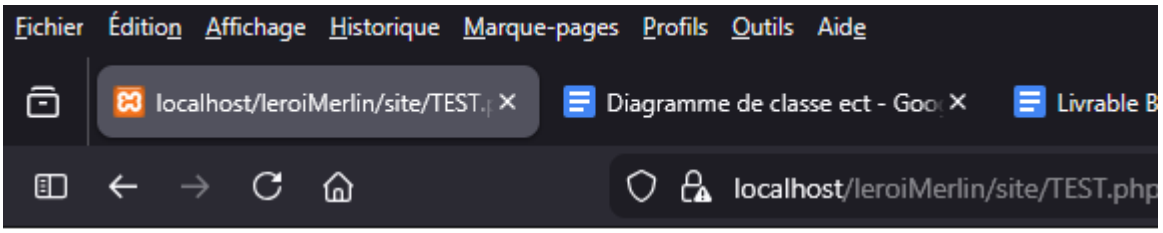
```php
<?php

require "./connec.inc.php";

$idDeclinaison = 42; // ID de la déclinaison à tester
$tab_photos = [];

try {
    // Appel de la procédure stockée
    $stmt = $connection->prepare("CALL getPhotosProduit(:idDeclinaison)");
    $stmt->bindParam(':idDeclinaison', $idDeclinaison, PDO::PARAM_INT);
    $stmt->execute();

    // Récupération des URLs retournées
    while ($ligne = $stmt->fetch(PDO::FETCH_ASSOC)) {
        $tab_photos[] = $ligne['url'];
    }

    $stmt->closeCursor();

    // Affichage du résultat
    echo "<pre>";
    var_dump($tab_photos);
    echo "</pre>";
} catch (PDOException $e) {
    // Gestion exception
    echo "Erreur : " . $e->getMessage() ." ".$idDeclinaison;
}
?>
```

Résultat obtenu avec l'id declinaison 42 qui ne possède aucune photo dans la BD



Erreur : SQLSTATE[45000]: <>: 1644 Aucune photo trouvée pour cette déclinaison 42

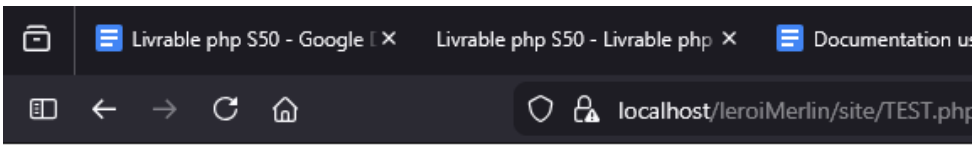
Résultat obtenu avec l'id declinaison 100 qui n'existe pas



Erreur : SQLSTATE[45000]: <>: 1644 Déclinaison innexistante ! 100

Résultat obtenu avec l'id declinaison 2 qui possède bien des photos enregistrées

idPhoto	url	idDeclinaison
1	images/manaMineure1.jpg	2
2	images/manaMineure2.jpg	2
3	images/manaMineure3.jpg	2
4	images/manaMineure4.jpg	2




```
array(4) {
  [0]=>
  string(23) "images/manaMineure1.jpg"
  [1]=>
  string(23) "images/manaMineure2.jpg"
  [2]=>
  string(23) "images/manaMineure3.jpg"
  [3]=>
  string(23) "images/manaMineure4.jpg"
}
```

Fonction permettant de savoir si un utilisateur à le droit de déposer un avis sur un produit

1) Code

```
DELIMITER $$
CREATE FUNCTION `isProduitAvisPossible`(`idClient` INT, `idDeclinaison` INT)
RETURNS tinyint(1)
BEGIN

DECLARE vNbAchete INT;
DECLARE vCountClient INT;
DECLARE vNbAvis INT;

-- On vérifie que le client existe
SELECT COUNT(*) INTO vCountClient FROM Client C WHERE C.idClient = idClient;
IF vCountClient = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Aucun client trouvé pour cet ID !';
    END IF;

-- On compte combien de fois il à été acheté
SELECT COUNT(*) INTO vNbAchete FROM Commande C, LigneCommande L
WHERE C.idClient = idClient AND L.idCommande = C.idCommande AND
L.idDeclinaison = idDeclinaison;

-- On compte le nombre de fois que le client a déposé un avis sur ce produit
SELECT COUNT(*) INTO vNbAvis FROM Avis A WHERE A.idClient = idClient AND
A.idDeclinaisonProduit = idDeclinaison;

-- Si le produit n'est pas acheté / un avis est déja déposé on return FALSE
IF vNbAchete = 0 OR vNbAvis != 0 THEN
    RETURN FALSE;
ELSE -- Sinon True, il a le droit de déposer un avis
    RETURN TRUE;
END IF;

END$$
DELIMITER ;
```

2) Test depuis php

Code exemple de mes divers tests suivants

```php
<?php

require "./connec.inc.php";
try {
    // Paramètres d'entrée
    $idClient = 2;
    $idDeclinaison = 15;

    // Appel de la fonction stockée dans MySQL
    $stmt = $connection->prepare("SELECT isProduitAvisPossible(:idClient,
:idDeclinaison) AS possible");
    $stmt->bindParam(':idClient', $idClient, PDO::PARAM_INT);
    $stmt->bindParam(':idDeclinaison', $idDeclinaison, PDO::PARAM_INT);
    $stmt->execute();

    // Récupération du résultat
    $resultat = $stmt->fetch(PDO::FETCH_ASSOC);
    $possible = $resultat['possible'];
    if ($possible) {
        echo "Le client ".$idClient." peut déposer un avis sur ce produit
".$idDeclinaison;
    } else {
        echo "Le client ".$idClient." ne peut pas déposer un avis sur ce
produit ".$idDeclinaison;
    }
    $stmt->closeCursor();
} catch (PDOException $e) {
    // Gestion des erreurs / exceptions
    echo "Erreur : " . $e->getMessage();
}
?>
```

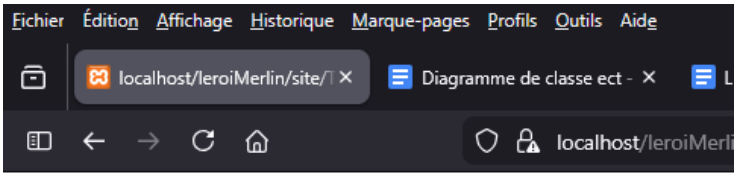
Test avec le client 2 qui essaie de déposer un avis sur un produit qu'il n'a jamais commandé :

On regarde quels produits ont été achetés par le client 2 :

```
SELECT C.idCommande, idDeclinaison FROM Commande C, LigneCommande L
WHERE L.idCommande = C.idCommande AND C.idClient = idClient AND
idClient = 2;
```

idCommande	idDeclinaison
2	7
2	52
2	55
2	62
2	82
61	7
61	55
61	82

Le produit 15 n'a été acheté par le client 2 donc il ne peut pas laisser un avis dessus. Voici le résultat obtenu dans php :



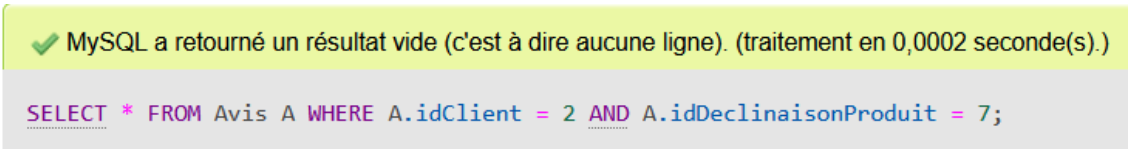
Le client 2 ne peut pas déposer un avis sur ce produit 15

Test avec le client 2 qui essaie de déposer un avis sur un produit qu'il a commandé (mais où il n'a pas encore mis d'avis):

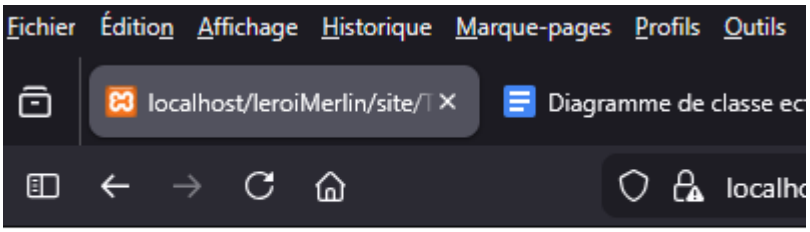
Comme vu sur la capture précédente, le client 2 a acheté le produit 7.

On vérifie qu'il n'a pas posé d'avis dessus encore :

```
SELECT * FROM Avis A WHERE A.idClient = 2
AND A.idDeclinaisonProduit = 7;
```



Résultat obtenu depuis php :



Le client a le droit de déposer un avis sur ce produit, il n'en a jamais déposé un.

Le client 2 peut déposer un avis sur ce produit 7

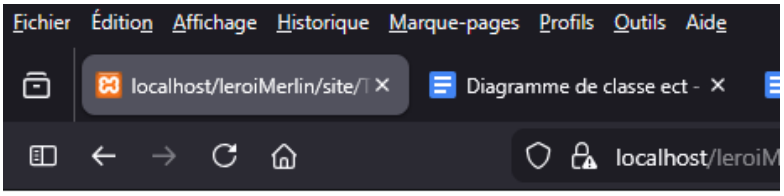
Test avec le client 2 qui essaie de déposer un avis sur un produit qu'il a commandé (et où il a déjà mis un avis):

On insère un avis du client 2 sur le produit 7 :

```
INSERT INTO `avis` (`idClient`, `idDeclinaisonProduit`, `note`, `commentaire`,
`dateAvis`)
VALUES ('2', '7', '5', 'Voici mon avis ', '2025-12-03 08:55:46')
```

idAvis	idClient	note	idDeclinaisonProduit	commentaire	dateAvis
26	2	5	7	Voici mon avis	2025-12-03 08:55:46

Résultat obtenu depuis php :

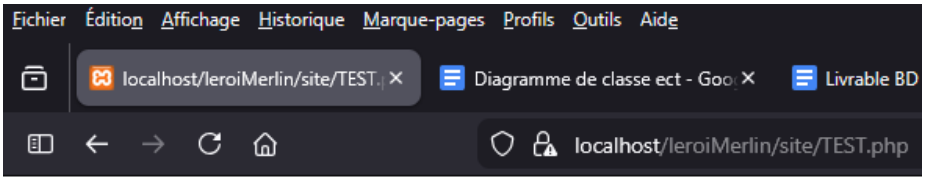


Le client n'a plus le droit de déposer un avis sur ce produit, il en a déjà déposé un.

Le client 2 ne peut pas déposer un avis sur ce produit 7

Test avec le client 9999 qui n'existe pas :

Résultat obtenu depuis php :



Erreur : SQLSTATE[45000]: <>: 1644 Aucun client trouvé pour cet ID !

Procédure permettant de recalculer les points de fidélité d'un client

1) Code

```
DELIMITER $$
CREATE PROCEDURE recalculerPointsFidelite(IN pIdClient INT, OUT totalPoints
DECIMAL(10,2))
BEGIN
    DECLARE fini INT DEFAULT FALSE;          -- Fin du curseur
    DECLARE vPrixTotal DECIMAL(10,2);        -- Valeur lue par le curseur
    DECLARE points INT(11) DEFAULT 0;  -- Total des points
       DECLARE nbClients INT;
    -- Curseur : commandes du client déjà payées / livrées
    DECLARE commandes_cursor CURSOR FOR
        SELECT prixTotal -- On récup le prix total de la commande
        FROM Commande
        WHERE idClient = pIdClient
       AND statutCommande IN ('payee', 'expediee', 'livree');
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET fini = TRUE; -- Fin du curseur

 -- Vérifier que le client existe
    SELECT COUNT(*) INTO nbClients
    FROM Client
    WHERE idClient = pIdClient;

    IF nbClients = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Client inexistant !';
    END IF;

       OPEN commandes_cursor;
    read_loop: LOOP
        FETCH commandes_cursor INTO vPrixTotal; -- Récupère une ligne
        IF fini THEN LEAVE read_loop; END IF;   -- Arret si plus de ligne
trouvée
        SET points = points + (vPrixTotal * 0.10); -- +10% en points
    END LOOP;
    CLOSE commandes_cursor;
    SET totalPoints = points; -- Valeur retournée
END$$
DELIMITER ;
```

2) Test depuis php

Code exemple de mes divers tests suivants

```php
<?php

require "./connec.inc.php";

try {
    // Paramètre d'entrée : ID du client
    $idClient = 2;

    // Appel de la procédure stockée avec param OUT
    $appelProcStock = 'CALL recalculerPointsFidelite(:idClient,
@totalPoints)';
    $stmt = $connection->prepare($appelProcStock);
    $stmt->bindParam(':idClient', $idClient, PDO::PARAM_INT);
    $stmt->execute();

    // Fermer le curseur pour récupérer la valeur OUT
    $stmt->closeCursor();

    // Récupération de la valeur OUT
    $resultat = $connection->prepare("SELECT @totalPoints AS points");
    $resultat->execute();
    $totalPoints = $resultat->fetch(PDO::FETCH_ASSOC);

    echo "Le total des points de fidélité du client $idClient est : " .
$totalPoints['points'];

} catch (PDOException $e) {
    // Gestion des exceptions
     echo "Erreur : " . $e->getMessage()." id client : ".$idClient;
}

?>
```

Test du calcul des points totaux d'un client existant

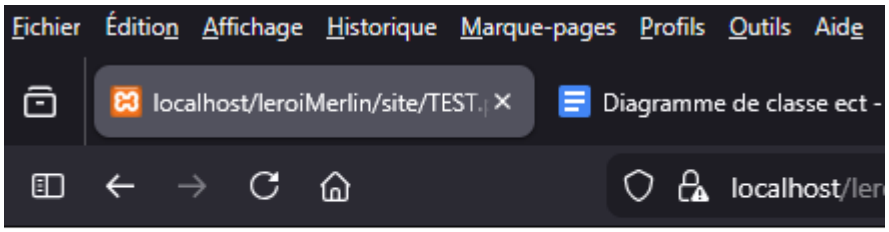
On récupère toute les commandes du client et on calcule les points de fidélité qu'il devrait avoir grâce aux prix totaux des commandes

```
SELECT idCommande, prixTotal FROM Commande WHERE idClient = 2;
```

idCommande	prixTotal
2	619.66
61	439.76

Donc **619.66 * 0.1 + 439.76*0.1 = 105,942** qui est arrondi à 106 points de fidelité.

Résultat obtenu dans php :



Le calcul des points de fidélité grâce à la procédure stockée est correct.

Le total des points de fidélité du client 2 est : 106.00

Test avec un client qui n'a jamais passé de commande

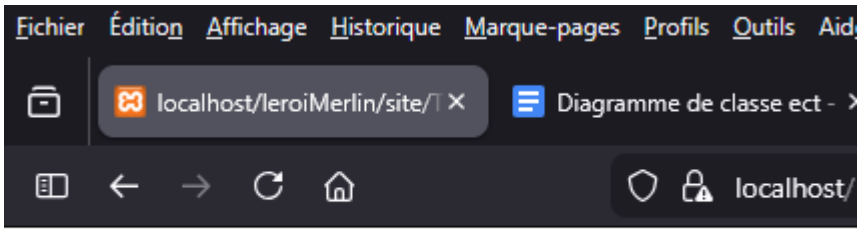
On récupère l'id d'un client n'ayant JAMAIS passé de commande :

```
SELECT Client.idClient
FROM Client
LEFT OUTER JOIN Commande
    ON Client.idClient = Commande.idClient
WHERE Commande.idClient IS NULL;
```

idClient
56

Dans php on teste la procédure avec le client 56.

Résultat obtenu dans php :



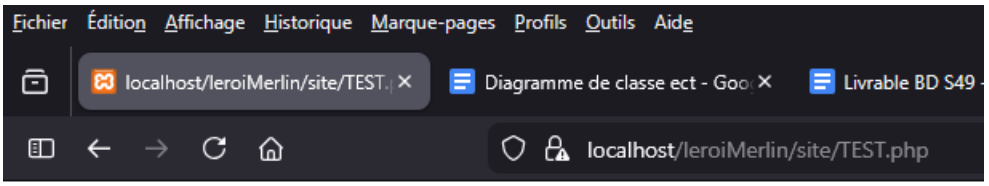
Le client 56 n'a jamais passé de commande donc il n'a aucun point de fidélité, le résultat de la procédure est cohérent.

Le total des points de fidélité du client 56 est : 0.00

Test avec un client inexistant

Le client 999999 n’existe pas.

Résultat dans php :



Erreur : SQLSTATE[45000]: <>: 1644 Client inexistant ! id client : 999999