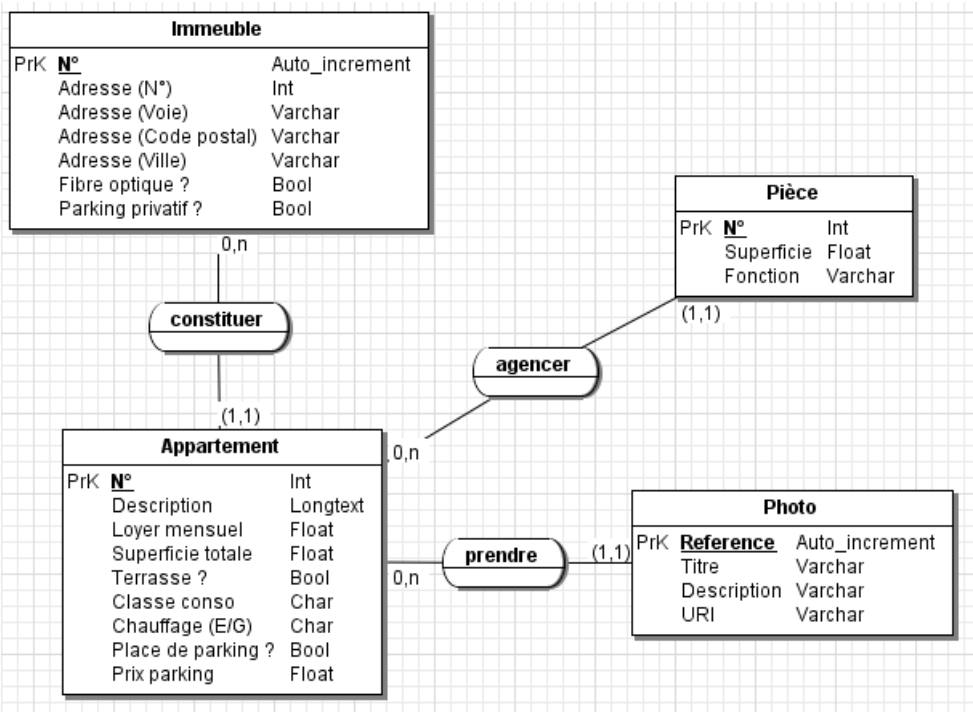
TP N°5 : SQL et triggers

Thème : à la découverte des triggers

Exercice 1 : parc immobilier (exemple)

Sujet :

Soit le modèle de données suivant :



Immeuble(id, adrNum, adrVoie, adrCodePostal, adrVille, fibreOptique, parkingPrivatif)
Clef primaire : id

Appartement(#immeuble, num, description, loyer, superficie, terrasse, classeConso, chauffage, placeParking, prixParking)
Clef primaire : immeuble, num
Clef étrangère : immeuble en référence à Immeuble(id)

Piece(#(immeuble, appartement), num, superficie, fonction)
Clef primaire : immeuble, appartement, num
Clefs étrangères : (immeuble, appartement) en référence à Appartement(immeuble, num)

Photo(num, titre, description, uri, #(immeuble, appartement))
Clef primaire : num
Clef étrangère : (immeuble, appartement) en référence à Appartement(immeuble, num)

En voici le script de création de tables :

```
CREATE TABLE Immeuble(
    Id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    adrNum VARCHAR(7) NOT NULL, adrVoie VARCHAR(100) NOT NULL,
    adrCodePostal VARCHAR(5) NOT NULL,  adrVille VARCHAR(30) NOT NULL,
    fibreOptique TINYINT  NOT NULL,  parkingPrivatif TINYINT  NOT NULL
);

CREATE TABLE Appartement(
    immeuble INT(11), num INT(3) NOT NULL, description LONGTEXT,
    loyer DOUBLE NOT NULL, superficie DOUBLE NOT NULL,
    terrasse TINYINT(1) NOT NULL, classeConso CHAR(1) NOT NULL, chauffage CHAR(1) NOT NULL,
    placeParking TINYINT(1) NOT NULL, prixParking DOUBLE,
    CONSTRAINT pk_appartement PRIMARY KEY (immeuble, num),
    CONSTRAINT fk_immeuble FOREIGN KEY (immeuble) REFERENCES Immeuble(id)
);

CREATE TABLE Photo(
    immeuble INT(11), appartement INT(3), reference INT(11) NOT NULL,
    titre VARCHAR(75), description VARCHAR(255), uri VARCHAR(120) NOT NULL,
    CONSTRAINT pk_photo PRIMARY KEY (immeuble, appartement, reference),
    CONSTRAINT fk_appartement_photo
        FOREIGN KEY (immeuble, appartement) REFERENCES Appartement(immeuble, num)
);

CREATE TABLE Piece(
    immeuble INT(11), appartement INT(3), num INT(2) NOT NULL,
    superficie DOUBLE ,
    fonction VARCHAR(30),
    CONSTRAINT pk_piece PRIMARY KEY (immeuble, appartement, num),
    CONSTRAINT fk_appartement_piece
        FOREIGN KEY (immeuble, appartement) REFERENCES Appartement(immeuble, num)
);
```

Créer ses tables dans une base de données sous MySQL.

Questions :

1. Rédiger le trigger permettant de vérifier la contrainte suivante : le prix de la place de parking d'un appartement peut et doit être NULL si l'appartement ne possède pas de place de parking. Tester le bon fonctionnement de votre trigger.

```
DROP TRIGGER IF EXISTS check_appartment;
DELIMITER $$
CREATE TRIGGER check_appartment_before_insert BEFORE INSERT ON appartement FOR EACH ROW
BEGIN
   IF NEW.placeParking = 0 THEN
      SET NEW.prixParking := NULL;
   END IF;
```

```
END $$
DELIMITER ; -- N.B. : le même trigger doit encore être ajouté « BEFORE UPDATE »
```

Alternative empêchant littéralement l'insertion (méthode plus classique) :

```
…
   IF NEW.placeParking = 0 THEN
      -- Interrompt la transaction en levant une erreur, i.e. empêche l'insertion (ou la modification)
      SIGNAL sqlstate '45000' SET message_text = 'Constraint violation: expected "placeParking"=0';
   END IF;
…
```

Tests unitaires :

```
-- Insertion d'un immeuble
INSERT INTO immeuble
    (Id, adrNum, adrVoie, adrCodePostal, adrVille, fibreOptique, parkingPrivatif)
VALUES
    (1, '7', 'Place de l\'étoile', '45000', 'ORLEANS', 1, 1);
-- Insertion de deux appartements, l'un avec une place de parking, l'autre sans place de parking, et
-- les deux ayant un prix de place de parking
-- La première insertion doit échouer, la seconde réussir.
INSERT INTO appartement
    (immeuble, num, description, loyer, superficie, terrasse, classeConso, chauffage, placeParking, prixParking)
VALUES
    (1, 1, 'Appartement 1.1', 825.00, 86, 1, 'C', 'E', 0, 75.00),
    (1, 2, 'Appartement 1.2', 750.00, 86, 1, 'C', 'E', 1, 80.00);
```

2. On souhaite que la contrainte suivante soit vérifiée : la superficie totale d'un appartement doit être égale à la somme de la superficie de chacune de ses pièces. Pour ce faire, créer le trigger qui permet de mettre à jour la superficie d'un appartement à l'insertion d'une pièce.

Version longue :

```
DELIMITER $$
CREATE TRIGGER maj_supercifie_appartment_before_insert AFTER INSERT ON piece FOR EACH ROW
BEGIN
    DECLARE superficieTotale INT;
    -- calcule la superficie totale de l'appartement
    SELECT sum(superficie) INTO superficieTotale
    FROM piece
    WHERE immeuble = NEW.immeuble
    AND appartement = NEW.appartement;
    -- met à jour la superficie de l'appartement
    UPDATE appartement
    SET superficie = superficieTotale
    WHERE immeuble = NEW.immeuble
    AND num = NEW.appartement;
```

```
END $$
DELIMITER ;
```

Notation allégée pour les variables :

```
DELIMITER $$
CREATE TRIGGER maj_supercifie_appartment_before_insert AFTER INSERT ON piece FOR EACH ROW
BEGIN
    SELECT sum(superficie) INTO @superficieTotale
    FROM piece
    WHERE immeuble = NEW.immeuble
    AND appartement = NEW.appartement;
    -- met à jour la superficie de l'appartement
    UPDATE appartement
    SET superficie = @superficieTotale
    WHERE immeuble = NEW.immeuble
    AND num = NEW.appartement;
END $$
DELIMITER ;
```

Version courte et soit dit en passant plus performante :

```
DELIMITER $$
CREATE TRIGGER maj_supercifie_apprt_before_insert BEFORE INSERT ON piece FOR EACH ROW
BEGIN
    -- on incrémente très simplement la superficie de l'appartement
   UPDATE appartement
   SET superficie = superficie + NEW.superficie
   WHERE immeuble = NEW.immeuble
   AND num = New.appartement;
END $$
DELIMITER ;
```

Test unitaire :

Insérer une pièce de superficie X et constater que la superficie de l'appartement à augmenter de X.

3. Adapter le trigger de la question 1 afin :
- de vérifier la contrainte suivante : un appartement ne peut avoir de place de parking si l'immeuble n'a pas de parking privatif ;
- d'initialiser la superficie de l'appartement à 0 à l'insertion d'un appartement ;
- d'empêcher la modification de la superficie d'un appartement en cas de mise à jour d'un appartement.

```
CREATE TRIGGER check_appartment_before_insert BEFORE INSERT ON appartement FOR EACH ROW
```

```
BEGIN
    -- 1er point :
    SELECT parkingPrivatif INTO @privatif
    FROM Immeuble
    WHERE id = New.immeuble ;
    IF @privatif = 0 THEN
        SET New.placeParking := 0
        SET New.PrixParking := NULL ;
    END IF ;
    -- 2ème point :
    SET New.superficie = 0 ;
END ;
-- 3ème point :
CREATE TRIGGER check_appt_before_update BEFORE INSERT ON appartement FOR EACH ROW
BEGIN
    IF New.Superficie != Old.Superficie THEN
        SIGNAL sqlstate '45000' SET message_text = 'Superficie non modifiable';
    END IF ;
END ;
```

4. En vous inspirant du trigger de la question 2, rédiger celui qui permet de mettre à jour la superficie d'un appartement à la mise à jour d'une pièce. Rédiger également le trigger qui met à jour la superficie d'un appartement à la suppression d'une pièce.

N.B. : si NEW permet de manipuler en lecture/écriture la ligne nouvellement insérée ou modifiée, OLD permet de manipuler la ligne avant modification ou suppression.

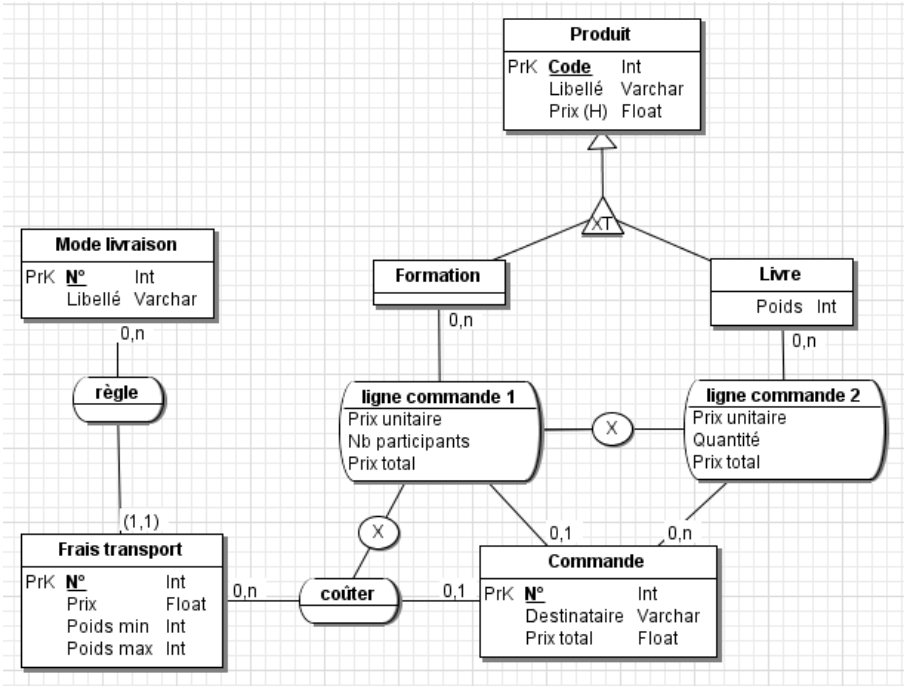
```
CREATE TRIGGER check_sup_piece_before_update BEFORE UPDATE ON Piece FOR EACH ROW
BEGIN
    UPDATE Appartement
    SET superficie = superficie + NEW.superficie - OLD.superficie
    WHERE num = Piece.appartement
    AND Appartement.immeuble = Piece.immeuble ;
END ;
```

```
CREATE TRIGGER check_sup_piece_after_delete AFTER DELETE ON Piece FOR EACH ROW
BEGIN
    UPDATE Appartement
    SET superficie = superficie - OLD.superficie
    WHERE num = Piece.appartement
    AND Appartement.immeuble = Piece.immeuble ;
END ;
```

Exercice 2 : des produits au prix variant au fil du temps

Sujet :

Soit le modèle de données suivant :



Produit(<u>code</u>, libelle, prix, formation, poids)
Clef primaire : code
Commentaire : le champ discrimination formation prend la valeur 1 (vrai) ou 0 (faux)
Mode_Livraison(<u>num</u>, libelle)
Clef primaire : num

Frais_Transport(#<u>livraison</u>, <u>num</u>, prix, poidsMin, poidsMax)
Clef primaire : livraison, num
Clefs étrangères : livraison en référence à Mode_Livraison(num)

Commande(<u>num</u>, destinataire, prixTotal, #(livraion, transport))
Clef primaire : num
Clef étrangère : (livraison, transport) en référence à Frais_Transport(livraison, num)

Ligne_Commande(#<u>commande</u>, #<u>produit</u>, prixu, quantite, prixt)
Clef primaire : commande, produit
Clef étrangères :
- commande en référence à Commande(num)
- produit en référence à Produit(code)

Questions :

1. Rédiger le script de création de table relatif à la base de données ci-avant décrite.

```
CREATE TABLE Produit(
    code VARCHAR(20) NOT NULL PRIMARY KEY,
    libelle VARCHAR(255) NOT NULL,
    formation TINIYINT NOT NULL ,
    poids INT(3)
);
-- Remarque : le champ formation est ici une propriété discriminante. Il permet de savoir si le produit est une livre
-- ou une formation : 0 (False) s'il s'agit d'une formation, 1 (True) s'il s'agit d'un livre.

CREATE TABLE Mode_Livraison(
    num INT(20) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    libelle VARCHAR(100) NOT NULL
);

CREATE TABLE Frais_Transport(
    livraison INT(20) NOT NULL,
    num INT(3) NOT NULL,
    prix DECIMAL(10, 2) NOT NULL,
    poidsMin INT(3) NOT NULL CHECK (poidsMin >= 0) ,
    poidsMax INT(3) NOT NULL CHECK (poidsMax >= poidsMin) ,
    CONSTRAINT pk_transport PRIMARY KEY (livraison, num),
    CONSTRAINT fk_livraison FOREIGN KEY (livraison) REFERENCES Mode_Livraison(num)
);

CREATE TABLE Commande (
    num INT(24) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    destinataire VARCHAR(100) NOT NULL,
    prixTotal DECIMAL(10, 2) NOT NULL CHECK (prixTotal >= 0) ,
    livraison INT(20) NULL,
    transport INT(3) NULL,
    CONSTRAINT fk_transport FOREIGN KEY (livraison, transport)
        REFERENCES Frais_Transport(livraison, num)
);

CREATE TABLE Ligne_Commande (
    commande INT(24) NOT NULL,
    produit VARCHAR(20) NOT NULL,
    prixu DECIMAL(10, 2) NOT NULL CHECK (prixu >= 0) ,
    quantite INT(3) NOT NULL CHECK (quantite >= 0) ,
    prixt DECIMAL(10, 2) NOT NULL CHECK (prixt >= 0) ,
    CONSTRAINT pk_ligne_commande PRIMARY KEY (commande, produit),
    CONSTRAINT fk_ligne_commande FOREIGN KEY (commande) REFERENCES Commande (num),
    CONSTRAINT fk_ligne_produit FOREIGN KEY (produit) REFERENCES Produit (code)
);
```

2. Rédiger le trigger permettant de vérifier la contrainte suivante : si une commande porte sur une formation, alors elle ne peut porter sur un ou plusieurs livres.

```
CREATE TRIGGER check_ligne_before_insert BEFORE INSERT ON Ligne_Commande FOR EACH ROW
BEGIN
   -- Teste si la commande comporte une formation
   SELECT COUNT(*) INTO @formations
   FROM Ligne_Commande AS L INNER JOIN Produit AS P ON L.produit = P.code
   WHERE commande = New.commande
   AND P.formation = 1;
   IF @formations > 0 THEN
      -- Si tel est le cas, on ne peut insérer de livre.
      -- (1) D'ailleurs, on devrait faire le "ROLLBACK" tout de suite car la commande ne doit
      -- alors comporter qu'une et une seule formation.
      SELECT P.formation INTO @formation
      FROM Produit AS P
      WHERE code = New.produit;
      IF @formation = 0 THEN
         SIGNAL sqlstate '45000' SET message_text = 'Si formation, alors pas de livre !!!';
      END IF ;
   END IF ;
END ;
```

3. Rédiger le trigger permettant de vérifier la contrainte réciproque : si une commande porte sur un ou plusieurs livres, alors elle ne peut porter sur une formation.

```
CREATE TRIGGER check_ligne_before_insert BEFORE INSERT ON Ligne_Commande FOR EACH ROW
BEGIN
   ...
   -- Teste si la commande porte sur au moins un livre
   SELECT COUNT(*) INTO @livres
   FROM Ligne_Commande AS L INNER JOIN Produit AS P ON L.produit = P.code
   WHERE commande = New.commande
   AND P.formation = 0;
   IF @livres > 0 THEN
      -- Si tel est le cas, on ne peut y insérer de formation
      SELECT P.formation INTO @formation
      FROM Produit AS P
      WHERE code = New.produit;
      IF @formation = 1 THEN
         SIGNAL sqlstate '45000' SET message_text = 'Si livre(s), alors pas de formation !!!';
      END IF ;
   END IF ;
END ;
```

4. Adapter le trigger de la question 2 ou 3 afin de vérifier la contrainte suivante : si la commande porte sur une formation, alors il ne doit pas y avoir de frais de transport.

```
CREATE TRIGGER check_commandef_before_insert BEFORE INSERT ON Commande FOR EACH ROW
BEGIN
   SELECT COUNT(*) INTO @formations
   FROM Ligne_Commande AS L INNER JOIN Produit AS P ON L.produit = P.code
   WHERE commande = New.commande
   AND P.formation = 1;
   IF @formations > 0 THEN
      IF NOT New.livraison IS NULL OR NOT New.transport IS NULL THEN
         SIGNAL sqlstate '45000' SET message_text = 'Si formation(s), alors pas de frais de trans. !!!';
      END IF ;
   END IF ;
END ;
```

5. Adapter le trigger de la question 2 ou 3 afin que le prix total d'une commande soit automatiquement mis à jour à lorsqu'une ligne de commande est insérée.

```
CREATE TRIGGER check_ligne_before_insert BEFORE INSERT ON Ligne_Commande FOR EACH ROW
BEGIN
   …
   -- Calcul du prix de la ligne
   New.prixt = New.prixu * New.quantite;
   -- Prix courant de la commande
   SELECT SUM(prixt) INTO @total
   FROM Ligne_Commande
   WHERE commande = New.commande;
   -- Frais de transport de la commande
   SELECT F.prix INTO @frais
   FROM Commande AS C
   INNER JOIN Frais_Transport AS F ON C.livraison = F.livraison AND C.transport = F.num
   WHERE C.num = New.Commande;
   -- Frais de transport de la commande
   UPDATE Commande
   SET prixTotal = New.prixt + @total + @frais
   WHERE num = New.commande ;
END ;
```

6. Préciser les cas que nous n'avons pas traités au travers des questions 1 à 5.

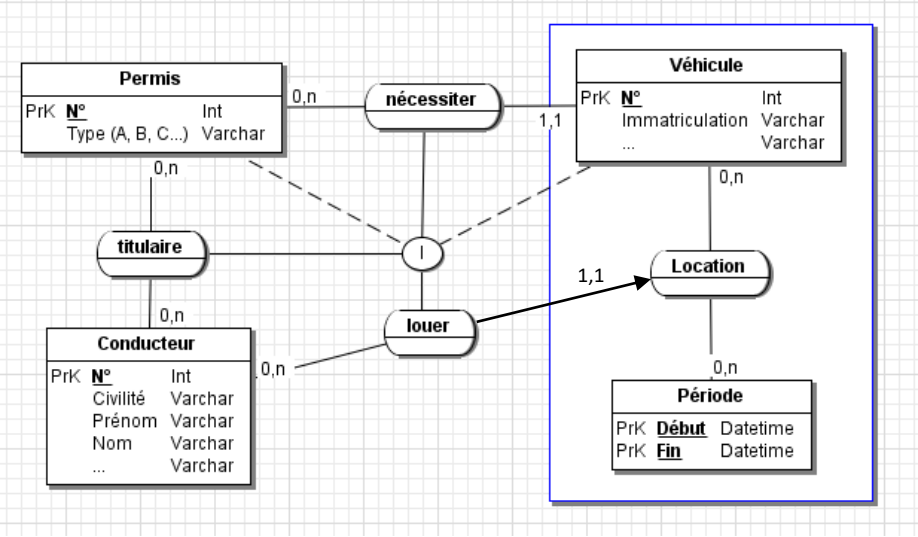
N'ont par exemple pas été gérés les cas suivants :

- Mise à jour du prix de la commande en cas de suppression d'une ligne de commande.
- Vérification de l'unicité de la formation en cas de commande de formation.

Exercice 3 : location de véhicules

Sujet :

Soit le modèle de données suivant :



Questions :

1. Créer sous MySQL une base de données correspondant au modèle de données ci-dessus (fournir le script de création de tables).

```
CREATE TABLE Permis(
    num INT(20) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    typee VARCHAR(3) NOT NULL
);
-- Attention ! Le nom de champ “type” est tentant mais “type” est malheureusement un mot-clef MySQL…

CREATE TABLE Conducteur(
    num INT(20) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    civilite VARCHAR(3) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    nom VARCHAR(50) NOT NULL,
    …
);

CREATE TABLE Titulaire(
    permis INT(20) NOT NULL FOREIGN KEY REFERENCES Permis(num),
    conducteur INT(20) NOT NULL FOREIGN KEY REFERENCES Conducteur(num)
);

CREATE TABLE Vehicule(
    num INT(20) NOT NULL AUTO_INCREMENT PRIMARY KEY,
```

```
    immatriculation VARCHAR(10) NOT NULL,
    …,
    -- Hypothèse : pas de véhicule sans permis
    permis INT(20) NOT NULL FOREIGN KEY (permis) REFERENCES Permis(num)
);

CREATE TABLE Location (
    num INT(20) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    vehicule INT(20) NOT NULL FOREIGN KEY REFERENCES Vehicule(num),
    conducteur INT(20) NOT NULL FOREIGN KEY REFERENCES Conducteur(num) ,
    debut DATETIME NOT NULL,
    fin DATETIME NOT NULL
);
-- Remarque : par commodité (valable et plus simple), on choisit une simple compteur comme clef primaire. En
-- pratique, on privilégiera les clefs simple plutôt que composées, quitte à rajouter des contraites unique/index.
-- Il est bien sûr valable (mais moins pratique) de choisir (vehicule, conducteur, debut) en clef primaire.
```

2. Rédiger le trigger qui permet, à l'insertion et à la modification d'une Location (INSERT ou UPDATE sur la table « Location »), de vérifier que le conducteur spécifié correspond à un conducteur ayant parmi ses permis celui requis pour conduire le véhicule.

```
CREATE TRIGGER check_premis_before_save BEFORE INSERT, UPDATE ON Location FOR EACH ROW
BEGIN
    -- Récupération du permis requis
    SELECT permis INTO @permis
    FROM Vehicule
    WHERE num = New.vehicule;
    -- Vérification d'existance côté conducteur
    SELECT COUNT(*) INTO @n
    FROM Titulaire
    WHERE num = New.conducteur
    AND permis = @permis;
    -- Si le conducteur n'a pas le bon permis, on l'arrête dans sa lancée…
    IF @n = 0 THEN
        SIGNAL sqlstate '45000' SET message_text = 'Pas de conducteur sans le permis approprié !!!';
    END IF ;
END ;
```