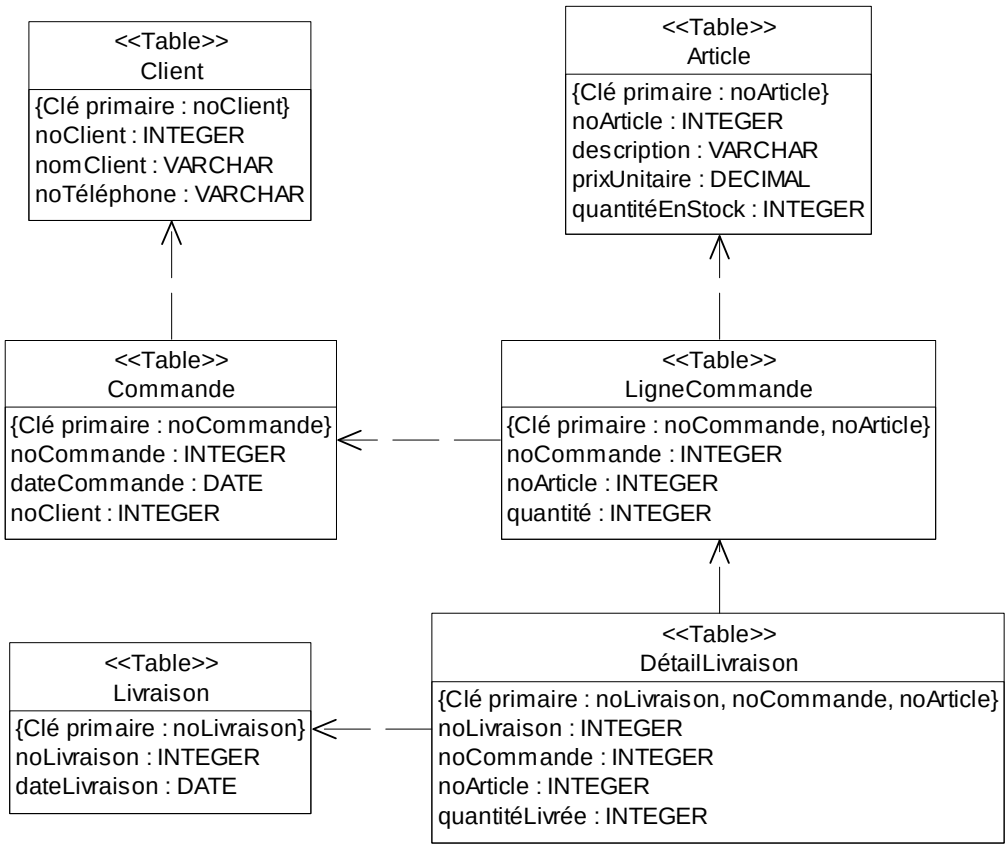
INF3180 : Fichiers et bases de données
Solutionnaire



1) Pour chacun des cas qui suit, déterminer comment faire respecter les contraintes d'intégrité en SQL pour le schéma *PleinDeFoin*. Donnez le code SQL correspondant. Supposez que la base de données ne contient pas encore de données.

a) La quantité commandée ne peut être supérieure à 5 pour les *Articles* dont le *noArticle* est supérieur à 10000.

b) Supposez qu'on ait ajouté à la table *Commande* une nouvelle colonne *totalCommande*. Le *totalCommande* doit être égal au total des montants de chacune des lignes de la commande. Le montant de chacune des lignes correspond à la *quantité* commandée multipliée par le *prixUnitaire* de l'*Article*. La modification des *LigneCommandes* et des *prixUnitaire* doit être interdite.

c) Le *prixUnitaire* d'un *Article* ne peut diminuer.

d) Supposez qu'on ait ajouté à la table *Commande* une nouvelle colonne *totalCommande* et à la table *LigneCommande* une nouvelle colonne *totalLigne*. Le *totalCommande* doit être égal au total des montants de chacune des lignes de la commande. Le *totalLigne* correspond à la *quantité* commandée multipliée par le *prixUnitaire* de l'*Article*.. La modification d'une *LigneCommande* est permise sauf lorsqu'il y a une ligne de *DétailLivraison* qui fait référence à *LigneCommande*. La modification du *prixUnitaire* est permise sous les mêmes conditions.

e) Pour la question précédente, concevez un TRIGGER qui modifie directement le *totalLigne* des *LigneCommande* ainsi que le *totalCommande* des *Commande* suite à la modification du *prixUnitaire* de l'*Article*.

f) Supposez qu'on ait ajouté à la table *LigneCommande* une nouvelle colonne *quantitéEnAttente*. La *quantitéEnAttente* d'une *LigneCommande* est égale à la *quantité* commandée moins le total des *quantitéLivrées* des *DétailLivraison* correspondant à la *LigneCommande*. La *quantitéEnAttente* doit être initialisée à la même valeur que la *quantité* commandée lors d'une insertion de *LigneCommande*. La *quantitéEnStock* doit être ajustée suite à l'insertion d'une ligne dans *DétailLivraison*.

Il est interdit d'insérer une ligne dans *DétailLivraison* si la *quantitéEnStock* de l'*Article* est insuffisante ou si la *quantitéLivrée* dépasse la *quantitéEnAttente* de la *LigneCommande*.

g) Il est interdit de supprimer une *Commande* s'il y a des *LigneCommandes* qui y font référence.

h) La *dateLivraison* ne peut précéder la *dateCommande*.

i) Une livraison ne touche toujours qu'un seul *Client*, c'est-à-dire ne peut être liée à des *Commandes* de plusieurs *Clients*.

Solutions des exercices du chapitre 6 selon la syntaxe du dialecte Oracle

1. a) Ajouter un CHECK sur la table LigneCommande

```
ALTER TABLE LigneCommande
ADD (CONSTRAINT XXX CHECK (noArticle <= 10000 OR quantité <= 5))
```

b)

```
CREATE TRIGGER initialiserTotalCommande
BEFORE INSERT ON Commande
FOR EACH ROW BEGIN
        :NEW.totalCommande:=0.0;
END;

CREATE TRIGGER modifierTotal
AFTER DELETE OR INSERT ON LigneCommande
FOR EACH ROW
DECLARE prix NUMBER(10,2);
BEGIN
        IF DELETING THEN
                LOCK TABLE Article IN ROW SHARE MODE;
                SELECT prixUnitaire INTO prix FROM Article
                        WHERE noArticle = :OLD.noArticle;
                UPDATE Commande
                SET totalCommande = totalCommande – :OLD.quantité*prix
                WHERE noCommande  = :OLD.noCommande ;
        END IF;
        IF INSERTING THEN
                LOCK TABLE Article IN ROW SHARE MODE;
                SELECT prixUnitaire INTO prix FROM Article
                        WHERE noArticle = :NEW.noArticle;
                UPDATE Commande
                SET totalCommande = totalCommande + :NEW.quantité*prix
                WHERE noCommande = :NEW.noCommande ;
        END IF;
END;

CREATE TRIGGER empecherModificationLignes
BEFORE UPDATE ON LigneCommande
FOR EACH ROW BEGIN
        raise_application_error(-20100, 'les lignes de commande ne peuvent être modifiées');
END;
CREATE TRIGGER empecherModificationPrix
BEFORE UPDATE OF prixUnitaire ON Article
FOR EACH ROW BEGIN
        raise_application_error(-20101, 'le prix ne peut être modifié');
END;
```

c)

```
CREATE OR REPLACE TRIGGER prixNePeutDiminuer
BEFORE UPDATE OF prixUnitaire ON Article
FOR EACH ROW
WHEN (OLD.prixUnitaire > NEW.prixUnitaire)
BEGIN
         raise_application_error(-20100, 'le prix d''un produit ne peut diminuer');
```

```
END;

d)
CREATE TRIGGER initialiser_total_commande
BEFORE INSERT ON COMMANDES
FOR EACH ROW BEGIN
        :NEW.total_commande:=0.0;
END;

CREATE OR REPLACE TRIGGER initialiserTotalLigne
BEFORE INSERT ON LigneCommande
FOR EACH ROW
DECLARE totalLigne NUMBER(10,2);
BEGIN
        LOCK TABLE Article IN ROW SHARE MODE;
        SELECT prixUnirtaire*:NEW.quantité INTO totalLigne
                FROM Article WHERE noArticle = :NEW.noArticle;
        :NEW.totalLigne:=totalLigne;
END;

CREATE OR REPLACE TRIGGER modifierTotalLigne
BEFORE UPDATE of noCommande, quantité, noArticle ON LigneCommmande
/*On ne peut intercepter les tentatives de modification de totalLigne car lorsque déclenché par
le TRIGGER aprèsModificationPrix, il est interdit ici d'accéder au prix en cours de modification!*/

FOR EACH ROW
DECLARE
        totalLigne NUMBER(10,2);
        nbLivraison INTEGER;
BEGIN
        LOCK TABLE DétailLivraison IN SHARE MODE;
        SELECT COUNT(*) INTO nbLivraison FROM DétailLivraison
        WHERE noCommande = :OLD.noCommande AND
                noArticle = :OLD.noArticle ;
        IF nbLivraison = 0 THEN
                LOCK TABLE Article IN ROW SHARE MODE;
                SELECT prixUnitaire*:NEW.quantité INTO totalLigne
                FROM Article WHERE noArticle = :NEW.noArticle;
                :NEW.totalLigne:=totalLigne;
        ELSE
                raise_application_error(-20100, 'il est interdit de modifier une ligne de commande
                                        lorsque des produits ont été livrés pour cette ligne');
        END IF;
END;

CREATE OR REPLACE TRIGGER aprèsMajLignes
AFTER DELETE OR INSERT OR UPDATE OF totalLigne, noCommande ON LigneCommande
FOR EACH ROW
BEGIN
        IF DELETING OR (UPDATING AND :OLD.noCommande != :NEW.noCommande) THEN
                UPDATE Commande
                SET totalCommande = totalCommande – :OLD.totalLigne
                WHERE noCommande  = :OLD.noCommande ;
        END IF;
        IF INSERTING OR (UPDATING AND :OLD.noCommande != :NEW.noCommande) THEN
                UPDATE COMMANDES
```

```
                SET totalCommande = totalCommande + :NEW.totalLigne
                WHERE noCommande = :NEW.noCommande ;
        END IF;
        IF (UPDATING AND  :OLD.noCommande = :NEW.noCommande AND :OLD.totalLigne !=
:NEW.totalLigne) THEN
                UPDATE Commande
                SET totalCommande = totalCommande – :OLD.totalLigne +:NEW.totalLigne
                WHERE noCommande  = :NEW.noCommande ;
        END IF;
END;

CREATE OR REPLACE TRIGGER aprèsModificationPrix
AFTER UPDATE OF prixUnitaire ON Article
FOR EACH ROW
DECLARE
BEGIN
                UPDATE LigneCommande
                SET totalLigne = quantité*:NEW.prixUnitaire
                WHERE  noArticle = :OLD.noArticle;
END;

e)
CREATE TRIGGER aprèsModificationPrix
AFTER UPDATE OF prixUnitaire ON Article
FOR EACH ROW
DECLARE
        vieuxTotal, NUMBER(10,2);
        nouveauTotal NUMBER(10,2);
        /* curseur avec un paramètre qui est le numéro d'article dont les lignes sont à modifier*/
        CURSOR ligneCursor (noA INTEGER) IS
                SELECT * FROM LigneCommande
                WHERE noArticle = noA
                FOR UPDATE OF totalLigne;

BEGIN
        FOR uneLigne IN ligneCursor(:OLD:noArticle) LOOP
                vieuxTotal:= uneLigne.totalLigne;
                nouveauTotal:=uneLigne.quantité*:NEW.prixUnitaire;
                UPDATE LigneCommande
                SET totalLigne = nouveauTotal
                WHERE  CURRENT OF ligneCursor;
                UPDATE Commande
                SET totalCommande = totalCommande + nouveauTotal-vieuxTotal
                WHERE noCommande = uneLigne.noCommande ;
        END LOOP;
END;

f)
CREATE OR REPLACE TRIGGER initQtéEnAttente
BEFORE INSERT ON LigneCommande
FOR EACH ROW BEGIN
        :NEW.quantitéEnAttente:=:NEW.quantité;
END;
```

Vérifier la quantité en stock et en attente

```
CREATE TRIGGER vérifierQuantitéEnStockEtEnAttente
BEFORE INSERT ON DétailLivraison
FOR EACH ROW
DECLARE
qtéStock, qtéDéjàLivrée, qtéCommandée NUMBER;
BEGIN
    SELECT      quantitéEnStock INTO qtéStock
    FROM        Article
    WHERE       noArticle = :NEW.noArticle
      FOR UPDATE OF quantitéEnStock;
    IF :NEW.quantitéLivrée > qtéStock THEN
      raise_application_error(-20100, 'quantité en stock insuffisante');
    END IF;

    LOCK TABLE DétailLivraison IN SHARE MODE;
    SELECT SUM(quantitéLivrée) INTO qtéDéjàLivrée
    FROM DétailLivraison
    WHERE noArticle = :NEW.noArticle AND
        noCommande = :NEW.noCommande;
    SELECT quantité INTO qtéCommandée
    FROM LigneCommande
    WHERE noArticle = :NEW.noArticle AND
        noCommande = :NEW.noCommande
        FOR UPDATE;

          IF :NEW.quantitéLivrée > qtéCommandée- qtéDéjàLivrée THEN
      raise_application_error(-20101, 'quantité livrée supérieure à quantité en attente');
    END IF;
END;


CREATE OR REPLACE TRIGGER ajusterQuantitéAttenteEtStock
AFTER INSERT ON DétailLivraison
FOR EACH ROW
DECLARE
BEGIN
  UPDATE LigneCommande
  SET
    quantitéEnAttente = quantitéEnAttente - :NEW.quantitéLivrée
  WHERE
          no_commande =:NEW.noCommande AND
          noArticle =:NEW.noArticle ;
  UPDATE Article
  SET
    quantitéEnStock = quantitéEnStock - :NEW.quantitéLivrée
  WHERE
   noArticle = :NEW.noArticle;
END;
```

g) Clause ON DELETE RESTRICT (NO ACTION) de la contrainte d'intégrité référentielle associée à la clé étrangère noCommande dans LigneCommande

h)

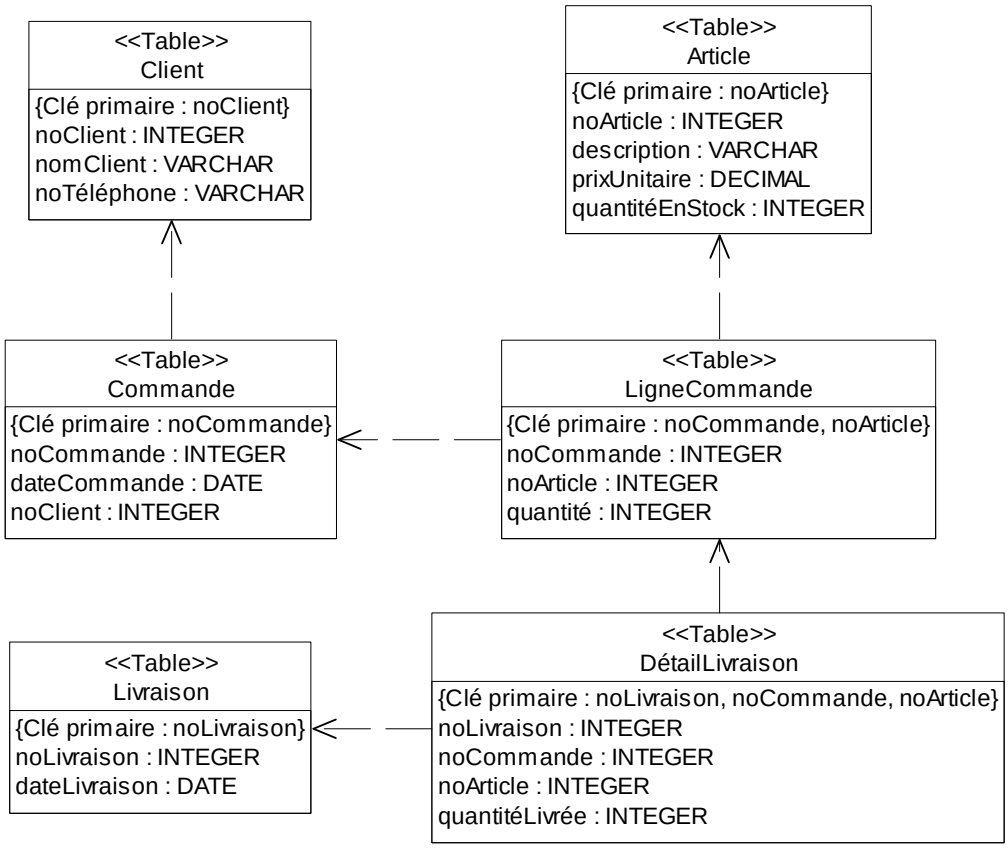
```
-- TRIGGER qui fait la vérification à l'insertion d'un DétailLivraison
-- NB il faut aussi vérifier les cas de modifications ...
```

```
CREATE OR REPLACE TRIGGER BIDetLivDateLivApresDatComm
BEFORE INSERT ON DétailLivraison
REFERENCING
	NEW AS ligneAprès
FOR EACH ROW
DECLARE
	laDateCommande			Commande.dateCommande%TYPE;
	laDateLivraison		Livraison.dateLivraison%TYPE;
BEGIN
	SELECT dateCommande INTO laDateCommande
	FROM Commande
	WHERE noCommande = :ligneAprès.noCommande;
	SELECT dateLivraison INTO laDateLivraison
	FROM Livraison
	WHERE noLivraison = :ligneAprès.noLivraison;
	IF laDateLivraison < laDateCommande THEN
		raise_application_error(-20100, 'la date de livraison ne peut précéder la date de la
commande');
	END IF;
END;

i)
-- On ne traite ici que le cas de l'insertion d'un DétailLivraison
CREATE OR REPLACE TRIGGER BIDetLivMemeClient
BEFORE INSERT ON DétailLivraison
REFERENCING
	OLD AS ligneAvant
	NEW AS ligneAprès
FOR EACH ROW
DECLARE
	leNouveauNoClient	INTEGER;
	leNoClient		INTEGER;
	CURSOR curseurLesNoClient(leNoLivraison DétailLivraison.noLivraison%TYPE)IS
		SELECT DISTINCT noClient
		FROM DétailLivraison D, Commande C
		WHERE C.noCommande = D.noCommande AND
			D.noLivraison = leNoLivraison;
BEGIN
	LOCK TABLE Commande IN SHARE MODE;
	LOCK TABLE DétailLivraison IN SHARE MODE;
	OPEN curseurLesNoClient(:ligneAprès.noLivraison);
	FETCH curseurLesNoClient INTO leNoClient;
	IF curseurLesNoClient%FOUND THEN
		CLOSE curseurLesNoClient;
		SELECT noClient
		INTO leNouveauNoClient
		FROM Commande C
		WHERE C.noCommande = :ligneAprès.noCommande;
		IF leNoClient <> leNouveauNoClient THEN
			raise_application_error(-20100, 'pas le même client pour les commandes');
		END IF;
	ELSE
		CLOSE curseurLesNoClient;
	END IF;
END;
```

Exercice supplémentaire

Codez en SQL la définition du schéma relationnel de l'application *VentesPleinDeFoin* représenté en UML dans la figure suivante. Incluez les définitions des clés primaire et étrangères.



Solution :

```
CREATE TABLE Client
(noClient         INTEGER          NOT NULL,
 nomClient        VARCHAR(20)      NOT NULL,
 noTéléphone      VARCHAR(15)      NOT NULL,
 PRIMARY KEY      (noClient)
)
CREATE TABLE Article
(noArticle        INTEGER          NOT NULL,
 description      VARCHAR(20)      NOT NULL,
 prixUnitaire     DECIMAL(10,2)    NOT NULL,
 quantitéEnStock  INTEGER          NOT NULL,
 PRIMARY KEY (noArticle))
CREATE TABLE Commande
(noCommande       INTEGER          NOT NULL,
 dateCommande     DATE             NOT NULL,
 noClient         INTEGER          NOT NULL,
 PRIMARY KEY      (noCommande),
 FOREIGN KEY      (noClient) REFERENCES Client
)
CREATE TABLE LigneCommande
(noCommande       INTEGER          NOT NULL,
 noArticle        INTEGER          NOT NULL,
 quantité         INTEGER          NOT NULL,
 PRIMARY KEY (noCommande, noArticle),
 FOREIGN KEY (noCommande) REFERENCES Commande,
 FOREIGN KEY (noArticle) REFERENCES Article
)
CREATE TABLE Livraison
(noLivraison      INTEGER          NOT NULL,
 dateLivraison    DATE             NOT NULL,
 PRIMARY KEY (noLivraison)
)
CREATE TABLE DétailLivraison
(noLivraison      INTEGER          NOT NULL,
 noCommande       INTEGER          NOT NULL,
 noArticle        INTEGER          NOT NULL,
 quantitéLivrée   INTEGER          NOT NULL,
 PRIMARY KEY (noLivraison, noCommande, noArticle),
 FOREIGN KEY (noLivraison) REFERENCES Livraison,
 FOREIGN KEY (noCommande, noArticle) REFERENCES LigneCommande
)
```