1. Vues

1.1 Créez une vue cli_view1 de la table client qui ne présente que le nom et l'adresse mail, tri par nom croissant

CREATE VIEW cli_view1 AS
SELECT clients.cli_nom, clients.cli_email FROM clients ORDER BY clients.cli_nom

1.2 Créez une autre vue cli_view2 identique à la précédente mais qui en plus renomme les colonnes d'origine

CREATE VIEW cli_view2 (nom, email) AS
SELECT clients.cli_nom, clients.cli_email FROM clients ORDER BY clients.cli_nom

1.3 Créez une vue de la table albums qui ne présente que le titre et le prix (en renommant les colonnes d'origine) des albums dont le prix est inférieur ou égal à 15 euros, tri par prix décroissant

CREATE VIEW alb_view1 (titre, prix) AS
SELECT albums.alb_titre, albums.alb_prix
FROM albums
WHERE albums.alb_prix <= 15
ORDER BY albums.alb_prix DESC, albums.alb_titre ASC

1.4 Créez une vue qui présente les informations suivantes :

- Nom de l'artiste
- Libellé complet du pays de l'artiste
- Libellé complet du genre de l'artiste
- Tri par nom d'artiste croissant
- Renommez les colonnes d'origine

CREATE VIEW art_view1(Nom, Pays, Genre) AS
SELECT artistes.art_nom, pays.pay_libelle, genres.gen_libelle
FROM artistes, pays, genres
WHERE artistes.pay_id = pays.pay_id
AND artistes.gen_id = genres.gen_id
ORDER BY artistes.art_nom ASC

Comment voir les vues ?

*SELECT * FROM information_schema.views*

2. Triggers

2.1 Ecrivez le trigger qui modifiera la colonne cli_ca de la table clients **à chaque création d'une nouvelle vente.**

```
DROP TRIGGER IF EXISTS upca;
DELIMITER //
CREATE TRIGGER upca
AFTER INSERT ON ventes
FOR EACH ROW
UPDATE clients
SET clients.cli_ca = clients.cli_ca + new.ven_prix
WHERE clients.cli_id = new.cli_id
//
DELIMITER ;
```

2.2 Ecrivez le trigger qui modifiera cette colonne à **chaque suppression d'une vente**.

```
DROP TRIGGER IF EXISTS upcadel;
DELIMITER //
CREATE TRIGGER upcadel
AFTER DELETE ON ventes
FOR EACH ROW
UPDATE clients
SET clients.cli_ca = clients.cli_ca - old.ven_prix
where clients.cli_id = old.cli_id
//
DELIMITER ;
```

2.3 Ecrivez le trigger qui modifiera cette colonne à **chaque modification d'une vente**.

```
DROP TRIGGER IF EXISTS upcamod;
DELIMITER //
CREATE TRIGGER upcamod
AFTER UPDATE ON ventes
FOR EACH ROW
UPDATE clients
SET clients.cli_ca = clients.cli_ca - old.ven_prix + new.ven_prix
WHERE clients.cli_id = new.cli_id
//
DELIMITER ;
```

Comment voir les triggers ?

```
SHOW TRIGGERS
```

Ou

```
SELECT trigger_schema, trigger_name, action_statement
FROM information_schema.triggers
```

Délimiteurs

Ce que l'on appelle délimiteur, c'est (par défaut), le caractère **;**

C'est le caractère qui permet de délimiter les instructions.

Il est possible de définir le délimiteur manuellement, pour que **;** ne signifie plus qu'une instruction se termine. Ainsi le caractère **;** pourra être utilisé à l'intérieur d'une instruction, et donc **dans le corps d'une procédure stockée.**

Pour changer le délimiteur, il suffit d'utiliser cette commande :
DELIMITER **|**

À partir de maintenant, vous devrez utiliser le caractère **|** pour signaler la fin d'une instruction.
; ne sera plus compris comme tel par votre session.

DELIMITER n'agit que **pour la session courante**.

Vous pouvez utiliser le ou les caractères de votre choix comme délimiteur, à condition de choisir quelque chose qui ne risque pas d'être utilisé dans une instruction. Il faut donc éviter les lettres, les chiffres, le @ qui sert pour les variables utilisateurs et le \ qui sert à échapper les caractères spéciaux.

Les deux délimiteurs suivants sont les plus couramment utilisés :
DELIMITER **//**
DELIMITER **|**