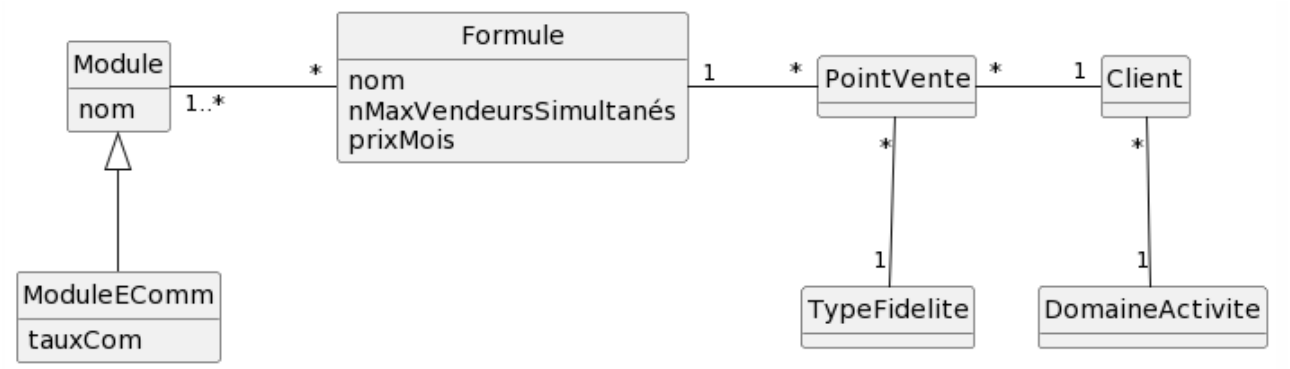
Question 1.1 (4 points)
A/ Quelles sont les risques de transmettre les identifiants de cette façon ?
B/ Proposez des solutions pour limiter les risques.

A/ Risque de vol de données de messagerie (hacker, personne indiscrète, ...)

B/ Une solution consiste à ce que le mot de passe transmis au client soit **transitoire** et **limité dans le temps**. Une fois le client connecté avec le mot de passe transmis en clair, l'application devra forcer le client à changer de lui-même son mot de passe, qu'il sera alors seul à connaître. Autre solution : transmettre un lien temporaire vers l'application, qui amène l'utilisateur à définir un mot de passe (action limitée dans le temps).

Question 1.2
Modifier le modèle d'analyse **ou** la structure de la base de données, afin de permettre la souscription en ligne du logiciel *WebCaisse*. (**12 points**)



TODO : 0..1 entre PV et formule (car un point de vente peut être en attente de choix de formule – ref annexe dans le sujet)
Remarque : Le montant de la réduction à appliquer en cas d'atteinte du seuil de « points de fidélité » est stocké dans PointVente.

Voir doc annexe pour une autre solution sans héritage et plus encore ainsi que le code source des diagrammes plantuml.

Question 1.3 (4 points)
indiquez quels sont les vérifications qu'il faudrait mener pour vérifier la sureté de ce code.

Ce code n'est sûr que si les variables utilisées (identifiant et mot de passe) ont été préparées pour leur usage (**filtrées, nettoyées afin de se prémunir d'une attaque par injection SQL**).

Il faut également vérifier la présence d'un **salage** par la fonction de hachage SHA256 pour plus de sécurité.

Utiliser la méthode statique Hash.SHA256("mot") produit un hash de "mot" en utilisant l'algorithme cryptographique SHA256. Le mot de passe clair ne peut pas être retrouvé à partir de l'empreinte

SHA256 stockée en base. Par contre si comme aucun salage n'est effectué on note 2 inconvénients majeurs :
2 comptes avec le même mot de passe clair ont la même empreinte donc en connaissant l'un on connaît l'autre.
Un hackeur, s'il récupère les empreintes peut utiliser des "rainbow tables" pour retrouver les mots de passe en clair.

Commentaire après analyse des réponses étudiants :
Les étudiants (majoritaires) qui ont grande difficulté à porter un raisonnement critique sur un code (document 3) . Voici un panorama de ce qu'ils ont proposé :

- de tester le code avec des identifiants (bon et mauvais) pour voir comment ce code se comporte
- de vérifier que le mot de passe de l'utilisateur est bien haché dans la base de données
- de vérifier que le hachage est bien réalisé
- de vérifier que le mot de passe est suffisamment long
- **de vérifier les capacités de ce code à résister aux injections SQL en testant des entrées SQL avec le formulaire (pas bête du tout, car on ne dit rien sur les fonctions utilisées !)**

Question 1.4 (4 points)
Expliquer en quoi la structure de la table ne permettra pas de gérer l'historique des formules souscrites, indispensable à la détermination du montant mensuel à régler par le client.

Principalement par le fait même de la nature composée de la clé primaire. La table est une **table de liaison** : elle ne peut mettre en relation qu'**une seule fois** un point de vente et une formule donnée. Le fait de noter que la table proposée ne contient pas la date à laquelle le changement de formule a eu lieu ne répond pas à la question de l'historicité.

Question 1.5 (4 points)
Proposer une correction de la structure de la table qui réponde au besoin exprimé.

FormuleSouscrite(idPointDevente, dateSouscription, idFormule)
clé primaire : idPointDeVente, dateSouscription
clé étrangère : idFormule en référence à id de Formule
idPointDeVente en référence à id de PointDeVente

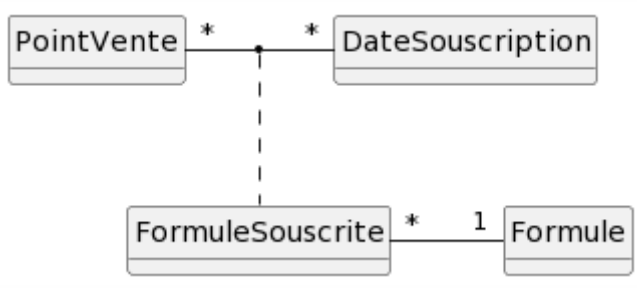
La contrainte « une seule formule par point de vente à un instant t» est gérée par la logique de l'application.

On acceptera également :
- au lieu de dateSouscription, une date de *début de souscription* et une date de *fin de souscription* dans la table **FormuleSouscrite,** la date de *début de souscription* faisant partie de la clé primaire de FormuleSouscrite ;

-

On acceptera la représentation de la solution avec un schéma entité-association ou un diagramme de classes.

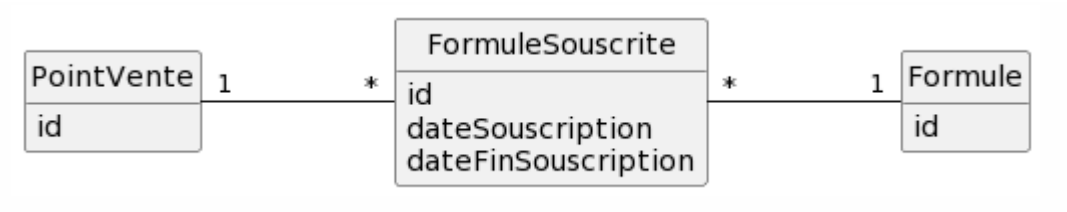
Exemple UML :



On acceptera la solution qui consiste à réifier FormuleSouscrite

FormuleSouscrite(id, idPointDevente, idFormule, dateSouscription, dateFinSouscription)

clé primaire : id,
clé étrangère : idFormule en référence à id de Formule
idPointDeVente en référence à id de PointDeVente



Mission 2

Mission 2 – Partie 1

Question 2.1 (10 points)
Écrire les requêtes permettant d'extraire les informations nécessaires de la base de données fournie dans le dossier documentaire.

a. la liste des consommateurs (nom, prénom, adresse de courriel) pour lesquels au moins une vente a été réalisée en 2017. **(3 points)**

```
SELECT distinct nom, prenom, mail
FROM Conso
JOIN Vente ON idConso = Conso.id
Where Year(dateVente) = 2017
```

On pourra trouver l'attribut « Conso.id » dans le SELECT pour garantir le distinct sur des homonymes dont la valeur du mail serait nulle.
On peut également accepter les solutions suivantes :

```
SELECT distinct nom, prenom, mail
FROM Conso, Vente
WHERE idConso = Conso.id AND Year(dateVente) = 2017
```

```
SELECT nom, prenom, mail
FROM Conso
Where id in ( select  distinct idConso from Vente
                Where Year(dateVente) = 2017)
```

On peut également accepter l'utilisation de EXISTS au lieu de in

b. le nombre de consommateurs ayant souscrit au programme de fidélité et appartenant à la tranche d'âge 18-30 ans. **(3 points)**

```
SELECT COUNT(*)  FROM ConsoFidele
WHERE TIMESTAMPDIFF(YEAR, dateNaiss , CURDATE()) BETWEEN 18 AND 30
```

On acceptera également la syntaxe `TIMESTAMPDIFF(YEAR, CURDATE(), dateNaiss)`

c. la liste des consommateurs (nom, prénom, adresse de courriel) avec le montant total des ventes réalisées pour chacun **(4 points)**

```
SELECT nom, prenom, mail, SUM(montantVente)
FROM Conso
JOIN Vente ON idConso = Conso.id
GROUP BY nom, prenom, mail
```

A la place de GROUP BY nom, prenom, mail on acceptera les GROUP BY suivants :

```
GROUP BY  Conso.id   (ou idConso)
GROUP BY mail
GROUP BY nom, prenom, mail, Conso.id   (ou idConso)
```

autre GROUP BY: pénalité de 1 point.

Question 2.2a

Expliquer le rôle du motif proposé par *Bob*.

L'expression régulière consiste à restreindre la donnée à :
01..09 ou 10..19 ou 20..29 ou 30 ou 31
suivi de /

Question 2.2b

Finaliser la conception de ce motif, selon le format de date attendu par le formulaire.

Date (du et au) : une date au format jj/mm/aa
Expression régulière : ^((0[1-9])|([12][0-9])|(3[01]))\/((0[1-9])|(1[012]))\/[0-9]{2}$
On peut accepter un modèle plus simple... : ^[0-9]{2}\/[0-9]{2}\/[0-9]{2}$

Question 2.2c

Donner, dans ce contexte, la limite de filtrage de cette expression régulière.

Ne tient pas compte des mois en 30 jours ou des années bissextiles

Remarque : Réponse d'un étudiant à la question 2.2c : (parlant d'une limite supérieure)
La limite de cette expression régulière (ou regex) est : 99/12/99 .

Mission 2 – Partie 2

Critique : l'attribut «double pointsFidélité » de la classe ConsoFidele est **maladroitement polymorphe** car, selon le type de fidélité retenue, il dénote soit un entier (un nombre de points cumulés), soit un réel (une accumulation de montants). Une ambiguïté qui peut nuire à la compréhension du sujet.

Question 2.3 (3 points)
Compléter la méthode *testInitConso* permettant de combler ce manque.

```
public void testInitConso() {
 ConsoFidele consoTest = new ConsoFidele("Lifo", "Paul", "lifo.paul@gmail.com",
"0600000000", new SimpleDateFormat("yyyy-MM-dd").parse("1961-01-03"),new
SimpleDateFormat("yyyy-MM-dd").parse("2017-01-05"));
assertEquals(0, consoTest.getPointsFidelite());
}
```

Question 2.4 (8 points)
Compléter la méthode *testAddMontant* permettant de valider les points de fidélité obtenus dans le cas d'un programme de fidélisation par points.

```
public void testAddMontant() {
            ConsoFidele consoTest = new ConsoFidele("Lifo", "Paul",
"lifo.paul@gmail.com",
        "0600000000", new SimpleDateFormat("yyyy-MM-dd").parse("1961-01-03"),
        new SimpleDateFormat("yyyy-MM-dd").parse("2017-01-05"));

      consoTest.addFidelite(3, 50);
      assertEquals("Erreur 1er achat à 50 euros (inférieur à 100 euros)",
            0, consoTest.getPointsFidelite());
      consoTest.addFidelite(3, 150);
      assertEquals("Erreur 2ème achat à 150 euros (entre 100 et 200 euros)",
                  10, consoTest.getPointsFidelite());
      consoTest.addFidelite(3, 250);
      assertEquals("Erreur 3ème achat à 250 euros (entre 200 et 500 euros)",
                  30, consoTest.getPointsFidelite());
      consoTest.addFidelite(3, 550);
      assertEquals("Erreur 4ème achat à 550 euros (supérieur à 500 euros)",
                  80, consoTest.getPointsFidelite());
}
```

On acceptera les tests effectués dans un ordre différent mais faisant apparaître une **valeur cumulée** cohérente des points de fidélité couvrant les cas nominaux (au minimum 3 tranches)

Mission 3

Question 3.1 (4 points)
Rédiger le commentaire de la méthode *statVente* de la classe Statistique expliquant ce qu'elle retourne.

La méthode retourne le pourcentage (ou taux) de ventes réalisées par des clients fidèles par rapport à un ensemble de ventes journalières fournies en paramètre.

Le terme "journalières" n'est pas exigé.

Question 3.2 (3 points)
Écrire la méthode *getNbVentes* de la classe Conso qui retourne le nombre de ventes enregistrées dans la collection des ventes du consommateur.

```
public int getNbVentes() {
        return lesVentes.size();
}
```

ou équivalent...

Question 3.3 (6 points)_
Écrire le constructeur de la classe VenteEcommerce qui permet d'initialiser tous les attributs d'une instance de la classe.

```
public VenteEcommerce (Date dateVente, Conso leConso, double montant,
                       String adresseLivr, String optionLivr) {
        super(dateVente, leConso, montant) ;
        this.adresseLivraison = adresseLivr ;
        this.optionLivraison = optionLivr ;
}
```

Au lieu de super, on acceptera toute autre syntaxe : base, parent, etc.

Question 3.4 (7 points)_
Compléter le code de la méthode *compareLieuVente.*

```
public static double compareLieuVente (ArrayList<ConsoFidele> lesConsommateurs)
{
  double totalEcom=0; //cumul des ventes ecommerce
  double totalMag=0;  // cumul des ventes en magasin
  for(ConsoFidele cf : lesConsososommateurs) //parcours de la liste
  {
    //parcours des ventes d'un consommateur
    for(Vente uneVente : cf.getVentes())
    {
        if (uneVente instanceof VenteEcommerce )
            // vente ecommerce
            // on ajoute le total de la vente au cumul ecommerce
            totalEcom += uneVente.getMontantVente();
        else if (uneVente instanceof VenteMagasin )
            // on ajoute le total de la vente au cumul magasin
            totalMag += uneVente.getMontantVente();
        }
    }
    return totalMag/totalEcom ; //calcul de l'indice et retour du résultat
}
```

On acceptera foreach au lieu de for même si le candidat répond en java, ou toute autre expression de boucle logique.
On acceptera un « else » sec (sans autre test) à la place du « else if »

Étant donné que le choix du langage est libre, à la place du test utilisant *instanceof* on pourra trouver un test réalisé sur la valeur ou présence d'un attribut d'une classe dérivée, ou tout autre façon permettant de déterminer le type précis de l'objet.

Question 3.5 (4 points)
3.5.a Expliquer en quoi la dernière instruction "**return totalMag/totalEcom**" de la méthode *compareLieuVente* peut poser un problème de robustesse

3.5.b Proposer une idée d'une solution possible corrigeant ce problème (sans l'implémenter).

3.5.a Si aucune vente eCommerce n'a été réalisée alors l'instruction provoquera une erreur de **division par zéro.**

3.5.b Une solution consiste à responsabiliser l'appelant en déclenchant une exception métier ou à changer l'interface de la fonction afin de retourner à l'appelant les deux valeurs de cumul, sans faire de division.

Question 3.6 (7 points)_
Ecrire la méthode répondant à ce besoin.

```
public ArrayList<Vente> getVentesSupMt (double montant)
{
        ArrayList<Vente> lesVentesSel ;
        lesVentesSel = new ArrayList<Vente>() ;
        for(Vente uneVente : lesVentes) //parcours de la liste
        {
                if (uneVente.getMontantVente() > montant)
                        //on ajoute la vente à la collection
                        lesVentesSel .add(uneVente) ;
                }
        }
        return lesVentesSel ;
}
```

On acceptera foreach au lieu de for même si le candidat répond en java, ou toute autre expression de boucle logique.

Mission 4 – Partie 1

Question 4.1 :
Où devrait être placé le SGBD pour un maximum de sécurité ? Justifiez.

La DMZ est accessible depuis l'internet. Le SGBD est donc accessible à d'éventuels hackeurs.

Une solution est de placer le SGBD dans le réseau LAN de l'entreprise et permettre au serveur Web d'accéder à cette seule machine dans le LAN.

(5 points)

Question 4.2 :
Critiquez le code et proposez les modifications qui permettraient d'améliorer la sécurité.

La directive ALL donne toutes les permissions au compte acces_webcaisse à la base webcaisse_db. Ce qui inclut les droits de création de base, modification de la structure, sauvegarde, etc.

Alors que seuls les permissions correspondant aux requêtes SQL SELECT, INSERT, UPDATE et DELETE devraient être autorisées.
Donc on pourrait plutôt écrire : GRANT SELECT, UPDATE, DELETE, INSERT ON webcaisse_db TO acces_webcaisse;

Mission 4 – Partie 2

<u>Question 4.3</u> (7 points)_
Rédiger une courte note à destination du chef de projet en justifiant le choix d'un hébergeur parmi les trois propositions.

Je vous transmets le comparatif des solutions d'hébergement et je conseille de choisir l'hébergeur ***A*** qui répond aux trois critères de *Nasdy*, à savoir :

- une sauvegarde en temps réel et une sauvegarde effectuée tous les lundis soirs
- une disponibilité de 99,9 %
- un chiffrement des échanges avec un certificat SSL

L'hébergeur ***B*** propose une sauvegarde journalière et la disponibilité mais pas de cryptage, mais le client peut le mettre en place car son serveur est dédié.

L'hébergeur ***C*** ne propose que la possibilité de faire une sauvegarde manuelle et la disponibilité avec une garantie du temps de disponibilité mais pas de cryptage.

1 point seulement si pas de justification.
(5 points)

DOC ANNEXES DU CORRIGÉ

Question 1.2

https://www.plantuml.com/plantuml/png/
RP5DQWCn38NtEeNmiZ21R4CNATDcDr1ewTxC22BmJx0rPKhQ0_KSlLZjHaCSg1S2l_SzoSB
xB3R9x1sCpkQCXvZyx0ZFW27wKWQxl56OQ4xvXOibDlnyPy1JucM8dE4BDDexfY1ulYhFaO
EKj51LLNpzED619t9SPRpIGvnqSXrTzHlZgO_U_xbYvwLN0ZHbjflDdS4EJQdh4Zk3DyUizFrdr
xLjOKsOhVPetxv3qxHeZCfrDQExA_alTk7QvK53ZVqOlUL03wFmUoME1SMHq-
W8BnTFSenJXZs5gVxQBm00

```
@startuml
class Formule {
 nom
 nMaxVendeursSimultanés
 prixMois
}

class Client {
}

class PointVente {
}
class TypeFidelite {
}

class Module {
  nom
}


class ModuleEComm {
  tauxCom
}



Module "1..*" - "*" Formule :"              "
Module <|-- ModuleEComm
Formule "1" - "*" PointVente : "         "
PointVente "*" - "1" Client : "      "
PointVente "*" -- "1" TypeFidelite
Client "*" -- "1" DomaineActivite
hide circle
hide methods
@enduml
```

Sans héritage
https://www.plantuml.com/plantuml/png/RT3FIiSm30VmUvyYv2ZucxiEXCbaDy63U2zjP8N-aJKLHNntMrAnWZqKzird8UcQMPzSWbV6wvnXJsSedk1JVQaURTvHv35vJ2xoSmt5IVZqyKgxi-HTY-5NFoGh7G5Y2lLcNTwt5E0MRc2QBBteuhdvBd6-NAuGhW7h_JFGWl3de59IQhxZSR05SB130QNrZ7qnaVypSUDcgXxvRnJtAMWNwSwmUsleg1gCEuqda8z0V2IRrKhHjjV-1W00

```
@startuml
class Formule {
}

class Client {
}

class PointVente {
}
class TypeFidelite {
}

class Module {
  nom
  tauxCom = 0 //default
}

Module "1..*" - "*" Formule :"              "

Formule "1" - "*" PointVente : "         "
PointVente "*" - "1" Client : "      "
PointVente "*" -- "1" TypeFidelite



Client "*" -- "1" DomaineActivite
hide circle
hide methods
@enduml
```