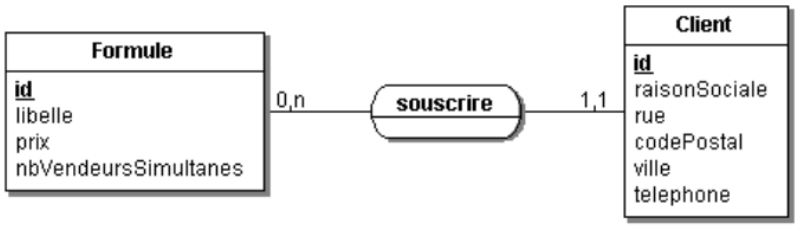
Mission 1 - Souscription en ligne

Partie 1 – Évolution de la base de données pour la souscription en ligne

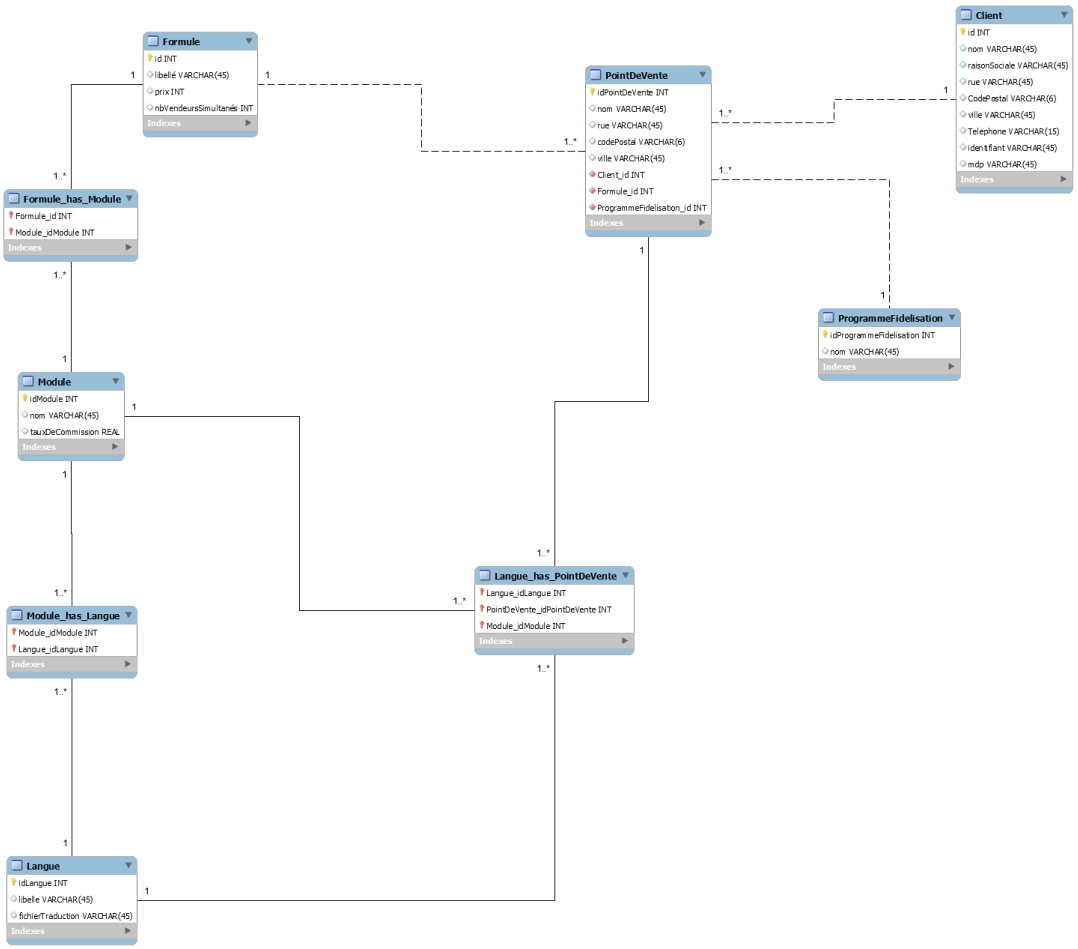
Question 1.1

Modifier la structure de la base de données utilisée par l'application AchatWebCaisse afin de permettre la souscription en ligne du logiciel WebCaisse.

Avant :



Après :



Partie 2 - Gestion du changement de formule

Question 1.2

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.

On ne peut pas garder l'historique car il n'y a pas de notion de date.

Question 1.3

Proposer une correction de la structure de la table qui réponde au besoin exprimé.

Pour résoudre ce problème il faut créer une relation ternaire avec une entité DateSouscription qui contient le jour, le mois et l'année sachant qu'on ne peut faire qu'une seule modification par jour.

FormuleSouscrite(idPointDeVente, idFormule, date)

clé primaire : idPointDeVente, idFormule, date

clés étrangères :

- idPointDeVente en référence à id de PointDeVente
- idFormule en référence à id de Formule
- date en référence à DateSouscription de date.

Mission 2 : Fidélisation des consommateurs

Partie 1 – Amélioration du module de gestion de la relation client (GRC)

Question 2.1

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.

```
SELECT nom, prenom, mail
FROM conso
JOIN vente on idConso = conso.id
WHERE year(dateVente) = 2017
GROUP BY conso.id having count(*) > = 1
```

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

```
SELECT count(*)
FROM consoFidele
WHERE TIMESTAMPDIFF(year, CURDATE(), dateNaissance) between 18 AND 30
```

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

```
SELECT nom, prenom, mail, sum(montantVente)
FROM conso
JOIN on vente on idConso = conso.id
GROUP BY conso.id
```

Question 2.2

Modifier la requête SQL de la méthode listeConsoAFideliser(int seuilVentes, String dateDeb, String dateFin) fournie par Sylvain Cho, afin de lister les consommateurs qui n'ont pas adhéré au programme de fidélisation et pour lesquels on a enregistré un nombre de ventes supérieur au seuil donné, durant la période donnée (le seuil et la période sont fournis en paramètre).

```
String requete = "SELECT nom, prenom, tel, mail, COUNT(*) AS nbVentes
FROM Conso
JOIN Vente ON idConso = Conso.id
WHERE Conso.id NOT IN (SELECT id FROM ConsoFidele)
AND dateVente BETWEEN 'dateDeb' AND 'dateFin'
GROUP BY nom, prenom, tel, mail
HAVING COUNT(*) > seuilVentes;"
```

Partie 2 – Réalisation de tests unitaires pour la méthode AddFidelite()

Question 2.3

Compléter la méthode testInitConso permettant de combler ce manque.

```
assertEquals("erreur calcul mise à 0 fidélité", 0 ,consoTest.getPointsFidelite());
```

Question 2.4

Compléter la méthode testAddMontant permettant de valider les points de fidélité obtenus dans le cas d'un de programme de fidélisation par points.

```
consoTest.addFidelite(3, 150);
assertEquals("erreur calcul 1er point", 10 ,consoTest.getPointsFidelite());
consoTest.addFidelite(3, 250);
assertEquals("erreur calcul 2ème point ", 30 ,consoTest.getPointsFidelite());
consoTest.addFidelite(3, 600);
assertEquals("erreur calcul 3ème point ", 80 ,consoTest.getPointsFidelite());
```

Mission 3 : Statistiques de ventes

Question 3.1

Rédiger le commentaire de la méthode statVente de la classe Statistique expliquant ce qu'elle retourne.

```
public static double statVente(ArrayList<Vente> lesVentesDuJour) {
    int nbVenteFidele = 0;
    foreach (Vente uneVente : lesVentesDuJour) {
        Conso c = uneVente.getLeConso();
        if (c.estFidele()) {
            nbVenteFidele++;
        }
    }
    return (nbVenteFidele * 100 / lesVentesDuJour.size());
}
```

La méthode indique le pourcentage de consommateurs fidèles qui ont réalisés des achats dans la journée.

Question 3.2

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();
}
```

Question 3.3

Écrire le constructeur de la classe VenteEcommerce qui permet d'initialiser tous les attributs d'une instance de la classe.

```
public VenteEcommerce(Date uneDateVente, Conso unConso, double unMontant,  String
adresseLivraison,  String optionLivraison)
{
    super(uneDateVente, unConso, unMontant);
    this.adresseLivraison = adresseLivraison;
    this.optionLivraison = optionLivraison;
}
```

Question 3.4

Compléter le code de la méthode compareLieuVente.

```
public static double compareLieuVente(ArrayList lesConsos)
{
    double totalEcom = 0; // cumul des montants des ventes e-commerce
    double totalMag = 0;  // cumul des montants des ventes en magasin
    // parcours de la liste des consommateurs fidèles
    foreach (ConsoFidele cf : lesConsos)
    {
        foreach (Vente v : cf.getLesVentes())
        {
            if (v instanceof VenteECommerce)
            {
                totalEcom += v.getMontantVente();
            }
            else
            {
                totalMag += v.getMontantVente();
            }
        }
    }
    return totalMag / totalEcom; // calcul de l'indice et retour du résultat }
}
```

Question 3.5

Expliquer en quoi la dernière instruction "return totalMag / totalEcom" de la méthode compareLieuVente peut poser problème.

En cas de division par 0 il y aura une erreur.

Question 3.6

Écrire la méthode répondant à ce besoin.

```
public ArrayList VentesSupérieurMontant(double montant)
{
      ArrayList lesVentesSup = new ArrayList();
      foreach(Vente v : lesVentes)
      {
          if (v.getMontantVente() > montant)
          {
               lesVentesSup.Add(v);
          }
       }
       return lesVentesSup;
}
```

Mission 4 : Seuil de rentabilité et solutions d'hébergement

Partie 1 - Analyse de la rentabilité

Question 4.1

a) Identifier les charges fixes et les charges variables dans le coût du projet présenté dans le dossier documentaire.

Charges fixes :

- Coût des équipements informatiques
- Coût salarial de l'équipe

Charges variables :

- Charges d'exploitation

b) Calculer le montant total de charges fixes et le montant total de charges variables.

Montant total des charges fixes :

- Coût salarial de l'équipe : 172 000
- Coût des équipements informatiques par année en tenant compte de l'amortissement : 8000
- Coût total : 180 000

Les charges variables sont représentées par les charges d'exploitation : 104 000

Question 4.2

Déterminer le point mort du projet en nombre de jours, sachant qu'une année comptable est équivalente à 360 jours. Commenter le résultat obtenu.

Calcul du seuil de rentabilité :

280 000 * 360 / 300 000 = 336

L'entreprise atteint le seuil de rentabilité juste à la fin de l'année (début décembre), elle ne va pas faire beaucoup de bénéfices.

Partie 2 - Choix de l'hébergeur

Question 4.3

Rédiger une courte note à destination du chef de projet en justifiant le choix d'un hébergeur parmi les trois propositions.

Bonjour Renaud, voici un tableau qui analyse les différentes offres.

	Offre A	Offre B	Offre C
Une sauvegarde quotidienne des données du portail	Oui	Oui	Oui mais manuel
De la haute disponibilité	Oui mais pas de réplication	Oui	Oui
Une sécurisation des échanges et des données par cryptage	Oui SSL	Non	Non

Après analyse il apparait que la solution 1 est plus avantageuse, elle répond à tous les besoins de l'entreprise en plus les données sont situées en France et un service technique est disponible.