Note technique explicative - projet Cooking - BDD

I. Organisation de la base de données

Tout d'abord, notre BD contient 5 entités principales (client, commande, recette, produit et fournisseur). Avec cela, on retrouve 5 associations (passe, crée, livrent, contient, utilise) et les deux dernières associations deviennent des tables car relations < n,n >.

Pour rentrer plus dans le détail, nous avons choisi de ne pas créer de table CdR mais de l'inclure, en tant que booléen, dans la table client, car un CdR n'est qu'un client privilégié finalement.

La table contient permet de faire l'association entre commande et recette, son attribut quantiteRecette compte le nombre de fois qu'une recette en particulier est commandée dans une commande. La table utilise permet de faire l'association entre recette et produit, son attribut quantiteProduit compte le nombre de fois qu'un produit en particulier est utilisée dans une recette.

Afin de commencer notre projet, nous avons créé une BDD de référence composée de :

- 4 clients dont 2 CdR
- 5 recettes
- 3 fournisseurs
- 7 produits
- 3 commandes
- 10 "utilise"
- 5 "contient"

Après avoir créé tout cela dans MySQL Workbench, nous avons décidé de créer une première fonction dans notre projet C#, qui nous permet de reset la database, sans passer sur Workbench. Pour faire cela, nous avons créé trois fichiers txt dropData, creationTable et peuplementTable.

Le fichier dropData nous permet de supprimer toutes les tables.
Le fichier creationTable nous permet de les recréer.
Le fichier peuplementTable nous permet de les repeupler avec les données de la BDD de référence.

II. Options de codage

Avant de pouvoir nous attaquer au vif du sujet, nous avons mis en place dans notre projet C# trois fonctions importantes : Connexion_BDD2(), ConnexionBDDUneColonne(), ConnexionBDDDeuxColonne().

La première fonction nous permet d'exécuter simplement une commande SQL (sans affiche console, ni return), notamment utilisée pour les INSERT INTO et UPDATE.

La deuxième fonction nous permet d'exécuter une commande SQL (qui apparait sur Workbench avec une seule colonne; un seul attribut dans SELECT) et retourne cette colonne en tant que liste. La dernière fonction nous permet d'exécuter une commande SQL (qui apparait sur Workbench avec deux colonnes; deux attributs dans SELECT) et retourne deux colonne en tant que liste unique (la 1ère ligne Workbench (2 attributs) représente les deux premiers éléments de la liste, la 2ème ligne Workbench représente les deux éléments suivants de la liste etc.).

L'utilisation de ces trois fonctions (ainsi que de leurs dérivées comme ConnexionBDDTroisColonne() ou ConnexionBDDSpecialReappro()) nous permet de réaliser la quasi-totalité du projet.

Aussi point important, les fonctions test présentes dans notre projet ne sont pas indispensables au projet, mais nous ont permis de pouvoir tester chaque fonction une à une, avant de rassembler le tout.

Dans la réalisation du menu, nous avons décidé de proposer à l'utilisateur 5 choix principaux (Identification du client, gestionnaire Cooking, mode Démo, reset BDD, quitter Menu). Cela nous permet entre autres de pouvoir rentrer en tant que client ou que gestionnaire rapidement, de même pour pouvoir reset notre base de données.

Dans notre code, nous avons organisé nos fonctions, faisant en sorte que les fonctions principales et secondaires d'une partie du menu, comme gestionnaire Cooking, soient rassemblées dans un même bloc de fonctions. Idem pour les autres parties du menu.

Nous avons également rajouté quelques bibliothèques, telle que System.Xml.Serialization ou encore System.Globalization, qui nous ont servi pour des fonctions telles que l'éditeurXML() ou pour pouvoir travailler avec le temps dans la fonction MiseAJour().

III. Choix de programmation - Difficultés rencontrées

Tout d'abord nous avons créé beaucoup de fonctions qui ont pour simple but de savoir si la réponse entrée par l'utilisateur dans la console est valide ou non.

Nous avons aussi décidé dans le menu de bien séparer les clients CdR de ceux qui ne le sont pas. En effet, dans le menu cooking, un CdR aura (en plus d'un client normal) la possibilité de créer une recette, mais aussi d'afficher la liste de ses recettes. Pour le reste, les fonctionnalités sont les mêmes (afficher son solde et passer une commande).

Dans la fonction CréerUneRecette, nous avons fait le choix de donner au CdR seulement la possibilité de créer sa recette à partir des produits qui sont déjà dans la BDD. Nous avons fait ce choix car nous pensons qu'en réalité le site cooking possèdera déjà sa BDD de produits qui sera suffisante pour que les CdR puissent créer leurs recettes (et surtout Cooking aura déjà des accords avec des fournisseurs). C'est pour cela que nous pensions qu'un CdR n'aurait pas besoin de nouveaux produits (et pas l'autorisation d'en utiliser de nouveaux : problème de fournisseurs).

Dans la fonction PasserCommande, nous avons fait le choix de laisser le client commander autant de recettes qu'il le souhaite, quitte à ne pas avoir assez de stock, et seulement ensuite vérifier si à partir de sa commande les stocks de produits sont suffisants. Les stocks finaux des produits (c'est-à-dire si la commande est validée quelles seront les stocks des produits) lui sont affichés, de ce fait, si sa commande est refusée à cause du manque de stock, il saura quels produits n'ont plus assez de stock.
Par contre si les stocks finaux sont suffisants alors il peut procéder au paiement de sa commande. Pour le paiement, on vérifie le solde du client, et s'il n'est pas suffisant pour payer sa commande, nous l'incrémentons automatiquement de 20 cooks jusqu'à ce que son solde soit suffisant pour payer. De plus dans cette fonction, nous avons affiché dans la console certaines listes de notre fonction qui ne sont pas indispensables pour l'utilisateur, mais qui peuvent aider les correcteurs à comprendre le fonctionnement de la fonction PasserCommande.

Ensuite, dans les fonctions de vérification de données telles que NewIdValide() qui vérifie si, lors de la création d'un compte, le pseudo entré par l'utilisateur répond à nos critères (pas plus de 20 caractères...), nous avons décidé de vérifier le pseudo ne comportait pas d'espace et surtout aucune apostrophe, car une fois rentré dans la base de données, comme les valeurs sont déjà entre apostrophes, en rajouter risquerait de fausser tout le tuple.

De même pour toutes ces vérifications de caractères, une fonction a été créée dans ce but-là. Et pour nous faciliter la tâche, nous avons fait appel à la table ASCII pour désigner chacun caractère indésirable pour son code respectif.

De plus, sur la globalité de notre code, nous avons fait le choix d'utiliser des listes List<> au dépend de tableaux [], car elles permettent une utilisation plus dynamique, notamment avec la fonction .Add pour ajouter plus rapidement des éléments à la liste sans se soucier de la taille de cette dernière.

Pour pouvoir éditer une fichier XML, nous avons créé une classe Commande Fournisseur (qui se trouve du coup en dehors de class Program) qui contient uniquement la liste des informations nécessaires aux fournisseurs lors d'une commande des produits (nom du fournisseur, produit à commander et la quantité demandée). Cette classe nous permet de faire plus rapidement le lien entre C# et XML, car beaucoup plus facile de sérialiser des objets. Enfin, après avoir créé le fichier XML qui contient toutes ces informations, les stocks actuels restent inchangés (càd lorsque sur la BDD, avant la commande fournisseur, le stock de carotte est à 4, et que sur le fichier XML, il est écrit qu'il faut réapprovisionner 10 carottes, le stock de carotte reste à 4) car nous estimons que cela dépendait ensuite de Cooking et de ses fournisseurs.

Enfin, nous avons également pris l'habitude de commenter chaque fonction grâce à la fonctionnalité /// à écrire au-dessus de chaque fonction, nous permettant de renseigner le rôle de chaque fonction, ce qu'elle prend en paramètre, ce qu'elle retourne... et que cela ne surcharge pas la fonction en elle-même.