Rapport du Projet
« Startup Cooking »

Base de Données

ESIV Paris la Défense

Sommaire

I. Les Tables

1. Les Acteurs et le Login

Dans le but de permettre à tous les acteurs de pouvoir s'identifier sur le même portail, nous avons décidé de référencer tous les acteurs dans la table client. Cela permet d'effectuer une seule et même requête SQL d'identification sur la table client (Un seul point d'entrée => plus de sécurité). Lorsqu'un utilisateur s'identifie, le programme effectue une requête dans la table client join left admin et recipe_creator. Il nous est alors facile de savoir si l'utilisateur est admin, recipe_creator ou client.

Dans cet ordre : Si admin_id n'est pas null alors l'utilisateur est admin, si recipe_creator_id n'est pas null alors l'utilisateur est recipe_creator, sinon l'utilisateur est juste client.

2. Les Produits & Fournisseurs

Chaque produit possède un supplier_id qui fait référence à un fournisseur, si le fournisseur est supprimé, alors le produit est également supprimé. Chaque produit à une quantité minimum et maximum, ces variables sont mises à jour chaque lundi par un event qui enclenche le réajustement de celle-ci en fonction des ventes passées.

De plus chaque produit à une quantité courante, si un client tente de faire une commande qui nécessite plus de produits que disponible, alors la base de données va lever une MySqlException code 1690 et le programme affichera à l'utilisateur que cette commande ne peut pas être réalisé car la quantité de produit n'est pas assez grande.

3. Les Recettes et Commandes

Les recettes ont un attribut is_validated qui est par défaut à false, un Admin doit pour rendre cette recette publique, la valider depuis l'espace Admin. Lorsqu'une recette est achetée, toutes les fonctionnalités nécessaires sont enclenchées automatiquement par les triggers : décrémentation des stocks de produits, incrémentation du nombre de recette vendu...

Pour passer une commande, le programme ajoute d'abord une ligne dans la table ORDERTYPE puis ajoute des recettes dans cette commande via la table IS_IN_ORDERTYPE. Même procédure pour le panier d'un client, sauf que le client ne possède qu'un seul panier.

4. Le Dashboard

Le Dashboard contient toutes les informations des meilleures recettes/créateurs de recettes de la semaine/depuis toujours. Cette table est mise à jour par les triggers à chaque fois qu'une commande est réalisé ou bien qu'une recette est supprimée.

II. Les Triggers

1. Après la réalisation d'une Commande

1. On vide le panier du client, donc toutes les lignes dans IS_IN_CART_OF qui font référence au client qui vient de passer la commande.
2. On met à jour le Dashboard.

2. Avant la mise à jour d'une Recette

1. On met à jour le prix et la recette ainsi que la commission du créateur de recette en fonction du nombre de recettes vendu.

3. Après l'achat d'une Recette

1. On décrémente la quantité disponible de chaque produit de la recette en fonction des quantités dans HAS_PRODUCT et IS_IN_ORDERTYPE.
2. On incrémente le nombre de recette vendu en fonction du nombre de commandes.
3. On paye le créateur de recette.
4. On décrémente la cook_balance de l'acheteur s'il est un créateur de recette.

4. Après la suppression d'une Recette

1. On remet à jour le Dashboard pour éviter des erreurs dans le cas ou la recette supprimée était dans un des tops 5.

III. Les Events

Chaque lundi est mis à jour la quantité de tous les produits ainsi que le réajustement des quantités min et max en fonction de la quantité actuelle disponible du produit comparé aux variables min et max. Cela permet de réguler automatiquement les quantités min et max en fonction de la demande cliente.

IV. La Database en C#

1. ProtectedQueries.cs

Les méthodes de cette Class ne doivent être accessibles que par la Class Database.cs. Elle à pour but de référencer les requêtes nécessaires pour d'autres résultats de requêtes.

Par exemple, la requête Login du fichier Database.cs nécessite de récupérer également le panier actuel du client qui vient de se logger, nous n'avons jamais besoin d'accéder juste au panier d'un client dans autre cas.

Ainsi toutes les requêtes qui ne font sens que pour d'autres requêtes sont écrites dans ce fichier.

2. Database.cs

Ce fichier référence toutes les requêtes SQL nécessaire au programme, centraliser les requêtes dans un objet Database.cs nous permet de mieux ranger notre projet et de savoir ou regarder lorsqu'une erreur parvient. De plus cela nous permet également d'éviter les requêtes redondantes sur plusieurs pages différentes.

3. Conversion des objets SQL en C#

Tous nos objets SQL hormis les entités faibles on été convertie en objet C#. Ainsi pour convertir par exemple un Client SQL en Client C#, nous passons simplement en paramètre du constructeur Client le MySqlDataReader de la requête. Puis dans le constructeur chaque attribut C# prend la valeur de son attribut équivalent SQL.

Pour les entités faibles comme le panier par exemple, les dictionnaires C# ont été utilisés. Ainsi chaque Client possède un panier modélié par un dictionnaire ou la clef est la recette, et la valeur sa quantité dans le panier. Cette même technique est utilisé pour la modélisation des trois tables HAS_PRODUCT, IS_IN_CART_OF et IS_IN_ORDERTYPE.

De plus pour rendre flexible notre programme à des modifications/besoins futurs, les méthodes Update sont les mêmes peu importe l'attribut qui doit être mis à jour.

V. Le mode Demo/Test

Nous avons fusionné nos tests unitaires avec le peuplement de notre base de données ainsi que le mode démo demandé (Voir la barre de chargement et la Console pour la liste des Tests).

En effet, dès la première connexion à la base de données, le programme vérifie si tous les objets SQL de l'application sont déjà créés (Tables, Triggers, Events):

1. Si oui, alors une pop-up vous demande si vous souhaitez lancer le processus de reinitialisation de la base de données puis son peuplement.
2. Sinon alors se lance directement le processus de création des tables, triggers, events puis de nombreuses insertions/mises à jour/suppressions de lignes dans les tables.

Après chaque appel de méthode dans le fichier PopulateDatabase.cs est vérifié que le résultat attendu est correct. Par exemple, après l'insertion de trois lignes dans la table Client, nous vérifions que le nombre de ligne inséré est bien trois, puis nous sur-vérifions en rappelant ces trois lignes via la méthode select...