Le site de vente de la sorcière Giselle

Rapport de projet

**GiselleMagicArts.com**



Table des matières

[I. Présentation du projet 3](#_Toc479192940)

[1. Contexte 3](#_Toc479192941)

[2. Besoins 4](#_Toc479192942)

[3. L’équipe projet 4](#_Toc479192943)

[II. Planification du projet 5](#_Toc479192944)

[1. Découpage du projet : le WBS 5](#_Toc479192945)

[2. Planning prévisionnel 5](#_Toc479192946)

[3. Planning effectif 8](#_Toc479192947)

[III. Mise en place de la solution proposée 9](#_Toc479192948)

[1. Modélisation : la méthode MERISE 9](#_Toc479192949)

[a. Modèle conceptuel de données (MCD) 9](#_Toc479192950)

[b. Modèle logique de données (MLD) 13](#_Toc479192951)

[c. Choix du SGBDR 14](#_Toc479192952)

[d. Modèle physique de données (MPD) 14](#_Toc479192953)

[2. Les requêtes SQL 15](#_Toc479192954)

[a. Remplissage de la base de données 15](#_Toc479192955)

[b. Procédures stockées 15](#_Toc479192956)

[c. Soumission de recettes 17](#_Toc479192957)

[d. Rechercher et consulter 17](#_Toc479192958)

[e. Sauvegarder 17](#_Toc479192959)

[3. Tests 17](#_Toc479192960)

[4. Retour sur expérience 17](#_Toc479192961)

[a. Synthèse des problèmes rencontrés 17](#_Toc479192962)

[b. Bilan de l’équipe 17](#_Toc479192963)

[c. Bilans individuels 17](#_Toc479192964)

[ANNEXE 1 : Créer un serveur MySQL sur un Raspberry Pi 18](#_Toc479192965)

[ANNEXE 2 : Script de création de la base de données 19](#_Toc479192966)

[ANNEXE 3 : Manuel d’utilisation de la base de données 23](#_Toc479192967)

[ANNEXE 4 : Ressources 25](#_Toc479192968)

# Présentation du projet

## Contexte

Giselle, jeune sorcière ayant obtenue récemment son permis de magie souhaite se lancer dans la vente de potions et d’ingrédients. Elle souhaite donc réaliser un site e-commerce de magie : gisellemagicarts.com.

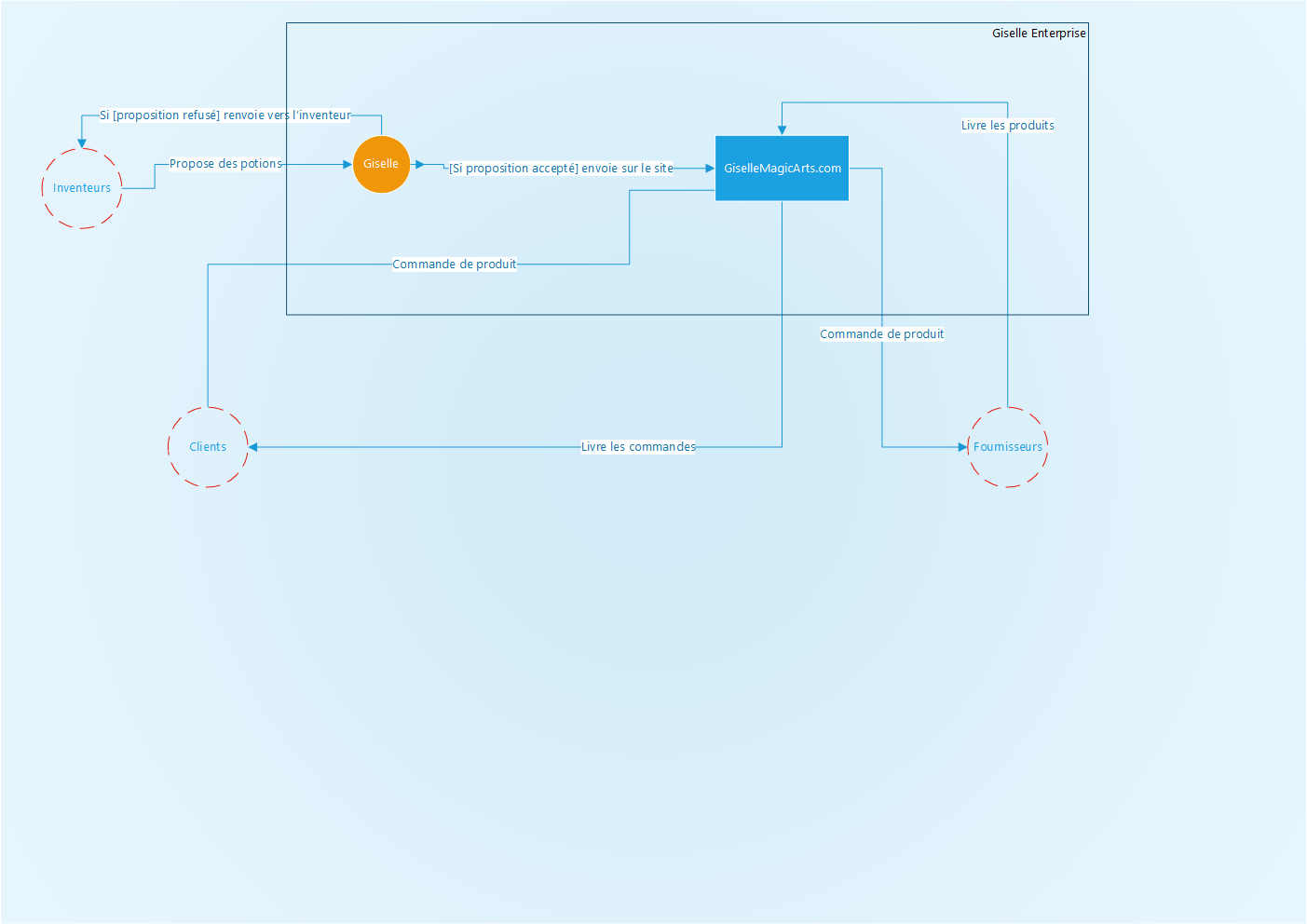
La sorcière a donc besoin d’une interface web ainsi que d’une base de données pour stocker toutes les informations nécessaires au bon fonctionnement de son site. Sur ce dernier sera vendu :

* Des potions
* Des onguents
* Des ingrédients (pour préparer des potions et des onguents)

Pour expliquer le fonctionnement du site GiselleMagicArts nous avons décidé d’établir un diagramme de flux. Le but est de bien comprendre le fonctionnement du futur site pour développer un modèle contextuel de données adaptées.

Tout d’abord listons quelques informations complémentaires concernant le fonctionnement du site :

* Giselle ne se procure pas elle-même les ingrédients mais fait appel à des fournisseurs.
* Les clients commandent puis sont livrés
* De nouveaux inventeurs peuvent proposer des recettes de nouvelles potions. Giselle choisit ensuite si elle souhaite ajouter les propositions de potions au site ou non.

Les acteurs externes sont représentés par des cercles à traits pointillés, les acteurs internes par des cercles à traits pleins et les flux par des flèches de l’émetteur vers le récepteur.

## Besoins

Un ami de Giselle, Miguel se charge de réaliser le site web. Ce projet consistera donc en la création d’une base de données et des requêtes nécessaires au bon fonctionnement de ce site.

Giselle souhaite surveiller de manière consciencieuse la fraîcheur de chaque ingrédient. En effet, la fraîcheur d’un ingrédient peut avoir un effet sur la qualité d’une potion. Elle souhaite également que les ingrédients qui ne sont plus frais soient automatiquement supprimés de la base de données. De plus les clients peuvent préciser la fraîcheur souhaitée des ingrédients qu’ils commandent (un même ingrédient peut être présent deux fois dans une commande mais avec des fraîcheurs différentes).

Les clients, lorsqu’ils commandent des potions, doivent choisir dans quel récipient celle-ci sera contenu (fioles, pots…). Le prix variera en fonction du type de récipient.

Il faudra préparer un certain nombre de requêtes en langage SQL pour extraire de la base de données des informations clés pour la sorcière.

Giselle veut mettre en place un système de proposition de recettes. Les clients pourront proposer de nouvelles recettes et Giselle choisira si elle décide de les intégrer ou non dans son catalogue.

Le dernier point important pour la base de données de Giselle est que celle-ci doit être sauvegardée. Ainsi, elle pourra être facilement restaurer en cas d’éventuel problèmes ou de migration.

## L’équipe projet

L’équipe chargée de réaliser le projet de Giselle est composée de quatre membres :

* Florian FRITSCH
* Aurélien KLEIN
* Pierre GEERAERT
* Alexandre LECOMTE (chef de projet)

Dans la suite de ce rapport, « nous » désignera l’équipe du projet présentée ci-dessus.

Nous disposons de 7 jours ouvrés pour une équipe de quatre personnes soit 28 jours homme pour réaliser une base de données fonctionnelle et répondant aux besoins du futur site web de la sorcière Giselle.

# Planification du projet

## Découpage du projet : le WBS

Afin de se répartir correctement les tâches et avoir une vision globale du travail pour réaliser le projet nous avons décidé de définir un Work Breakdown Structure (WBS).

Le WBS est définit par un « découpage hiérarchique en livrables spécifiques des travaux à exécuter ». On trouve plusieurs niveaux dans un WBS, chaque niveau détaille un peu plus les tâches à réaliser jusqu’à arriver au niveau final. L’idée est d’obtenir un diagramme facilement compréhensible reprenant tout ce que l’équipe du projet doit réaliser.

Le dernier niveau doit correspondre à une tâche que l’on peut attribuer à une ou plusieurs personnes de l’équipe.

Après avoir clarifié le contexte et défini le périmètre du projet dans la partie précédente, nous avons donc découpé le travail de la manière représenté par la figure 1 de la page suivante.

## Planning prévisionnel

A partir de ce WBS nous avons établi une répartition des tâches (figure 2) ainsi qu’une estimation de la durée de celles-ci.

A partir du tableau de répartition des tâches nous avons construit un diagramme de Gantt correspondant (figure 3).



Figure - WBS du projet



Figure - Répartition des tâches

Figure - Diagramme de Gantt

## Planning effectif

# Mise en place de la solution proposée

## Modélisation : la méthode MERISE

Merise a principalement été utilisée en France dans les administrations publiques ou privés et par certaines entreprises de services du numérique. La méthode Merise est une méthode d'analyse, de conception et de réalisation de systèmes d'informations.

La méthode Merise se caractérise par ces différents axes :

* Identifier les acteurs du projet
* Réaliser le schéma directeur (objectifs/stratégie de l’entreprise, cadre du projet)
* Faire l’étude préalable (description des besoins et des attentes des utilisateurs, ébauche des divers modèles)
* Faire l’étude détaillée
* Réalisation de l’étude technique (moyens techniques nécessaires)
* Mise en production
* Gestion de la maintenance (informations sur les acteurs, les documentations et les formations)

La méthode MERISE se déroule en trois grandes étapes :

* Le modèle conceptuel de données (MCD) : Quelles relations lient les données entre elles ?
* Le modèle logique de données (MLD) : Quelles sont les liens qui découlent de ces relations ?
* Le modèle physique de données (MPD) : C’est l’implémentation en langage SQL (Structured Query Language) de la base de données pour un système de gestion choisit.

Nous avons décidé de choisir la méthode Merise pour sa fiabilité, sa rapidité et son efficacité. De plus, c’est sur cette méthode que les membres de l’équipe ont été formés.

Pour formaliser les différentes étapes de la méthode Merise nous avons choisi le logiciel JMERISE, distribué gratuitement au téléchargement. Ce logiciel permet de formaliser un MCD puis de le traduire automatiquement en MLD puis en script SQL (MPD). La conversion des différents modèles est rapide et efficace. En outre, JMERISE gère la version 2 de Merise 2 ; nous permettant notamment d’implémenter la notion d’héritage dans notre MCD.

### Modèle conceptuel de données (MCD)

Pour pouvoir construire le modèle conceptuel de données[[1]](#footnote-1) on procède en deux étapes :

* La création d’un dictionnaire de données (DD)
* La création d’une matrice des dépendances fonctionnelles (Matrice des DF)

Le dictionnaire de données ou recueil d'informations consiste à collecter les données. Nous avons donc répertorié l’ensemble des données que nous jugions nécessaires en analysant le sujet du projet.

Nous aboutissons au dictionnaire de données suivant :

Figure - Dictionnaire de données (DD)

La deuxième phase de conception du MCD est l’analyse des dépendances. Les dépendances sont les liens qui lient les différentes données. Le document résultant de cette étape est un diagramme ou une matrice des dépendances.

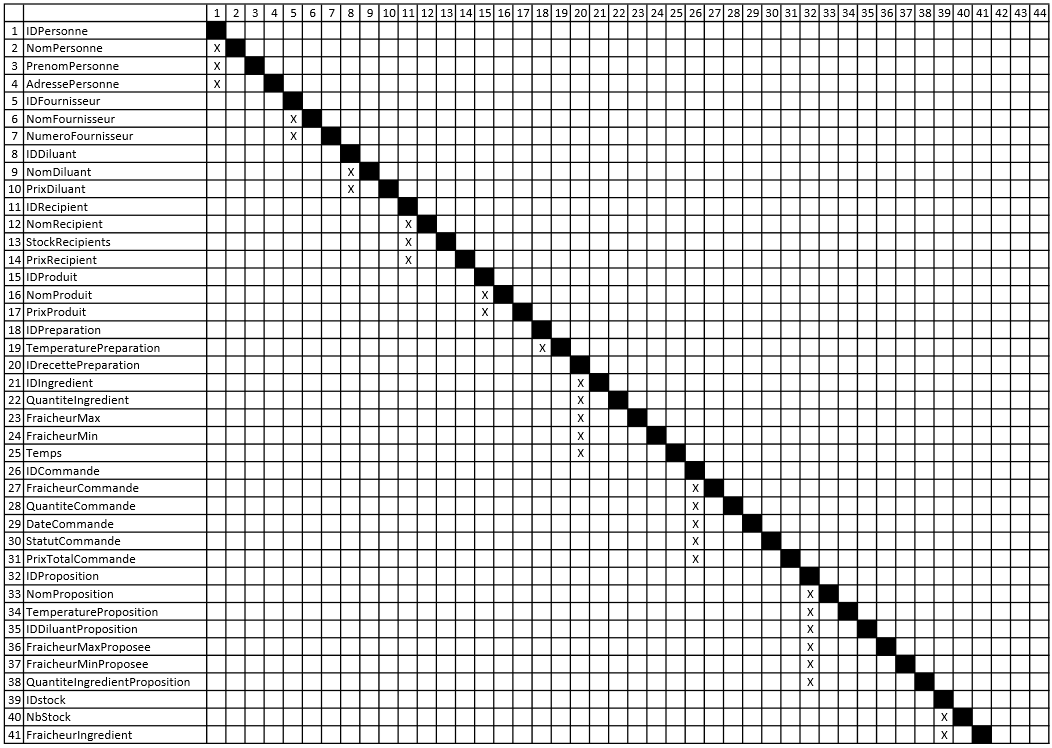
A partir du dictionnaire précédant on construit la matrice des dépendances fonctionnelles suivantes :

Figure - Matrice des dépendances fonctionnelles

A l’aide de ces deux documents (DD et matrice des DF) on établit le modèle conceptuel de données sur le logiciel JMERISE.

On obtient le MCD suivant :

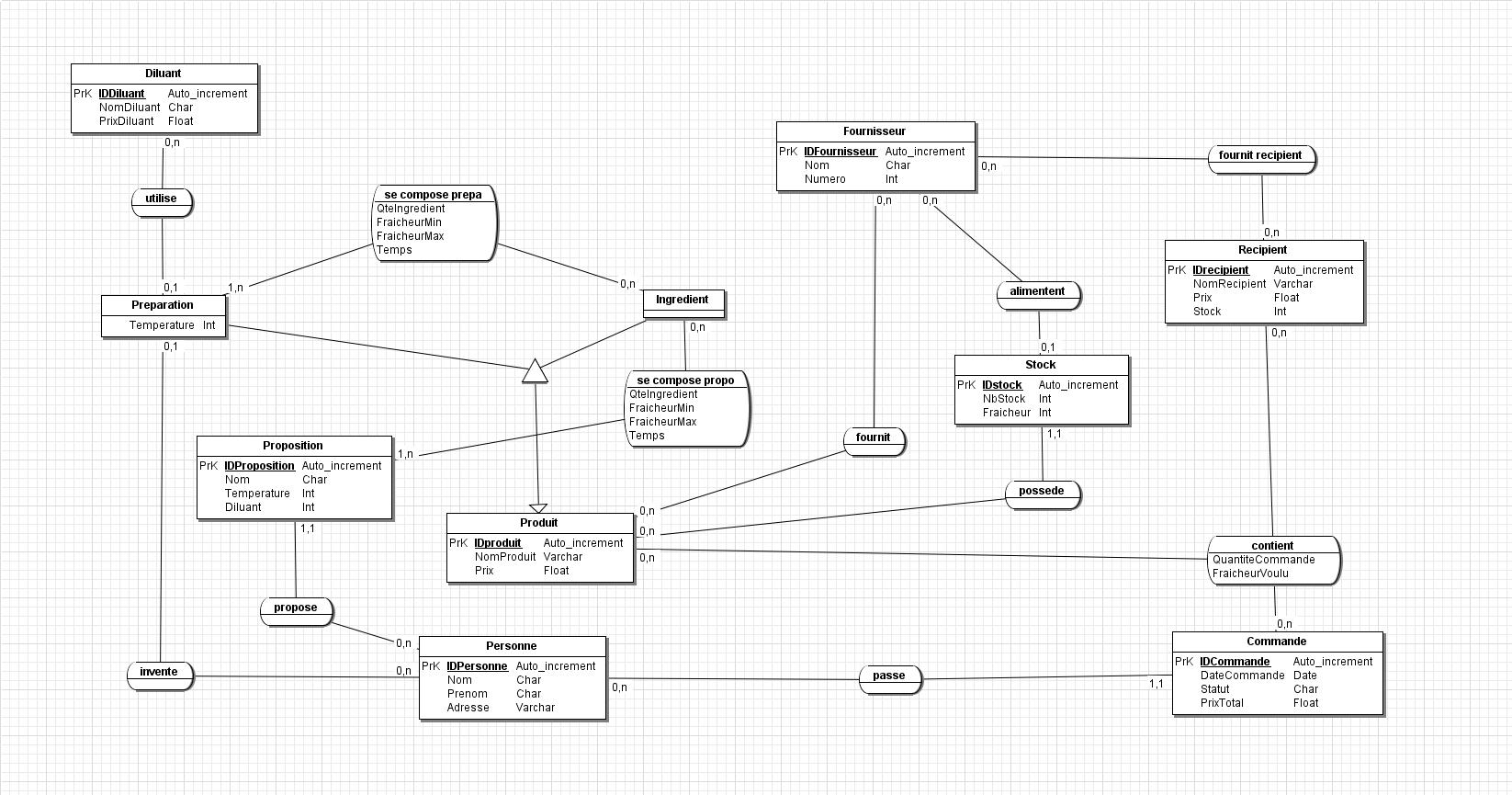
La figure 6 présente le modèle conceptuel de données de la future base de données du site GiselleMagicArts.com.

Figure - Modèle Conceptuel de données

Tout d’abord, l’entité produit rassemble tout ce que la sorcière Giselle vend sur son site ; c’est-à-dire potions, onguents et des ingrédients seuls. Chacun de ces produits sont identifiés de manière unique par un numéro (un peu comme une référence). Cette entité produit est l’entité mère d’une relation d’héritage dans laquelle les deux entités filles sont :

* Ingrédient : entité contenant les références des produits qui sont des ingrédients
* Préparation : entité contenant les références des produits qui sont des préparations (onguents et potions)

Ainsi, l’entité préparation contient aussi bien les onguents que les potions. Pour identifier les onguents des potions et vice versa il suffira de distinguer les préparations qui utilisent un diluant (potions) et celles qui n’en utilisent pas (onguents).

Une préparation utilise un ou aucun diluant (cas des onguents).

L’entité personne rassemble les clients mais aussi les inventeurs. En effet, un inventeur peut aussi être un client et inversement. C’est pour cette raison que la cardinalité du côté de l’entité personne de l’association invente est 0,n. Une personne peut inventer plusieurs ou aucune préparation. L’autre cardinalité de cette association montre que pour une préparation on ne connaît pas toujours l’inventeur.

Une personne passe des commandes qui contiennent des produits ainsi que des récipients. En effet, pour chaque produit commandé il faudra indiquer un récipient pour le transport et sa conservation.

Les personnes peuvent également proposer des idées de préparations qui sont composées de produits (Ingrédients, potions et onguents). De ce fait, il sera possible de créer une potion dans laquelle un des ingrédients est une autre potion ou un onguent.

L’entité fournisseur rassemble l’ensemble des fournisseurs que Giselle sollicitera. Ainsi, les fournisseurs fournissent des produits ; c’est-à-dire des ingrédients mais aussi des potions et des onguents. De cette manière, si Giselle souhaite par la suite acheter des potions ou des onguents directement aux fournisseurs il ne sera pas nécessaire de modifier le MCD. Les fournisseurs peuvent également fournir des récipients. Tout en sachant qu’un même type de récipient ou un même ingrédient peut être fournit par plusieurs fournisseurs différents. Ils alimentent donc un stock de produits.

Un identifiant de stock représente un ingrédient avec une fraîcheur précise. Par conséquent, un même ingrédient peut se retrouver plusieurs fois dans le stock mais avec des fraicheurs différentes.

### Modèle logique de données (MLD)

Maintenant nous devons transformer le MCD en modèle logique de données (MLD). La quatrième étape de l’analyse des données est donc la traduction du MCD en MLD. Cette traduction s’effectue en suivant des règles précises.

Le MLD est toujours indépendant du SGBD utilisé. Il est composé uniquement de relations. Une relation est composée d’attributs et possède systématiquement une et une seule clé primaire qui permet d’identifier sans ambiguïté chaque occurrence de cette relation.

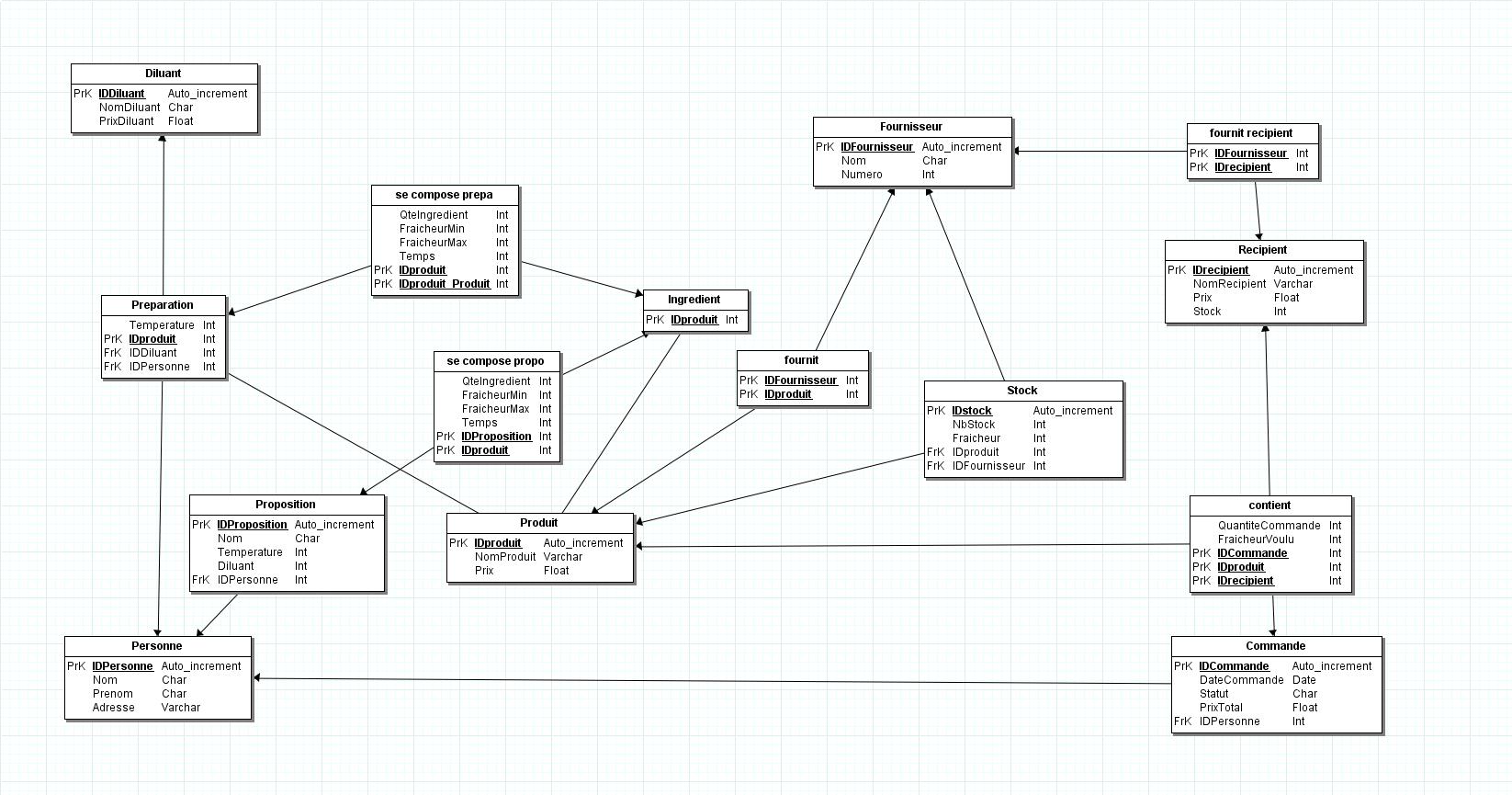


Figure - Modèle logique de données

### Choix du SGBDR

Nous avons choisi le Système de Gestion de Base de Données Relationnel (SGBDR) : MySQL. Comme la majeure partie des SGBDR, il utilise le langage SQL.

Dans un premier temps, nous avons été formés à l’école sur MySQL. Cela nous a permis d’avoir plus d’aisance dans la création de la base de données sur ce SGBDR.

Figure - Logo de MySQL

De plus MySQL est gratuit et disponible sur plusieurs systèmes d’exploitation. Il est multi-utilisateur, cela nous permet donc de travailler tous ensemble sur la même base de données. Pour cela nous avons donc hébergé notre serveur MySQL sur un Raspberry Pi 3[[2]](#footnote-2).

MySQL dispose également de fonctionnalité de sécurité. Ce SGBDR contient un système de privilèges qui nous permet de gérer les différents accès sur ce serveur et il dispose d’un système de sauvegardes.

### Modèle physique de données (MPD)

Le modèle physique de données est l’implémentation du MLD par un logiciel. On va devoir traduire le MLD pour obtenir un MPD. On va notamment préciser le type et la taille (octets, bits) des données pour savoir comment les stocker.

La traduction d’un MLD relationnel en MPD passe par la création d’une base de données hébergée par un SGBDR (ici MySQL).

Le MPD est donc le script de création de l’ensemble des tables et des relations entre elles en langage SQL.

Vous trouverez le fichier script\_creation\_bdd dans l’annexe 2 ainsi que sur le dépôt GitHub du projet.

## Les requêtes SQL

Maintenant que la structure de la base de données est valide et implémentée, nous devons insérer des données et réaliser un certain nombre de traitements que nous aborderons dans cette partie.

### Remplissage de la base de données

Afin d’insérer un jeu de données cohérent et significatif nous avons décidé de procéder de cette manière :

* 1ère étape : Créer un classeur Excel dans lequel chaque feuille représente une table de la base de données. On ajoute des données dans ces tableaux.
* 2ème étape : Enregistrement de chaque feuille dans un fichier texte au format CSV[[3]](#footnote-3).
* 3ème étape : Insertion du contenu des fichiers CSV à l’aide d’un script SQL.

Vous trouverez le fichier Excel du jeu de données, les fichiers CSV et le script de remplissage sur le dépôt GitHub du projet.

Nous avons choisi d’utiliser les fichiers CSV pour l’importation des données car cette méthode nous apparaissait comme la plus simple, la plus efficace et surtout la plus rapide. De plus, cette méthode présente l’avantage de pouvoir afficher les données contenues dans les différents fichiers sous la forme d’un tableau structuré. Il est possible de modifier ces fichiers à l’aide d’un logiciel tableur.

### Procédures stockées

Les procédures stockées permettent d’automatiser des actions. Une procédure stockée est une sorte de script, une série d’instruction SQL désignée par un nom. Cette procédure est enregistrée dans la BDD qu’on utilise et on peut l’appeler pour exécuter les instructions qu’elle contient. C’est un peu comme une fonction dans un code.

Nous avons choisi de définir des procédures stockées[[4]](#footnote-4) pour satisfaire les besoins suivants :

* Affichage du stock des ingrédients et mise à jour à la réception de nouveaux ingrédients
* Consultation des commandes d’un client
* Suppression d’un ingrédient arrivé à expiration (pourra être éventuellement automatisé)
* Mise à jour du stock après envoi d’une commande (la commande devra changer de statut)

**Affichage du stock des ingrédients et mise à jour à la réception de nouveaux ingrédients**

Cette procédure prend comme paramètres le nom de l'ingrédient, la quantité à ajouter et sa fraicheur.

Tout d'abord on affiche le stock de l'ingrédient en question avant tout changement avec une jointure entre les tables Stock, Ingredient, et Produit.

Ensuite on réalise la modification souhaitée grâce à une condition sur le nom de l'ingrédient concerné. On modifie le stock en effectuant une jointure entre la table stock et la table produit où nous cherchons à ajouter la variable nbr\_add a la quantité du stock de l'ingrédient renseigné avec une fraicheur existante dans la base de données.

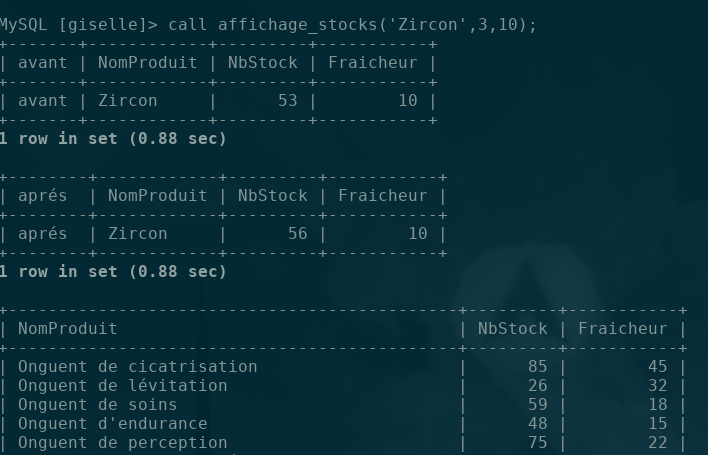
Pour finir, nous affichons l'intégralité des articles après modifications pour permette un inventaire des stocks.

Figure - Procédure d'affichage et de mise à jour du stock des ingrédients

**Historique des commandes d’un client**

Cette procédure prend en paramètre le nom du client pour lequel on souhaite obtenir la liste des commandes.

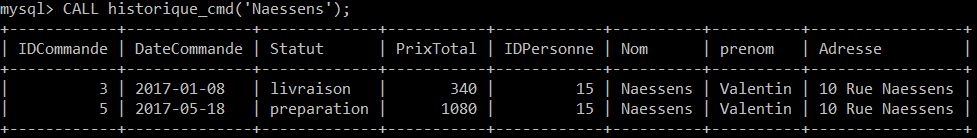
On réalise une jointure entre les tables Commande et Personne puisqu’une commande possède l’identifiant de la personne qui l’a passée. On restreint cette jointure avec une condition sur le nom de la personne dont l’on souhaite connaître l’historique de commandes.

Figure - Procédure affichant l'historique des commandes d'un client

**Suppression d’un ingrédient arrivé à expiration**

Cette procédure permet de décrémenter d'une unité (journée) la fraicheur chaque jour. Elle est programmée pour s’exécuter tous les soirs à minuit.

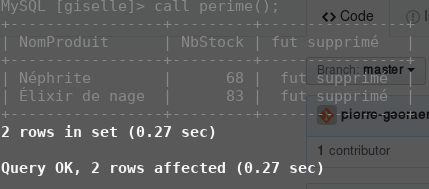
La procédure supprime également les éléments dont la fraicheur est égale a 0 en effectuant une jointure entre Stock et Produit.

Figure - Procédure de suppression d'un ingrédient arrivé à expiration

**Mise à jour du stock après envoie d’une commande**

### Soumission de recettes

La sorcière Giselle souhaite mettre en place un système de soumission de nouvelles recettes sur son site e-commerce. Nous avons donc développé la base de données en ce sens. Une table proposition permet de stocker les propositions des clients.

Nous devions donc pouvoir ajouter une proposition de recette et ensuite permettre à Giselle de valider ou de refuser cette proposition.

**Ajouter une proposition de recette**

Cette procédure stockée prend en paramètres tous les éléments de la recette que l’on souhaite proposée. La procédure récupère et insère les informations passées en paramètres dans les tables Proposition et se\_compose\_propo.

La table se\_compose\_propo ne contient que des ID, il faut donc récupérer les ID correspondant aux ingrédients passés en paramètres. On utilise pour cela des requêtes de type INSERT INTO … SELECT …

Par exemple, pour ajouter le deuxième ingrédient dans la recette :

**INSERT INTO** se\_compose\_propo(`QteIngredient`,`FraicheurMin`,`FraicheurMax`,`Temps`,`IDProposition`,`IDproduit`)

**SELECT** qte2, fraichMin2,fraichMax2,temps2,@IDPropo,IDproduit

**FROM** Produit

**WHERE** NomProduit = ingredient2;

Une fois la proposition de recette ajoutée on affiche le récapitulatif de celle-ci.

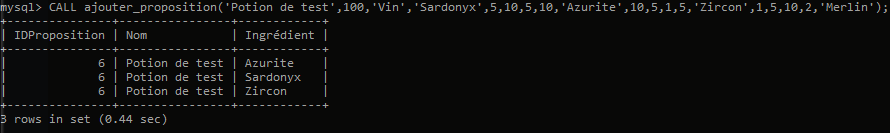
**Valider une recette qui a été acceptée**

Figure – Procédure d’ajout d’une proposition de recette

La table Proposition stocke les recettes proposées par les clients. Cette table permet de stocker temporairement les recettes en attentes de validation. La sorcière Giselle pourra utiliser la procédure stockée suivante pour valider une proposition de recette et l’ajouter dans les préparations officielles.

La procédure valider\_proposition prend en paramètre l’ID de la proposition pour laquelle on veut transférer la recette dans les recettes officielles. Une fois la proposition ajoutée dans les tables Produit, Preparation et se\_compose\_prepa on la supprime des tables Proposition et se\_compose\_propo.

**Supprimer une recette qui n’a pas été retenue**

Cette procédure prend un paramètre l’ID de la proposition que l’on souhaite supprimée car Giselle ne l’a pas retenue. L’exécution de la procédure se fait à l’aide de la commande :

**CALL** refus\_proposition(id)

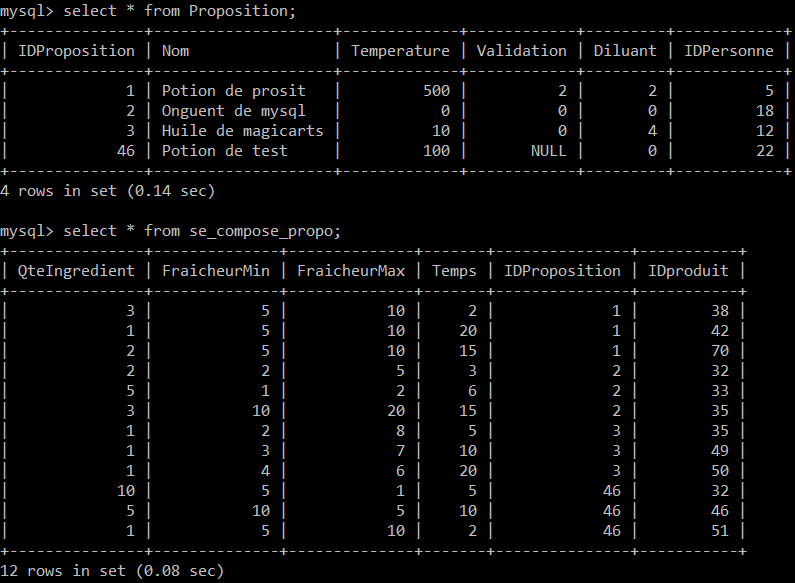
Une fois la commande exécutée, l’enregistrement correspondant est supprimé de la table Proposition et la composition de sa recette est retirée de la table se\_compose\_propo.

Figure - Résultat de la procédure de suppression d'une proposition qui n'a pas été retenue

### Rechercher et consulter

La sorcière Giselle avait également besoin de requêtes de recherche/consultation pour pouvoir utiliser le plus facilement et le plus efficacement possible la base de données de son site web. Giselle souhaitait :

* Les potions ou les onguents réalisables avec un ingrédient précis
* Les couples (potions, onguents) utilisant la même liste d’ingrédients
* Les potions réalisables avec un diluant précis
* La liste des potions classée par température de préparation
* La comparaison entre le prix de vente d’une potion et le total du prix des ingrédients (et diluant) nécessaires à sa réalisation
* Le nombre moyen d’ingrédients utilisés pour fabriquer une potion (ou onguent)

Outre, les besoins de Giselle nous avons pris la liberté d’ajouter des requêtes adaptées à notre MCD dans le but d’accéder à toutes les informations facilement. Nous pouvons donc effectuer les traitements suivants :

* Le contenu d’une commande
* La liste des Potions
* La liste des Onguents
* La liste des produits vendus par Giselle ainsi que leur prix
* Les recettes proposées selon les clients
* Le stock de produits disponibles
* La recette d’une préparation (potion ou onguent) précise
* La recette d’une proposition précise

**Les potions ou les onguents réalisables avec un ingrédient précis**

Cette requête prend la forme d’une procédure stockée prenant en paramètre le nom de l’ingrédient pour lequel on veut connaître la liste des potions/onguents qui le contiennent.

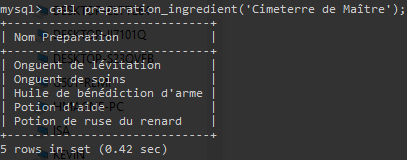
Pour cela on réalise une intersection entre les toutes les préparations et les préparations possédant l’ingrédient passé en paramètre dans leur composition (on utilise une condition).

Figure - Procédure listant les préparations réalisables avec un ingrédient précis

**Les couples potions/onguents utilisant la même liste d’ingrédients**

On réalise cette requête sous la forme d’une procédure stockée. Cette dernière utilise deux vues : la liste des potions et la liste des onguents. On réalise une jointure pour chacune de ces vues avec la table se\_compose\_prepa pour avoir la composition des preparations. Ensuite on réalise une jointure à partir des deux sous-requêtes précédentes. On fait la jointure sur les identifiants de produit pour afficher côte à côte les potions et les onguents possédant le/les mêmes ingrédients.

Pour finir on ajoute les conditions suivantes :

* L’onguent et la potion doivent avoir le même nombre d’ingrédients
* Ce nombre d’ingrédients doit être égale au maximum du nombre d’ingrédient entre la potion et l’onguent

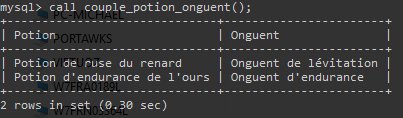
Si ces conditions sont vérifiées on obtiendra uniquement les potions et les onguents utilisant exactement la même liste d’ingrédients.

Figure - Procédure d'affichage des couples (potions,onguents) utilisant la même liste d'ingrédients

**Les potions réalisables avec un ingrédient précis**

Pour réaliser cette requête on utilise une procédure stockée prenant en paramètre le nom du diluant pour lequel on souhaite obtenir la liste des potions utilisant ce diluant.

Tout d’abord, on sélectionne les noms de produits de la table produit ainsi que le nom des diluant.

Puis on place une condition pour réaliser une jointure entre les tables produit, Preparation et Diluant.

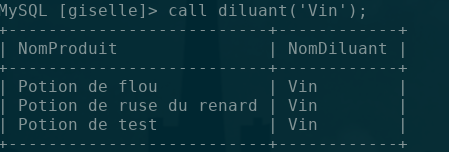
Ensuite nous spécifions le nom du diluant à rechercher grâce à une condition où la paramètre de la procédure permet de repérer quelle(s) potion(s) utilise le diluant choisit et donc celles que nous devons afficher.

Figure - Procédure affichant les potions réalisables avec un ingrédient précis

**La comparaison entre le prix de vente d’une potion et le total du prix des ingrédients (et diluant) nécessaires à sa réalisation**

Cette requête consiste en fait à calculer la marge que dégage Giselle pour chaque préparation qu’elle vend.

On doit donc faire la différence entre le prix des produits (onguents et potions), la somme des prix de chaque ingrédient multiplié par leur quantité et le prix du diluant si c’est une potion.

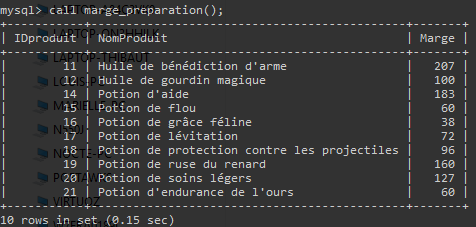
On récupère le prix total de la préparation et le prix du diluant grâce à une sous requête.

Figure - Procédure affichant la marge que dégage Giselle pour chaque préparation

**Le nombre moyen d’ingrédients utilisés pour fabriquer une potion (ou onguent)**

### Sauvegarder

## Tests

## Retour sur expérience

### Synthèse des problèmes rencontrés

### Bilan de l’équipe

### Bilans individuels

# ANNEXE 1 : Créer un serveur MySQL sur un Raspberry Pi

Nous voulions que l’ensemble des membres de l’équipe du projet puisse se connecter et travailler sur la base de données de manière simultanée. Nous avons donc décidé d’installer un serveur MySQL (version 5.7) sur un Raspberry pi 3. Ce dernier est connecté au domicile d’un membre de l’équipe et accessible depuis l’extérieur.

A l’aide de l’adresse IP publique du routeur domestique et à la création de comptes utilisateurs sur la base de données MySQL, les membres de l’équipe peuvent travailler sur la base de données en même temps.

Détaillons les étapes de déploiement de ce serveur sur un Raspberry pi 3.

Tout d’abord nous conseillons de mettre à jour l’ensemble du système d’exploitation de l’ordinateur.

Capture d’écranEnsuite on utilise la commande suivante pour récupérer le paquet d’installation du serveur MySQL version 5.7 (dernière version stable actuellement) :

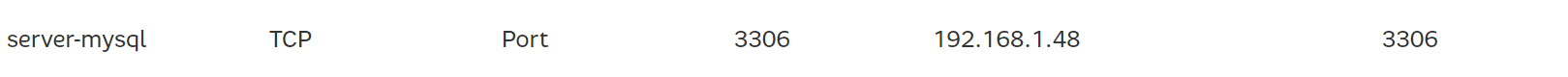
Capture d’écranAprès quelques minutes, le serveur MySQL est en place, pour le démarrer on utilise la commande :

Capture d’écranOn peut maintenant se connecter en local au serveur MySQL en utilisant la commande :

Capture d’écranUne fois connecté, on crée un compte utilisateur pour chaque membre de l’équipe du projet à l’aide de la clause GRANT.

Ici on attribue tous les privilèges à l’utilisateur florian qui peut se connecter depuis n’importe quelle adresse IP avec le mot de passe password.

Ensuite sur le routeur, on créer une règle de redirection du port 3306 (port MySQL par défaut) arrivant de l’extérieur vers l’adresse IP local du Raspberry comme le montre la capture d’écran suivante.



Grâce à cette règle de redirection peut maintenant se connecter à distance au serveur MySQL et donc à la base de données du projet. Pour l’administrer on pourra utiliser un terminal ou alors phpmyadmin.

L’installation de phpmyadmin requiert la mise en place d’un serveur Apache et la redirection du port 80.

# ANNEXE 2 : Script de création de la base de données

#------------------------------------------------------------

# Script MySQL.

#------------------------------------------------------------

#------------------------------------------------------------

# Table: Personne

#------------------------------------------------------------

**CREATE TABLE** Personne(

IDPersonne int (11) Auto\_increment NOT NULL ,

Nom Char (25) NOT NULL ,

Prenom Char (25) NOT NULL ,

Adresse Varchar (25) NOT NULL ,

**PRIMARY KEY** (IDPersonne )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: Ingredient

#------------------------------------------------------------

**CREATE TABLE** Ingredient(

IDproduit Int NOT NULL ,

**PRIMARY KEY** (IDproduit )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: Diluant

#------------------------------------------------------------

**CREATE TABLE** Diluant(

IDDiluant int (11) Auto\_increment NOT NULL ,

NomDiluant Char (25) NOT NULL ,

PrixDiluant Float NOT NULL ,

**PRIMARY KEY** (IDDiluant )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: Recipient

#------------------------------------------------------------

**CREATE TABLE** Recipient(

IDrecipient int (11) Auto\_increment NOT NULL ,

NomRecipient Varchar (255) ,

Prix Float NOT NULL ,

Stock Int ,

**PRIMARY KEY** (IDrecipient )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: Commande

#------------------------------------------------------------

**CREATE TABLE** Commande(

IDCommande int (11) Auto\_increment NOT NULL ,

DateCommande Date NOT NULL ,

Statut Char (25) ,

PrixTotal Float NOT NULL ,

IDPersonne Int NOT NULL ,

**PRIMARY KEY** (IDCommande )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: Fournisseur

#------------------------------------------------------------

**CREATE TABLE** Fournisseur(

IDFournisseur int (11) Auto\_increment NOT NULL ,

Nom Char (25) NOT NULL ,

Numero Int NOT NULL ,

**PRIMARY KEY** (IDFournisseur )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: Proposition

#------------------------------------------------------------

**CREATE TABLE** Proposition(

IDProposition int (11) Auto\_increment NOT NULL ,

Nom Char (25) NOT NULL ,

Temperature Int ,

Diluant Int ,

IDPersonne Int ,

**PRIMARY KEY** (IDProposition )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: Stock

#------------------------------------------------------------

**CREATE TABLE** Stock(

IDstock int (11) Auto\_increment NOT NULL ,

NbStock Int NOT NULL ,

Fraicheur Int ,

IDproduit Int ,

IDFournisseur Int ,

**PRIMARY KEY** (IDstock )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: Produit

#------------------------------------------------------------

**CREATE TABLE** Produit(

IDproduit int (11) Auto\_increment NOT NULL ,

NomProduit Varchar (255) ,

Prix Float ,

**PRIMARY KEY** (IDproduit )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: Preparation

#------------------------------------------------------------

**CREATE TABLE** Preparation(

Temperature Int ,

IDproduit Int NOT NULL ,

IDDiluant Int ,

IDPersonne Int ,

**PRIMARY KEY** (IDproduit )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: contient

#------------------------------------------------------------

**CREATE TABLE** contient(

QuantiteCommande Int ,

FraicheurVoulu Int ,

IDCommande Int NOT NULL ,

IDproduit Int NOT NULL ,

IDrecipient Int NOT NULL ,

**PRIMARY KEY** (IDCommande ,IDproduit ,IDrecipient )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: fournit

#------------------------------------------------------------

**CREATE TABLE** fournit(

IDFournisseur Int NOT NULL ,

IDproduit Int NOT NULL ,

**PRIMARY KEY** (IDFournisseur ,IDproduit )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: se compose prepa

#------------------------------------------------------------

**CREATE TABLE** se\_compose\_prepa(

QteIngredient Int ,

FraicheurMin Int ,

FraicheurMax Int ,

Temps Int ,

IDproduit Int NOT NULL ,

IDproduitFinal Int NOT NULL ,

**PRIMARY KEY** (IDproduit ,IDproduitFinal )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: se compose propo

#------------------------------------------------------------

**CREATE TABLE** se\_compose\_propo(

QteIngredient Int ,

FraicheurMin Int ,

FraicheurMax Int ,

Temps Int ,

IDProposition Int NOT NULL ,

IDproduit Int NOT NULL ,

**PRIMARY KEY** (IDProposition ,IDproduit )

)**ENGINE**=InnoDB;

#------------------------------------------------------------

# Table: fournit recipient

#------------------------------------------------------------

**CREATE TABLE** fournit\_recipient(

IDFournisseur Int NOT NULL ,

IDrecipient Int NOT NULL ,

**PRIMARY KEY** (IDFournisseur ,IDrecipient )

)**ENGINE**=InnoDB;

**ALTER TABLE** Ingredient **ADD CONSTRAINT** FK\_Ingredient\_IDproduit **FOREIGN KEY** (IDproduit) **REFERENCES** Produit(IDproduit);

**ALTER TABLE** Commande **ADD CONSTRAINT** FK\_Commande\_IDPersonne **FOREIGN KEY** (IDPersonne) **REFERENCES** Personne(IDPersonne);

**ALTER TABLE** Proposition **ADD CONSTRAINT** FK\_Proposition\_IDPersonne **FOREIGN KEY** (IDPersonne) **REFERENCES** Personne(IDPersonne);

**ALTER TABLE** Stock **ADD CONSTRAINT** FK\_Stock\_IDproduit **FOREIGN KEY** (IDproduit) **REFERENCES** Produit(IDproduit);

**ALTER TABLE** Stock **ADD CONSTRAINT** FK\_Stock\_IDFournisseur **FOREIGN KEY** (IDFournisseur) **REFERENCES** Fournisseur(IDFournisseur);

**ALTER TABLE** Preparation **ADD CONSTRAINT** FK\_Preparation\_IDproduit **FOREIGN KEY** (IDproduit) **REFERENCES** Produit(IDproduit) ON DELETE CASCADE ;

**ALTER TABLE** Preparation **ADD CONSTRAINT** FK\_Preparation\_IDDiluant **FOREIGN KEY** (IDDiluant) **REFERENCES** Diluant(IDDiluant);

**ALTER TABLE** Preparation **ADD CONSTRAINT** FK\_Preparation\_IDPersonne **FOREIGN KEY** (IDPersonne) **REFERENCES** Personne(IDPersonne);

**ALTER TABLE** contient **ADD CONSTRAINT** FK\_contient\_IDCommande **FOREIGN KEY** (IDCommande) **REFERENCES** Commande(IDCommande);

**ALTER TABLE** contient **ADD CONSTRAINT** FK\_contient\_IDproduit **FOREIGN KEY** (IDproduit) **REFERENCES** Produit(IDproduit);

**ALTER TABLE** contient **ADD CONSTRAINT** FK\_contient\_IDrecipient **FOREIGN KEY** (IDrecipient) **REFERENCES** Recipient(IDrecipient);

**ALTER TABLE** fournit **ADD CONSTRAINT** FK\_fournit\_IDFournisseur **FOREIGN KEY** (IDFournisseur) **REFERENCES** Fournisseur(IDFournisseur);

**ALTER TABLE** fournit **ADD CONSTRAINT** FK\_fournit\_IDproduit **FOREIGN KEY** (IDproduit) **REFERENCES** Produit(IDproduit);

**ALTER TABLE** se\_compose\_prepa **ADD CONSTRAINT** FK\_se\_compose\_prepa\_IDproduit **FOREIGN KEY** (IDproduit) **REFERENCES** Produit(IDproduit);

**ALTER TABLE** se\_compose\_prepa **ADD CONSTRAINT** FK\_se\_compose\_prepa\_IDproduitFinal **FOREIGN KEY** (IDproduitFinal) **REFERENCES** Produit(IDproduit) ON DELETE CASCADE ;

**ALTER TABLE** se\_compose\_propo **ADD CONSTRAINT** FK\_se\_compose\_propo\_IDProposition **FOREIGN KEY** (IDProposition) **REFERENCES** Proposition(IDProposition) ON DELETE CASCADE ;

**ALTER TABLE** se\_compose\_propo **ADD CONSTRAINT** FK\_se\_compose\_propo\_IDproduit **FOREIGN KEY** (IDproduit) **REFERENCES** Produit(IDproduit);

**ALTER TABLE** fournit\_recipient **ADD CONSTRAINT** FK\_fournit\_recipient\_IDFournisseur **FOREIGN KEY** (IDFournisseur) **REFERENCES** Fournisseur(IDFournisseur);

**ALTER TABLE** fournit\_recipient **ADD CONSTRAINT** FK\_fournit\_recipient\_IDrecipient **FOREIGN KEY** (IDrecipient) **REFERENCES** Recipient(IDrecipient);

# ANNEXE 3 : Manuel d’utilisation de la base de données

Afin que la sorcière Giselle puisse utiliser la base de données directement, nous avons recensé dans cette annexe l’ensemble des commandes qu’elle pourrait avoir besoin quotidiennement.

**Afficher et mettre à jour le stock à la réception de nouveaux ingrédients :**

Pour mettre à jour le stock des ingrédients on va devoir effectuer la commande avec la syntaxe suivante :

|  |
| --- |
| CALL Affichage\_stocks([”](https://dev.mysql.com/doc/refman/5.7/en/case.html)NomIngredient[”](https://dev.mysql.com/doc/refman/5.7/en/case.html" \o "14.6.5.1 CASE Syntax), QuantiteIngredient, Fraicheur) ; |

Par exemple :

On demande l’ingrédient « Alexandrite », on veut ajouter une quantité de 3 ingrédients et on souhaite que ces ingrédients disposent d’une fraicheur de 40.

Pour afficher le stock des ingrédients on va devoir effectuer la commande précédente mais sans paramètres.

**Consulter les commandes d’un client :**

Pour consulter les commandes d’une personne dans un premier temps nous allons afficher la table « Personne » pour pouvoir consulter les commandes d’une personne existante :

On va devoir effectuer la commande avec la syntaxe suivante pour consulter la commande d’un client défini :

|  |
| --- |
| CALL historique\_cmd(‘NomPersonne’) ; |

Par exemple :

On veut ici connaitre la ou les commandes de la personne portant le nom « Naessens »

**Supprimer un ingrédient arrivé à expiration :**

La suppression d’un ingrédient arrivé à expiration est automatisée tous les jours à minuit il est inutile d’effectuer une commande.

**Mettre à jour le stock après l’envoi d’une commande :**

Pour mettre à jour le stock après l’envoi d’une commande on va commencer par afficher la table « Commande » afin de choisir une commande existante, pour cela le statut de la commande doit être en préparation :

On va devoir effectuer la commande avec la syntaxe suivante pour :

|  |
| --- |
| CALL envoi\_commande(NumeroCommande); |

Par exemple :

On veut ici retirer le contenu de la commande « 2 » dans le stock afin de modifier le statut de cette commande.

**Ajouter une ou plusieurs nouvelles recettes :**

Pour ajouter une nouvelle recette on va utiliser la syntaxe suivante :

|  |
| --- |
| CALL ajouter\_proposition(‘NomPotion’, TemperatureDePreparation, ‘NomDiluant’,  ‘NomIngredient1’, QuantiteIngredient1, FraicheurMin1, FraicheurMax1, Temps1,  ‘NomIngredient2’, QuantiteIngredient2, FraicheurMin2, FraicheurMax2, Temps2,  ‘NomIngredient3’, QuantiteIngredient3, FraicheurMin3, FraicheurMax3, Temps3,  ‘NomInventeur’) ; |

Par exemple :

# ANNEXE 4 : Ressources

Outils de gestion de projet utilisé (WBS, Gantt, Plannings, listes des tâches à réaliser…) :

* <https://www.matchware.com/fr/logiciel-de-mind-mapping>
* <https://trello.com/>

Ressources pour la gestion de projet :

* <http://www.workbreakdownstructure.fr/>
* <http://www.blog-gestion-de-projet.com/conseils-pour-reussir-votre-planification-de-projet/>

La méthode MERISE :

* <http://perso.modulonet.fr/~placurie/Cgo1.htm> (chapitres 5 à 8)

Logiciel de formalisation du MCD, MLD et MPD de la méthode Merise :

* <http://www.jfreesoft.com/JMerise/>

Documentation officielle de MySQL :

* <https://dev.mysql.com/doc/refman/5.7/en/>

Requêtes SQL :

* SQL, *Ronald R. Plew & Ryan K. Stephens, édition CampusPress, traduction de l’ouvrage Teach Yourself SQL in 24 Hours – Second Edition.*
* <https://sql.sh/>

1. Tous les documents concernant la modélisation de la base de données sont disponibles dans le dossier modélisation sur le GitHub du projet. [↑](#footnote-ref-1)
2. Pour plus d’informations sur la création du serveur MySQL sur un Raspberry Pi 3 se reporter à l’annexe 1 [↑](#footnote-ref-2)
3. Comma-separated values permet de représenter les données d’un tableau sous formes de valeurs séparées par des virgules. [↑](#footnote-ref-3)
4. Le manuel d’utilisation en annexe 3 reprend le rôle et le mode d’emploi de ces procédures [↑](#footnote-ref-4)