

**Rapport du Projet RAD**

Gestion de la vente de vin

Table des matières

[1. Introduction 3](#_Toc463413615)

[1.1. Description du projet 3](#_Toc463413616)

[1.2. Description de l’application 3](#_Toc463413617)

[1.3. Objectifs 3](#_Toc463413618)

[2. Respect des Contraintes 3](#_Toc463413619)

[2.1. Types de données 3](#_Toc463413620)

[2.2. Association 1.n 4](#_Toc463413621)

[2.3. Association n.n 4](#_Toc463413622)

[2.4. Association réflexive 4](#_Toc463413623)

[2.5. Séquences, triggers et contraintes 4](#_Toc463413624)

[3. Application 4](#_Toc463413625)

[3.1. Connexion 4](#_Toc463413626)

[3.2. Modèles de données 4](#_Toc463413627)

[3.2.1. Modèle conceptuel de données / Logical Model 5](#_Toc463413628)

[3.2.2. Modèle logique de données / Relational Model 6](#_Toc463413629)

[3.3. Fonctionnalités choisies 6](#_Toc463413630)

[3.4. Meilleure fonctionnalité 7](#_Toc463413631)

[4. Conclusion 7](#_Toc463413632)

[5. Références 7](#_Toc463413633)

# Introduction

## Description du projet

Notre projet est une application de gestion pour la vente de vins, *Gesto’Vin*. Notre entreprise se propose en tant que revendeur pour les particuliers produisant leur propre vin. Nous agissons donc en tant qu’intermédiaire entre ceux-ci et les clients.

## Description de l’application

*Gesto’Vin* comprend tout d’abord la gestion des utilisateurs, c’est-à-dire les collaborateurs qui auront accès à l’application, car ce sont eux qui entreront les commandes des clients dans le système. Ils ont bien évidemment accès aux fonctions d’ajout, de modification, de suppression et de visualisation du contenu.

Les formulaires et états sont créés en fonction de nos tables et associations (modèle de données).

Un graphique permet aux utilisateurs de visualiser les ventes de vin par canton à des fins statistiques.

Nous utilisons également la journalisation des enregistrements afin de garder des traces sur certaines informations (les commandes des clients par exemple).

## Objectifs

Notre objectif pour cette application de gestion est d’entrer les commandes des clients dans le système afin que tout soit informatisé et centralisé.

Les utilisateurs pourront par la suite effectuer des statistiques afin de prendre différentes décisions quant aux ventes de vin.

# Respect des Contraintes

## Types de données

Les types de données les plus utilisés sont représentés. Voici quelques exemples d’utilisation :

* **NUMBER :** *Clés (primaires et secondaires), quantités, …*
* **VARCHAR2 :** *Noms, descriptions, …*
* **DATE :** *Dates des commandes*
* **LOB (BLOB ou CLOB) :** *Photos des vins*

## Association 1.n

Des associations 1.n sont présentes dans notre modèle. Par exemple, pour la liaison entre un fournisseur et un vin : ***un producteur produit 1 ou plusieurs vins***.

## Association n.n

Deux associations de type n.n sont également présentes au niveau du modèle de l’application. Elles se trouve au niveau des commandes de vin par le client (***un client commande 1 ou plusieurs vins. Un vin est commandé par 1 ou plusieurs clients***) et au niveau de la définition du prix en fonction du volume de la bouteille et du vin. Les tables associatives *Commande* et *Bouteille* sont donc créées.

## Association réflexive

Il est possible pour un fournisseur (particulier) d’avoir un représentant qui se charge du contact avec notre entreprise. Cela représentera une association réflexive au niveau de la table *Personne*.

## Séquences, triggers et contraintes

Chaque clé primaire est alimentée par une séquence. Cela est possible grâce à des triggers présents sur chaque table.

Les noms des contraintes respectent les standards définis dans la donnée du cas pratique.

# Application

## Connexion

Pour se connecter à l’application, nous utilisons une table d’utilisateur qui se trouve dans la base de données. La table contient tous les utilisateurs qui pourrons, par la suite, se connecter à notre application de gestion.

|  |  |
| --- | --- |
| ***Username*** | ***Password*** |
| ***murielle*** | *mumu16* |
| ***jean*** | *jeje16* |
| ***paule*** | *papa16* |

Dans cette application, nous ne faisons pas de distinction en les utilisateurs et les administrateurs car notre entreprise est une petite structure avec seulement quelques collaborateurs.

### Sécurité

Pour plus de sécurité, tous les mots de passe de la base de données sont hachés avant d’être stockés via le package ***API\_AUTHENTIFICATION*** présent dans notre base de données.

## Modèles de données

Les modèles de données ont été réalisés à l’aide du logiciel *Oracle SQL Developer Data modeler*.

### Modèle conceptuel de données / Logical Model

### Modèle logique de données / Relational Model

## Journalisation

Les informations des tables jugées importantes sont journalisées dans d’autres tables prévues à cet effet. Les tables journalisées sont les suivantes :

* ***Clients***
* ***Commandes***
* ***Personnes***
* ***Vins***

Cependant, les informations concernant les images liées aux vins ainsi que les images elles-mêmes ne sont pas journalisées car, en plus d’occuper beaucoup d’espace sur la base de données, nous avons jugé inutile de les consigner.

## Fonctionnalités choisies

Parmi les diverses fonctionnalités proposées dans le cas pratique, nous avons bien évidemment mis en place les fonctionnalités obligatoires :

* ***Etat classique et/ou état interactif***
* ***Formulaire simple et/ou formulaire tabulaire***
* ***Modèles d’authentification et d’autorisation***

De plus, nous avons intégré les fonctionnalités suivantes :

* ***Graphique***
* ***Image (logo)***

## Meilleure fonctionnalité

## Problèmes rencontrés / Solutions trouvées

### Nommage de champ Date

Dans notre table *Commandes*, le champ censé contenir la date à laquelle a été passée la commande était nommé *Date*. En conséquence, nous avons rencontré des problèmes avec nos triggers lorsque ceux-ci souhaitaient créer un type de donné repris de ce champ (*.DATE%TYPE*).

Il a été nécessaire de modifier le nom du champ et générer à nouveau les triggers de cette table pour que le problème se résolve.

### Modification du modèle relationnel

Lorsque nous souhaitions mettre en place la fonctionnalité permettant de passer une commande, nous nous sommes aperçu d’une erreur présente dans notre modèle : une ligne de commande était reliée à la table *Vins* et non à la table *Bouteilles* (celle-ci est une table associative composé du vin et de la dimension de la bouteille choisie). Nous avons donc modifié notre modèle relationnel en conséquence mais le code SQL créé générait des erreurs lors de son exécution. Nous avons effectué plusieurs tentatives afin de recréer notre base de données correctement (régénération des modèles, désactivation de contraintes, etc…) mais les erreurs persistaient.

La solution a été d’exécuter les morceaux de codes les uns après les autres, dans un ordre différent et non en exécutant le script SQL comme généré par Oracle Data Modeler. Tout d’abord, nous avons créé les séquences ainsi que les tables, ensuite les triggers, les déclarations de nos packages puis leurs corps contenant les implémentations.

### Tablespace plein

Lors de modifications sur notre table *Vins*, nous avons eu un problème lié à un tablespace : celui-ci était plein et ne permettait plus de faire de modification dans la base de données. L’erreur nous indiquait qu’il était impossible d’étendre la capacité de stockage de celui-ci.

Pour pallier à ce problème, nous avons supprimé les champs *Noming*, *Type\_mime*, *Date\_modif\_img* et *Image* de notre table *Vins\_JN* car ceux-ci stockaient des informations relatives aux images insérées pour nous vins. Le champ *Image* stockait le fichier sous forme de Blob et occupait donc un espace considérable, en plus de ne pas être très utile en termes de traçabilité.

# Conclusion

# Références

Camus, Fabrice. *AGL : Données – MCD*, février 2016.

Camus, Fabrice. *AGL : Données – MLDR-MPD*, mars 2016.