19/06/2019

PROJET RAPIZZ

(3I-IN-10)

Sarah CE-OUGNA (E3FI)

Pierre PERRIER-RIDET (E3FI)

ESIEE PARIS

Sommaire

I – Introduction

a) Contexte

RaPizz est une entreprise de fabrication et livraison de pizzas, pour laquelle une application de gestion et suivi d'activité a été développée.

Le programme permet aux clients de s'inscrire, d'ajouter du solde et de commander une ou plusieurs pizzas parmi le catalogue proposé. Elle permet également aux livreurs de valider leur temps de livraison, et aux administrateurs de créer de nouvelles pizzas ainsi que de suivre l'activité de leur société.

L'application a été développée en JAVA, en se basant sur une base de données MySQL. La première partie de ce rapport expliquera donc la création de la base de données et de ses composantes, tandis que la deuxième partie portera sur la structure de l'application. Enfin des exemples du code en JAVA seront donnés en annexe.

b) Vidéo

Une vidéo est fournie avec le présent rapport, voici les fonctionnalités qu'elle présente :

00:00 – 00:45 : Création d'un nouveau compte client (client1)

00:45 – 02:12 : Connexion en tant que client1, ajout de crédit et nouvelle commande

02:12 – 03:40 : Connexion en tant que client2, mise à jour des informations personnelles et nouvelle commande

03:40 – 04:38 : Connexion en tant que client3, commande de la 10ème pizza (gratuite)

04:38 : 07:14 : Connexion en tant qu'administrateur, vue sur les statistiques, consultation des 3 nouvelles fiches de livraison, création d'une nouvelle pizza

07:14 – 08:15 : Connexion en tant que livreur1, confirmation des livraisons pour les clients 1 et 3

08:15 – 09:05 : Connexion en tant que client1, vérification du statut de la commande et du débit sur le solde, vérification de la nouvelle pizza créée par l'administrateur

09:05 – 09:52 : Connexion en tant que client3, vérification du statut de la commande et de la non-facturation pour fidélité

09 :52 – 10:28 : Connexion en tant que livreur2, confirmation de la livraison du client2, avec simulation d'un retard >30min

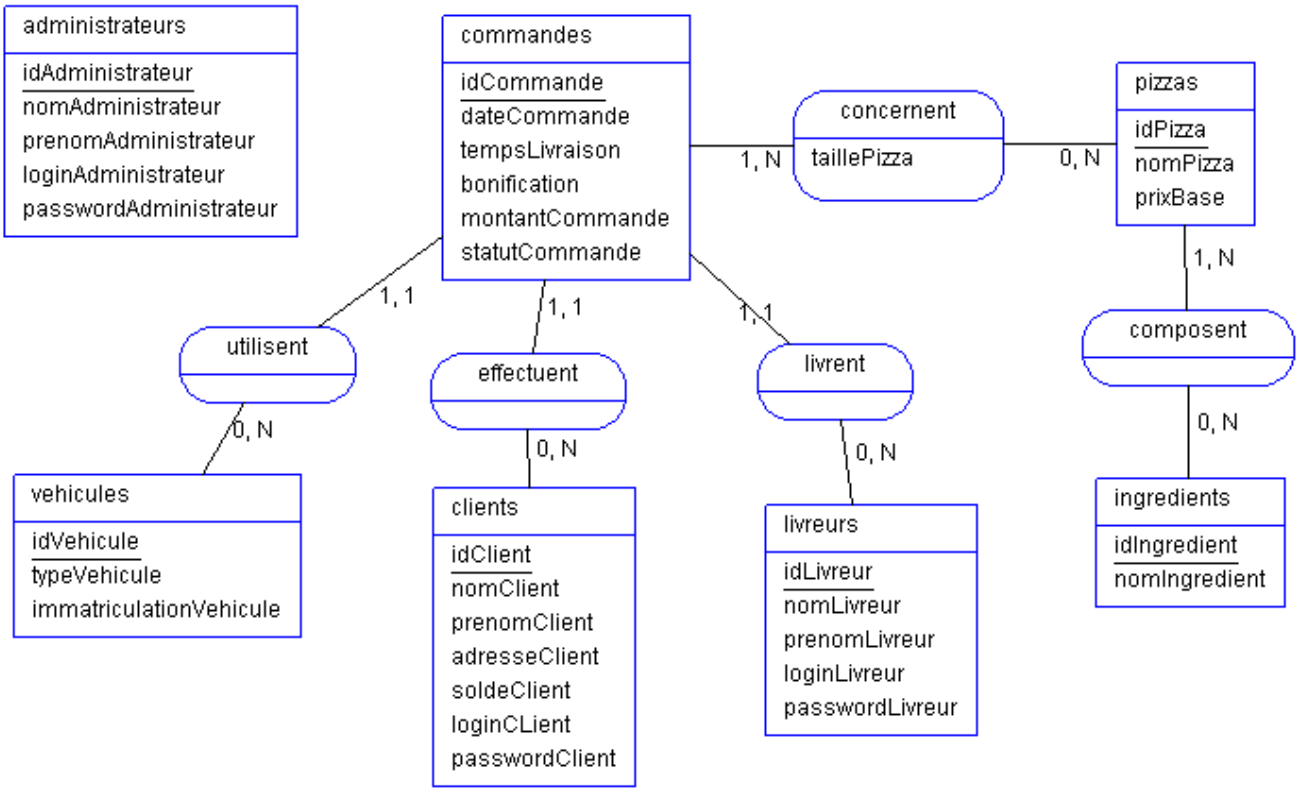
10:28 – 11:05 : Connexion en tant que client2, vérification du statut de la commande et de la non-facturation pour retard

11:05 – 11:43 : Connexion en tant qu'administrateur, vue sur les nouvelles statistiques

II – Base de données

La base de données a été créée à l'aide de *MySQL Workbench* ; elle est hébergée sur un serveur MySQL local. L'outil *HeidiSQL* a aussi permis de gérer les comptes et droits des membres de l'équipe sur le serveur et la base de données.

a) MCD



b) MLD

```
administrateurs (idAdministrateur, nomAdministrateur, prenomAdministrateur,
loginAdministrateur, passwordAdministrateur)

commandes (idCommande, dateCommande, tempsLivraison, bonification,
montantCommande, statutCommande, #idVehicule, #idClient, #idLivreur)

vehicules (idVehicule, typeVehicule, immatriculationVehicule)

clients (idClient, nomClient, prenomClient, adresseClient, soldeClient,
loginCLient, passwordClient)

livreurs (idLivreur, nomLivreur, prenomLivreur, loginLivreur,
passwordLivreur)

pizzas (idPizza, nomPizza, prixBase)

ingredients (idIngredient, nomIngredient)

commandes_pizzas (#idCommande, #idPizza, taillePizza)

pizzas_ingredients (#idPizza, #idIngredient)
```

c) Script de création des tables

```
CREATE DATABASE IF NOT EXISTS `rapizz`;
USE `rapizz`;

-- Table `ingredients`
DROP TABLE IF EXISTS `ingredients`;
CREATE TABLE `ingredients` (
  `idIngredient` int(11) NOT NULL AUTO_INCREMENT,
  `nomIngredient` varchar(45) NOT NULL,
  PRIMARY KEY (`idIngredient`)
);

-- Table `pizzas`
DROP TABLE IF EXISTS `pizzas`;
CREATE TABLE `pizzas` (
  `idPizza` int(11) NOT NULL AUTO_INCREMENT,
  `nomPizza` varchar(45) NOT NULL,
  `prixBase` float NOT NULL,
  PRIMARY KEY (`idPizza`)
);

-- Table `pizzas_ingredients`
DROP TABLE IF EXISTS `pizzas_ingredients`;
CREATE TABLE `pizzas_ingredients` (
  `idPizza` int(11) NOT NULL,
  `idIngredient` int(11) NOT NULL,
  PRIMARY KEY (`idPizza`,`idIngredient`),
  KEY `idx_pizza_ingredient` (`idPizza`),
  KEY `idx_ingredient_pizza` (`idIngredient`),
  CONSTRAINT `fk_ingredient_pizza` FOREIGN KEY (`idIngredient`)
REFERENCES `ingredients` (`idIngredient`),
  CONSTRAINT `fk_pizza_ingredient` FOREIGN KEY (`idPizza`) REFERENCES
`pizzas` (`idPizza`)
);

-- Table `clients`
DROP TABLE IF EXISTS `clients`;
CREATE TABLE `clients` (
  `idClient` int(11) NOT NULL AUTO_INCREMENT,
  `nomClient` varchar(45) NOT NULL,
  `prenomClient` varchar(45) NOT NULL,
  `adresseClient` varchar(200) NOT NULL,
  `soldeClient` float NOT NULL DEFAULT '0',
  `loginClient` varchar(45) NOT NULL,
  `passwordClient` varchar(45) NOT NULL,
  PRIMARY KEY (`idClient`)
);
```

```
-- Table `livreurs`
DROP TABLE IF EXISTS `livreurs`;
CREATE TABLE `livreurs` (
  `idLivreur` int(11) NOT NULL AUTO_INCREMENT,
  `nomLivreur` varchar(45) NOT NULL,
  `prenomLivreur` varchar(45) NOT NULL,
  `loginLivreur` varchar(45) NOT NULL,
  `passwordLivreur` varchar(45) NOT NULL,
  PRIMARY KEY (`idLivreur`)
);

-- Table `vehicules`
DROP TABLE IF EXISTS `vehicules`;
CREATE TABLE `vehicules` (
  `idVehicule` int(11) NOT NULL AUTO_INCREMENT,
  `typeVehicule` enum('Voiture','Moto') NOT NULL,
  `immatriculationVehicule` varchar(10) NOT NULL,
  PRIMARY KEY (`idVehicule`)
);

-- Table `commandes`
DROP TABLE IF EXISTS `commandes`;
CREATE TABLE `commandes` (
  `idCommande` int(11) NOT NULL AUTO_INCREMENT,
  `idClient` int(11) NOT NULL,
  `idLivreur` int(11) NOT NULL,
  `idVehicule` int(11) NOT NULL,
  `dateCommande` datetime NOT NULL,
  `tempsLivraison` int(11) NOT NULL DEFAULT '0',
  `bonification` enum('Retard','Fidélité') DEFAULT NULL,
  `montantCommande` float NOT NULL DEFAULT '0',
  `statutCommande` enum('En cours','Terminée') NOT NULL,
  PRIMARY KEY (`idCommande`),
  KEY `idx_commande_client` (`idClient`),
  KEY `idx_commande_livreur` (`idLivreur`),
  KEY `idx_commande_vehicule` (`idVehicule`),
  CONSTRAINT `fk_commande_client` FOREIGN KEY (`idClient`) REFERENCES
`clients` (`idClient`),
  CONSTRAINT `fk_commande_livreur` FOREIGN KEY (`idLivreur`) REFERENCES
`livreurs` (`idLivreur`),
  CONSTRAINT `fk_commande_vehicule` FOREIGN KEY (`idVehicule`)
REFERENCES `vehicules` (`idVehicule`)
);

-- Table `commandes_pizzas`
DROP TABLE IF EXISTS `commandes_pizzas`;
CREATE TABLE `commandes_pizzas` (
  `idCommande` int(11) NOT NULL,
  `idPizza` int(11) NOT NULL,
  `taillePizza` enum('Naine','Humaine','Ogresse') NOT NULL,
```

```
  PRIMARY KEY (`idCommande`,`idPizza`),
  KEY `idx_commande_pizza` (`idCommande`),
  KEY `idx_pizza_commande` (`idPizza`),
  CONSTRAINT `fk_commande_pizza` FOREIGN KEY (`idCommande`)
REFERENCES `commandes` (`idCommande`),
  CONSTRAINT `fk_pizza_commande` FOREIGN KEY (`idPizza`) REFERENCES
`pizzas` (`idPizza`)
);

-- Table `administrateurs`
DROP TABLE IF EXISTS `administrateurs`;
CREATE TABLE `administrateurs` (
  `idAdministrateur` int(11) NOT NULL AUTO_INCREMENT,
  `nomAdministrateur` varchar(45) NOT NULL,
  `prenomAdministrateur` varchar(45) NOT NULL,
  `loginAdministrateur` varchar(45) NOT NULL,
  `passwordAdministrateur` varchar(45) NOT NULL,
  PRIMARY KEY (`idAdministrateur`)
);
```

d) Script d'insertion des données fictives

```
-- Table `ingredients`
INSERT INTO `ingredients` VALUES
(1,'Tomates'),(2,'Mozzarella'),(3,'Olives'),(4,'Jambon'),(5,'Champignons'),(6,'Crème'),(
7,'Lardons'),(8,'Oignons'),(9,'Anchois'),(10,'Roquefort'),(11,'Emmental'),(12,'Chèvre'),
(13,'Pommes de terre'),(14,'Reblochon'),(15,'Origan'),(16,'Chorizo');

-- Table `pizzas`
INSERT INTO `pizzas` VALUES
(1,'Margarita',7.5),(2,'Reine',7.5),(3,'Carbonara',8.5),(4,'Napolitaine',8),(5,'Végétarien
ne',8),(6,'4 fromages',8.5),(7,'Calzone',9),(8,'Savoyarde',9),(14,'Chorizo',7.5);

-- Table `pizzas_ingredients`
INSERT INTO `pizzas_ingredients` VALUES
(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(2,4),(2,5),(3,2),(3,3),(3,6),(3,7),(3,8),(4,1),(4,2),(4,3),
(4,9),(4,15),(5,1),(5,2),(5,3),(5,5),(5,8),(5,11),(6,1),(6,2),(6,3),(6,10),(6,11),(6,12),(7,1)
,(7,2),(7,4),(7,5),(7,11),(8,2),(8,3),(8,6),(8,7),(8,8),(8,13),(8,14),(8,15),(14,1),(14,2),(1
4,3),(14,8),(14,16);

-- Table `clients`
INSERT INTO `clients` VALUES (1,'Bernard','Jeanne','12 rue Andrieux 75000
Paris',4.66667,'jeanne.bernard@gmail.com','test'),(2,'Thomas','Catherine','23 rue
Bayard 75000 Paris',2,'catherine.thomas@gmail.com','test'),(3,'Petit','Philippe','30
rue des Joncs',15,'philippe.petit@gmail.com','test'),(4,'Robert','Louis','3 avenue
Gabriel 75000 Paris',5,'louis.robert@gmail.com','test'),(5,'Richard','Jacques','1
impasse Dany 75000
Paris',1,'jacques.richard@gmail.com','test'),(6,'Durand','Daniel','54 rue de Naples
75000 Paris',12,'daniel.durand@gmail.com','test'),(7,'Dubois','Paul','6 allée Marcel-
Proust 75000 Paris',4,'paul.dubois@gmail.com','test'),(8,'Laurent','Hélène','29 rue
```

```
Vernet 75000 Paris',21.3334,'helene.laurent@gmail.com','test'),(9,'Simon','Nicolas','8
rue Saint-Honoré 75000
Paris',5.33333,'nicolas.simon@gmail.com','test'),(10,'Michel','Julien','14 rue de
Moscou 75000 Paris',20,'julien.michel@gmail.com','test'),(11,'Fregier','Benjamin','5
allée du Mail 75000
Paris',8.6667,'benjamin.fregier@gmail.com','test'),(12,'Moulin','Jean','3 rue des
poiriers 75000 Paris',2.5,'jean.moulin@gmail.com','test'),(13,'Jonathan','Prada','7 rue
des Lilas 75000 Paris',0,'jonathan.prada@gmail.com','test');

-- Table `livreurs`
INSERT INTO `livreurs` VALUES
(1,'Leroy','Eric','eric.leroy@gmail.com','test'),(2,'Roux','Olivier','olivier.roux@gmail.co
m','test'),(3,'Bonnet','Thomas','thomas.bonnet@gmail.com','test'),(4,'Fournier','Julie','j
ulie.fournier@gmail.com','test'),(5,'Girard','Marc','marc.girard@gmail.com','test');

-- Table `vehicules`
INSERT INTO `vehicules` VALUES (1,'Voiture','RT-327-HJ'),(2,'Voiture','VF-853-
MY'),(3,'Moto','NH-372-SD'),(4,'Moto','JZ-582-KJ'),(5,'Moto','YJ-245-QZ');

-- Table `commandes`
INSERT INTO `commandes` VALUES (1,1,1,1,'2019-06-16
04:45:19',49,'Retard',0,'Terminée'),(2,3,3,1,'2019-06-16
04:53:45',45,'Retard',0,'Terminée'),(3,7,5,5,'2019-06-16
05:14:03',27,NULL,16,'Terminée'),(4,6,1,5,'2019-06-16
05:19:02',19,NULL,5.66667,'Terminée'),(5,8,3,3,'2019-06-16
05:23:47',16,NULL,8.5,'Terminée'),(6,10,4,5,'2019-06-16
05:28:04',12,NULL,10,'Terminée'),(7,1,3,4,'2019-06-16
06:18:48',36,'Retard',0,'Terminée'),(8,8,3,1,'2019-06-17
20:00:57',34,'Retard',0,'Terminée'),(9,8,4,3,'2019-06-17
20:01:18',33,'Retard',0,'Terminée'),(10,8,1,5,'2019-06-17
20:01:30',34,'Retard',0,'Terminée'),(11,8,1,4,'2019-06-17
20:02:12',33,'Retard',0,'Terminée'),(12,8,5,3,'2019-06-17
20:02:20',18,NULL,11,'Terminée'),(13,8,3,4,'2019-06-17
20:02:29',32,'Retard',0,'Terminée'),(14,8,4,5,'2019-06-17
20:02:38',31,'Retard',0,'Terminée'),(15,8,5,5,'2019-06-17
20:02:50',18,NULL,11.3333,'Terminée'),(16,11,1,3,'2019-06-17
20:09:10',26,NULL,21.3333,'Terminée'),(17,6,4,4,'2019-06-17
20:14:45',19,NULL,5.33333,'Terminée'),(18,2,1,4,'2019-06-17
20:16:29',19,NULL,8,'Terminée'),(19,4,1,3,'2019-06-17
20:17:17',18,NULL,10,'Terminée'),(20,5,1,1,'2019-06-17
20:17:44',18,NULL,9,'Terminée'),(21,9,4,1,'2019-06-17
20:18:21',16,NULL,5.66667,'Terminée'),(22,1,4,1,'2019-06-17
20:18:49',15,NULL,5.33333,'Terminée'),(23,6,1,4,'2019-06-17
20:19:21',16,NULL,5,'Terminée'),(24,12,3,4,'2019-06-17
20:43:16',10,NULL,7.5,'Terminée');

-- Table `commandes_pizzas`
INSERT INTO `commandes_pizzas` VALUES
(1,3,'Naine'),(2,4,'Ogresse'),(3,1,'Humaine'),(3,6,'Humaine'),(4,3,'Naine'),(5,3,'Humai
ne'),(6,1,'Ogresse'),(7,8,'Humaine'),(8,3,'Humaine'),(9,3,'Ogresse'),(9,8,'Naine'),(10,8
```

```
,'Ogresse'),(11,8,'Naine'),(12,3,'Naine'),(12,4,'Naine'),(13,8,'Naine'),(14,4,'Humaine'),(
15,3,'Ogresse'),(16,6,'Ogresse'),(16,14,'Ogresse'),(17,5,'Naine'),(18,5,'Humaine'),(19
,2,'Ogresse'),(20,7,'Humaine'),(21,6,'Naine'),(22,4,'Naine'),(23,1,'Naine'),(24,2,'Humai
ne');

-- Table `administrateurs`
INSERT INTO `administrateurs` VALUES (1,'Martin','Benoît','admin','admin');
```

e) Requêtes spécifiques demandées

```
-- La carte
SELECT nomPizza AS 'PIZZA', CONCAT(ROUND(prixBase-(1/3*prixBase),2), ' €')
as 'NAINE', CONCAT(ROUND(prixBase,2), ' €') as 'HUMAINE',
CONCAT(ROUND(prixBase+(1/3*prixBase),2), ' €') as 'OGRESSE',
GROUP_CONCAT(nomIngredient) as 'INGREDIENTS'
FROM pizzas
INNER JOIN pizzas_ingredients ON pizzas.idPizza=pizzas_ingredients.idPizza
INNER JOIN ingredients ON
pizzas_ingredients.idIngredient=ingredients.idIngredient
GROUP BY nomPizza;

-- Fiche de livraison
SELECT CONCAT(prenomLivreur, ' ', nomLivreur) as 'LIVREUR', typeVehicule as
'VEHICULE', CONCAT(prenomClient, ' ', nomClient) as 'CLIENT', DATE(dateVente)
as 'DATE DE LA COMMANDE', CONCAT(GREATEST(tempsLivraison-30,0), ' min')
as RETARD, nomPizza as 'PIZZA', prixBase as 'PRIX (BASE)'
FROM ventes
INNER JOIN livreurs on ventes.idLivreur=livreurs.idLivreur
INNER JOIN vehicules ON ventes.idVehicule = vehicules.idVehicule
INNER JOIN clients ON ventes.idClient = clients.idClient
INNER JOIN pizzas ON ventes.idPizza = pizzas.idPizza;

-- Vehicules n'ayant jamais servi
SELECT typeVehicule, modeleVehicule, immatriculationVehicule
FROM vehicules
LEFT JOIN ventes ON ventes.idVehicule = vehicules.idVehicule
WHERE ventes.idVehicule IS NULL;

-- Nombre de commandes par client
SELECT CONCAT(prenomClient, ' ', nomClient) AS 'CLIENT', COUNT(idVente) AS
'NOMBRE DE COMMANDES'
FROM clients
INNER JOIN ventes ON ventes.idClient = clients.idClient
GROUP BY ventes.idClient;

-- Moyenne des montants des commandes
SELECT ROUND(AVG(montantCommande),2) AS 'MOYENNE DES COMMANDES'
FROM ventes;
```

```
-- Clients ayant commandé plus que la moyenne
SELECT CONCAT(prenomClient, ' ', nomClient) AS 'CLIENT',
ROUND(AVG(montantCommande),2) AS 'MOYENNE DES COMMANDES'
FROM clients
INNER JOIN ventes ON ventes.IdClient = clients.idClient
GROUP BY ventes.idClient
HAVING AVG(montantCommande) > (SELECT AVG(montantCommande) FROM
ventes);
```

f) Procédures

Quatre procédures ont été créées afin d'afficher les données statistiques dans l'application.

```
-- Meilleur client
DROP PROCEDURE IF EXISTS meilleurClient;
DELIMITER $$
CREATE PROCEDURE meilleurClient()
BEGIN
	SELECT CONCAT(prenomClient,' ',nomClient) as 'Client',
COUNT(idCommande) as 'Nombre de commandes',
ROUND(SUM(montantCommande),2) as 'Montant total'
	FROM commandes, clients
	WHERE commandes.idClient = clients.idClient
	GROUP BY commandes.idClient
	ORDER BY COUNT(idCommande) DESC;
END$$
DELIMITER ;

-- Pire Livreur
DROP PROCEDURE IF EXISTS pireLivreur;
DELIMITER $$
CREATE PROCEDURE pireLivreur()
BEGIN
	SELECT livreurs.idLivreur as id, CONCAT(prenomLivreur,' ',nomLivreur) as
'Livreur', (SELECT COUNT(idCommande) FROM commandes WHERE idLivreur=id
AND bonification='Retard') as 'Nombre de retard', ROUND(AVG(tempsLivraison)) as
'Temps de livraison moyen'
	FROM commandes, livreurs
	WHERE commandes.idLivreur = livreurs.idLivreur
	GROUP BY commandes.idLivreur
	ORDER BY  'Nombre de retard' DESC;
END$$
DELIMITER ;

-- Pizza favorite
DROP PROCEDURE IF EXISTS pizzaFavorite;
DELIMITER $$
CREATE PROCEDURE pizzaFavorite()
BEGIN
```

```
	SELECT nomPizza as 'Pizza', COUNT(idCommande) as 'Nombre de
commandes'
	FROM commandes_pizzas, pizzas
	WHERE commandes_pizzas.idPizza = pizzas.idPizza
	GROUP BY pizzas.idPizza
	ORDER BY COUNT(idCommande) DESC;
END$$
DELIMITER ;

-- Ingrédient favori
DROP PROCEDURE IF EXISTS ingredientFavori;
DELIMITER $$
CREATE PROCEDURE ingredientFavori()
BEGIN
	SELECT nomIngredient as 'Ingrédient', COUNT(idPizza) as 'Nombre de
pizzas'
	FROM pizzas_ingredients, ingredients
	WHERE pizzas_ingredients.idIngredient = ingredients.idIngredient
	GROUP BY ingredients.idIngredient
	ORDER BY COUNT(idPizza) DESC;
END$$
DELIMITER ;
```

g) Trigger

Enfin, un trigger a également été ajouté avant l'insertion d'une nouvelle commande, afin de gérer notamment la bonification de fidélité.

```
DELIMITER $$
DROP TRIGGER IF EXISTS setCommande $$

CREATE TRIGGER setCommande
   BEFORE INSERT ON commandes
   FOR EACH ROW
BEGIN
	IF ((MOD((SELECT COUNT(idCommande) FROM commandes WHERE
idClient = NEW.idClient)+1,10)=0) AND ((SELECT COUNT(idCommande) FROM
commandes WHERE idClient = NEW.idClient)>0)) THEN
   SET NEW.bonification = 'Fidelité';
   SET NEW.montantCommande = 0;
	END IF;
   SET NEW.statutCommande = 'En cours';
   SET NEW.tempsLivraison = 0;
   SET NEW.idLivreur=(SELECT idLivreur FROM livreurs ORDER BY rand() LIMIT
1);
   SET NEW.idVehicule = (SELECT idVehicule FROM vehicules ORDER BY rand()
LIMIT 1);
END$$
DELIMITER ;
```

III – Application JAVA

Afin de développer l'application, l'IDE Eclipse a été utilisée grâce au driver JDBC.

a) Singleton

Pour se connecter à la base de données, le pattern Singleton a été mis en place. Il permet de n'avoir qu'une seule et même instance de connexion pour toutes les données à récupérer. Cela prodigue un avantage au niveau de la sécurité, mais aussi au niveau du temps et du travail à la charge du serveur.

Pour le mettre en place, une classe appelée Singleton a été créée : son constructeur est privé, si bien qu'on ne peut s'y connecter en dehors de la classe même. La fonction *getConnection()* vérifie si une instance du Singleton existe déjà, et si non, en crée une. Le code de la classe est visible en Annexe 1.

La connexion que retourne le Singleton est ensuite utilisée par les autres classes qui interagissent avec la base de données.

b) DAOs

Parmi ces classes qui utilisent le Singleton se trouvent les DAOs créés pour cette application : étant donné les ressemblances des tables Clients, Livreurs, et Administrateur au niveau de leurs attributs, elles ont été rassemblées sous une classe User. Les classes Pizzas et Commandes ont aussi été rajoutées.

Ces trois entités ont chacune leur propre DAO, certaines fonctions concernant les ingrédients ont été ajoutée dans le DAOPizza en raison de leur lien avec cette table.

Dans le DAOUser, la fonction *checkAuth* s'occupe de vérifier si les identifiants entrés par l'utilisateur correspondent bien à un Client/Livreur/Administrateur dans la base de données. D'autres fonctions s'occupent de récupérer les utilisateurs, et de mettre à jour les tables. Le code du DAOUser est visible en Annexe 2. Les DAOPizza et DAOCommande permettent les mêmes types d'action sur les tables correspondantes. Les méthodes retournent des objets de type User, Pizza et Commande, cela permettant de se rapprocher du modèle des tables de la base de données.

c) Fenêtres et Contrôleurs

Pour la partie graphique, la bibliothèque SWING a été utilisée. Elle permet en effet d'ajouter des éléments graphiques, tels que des champs à remplir, des boutons, des tableaux etc… via une interface graphique. L'application compte en tout 14 fenêtres.

Un contrôleur a été mis en place pour chacune d'entre elles, permettant de gérer les champs et l'affichage des données. Pour la fenêtre de connexion par exemple, le contrôleur récupère les champs email et mot de passe remplis, et appelle la fonction *checkAuth* du DAOUser afin de vérifier un utilisateur correspond aux identifiants entrés, dans la table indiquée. Si ce n'est pas le cas, un message d'alerte s'affiche indiquant qu'une mauvaise combinaison identifiant/mot de passe a été entrée grâce à un objet de type JOptionPane. Si un User correspond au couple d'identifiants, alors une fenêtre d'accueil est ouverte, en fonction du statut de l'utilisateur indiqué.

Note : le type de champ PasswordField a été utilisé pour les mots de passe, car cela permet de ne pas les afficher en clair lorsqu'ils sont entrés par l'utilisateur.

Pour l'affichage des tableaux (scrollPane) des JTableModel ont été créés, permettant de définir le type de données à afficher, le nombre de colonnes, leur nom, etc… Le code du Contrôleur pour la page de nouvelle commande ainsi que le JTableModel correspondant au panier ont été rajoutés en Annexe 3 et Annexe 4.

La navigation entre les fenêtres se fait grâce à des Listeners ajoutés aux boutons. Des champs de types ComboBox et JTextField ont été utilisés pour l'affichage des données. Un exemple de code graphique est fourni en Annexe 5, correspondant à la page Login. L'application démarre tout simplement en créant un nouveau contrôleur login en lui passant en arguments une nouvelle fenêtre, une connexion par singleton et les DAOs nécessaires.

IV – Conclusion

L'application fonctionne de manière fluide et permet à la fois aux clients d'interagir avec l'entreprise, aux livreurs de justifier leur travail et aux responsables de suivre l'activité de RaPizz.

Une amélioration future pourrait être l'ajout de pizzas personnalisées, au choix des clients.

V – Annexes

Annexe 1 : Singleton

```
package rapizz;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Singleton {
		private Connection cnx;
		private static Singleton instance;

		private Singleton() {
			try {
				Class.forName("com.mysql.cj.jdbc.Driver");
			} catch (ClassNotFoundException e) {
				e.printStackTrace();
				System.out.println("PB Chargement Driver MySQL");
			}

			try {
				cnx =
DriverManager.getConnection("jdbc:mysql://localhost:3306/rapizz?user=user&password
=user&serverTimezone=UTC");
			} catch (SQLException e) {
				e.printStackTrace();
				System.out.println("PB Connexion à la BD");
			}
		}

		public static Singleton getInstance() {
			if(instance == null) {
				instance = new Singleton();
			}
			return instance;
		}

		public Connection getConnection() {
			return cnx;
		}
}
```

Annexe 2 : DAOUser

```
package rapizz;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class DAOUser {
	private Connection cnx;
	private String table;

	public DAOUser(Connection connect) {
		this.cnx = connect;
	}

	public User checkAuth(String statut, String email, String motdepasse) {
		PreparedStatement pstatement;
		ResultSet rs;
		User user=null;
		this.table = statut.toLowerCase()+"s" ;
		String loginField = "login"+statut;
		String passwordField = "password"+statut;

		String SQL = "SELECT * FROM "+this.table+" WHERE "+loginField+"=?
AND "+passwordField+"=?";
		try {
			pstatement = cnx.prepareStatement(SQL);
			pstatement.setString(1, email);
			pstatement.setString(2, motdepasse);

			rs = pstatement.executeQuery();

			user = new User();

			switch(statut) {
				case "Livreur":
					if(rs.next()==true) {
						user.setId(rs.getInt("idLivreur"));
						user.setNom(rs.getString("nomLivreur"));

	user.setPrenom(rs.getString("prenomLivreur"));

	user.setLogin(rs.getString("loginLivreur"));

	user.setPassword(rs.getString("passwordLivreur"));
					}
					break;
				case "Administrateur":
					if(rs.next()==true) {
						user.setId(rs.getInt("idAdministrateur"));

	user.setNom(rs.getString("nomAdministrateur"));

	user.setPrenom(rs.getString("prenomAdministrateur"));

	user.setLogin(rs.getString("loginAdministrateur"));
```

```
	user.setPassword(rs.getString("passwordAdministrateur"));
				}
				break;
			default:
				if(rs.next()==true) {
					user.setId(rs.getInt("idClient"));
					user.setNom(rs.getString("nomClient"));

	user.setPrenom(rs.getString("prenomClient"));

	user.setAdresse(rs.getString("adresseClient"));
					user.setSolde(rs.getFloat("soldeClient"));

	user.setLogin(rs.getString("loginClient"));

	user.setPassword(rs.getString("passwordClient"));
				}
				break;
			}



		} catch (SQLException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		return user;
	}

	public void UpdateClientInformations(User user) {
		PreparedStatement pStatement;
		String SQL="UPDATE "+this.table+" SET nomClient=?, prenomClient=?,
adresseClient=?, loginClient=?, passwordClient=? WHERE idClient=?";
		try {
			pStatement = cnx.prepareStatement(SQL);
			pStatement.setString(1, user.getNom());
			pStatement.setString(2, user.getPrenom());
			pStatement.setString(3, user.getAdresse());
			pStatement.setString(4, user.getLogin());
			pStatement.setString(5, user.getPassword());
			pStatement.setInt(6, user.getId());
			pStatement.execute();
		} catch (SQLException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public void UpdateSoldeClient(User user) {
		PreparedStatement pStatement;
		String SQL="UPDATE "+this.table+" SET soldeClient=? WHERE
idClient=?";
		try {
			pStatement = cnx.prepareStatement(SQL);
			pStatement.setFloat(1, user.getSolde());
			pStatement.setInt(2, user.getId());
```

```
                pStatement.execute();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        public void addClient(String nom, String prenom, String adresse, String
login, String password) {
            PreparedStatement pStatement;
            String SQL="INSERT INTO clients(nomClient, prenomClient,
adresseClient, loginClient, passwordClient, soldeClient) VALUES(?,?,?,?,?,?)";
            try {
                pStatement = cnx.prepareStatement(SQL);
                pStatement.setString(1, nom);
                pStatement.setString(2, prenom);
                pStatement.setString(3, adresse);
                pStatement.setString(4, login);
                pStatement.setString(5, password);
                pStatement.setFloat(6, 0);
                pStatement.execute();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        public void UpdateLivreurInformations(User user) {
            PreparedStatement pStatement;
            String SQL="UPDATE livreurs SET nomLivreur=?, prenomLivreur=?,
loginLivreur=?, passwordLivreur=? WHERE idLivreur=?";
            try {
                pStatement = cnx.prepareStatement(SQL);
                pStatement.setString(1, user.getNom());
                pStatement.setString(2, user.getPrenom());
                pStatement.setString(3, user.getLogin());
                pStatement.setString(4, user.getPassword());
                pStatement.setInt(5, user.getId());
                pStatement.execute();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        public List<String[]> meilleursClients(){
            PreparedStatement pStatement;
            ResultSet rs;
            List<String[]> meilleursclients = new ArrayList<>();
            String client="";
            String nbCommandes = "";
            String montantTotal = "";
            String SQL="CALL meilleurClient();";
            try {
                pStatement = cnx.prepareStatement(SQL);
                rs = pStatement.executeQuery();

                while(rs.next()) {
                    String [] infosClient = new String[3];
```

```
                        client = rs.getString("Client");
                        nbCommandes = rs.getInt("Nombre de commandes")+"";
                        montantTotal = rs.getFloat("Montant total")+" €";
                        infosClient[0]= client;
                        infosClient[1]= nbCommandes;
                        infosClient[2]= montantTotal;
                        meilleursclients.add(infosClient);
                  }

            } catch (SQLException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
            return meilleursclients;
      }

      public List<String[]> piresLireurs(){
            PreparedStatement pStatement;
            ResultSet rs;
            List<String[]> pireslivreurs = new ArrayList<>();
            String livreur="";
            String retards = "";
            String moyenne = "";
            String SQL="CALL pireLivreur();";
            try {
                  pStatement = cnx.prepareStatement(SQL);
                  rs = pStatement.executeQuery();

                  while(rs.next()) {
                        String [] infosLivreur= new String[3];
                        livreur = rs.getString("Livreur");
                        retards = rs.getInt("Nombre de retard")+"";
                        moyenne = rs.getFloat("Temps de livraison moyen")+"
min";

                        infosLivreur[0]= livreur;
                        infosLivreur[1]= retards;
                        infosLivreur[2]= moyenne;
                        pireslivreurs.add(infosLivreur);
                  }

            } catch (SQLException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
            return pireslivreurs;
      }

}
```

Annexe 3 : ControleurNouvelleCommande

```
package rapizz.Controleurs;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JOptionPane;
import rapizz.Commande;
import rapizz.CommandePizzasJTableModel;
import rapizz.DAOCommande;
import rapizz.DAOPizza;
import rapizz.DAOUser;
import rapizz.Pizza;
import rapizz.User;
import rapizz.Fenetres.FenetreAccueilClient;
import rapizz.Fenetres.FenetreLogin;
import rapizz.Fenetres.FenetreNouvelleCommande;
import java.sql.Timestamp;

public class ControleurNouvelleCommande {
	private User connectedUser;
	private float total = 0;
	private static DecimalFormat df = new DecimalFormat("0.00");

	public ControleurNouvelleCommande(FenetreLogin f0, FenetreAccueilClient f1,
FenetreNouvelleCommande f2, DAOUser daouser, User user, DAOCommande daocommande,
DAOPizza daopizza) {
		this.connectedUser = user;

		List<Pizza> pizzasObj = daopizza.getAllPizzas();
		List<String> pizzas = new ArrayList<>();

		for (Pizza p : pizzasObj) {
			String pizza = p.getNomPizza()+" :
"+daopizza.getIngredientsPizza(p.getIdPizza())+" => "+p.getPrixBase()+" €";
			pizzas.add(pizza);
		}

		f2.getComboBoxPizzas().setModel(new
DefaultComboBoxModel<>(pizzas.toArray()));
		f2.getComboBoxPizzas().setSelectedIndex(0);

		List<String> tailles = new ArrayList<>();
		tailles.add("Naine");
		tailles.add("Humaine");
		tailles.add("Ogresse");

		f2.getComboBoxTaille().setModel(new
DefaultComboBoxModel<>(tailles.toArray()));
		f2.getComboBoxTaille().setSelectedIndex(0);


		List<Pizza> data = new ArrayList<>();
		List<String> correspondingSizes= new ArrayList<>();
		List<Float> correspondingPrices= new ArrayList<>();
```

```
		f2.setVisible(true);

		f2.getBtnAjouter().addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				int pizzaIndex =
f2.getComboBoxPizzas().getSelectedIndex();
				int tailleIndex =
f2.getComboBoxTaille().getSelectedIndex();

				Pizza pizzaSelectionnee = pizzasObj.get(pizzaIndex);
				float prix =
daopizza.getPrixPizza(pizzaSelectionnee.getIdPizza(), tailles.get(tailleIndex));

				total += prix;
				f2.getTextPaneTotal().setText(df.format(total));

				data.add(pizzaSelectionnee);
				correspondingSizes.add(tailles.get(tailleIndex));
				correspondingPrices.add(prix);

				CommandePizzasJTableModel modele = new
CommandePizzasJTableModel(data, correspondingSizes, correspondingPrices,
daopizza);
				f2.getTable().setModel(modele);

				List<String> pizzaInfos =  new ArrayList<>();
				for (int i=0; i<data.size(); i++) {
					String s = data.get(i).getNomPizza()+"
"+correspondingSizes.get(i)+", "+correspondingPrices.get(i);
					pizzaInfos.add(s);
				}

				f2.getComboBoxRetirer().setModel(new
DefaultComboBoxModel<>(pizzaInfos.toArray()));
				if(pizzaInfos.size()>0) {
					f2.getComboBoxRetirer().setSelectedIndex(0);
				}
			}
		});

		f2.getBtnAnnuler().addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				f2.setVisible(false);
				f1.setVisible(true);
			}
		});


		f2.getBtnRetirerDuPanier().addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				int pizzaIndex =
f2.getComboBoxRetirer().getSelectedIndex();
```

```
                        data.remove(pizzaIndex);
                        correspondingSizes.remove(pizzaIndex);

                        total -= correspondingPrices.get(pizzaIndex);
                        correspondingPrices.remove(pizzaIndex);

                        f2.getTextPaneTotal().setText(df.format(total));

                        CommandePizzasJTableModel modele = new
CommandePizzasJTableModel(data, correspondingSizes, correspondingPrices,
daopizza);
                        f2.getTable().setModel(modele);

                        List<String> pizzaInfos =  new ArrayList<>();
                        for (int i=0; i<data.size(); i++) {
                              String s = data.get(i).getNomPizza()+"
"+correspondingSizes.get(i)+", "+correspondingPrices.get(i);
                              pizzaInfos.add(s);
                        }

                        f2.getComboBoxRetirer().setModel(new
DefaultComboBoxModel<>(pizzaInfos.toArray()));
                        if(pizzaInfos.size()>0) {
                              f2.getComboBoxRetirer().setSelectedIndex(0);
                        }

                  }
            });

            f2.getBtnCommander().addActionListener(new ActionListener() {

                  @Override
                  public void actionPerformed(ActionEvent e) {
                        if (JOptionPane.showConfirmDialog(null, "Confirmer la
commande ?", "WARNING",
                                    JOptionPane.YES_NO_OPTION) ==
JOptionPane.YES_OPTION) {
                              if(user.getSolde() >= total) {
                                    f2.setVisible(false);
                                    Commande commande = new Commande();
                                    commande.setIdClient(user.getId());
                                    commande.setDateCommande(new
Timestamp(System.currentTimeMillis()));
                                    commande.setMontantCommande(total);
                                    daocommande.addCommande(commande, data,
correspondingSizes);
                                    new ControleurAccueilClient(f0, f1,
daouser, connectedUser, daocommande, daopizza);
                              } else {
                                    JOptionPane.showMessageDialog(f2, "Solde
insuffisant !");
                              }

                        }
                  }
            });
      }

}
```

Annexe 4 : CommandePizzaJtableModel

```
package rapizz;
import java.util.List;
import javax.swing.table.DefaultTableModel;

public class CommandePizzasJTableModel extends DefaultTableModel {

	List<Pizza> data;
	List<String> sizes;
	List<Float> prices;
	DAOPizza daopizza;
	String[] nomColonnes = {"Pizza", "Taille", "Prix"};
	int taille=0;

	public CommandePizzasJTableModel(List<Pizza> l, List<String> s, List<Float>
p, DAOPizza daop) {
		data = l;
		sizes = s;
		prices = p;
		taille = data.size();
		this.daopizza = daop;
	}

	@Override
	public int getColumnCount() {
		return nomColonnes.length;
	}

	@Override
	public String getColumnName(int column) {
		return nomColonnes[column];
	}

	@Override
	public int getRowCount() {
		return taille;
	}

	@Override
	public Object getValueAt(int row, int column) {
		Pizza pizza = data.get(row);
		String taille = sizes.get(row);
		float prix = prices.get(row);

		Object val=null;
		switch(column) {
		case 0:
			val= pizza.getNomPizza();
			break;
		case 1:
			val = taille;
			break;
		case 2:
			val = prix;
			break;
		}
		return val;
	}
```

```
	@Override
	public boolean isCellEditable(int rowIndex, int columnIndex) {
		return false;
	}

}
```

Annexe 5 : FenetreLogin

```
package rapizz.Fenetres;
import java.awt.EventQueue;
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import java.awt.Font;
import javax.swing.SwingConstants;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JPasswordField;
import javax.swing.JComboBox;
import javax.swing.ImageIcon;

public class FenetreLogin extends JFrame {

	private JPanel contentPane;
	private JTextField txtLogin;
	private JButton btnConnexion;
	private JPasswordField passwordField;
	private JComboBox comboBox;
	private JButton btnInscription;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					FenetreLogin frame = new FenetreLogin();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}

	/**
	 * Create the frame.
	 */
	public FenetreLogin() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(750, 300, 450, 300);
		setResizable(false);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		contentPane.setLayout(new BorderLayout(0, 0));
		setContentPane(contentPane);

		JLabel lblAuthentification = new JLabel("Connexion");
		lblAuthentification.setHorizontalAlignment(SwingConstants.CENTER);
		lblAuthentification.setFont(new Font("Tahoma", Font.PLAIN, 20));
```

```
        contentPane.add(lblAuthentification, BorderLayout.NORTH);

        JPanel panel = new JPanel();
        contentPane.add(panel, BorderLayout.CENTER);
        panel.setLayout(null);

        JLabel lblLogin = new JLabel("Identifiant (Email) :");
        lblLogin.setBounds(10, 23, 131, 33);
        lblLogin.setHorizontalAlignment(SwingConstants.CENTER);
        lblLogin.setFont(new Font("Tahoma", Font.PLAIN, 15));
        panel.add(lblLogin);

        JLabel lblMotDePasse = new JLabel("Mot de passe :");
        lblMotDePasse.setFont(new Font("Tahoma", Font.PLAIN, 15));
        lblMotDePasse.setHorizontalAlignment(SwingConstants.CENTER);
        lblMotDePasse.setBounds(20, 70, 120, 40);
        panel.add(lblMotDePasse);

        txtLogin = new JTextField();
        txtLogin.setHorizontalAlignment(SwingConstants.CENTER);
        txtLogin.setBounds(151, 29, 245, 25);
        panel.add(txtLogin);
        txtLogin.setColumns(10);

        btnConnexion = new JButton("Connexion");
        btnConnexion.setActionCommand("Connexion");
        btnConnexion.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent arg0) {
              }
        });
        btnConnexion.setBounds(276, 170, 120, 33);
        panel.add(btnConnexion);

        passwordField = new JPasswordField();
        passwordField.setHorizontalAlignment(SwingConstants.CENTER);
        passwordField.setBounds(151, 80, 245, 25);
        panel.add(passwordField);

        btnInscription = new JButton("Inscription (Nouveau client)");
        btnInscription.setActionCommand("Connexion");
        btnInscription.setBounds(28, 170, 202, 33);
        panel.add(btnInscription);

        comboBox = new JComboBox();
        comboBox.setBounds(151, 127, 131, 20);
        panel.add(comboBox);

        ImageLabel bg = new ImageLabel(new ImageIcon("images/bg.png"));
        bg.setSize(225, 225);
        bg.setLocation(110, 0);
        panel.add(bg);
}
public JTextField getTxtLogin() {
        return txtLogin;
}
public JButton getBtnConnexion() {
        return btnConnexion;
}
public JPasswordField getPasswordField() {
```

```
		return passwordField;
	}
	public JComboBox getComboBox() {
		return comboBox;
	}
	public JButton getBtnInscription() {
		return btnInscription;
	}
}
```