Rapport Projet BD

WAGNER MARIE - MARMIER JULES - CONAN JEAN -
MEGHRAOUI ABDALLAH - SEKKAL AMINE - RODRIGUES THÉO

Introduction

L'objectif de ce projet est de formaliser une base de données à partir de l'énoncé afin de mettre en place un système de réservation de repas par groupe de 5-6. On se basera sur du SQL pour la base de données et du Java pour l'implémentation des interactions hommes-machines. Ce projet a duré 10 séances de 1h30 complétées par du travail personnel. Pour bien commencer ce projet nous avons mis en place notre environnement. Dans un premier temps nous avons créé un GitLab pour stocker la partie du projet qui est codée (Descripteur, Requêtes, Script SQL). Nous avons également fait un GoogleDrive pour mettre le brouillon, le rapport, le diagramme Entité/Associations… À cause de la panne informatique nous n'avions plus accès au GitLab. Pour pallier cela, nous avons donc transféré nos fichiers sur GitHub.

1.Étapes du Projet

- Analyse du sujet: Pour cela, nous avons chacun lu attentivement le sujet. Nous avons ensuite listé toutes les données mentionnées (cf Annexe).
- Contraintes: Nous avons listé les contraintes. Dans un premier temps, nous avons fait les contraintes sur les données comme le type (int, varchar..) mais aussi certaines valeurs comme celles pour les type de commandes ("livraison", "a emporter", "sur place"). Ensuite, nous avons écrit celles qu'il n'était pas possible de formaliser via des formules (cf Annexe).
- Dépendances fonctionnelles: Dans cette partie nous avons définies les relations entre attributs, ainsi que les multivaluations en prenant exemple sur les slides du cours (cf

Annexe). La partie la plus compliquée était d'être sûr que nous avions toutes les dépendances fonctionnelles.

- Diagramme Entités/Relations: Le diagramme est la partie où nous avons eu le plus de débat. En effet, nous n'avions pas tous la même manière de formaliser les relations. Par exemple pour les catégories à n profondeurs des restaurants. Nous avons finalement opté pour une méthode de relation mère/fille. C'est-à-dire qu' un restaurant est associé à une catégorie et on retrouve l'arbre correspondant grâce à une table de relations entre catégorie mère et catégorie fille. Ceci implique qu'il doit nécessairement exister un seul chemin à partir d'une catégorie fille(une catégorie de peut pas avoir 2 mères différentes). Pour faire les relations, nous nous sommes aidés de l'exemple du cours avec le capitaine Bouchbé, comme par exemple pour voir comment gérer le fait que les commandes puissent être de différents types (Livraison, A Emporter, Sur Place). Le schéma est en annexe.
- Création des tables: Avec le diagramme Entités/Relations, nous avons pu formaliser nos tables. Ainsi, nous avons repris les slides du cours et avons créé 13 tables. Elles sont toutes listées en Annexe.
- Formalisation relationnelle: Pour l'implémentation du script SQL de la base de données, nous avons utilisé l'IDE VSCode. Ce code se trouve dans le fichier projet_BD.sql. Nous avons pu implémenter une partie des contraintes directement dans les tables, afin de limiter au maximum des valeurs aberrantes dans les tables. Notamment, les contraintes sur le format des variables(date, prix non négatifs, chaine de caractère du longueur maximum…). Nous avons ensuite rempli en partie la base de données. Nous avons ajouté des valeurs cohérentes pour vérifier le bon fonctionnement du code mais également des valeurs volontairement fausses afin de tester nos fonctionnalités. Cette partie se trouve dans le fichier remplir_BD.sql.
- Implémentations des Requêtes: On a groupé nos requêtes dans le dossier "fonctionnalites" où on a implémenté les fonctionnalités nécessaires pour notre application.

- Connexion : Pour des raisons de factorisation du code, on a fait la partie connexion dans le démonstrateur java. Le client doit taper le bon identifiant ainsi que le mot de passe pour accéder à notre application.
- Parcours des Restaurants : on commence par demander à l'utilisateur les catégories de restaurants dans lesquelles il souhaite commander. S'il n'en donne, on prend alors les trois catégories où il a mis les meilleures notes (ou moins si il n'a pas mis des avis sur trois catégories différentes). Ensuite, on étend la liste des catégories pour aussi inclure récursivement les catégories filles. Comme il y a un risque de boucle infinie, un garde-fou fût rajouté (en plus des vérifications sur la base de données). On fait ensuite la sélection, puis on demande à l'utilisateur s'il souhaite filtrer pour un horaire précis. Comme le type gérant les horaires est un entier, on peut faire la comparaison dans une sélection, c'est pratique.
- Commande : On vérifie d'abord que le type de commande voulu par le client est bien disponible dans le restaurant choisi.Si le résultat de la requête est vide, on informe le client que malheureusement ce restaurant n'a pas ce type de commande. Ensuite, pour la deuxième étape, on vérifie , pour une commande sur place, qu'il y a bien assez de places au client pour réserver. Pour cela, on a réalisé des requêtes imbriquées à partir des relations "Commande" et "OuvertA" et "CommandeClient". Pour l'avant dernière étape, on associe un id_commande au client en select max(id_commande) et on incrémente de 1. Finalement, on insère dans les tables "Commande" et "CommandeClient" après avoir implémenter les méthodes nécessaires pour le calcul de leurs attributs.
- Les changements de statuts : de multiples vérifications sont en ordre, mais qui ne portent pas sur l'intégralité de la base, mais plutôt sur le bon sens. En effet, un client et un restaurant ne peuvent affecter les mêmes statuts, et un ordre logique de succession des statuts est à respecter.
- Évaluation : Cette class permet au client d'effectuer une évaluation, en effet, après avoir récupéré la note et l'avis du client, on ajoute celles-ci dans notre base de données avec l'identifiant adéquat. Après, on recalcule la note moyenne du

restaurant qui a probablement changé à l'issue de cette évaluation, et puis on fait une mise à jour de la note du restaurant.

- Le droit à l'oubli : Une fois que le client veut supprimer son compte, on fait appel à cette classe. Au début, on voulait juste changer l'identifiant du client en supprimant ses données, mais ce n'était pas possible vu que l'identifiant du client est une clé primaire. Pour cela on a dupliqué les données du client (les commandes qu'il a fait avec les évaluations) en lui donnant au même temps un nouveau identifiant (un nombre + le chiffre D, exemple : "16D") puis on a supprimé son mail, nom, prénom et mot de passe, et enfin on a décidé de supprimer aussi l'adresse de livraison puisqu'elle est personnelle au client.

- Démonstrateur JAVA: Pour le démonstrateur, on a implémenté une interface textuelle dans le fichier "Demonstrateur.java". L'exécution de ce fichier fait ouvrir l'application dans le terminal shell. Une fois que le client ou le restaurant entre dans l'application, il peut avoir accès aux différentes fonctionnalités qu'on a fourni.

2.Forme Normale de la base de donnée

Afin de valider le résultat obtenu, nous avons vérifié la forme normale de nos tables.

- 1 FN: Nos tables sont 1FN car toutes les valeurs sont atomiques. Il s'agit uniquement de chaînes de caractères et nombre. En effet, les dates sont des nombres correspondant au nombre de secondes depuis une date donnée (1er janvier 1970, il s'agit de la valeur associée au getTime() de java). De plus, aucune valeur est NULL, pour l'adresse de livraison d'une commande sur place ou à emporter, ce sera l'adresse du restaurant.
- 2FN: Nous avons des tables 1FN et de plus, tous les attributs non clés dépendent pleinement d'attributs clés. Pour l'exemple de la table Restaurant, nous avons comme clé le mail du restaurant qui est unique. Ainsi, tous les attributs non clés (nom, adresse, téléphone…) dépendent de cette clé car en connaissant uniquement le mail du restaurant, tous les autres attributs en découlent. Donc les tables sont 2FN.
- 3FN: Nous avons des tables 2FN. En outre, pour chaque attribut non clés, il ne dépend pas d'autres attributs non clés. Prenons l'exemple de la table Commande, une adresse

de livraison ne va pas permettre de retrouver la date ou l'id commande. De même pour tous les autres attributs non clés. Ainsi, nos tables sont toutes 3FN.

- 3FNBCK: Nous avons donc des tables 3FN. Pour prouver qu'elles sont aussi 3FNBCK, il faut reprendre nos dépendances fonctionnelles et vérifier qu'elles ont toutes au moins une clé dans les ensembles à gauche de la flèche (voir Annexe). C'est ce que nous avons fait et c'est effectivement le cas. Ainsi, nos tables sont 3FNBCK.

3.Répartition du travail

Pour l'analyse du sujet et des contraintes nous avons décidé de rester ensemble afin d'avoir l'avis de tous sur le sujet. Ceci nous a été utile afin que nous ayons tous la même compréhension du problème. Par la suite nous avons fait deux équipes de trois. La première équipe (Jules, Théo, Marie) à fait les parties script SQL avec la création des tables et leur remplissage, et la deuxième à implémenter les fonctionnalités et requêtes SQL. (Jean, Amine, Abdallah). Comme l'équipe chargée de la partie SQL avait fini, Théo et Jules ont commencé le rapport et Marie à fait le descripteur, ainsi que la preuve de la forme normale de nos tables. Ainsi, l'autre partie du groupe à pu débugger la partie java car ils avaient besoin des script SQL pour tester leur code.

4. Conclusion

À travers ce projet nous avons assimilé les compétences demandées en SQL et en Java. De plus ce projet n'était pas seulement un projet technique mais aussi humain. Nous nous sommes en effet répartis le travail de manière consciencieuse et organisée. Ce qui nous a permis de travailler en parallèle sur la partie SQL et Java et a facilité une mise en commun qui de fait nous a paru plus simple que ce que l'on pensait. Nous aurions cependant préféré finir totalement le projet, la phase de débogage était actuellement en cours lors de la panne informatique même si certaines fonctionnalités comme le "Droit à l'Oubli" et la connexion à l'application étaient fonctionnelles.

5.Annexe

Liste des Données:

Restaurant :

mail resto (varchar 100)
nom resto(varchar 100)
tel resto (varchar 15)
adresse resto (varchar 100)
liste de catégorie
horaires_midi_debut (int)
horaires_midi_fin (int)
horaires_soir_debut (int)
horaires_soir_fin (int)
nombre de places (unsigned int)
menu(liste de plats)
type de commande (à emporter, sur place, livraison)
note resto(float entre 0 et 5, moyenne des notes des évaluations)

Plat :

id plat (int unique)
id restaurant (int)
nom plat(varchar 100)
description plat (varchar 100)
prix plat (float > 0)
liste allergènes(optionnel) (varchar 100) -> si pas de liste d'allergènes la valeur = NULL)

Client :

id utilisateur(mail) (varchar unique 100)
mdp (varchar 100)
nom client (varchar 30)
prenom client (varchar 30)
adresse client (varchar 100)

Commande :

id commande (int unique) -> peut importe le resto (unique pour tous)
date commande (int)

heure commande (int)
heure terminée (int après heure commande ou heure avant l'heure commande et date terminée après date commande)
date terminée (égal ou après date commande)
client commande (id_client) (int)
restaurant commande (id restaurant)
adresse livraison -> on peut livrer pas chez soi
type commande (à emporter, sur place, livraison) (correspond au type de commande de id_restaurant)
contenu (liste d'id de plat), (int)
prix commande (unisgned float)
statut(attente de confirmation, validée, disponible, en livraison, annulée par le client, annulée par le restaurant, terminé, en préparation, litige)

Evaluation :

id évaluation(id evaluation = id commande car un seul client par commande) (int unique)
date évaluation (int)
avis evaluation (varchar 100)
note évaluation (entre 0 et 5) (float > 0)

Contraintes Relationnelles:

- ➔ Par service et par restaurant: le nombre de commandes de type sur place doit être au nombre de places du restaurant.
- ➔ Le type d'une commande d'un restaurant doit être dans les types de commande possible du restaurant.
- ➔ Pour une commande non en livraison, on ne peut pas avoir le statut en livraison.
- ➔ L'heure et la date d'évaluation doit être après l'heure et la date de la commande terminée.
- ➔ Si la livraison est sur place ou à emporter alors l'adresse client est nulle, sinon on doit mettre une adresse(évite de faire des livraisons sans adresse).
- ➔ Si la date de la commande ne correspond pas aux horaires d'ouverture du restaurant on ne peut pas passer de commandes.
- ➔ Si on veut supprimer le compte, les infos sont supprimées mais on garde l'évaluation avec un nouveau id client généré(on supprime tout et pour l'évaluation avec son id on change l'id et on le remplace par le nouvel id)
- ➔ Pour les catégories: 2 catégories différentes ont un nom différent(un seul arbre pour une catégorie donnée). Exemple: gastronomique → français ou bistrot → francais

Dépendances Fonctionnelles:

- Id resto → nom_resto, tel_resto, adresse_resto, nbr_places, notes_resto
- ->>type_commande
- ->>évaluations_resto
- ->> catégorie
- ->> plages horaires(plus flexible, peut tout décrire)
- ->> id_plat

- Id plat→nom plat, description plat
- ->> liste allergènes
- ->>(id_restaurant, prix_plat) //car le même plat mais le prix est différents dans différents restaurants

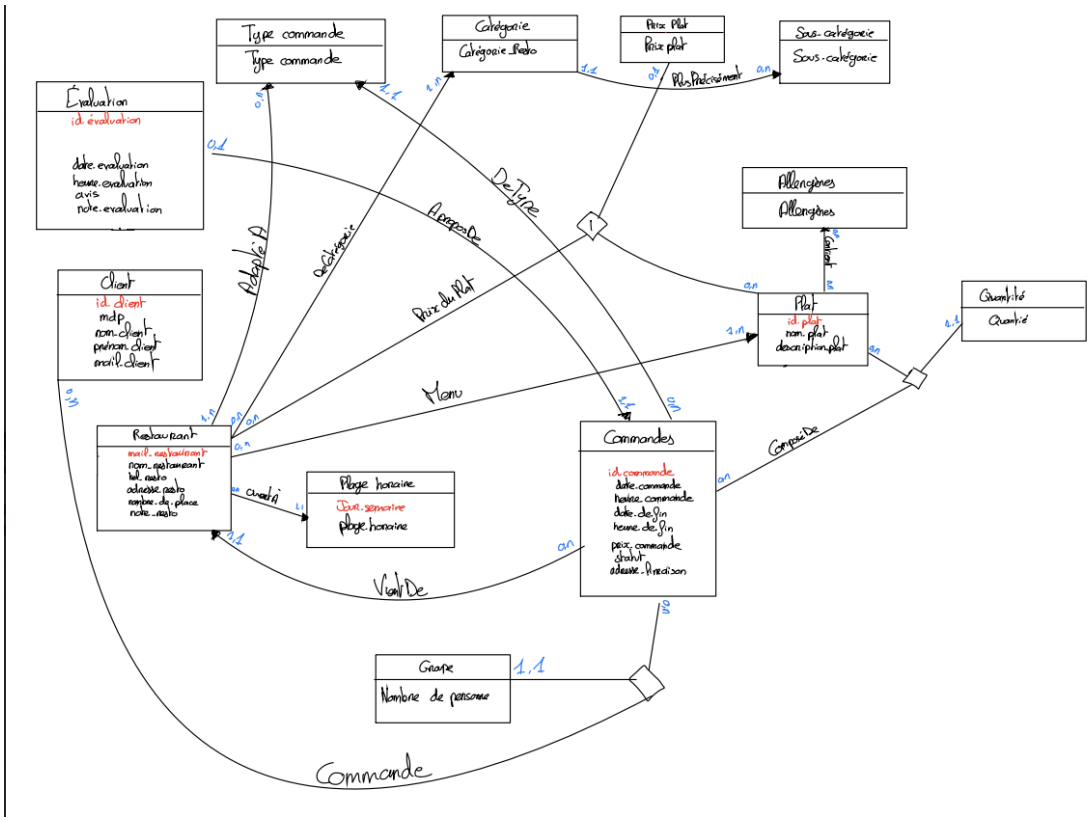
- id_client→ mdp, nom_client, prenom_client, mail_client

- (id_commande, id_restaurant) → date_commande,date_terminée, client_commande, type_commande, prix_commande, statut, adresse_livraison
- ->> contenu(id plats)

- (id_évaluation, id_restaurant) -> id_commande, date_évaluation, avis, note_évaluation

- (id_restaurant, Jour de la semaine) → horaire_matin_ouverture, horaire_matin_fermeture, horaire_soir_ouverture, horaire_soir_fermeture

Diagramme Entités/Relations:



Tables:

Restaurant	**Evaluation**	**ComposéDe**	**Menu**	**OuvertA**	**CatégorieResto**	**Client**
mail_resto	*id_evaluation*	*id_commande*	*mail_resto*	*mail_resto*	*mail_resto*	*id_client*
nom_resto	date_evaluation	id_plat	*id_plat*	*jour_semaine*	categorie_fille	mdp
tel_resto	avis	Quantité	nom_plat	horraire_midi_debut		nom_client
adresse_resto	note_evaluation		prix_plat	horraire_midi_fin		prenom_client
nbr_place				horraire_soir_debut		
note_resto				horraire_soir_fin		

CommandeClient	**Commande**	**Plat**	**Allergenes**	**TypeCommandePossible**	**RelationCategorie**
id_client	*id_commande*	*id_plat*	*id_plat*	*mail_resto*	*Categorie_fille*
id_commande	date_commande	nom_plat	allergenes	type_commande	Categorie_mere
nbr_personnes	date_fin	description_plat			
	prix_commande				
	adresse_livraison				
	mail_resto				

Le nom des tables est en gras et les clés sont en italiques.