Projet Base de Données

BARBERA Noe, CHAUVIN Etienne, DIDIER Anthony
MAZIER Titouan, ROCCHIA Sylvain

Novembre 2022

1 Analyse statique

Voici l'analyse statique que nous avons élaborée pour ce projet. Il s'agit ici de la notre première analyse sur feuille, qui comme vous pourrez le voir dès la formalisation Entités/Associations a légèrement évolué. Il s'agit uniquement de modifications superficielles (sur la forme) et non pas sur la structure (le fond) qui reste identique.

1.1 Propriétés :

{mailR, nomR, telR, adrR, catR, nbplacesR, presentationR, typCmdR, horaires, noteMoy, idP, nomP, descP, prixP, allergènesP, idCl, mailCl, mdpCl, nomCl, prenomCl, adrCl, idCmd, dateCmd, hCmd, typCmd, prixCmd, statutCmd, idLiv, adrLiv, textLiv, hLiv, idSP, nbCouvertsSP, hSP, dateEv, hEv, avisEv, noteEv}

1.2 Dépendances Fonctionnelles

mailR $\longrightarrow$ nomR, telR, adrR, nbplacesR, presentationR, noteMoy
mailR, idP $\longrightarrow$ nomP, descP, prixP
idCl $\longrightarrow$ mailCl
mailCl $\longrightarrow$ mdpCl, nomCl, prenomCl, adrCl
idCmd $\longrightarrow$ dateCmd, hCmd, idCl, mailR, typCmd, prixCmd, statutCmd
idLiv $\longrightarrow$ adrLiv, hLiv
idSP $\longrightarrow$ nbCouvertsSP, hSP
idCmd, dateEv $\longrightarrow$ hEv, avisEv, noteEv

1.3 Contraintes de valeur

typCmd $\in$ {sur place, à emporter, livraison}
statutCmd $\in$ {attente de confirmation, validée, disponible, en livraison, annulée client, annulée restaurant, terminée}
Ext(idLiv) $\subseteq$ Ext(idCmd)
Ext(idSP) $\subseteq$ Ext(idCmd)
Ext(idSP) $\cap$ Ext(idLiv) $= \emptyset$
nbCouvertsSP > 0
$0 \leq$ noteEv ≤ 5
$0 \leq$ noteMoy ≤ 5

1.4 Contraintes de Multiplicité

mailR $\longrightarrow\!\!\!\!\rightarrow$ catR, idP, typCmd
mailR $\not\longrightarrow\!\!\!\!\rightarrow$ horaires, idCmd
mailR, idP $\not\longrightarrow\!\!\!\!\rightarrow$ allergènesP
idCl $\not\longrightarrow$ mailCl
idCmd $\longrightarrow\!\!\!\!\rightarrow$ idP

`idLiv` $\not\rightarrow$ `textLiv`, `hLiv`
`idCmd` $\not\rightarrow$ `dateEv`

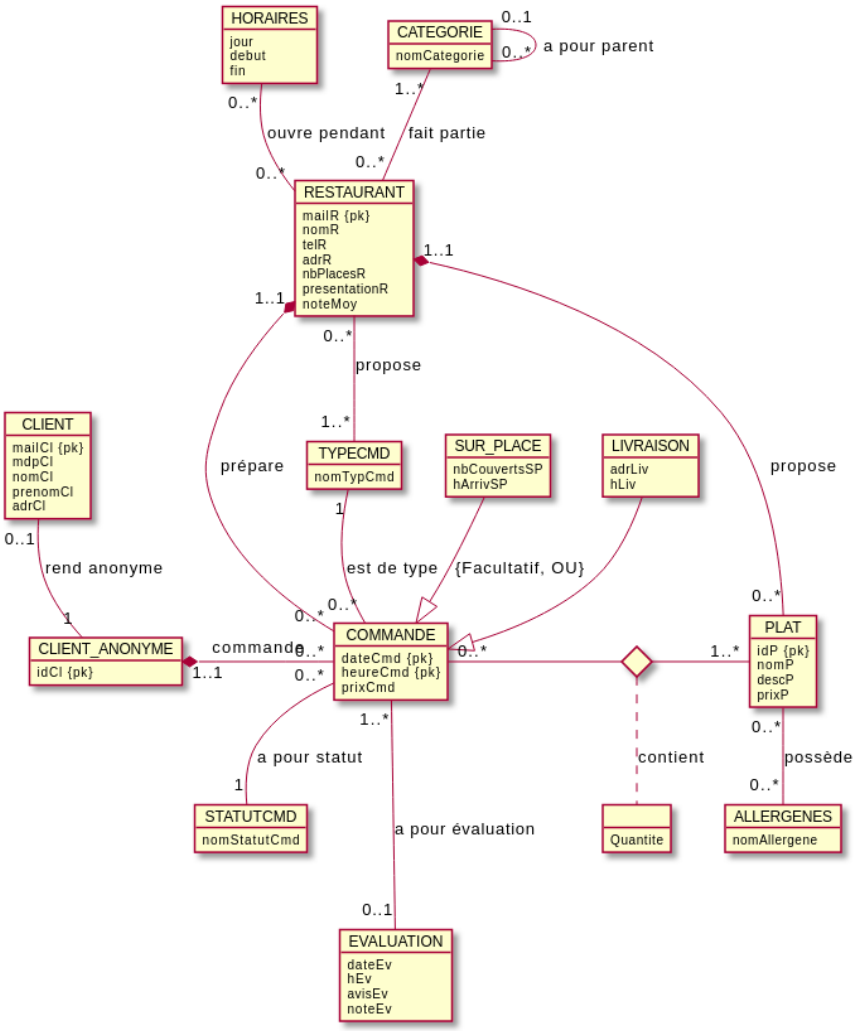
1.5 Autres contraintes

`catR` associé ou non à une autre `catR` parente.
pour chaque restaurant somme des `nbCouvertsSP` dans chaque créneau inférieure à `nbplacesR`.
`hSp` compatible avec `horaires` du restaurant.
`horaires` d'un restaturant ne se superposent pas.
si `statutCmd` == "annulée client" alors pas possible d'associer une évaluation à `idCmd`.

2 Formalisation Entités/Associations

2.1 Schéma

Comme nous pouvons le voir, les attributs clé `id` comme `idLiv` ou `idCmd` ont été remplacé par des relations entités faibles/entités fortes, ce qui nous a semblé plus intuitif.



2.2 Contraintes de valeur

`jour` $\in$ {Lundi, Mardi, Mercredi, Jeudi, Vendredi}
$0 \leq$ `noteMoy` ≤ 5
`nomTypCmd` $\in$ {sur place, à emporter, livraison}
`nbCouvertsSP` > 0
`nomStatutCmd` $\in$ {attente de confirmation, validée, disponible, en livraison, annulée client, annulée restaurant, terminée}
`quantite` > 0
`noteEv` $\in \{1, 2, 3, 4, 5\}$

2.3 Contraintes non listées :

Pour chaque `Restaurant`, la somme des `nbCouvertsSP` liés ne doit pas dépasser `nbplacesR`.
`hArrivSP` doit être compatible avec les `horaires` d'ouverture du `Restaurant`.
Les `horaires` d'un `Restaurant` ne se superposent pas.
Si le champ `statutCmd` d'une commande est à "annulé client" alors il n'est pas possible d'avoir une évaluation associée à la commande.

2.4 Explications des choix :

L'intégralité des données personnelles des clients sont enregistrées dans une entité client qui est dissocié de l'entité `ClientAnonyme` qui ne contient qu'un identifiant. Le reste de l'application interagit exclusivement avec l'entité `ClientAnonyme`. Cette séparation devrait permettre d'oublier un client par simple suppression de sa ligne dans la table `Client`, l'identifiant de `ClientAnonyme` permettant toujours de conserver la trace des commandes passées sans plus pouvoir les relier à l'identité du client.

Une `Commande` est associée à au mois un `Plat`, on considère que les commandes sur places ne sont pas simplement des réservation de table.

Un restaurant peut ne pas proposer de plats, il ne sera alors pas possible d'y effectuer des commandes. Cela simplifie la procédure d'enregistrement d'un `Restaurant`.

3 Passage au relationnel

3.1 Tables

Restaurant(mailR, nomR, telR, adrR, nbplacesR, presentationR, noteMoy)
- $0 \leq$ noteMoy ≤ 5

Plat(mailR, idP, nomP, descP, prixP)
- (mailR) référence Restaurant

Client(mailCl, mdpCl, nomCl, prenomCl, adrCl, idCl)
- (idCl) non nul et référence ClientAnonyme
- vérifier qu'un ClientAnonyme rend anonyme un seul Client

ClientAnonyme(idCl)

Commande(mailR, idCl, dateCmd, hCmd, prixCmd, nomTypCmd, nomStatutCmd)
- (mailR) référence Restaurant
- (idCl) référence ClientAnonyme
- (nomTypCmd) non nul et référence TypeCmd
- (nomStatutCmd) non nul et référence StatutCmd

TypeCmd(nomTypCmd)
- nomTypCmd $\in$ {sur place, à emporter, livraison}

StatutCmd(nomStatutCmd)
- nomStatutCmd $\in$ {attente de confirmation, validée, disponible, en livraison, annulée client, annulée restaurant, terminée}

Livraison(mailR, idCl, dateCmd, hCmd, adrLiv, hLiv)
- (mailR, idCl, dateCmd, hCmd) référence Commande

SurPlace(mailR, idCl, dateCmd, hCmd, nbCouvertsSP, hArrivSP)
- (mailR, idCl, dateCmd, hCmd) référence Commande
- Vérifier que hArrivSP soit compatible avec les horaires d'ouverture du Restaurant.
- Vérifier que la somme des nbCouvertsSP correspondant à un même Restaurant sur un service donné ne dépasse pas le champ nbPlacesR du dit Restaurant.

- nbCouvertsSP > 0

Evaluation(dateEv, hEv, avisEv, noteEv)
- noteEv $\in \{1, 2, 3, 4, 5\}$

Categorie(nomCategorie)

horaires(jour, debut, fin)
- jour $\in$ {Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche}

Allergenes(nomAllergene)

APourEvaluation(mailR, idCl, dateCmd, hCmd, dateEv, hEv, avisEv, noteEv)
- (mailR, idCl, dateCmd, hCmd) référence Commande
- (dateEv, hEv, avisEv, noteEv) référence Evaluation
- vérifier qu'une évaluation soit toujours utilisée
- Vérifier que le champ nomStatutCmd de la Commande ne soit pas à annulée client

ContientPlats(mailR, idCl, dateCmd, hCmd, mailR, idP, quantite)
- (mailR, idCl, dateCmd, hCmd) référence Commande
- (mailR, idP) référence Plat
- vérifier qu'une Commande à au moins un Plat
- quantite > 0

OuvrePendant(mailR, jour, debut, fin)
- (mailR) référence Restaurant
- (jour, debut, fin) référence horaires
- Vérifier que les horaires d'un Restauran ne se superposent pas.

FaitPartieCat(mailR, nomCategorie)
- (mailR) référence Restaurant
- (nomCategorie) référence Categorie
- vérifier qu'un Restaurant fait partie d'au moins une catégorie

ProposeTypeCmd(mailR, nomTypCmd)
- (mailR) référence Restaurant
- (nomTypCmd) référence TypeCmd

PossedeAllergene(mailR, idP, nomAllergene)
- (mailR, idP) référence Plat
- (nomAllergene) référence Allergenes

APourCatParent(nomCategorieFils, nomCategorieParent)
- (nomCategorieFils) référence Categorie
- (nomCategorieParent) référence Categorie

3.2 Formes normales

Les dépendances fonctionnelles associées aux relations de notre base de données sont donc les suivantes :

mailR ⟶ nomR, telR, adrR, nbPlacesR, presentationR, noteMoy

mailR, idP ⟶ nomP, descP, prixP

idCl ⟶

mailCl ⟶ mdpCl, nomCl, prenomCl, adrCl, idCl

mailR, idCl, dateCmd, heureCmd ⟶ prixCmd, nomTypCmd, nomStatutCmd adrLiv, hLiv, nbCouvertsSP, hArrivSP

mailR, idP, nomAllergene ⟶

nomCategorieFils, nomCategorieParent ⟶

mailR, idCl, dateCmd, heureCmd ⟶ dateEv, hEv, avisEv, noteEv

mailR, idCl, dateCmd, heureCmd, mailR, idP ⟶ quantite

mailR, debut, fin ⟶

mailR, nomCategorie ⟶

mailR, nomTypCmd ⟶

nomTypCmd ⟶

nomStatutCmd ⟶

Analysons ce modèle afin d'en déduire quelles formes normales il respecte :

- **Première forme normale**
  Nous pouvons voir qu'aucun attribut d'aucune relation n'est une liste, ainsi tous les attributs sont atomiques donc le modèle est 1FN.
- **Deuxième forme normale**
  Nous pouvons voir qu'aucun sous ensemble d'une clef ne référence les mêmes attributs que ceux référencés par cette clef, et cela pour chaque relation. De plus le modèle est déjà 1FN, alors il est aussi 2FN.
- **Troisième forme normale**
  Nous pouvons voir qu'aucun attribut non clef ne dépend d'un ou plusieurs attributs de la relation n'étant pas clef. De plus le modèle est déjà 2FN, donc il est aussi 3FN.
- **Forme normale de Boyce-Codd-Kent**
  Enfin, nous remarquons que tous les identifiants des dépendances fonctionnelles ci-dessus sont des clés du modèle. Autrement dit, aucun attribut non identifiant n'est source de dépendance pour une partie de son identifiant. Puisque le modèle est en 3FN, alors il est également 3FNBCK.

4 Fonctionnalités

4.1 Création de la base de données

Les scripts de création et de peuplement de la Base de Données sont disponibles dans le dossier `scripts_sql` joint à ce rapport.
Les scripts commencent par supprimer ou vider les tables existantes afin de s'assurer de toujours partir de la même base de données.
Certaines modifications ont à nouveau eu lieu vis à vis du modèle relationnel. En particulier concernant le typage des champs et plus spécifiquement des heures. elles étaient décrites jusqu'ici comme des champs uniques et ont été séparées en 2 champs distincts : un pour l'heure et un pour les minutes. Cela à pour but d'éviter des traitements postérieurs pour l'implémentation des contraintes additionnelles.

4.2 Connexion d'un utilisateur

Nous avons fait le choix de considérer que toutes les transactions demandées ne puissent être exécutées qu'après que l'utilisateur se soit connecté en entrant son mail et son mot de passe. La procédure de connexion correspond à une vérification de l'existence d'une ligne contenant bien les informations de connexion dans la table `Client` à l'aide de la requête SQL suivante :

```
1  SELECT mailCl, mdpCl FROM Client
2  Where mailCl = <mail fourni>
3  And mdpCl = <mot de passe fourni>
4  ;
```

Si cette requête renvoie une table vide, la connexion échoue. Sinon elle réussit.

4.3 Parcours des restaurants disponibles

La fonctionnalité de parcours des restaurants se découpe en trois sous fonctionnalités : L'affichage de tous les restaurants correspondant à une catégorie ou une de ses catégorie fille, l'affichage de tous les restaurants ouverts à une heure et/ou un jour donnés et l'affichage des détail sur un restaurant.

Parcours par catégories Le principe est le suivant : on commence par lister toutes les catégories filles de la catégorie donnée en appelant récursivement (i.e. pour chaque catégorie résultat) la requête suivante :

```
1  SELECT * FROM Categories
2  JOIN APourCatParent
3      ON APourCatParent.nomCatParent = <categorie>
4  JOIN APourCatParent
5      ON APourCatParent.nomCatFils = Categories.nomCategorie
6  ;
```

Une fois ceci fait on va récupérer tous les restaurants associés à au moins une de ces catégories. Pour cela on effectue les requêtes suivantes pour chaque catégorie obtenue avec la première étape.

```
SELECT mailR FROM FaitPartieCat
WHERE nomCategorie LIKE <categorie>
;

SELECT nomR FROM Restaurants
WHERE mailR LIKE <mail obtenu avec la requete precedente>
;
```

Parcours selon les horaires On va cette fois utiliser la table `OuvrePendant` avec la requête suivante :

```
SELECT * FROM OuvrePendant
WHERE jour LIKE <jour demande>
    AND hDebut = <heure demandee>
;
```

Si seule l'heure ou seul le jour ont été spécifiés, on ne conserve que la condition correspondante dans la requète.

Informations d'un restaurant On utilise les requêtes suivantes afin de récupérer en plus des données sur les restaurant la liste des catégories auxquelle le restaturant est rattaché et la liste des plats proposés par le restaturant :

```
SELECT * FROM Restaurants
WHERE nomR LIKE <restaurant demande>
;

SELECT * FROM FaitPartieCat
WHERE mailR LIKE <mail restaurant obtenu avec requete
precedente>
;

SELECT * FROM Plat
WHERE mailR = <mail restaurant obtenu avec requete precedente>
;
```

4.4 Commande

Pour une commande, il faut tout d'abord récolter les informations nécessaires :

- Dans quel restaurant la commande à lieu.
- Quel est l'id du client qui commande.
- La date et l'heure actuelle.
- Le type de la commande.
- Les plats commandés.

Certaines de ces informations sont déja disponible comme la date et l'heure. On peut également récupérer l'id du Client depuis la base de donnée grâce au mail utilisé pour la se connecter avec la requète suivante :

```
SELECT idCl FROM Client
WHERE mailcl = <mail fourni>
;
```

Pour les autres informations, il faut les demander à l'utilisateur et ensuite s'assurer que le client à bien demandé un élément qui existe et qui respecte les autres données déjà enregistrées. Pour cela, on dispose de deux solutions possibles : soit on commence par demander à l'utilisateur d'entrer l'information et on effectue ensuite une requête pour vérifier que l'élément demandé est bien enregistrée dans la base de données, soit on effectue une requête pour récupérer la liste des éléments disponible et l'utilisateur va ensuite en choisir un.
Comme il existe déjà une fonctionnalité dédiée à l'affichage des restaurants, on se permet d'utiliser la première méthode en utilisant la requête de vérification pour récupérer le mail associé au restaurant sélectionné :

```
SELECT mailR FROM Restaurant
WHERE nomR like <nom fourni>
;
```

Ici, on ne considère pas le cas où plusieurs restaurants auraient le même nom dans un souci de simplification.
Pour le type de la commande et les plats commandés en revanche on va commencer par effectuer une requête qui renvoie la liste des types de commandes et de plats disponible et afficher le résultat avant de faire choisir à l'utilisateur respectivement un et plusieurs d'entre eux. On calcule le prix de la commande à l'aide des prix de chaque plat sélectionné.
Une fois les informations récoltées, on peut enfin créer la commande. Pour cela on doit ajouter une ligne dans `Commande` et, pour chaque plat commandé, une ligne dans `ContientPlats`. On utilise pour cela les requêtes suivantes :

```
INSERT INTO Commande
VALUES (<mailR>, <idCl>, <dateCmd>, <hCmd>, <prixCmd>, <
nomtypCmd>, 'attente de confirmation');
```

```
INSERT INTO ContientPlats
VALUES (<mailR>, <idCl>, <dateCmd>, <hCmd>, <mailR>, <idP>, <
quantite>)
;
```

Si le type de la commande passé est une livraison, il faut également demander au client l'adresse de la livraison. On peut proposer au client d'utiliser l'adresse qu'il a renseigné comme la sienne. Si c'est le cas, on va récupérer cette adresse à l'aide de la requète suivante :

```
SELECT adrCl FROM Client
WHERE mailCl = <mail d`authentification>
;
```

Ensuite, il faut ajouter la livraison dans la table dédiée :

```
INSERT INTO Livraison
VALUES (mailR, idCl, dateCmd, hCmd, adrLiv, NULL)
;
```

De même, pour une commande sur place on va demander l'heure d'arrivée, nous assurer qu'elle correspond bien aux horaires d'ouverture auxquels on accède avec la requète suivante :

```
SELECT hDebut, minDebut, hFin, minFin FROM OuvrePendant
WHERE mailR = <mailR>
    AND jour = <jour de la semaine correspondant a dateCmd>
;
```

On doit ensuite demander le nombre de couverts et nous assurer qu'il reste assez de place dans le restaturant. Pour cela, on utilise les requètes suivantes :

```
SELECT nbPlacesR FROM Restaurant
WHERE mailR = <mailR>
;

SELECT nbCouvertsSP FROM Sur_Place
WHERE mailR = <mailR>
    AND dateCmd = <dateCmd>
    AND hCmd <= <14 ou 23 selon l`heure d`arrivee>
    AND hCmd >= <12 ou 19 selon l`heure d`arrivee>
;
```

Une fois les Conditions vérifiées, on ajoute notre commande sur place à la table dédiée :

```
INSERT INTO SurPlace
VALUES (<mailR>, <idCl>, <dateCmd>, <hCmd>, <nbCouvertsSP>, <
hArrivSP>)
;
```

4.5 Droit à l'oubli

Cette fonctionnalité est celle sur laquelle nous avons porté le plus de soin dès le début de la conception du produit. Grâce à cela, son implémentation est particulièrement directe puisqu'il suffit de retirer une ligne dans la table `Client` :

```
DELETE FROM Clients WHERE mailCl = <mail de connexion>;
```

Avant de supprimer les données, on demande à l'utilisateur d'entrer à nouveau son mot de passe afin de confirmer l'opération. Une fois la requète effectuée, on met fin à la session. L'utilisateur ne pourra plus se connecter avec la même adresse mail.

4.6 Implémentation

Les sources des codes java pour le démonstrateur sont disponibles dans les dossiers joints à ce rapport. Ils n'ont malheureusement pas étés complètement testés à cause des problèmes d'accès à la base de donnée que nous rencontrons depuis décembre. Il est donc probable que le démonstrateur ne soit pas encore pleinement fonctionnel. Nous espérons toutefois avoir suffisament détaillé le principe de son fonctionnement.

5 Conclusion

Le projet à pu être mené à bien malgré un résultat pas tout à fait fonctionnel. Il faudrait compter encore une journée pour le déploiement du projet avec une

grande partie de tests qui n'ont pas pu être effectués à cause des problèmes d'infrastucture.

La plus grande difficulté rencontrée a été de synchroniser les environnement de travail de tous les membres de l'équipe. Les changements ou précisions dans l'implémentation ont également ralenti le travail puisqu'il fallait mettre à jour plusieurs points du projets dont certains étaient parfois en développement.

La répartition des tâches c'est faite au fur et à mesure de l'avancée du projet avec dans l'idée de toujours essayer de parralléliser au maximum le travail. Malgré quelques goulots d'étranglement dus à la modélisation de la base de donnée au début cette logique à vite permis de tirer parti efficacement des effectifs de l'équipe.