Rapport du projet Base de Données

Ahamed Mouhamadou, Aiman Bakour, Tahitoa Arbelot, Naoto Lucas, Jean-Baptiste Gaeng

December 11, 2017

1 Analyse du sujet

L'analyse du sujet nous a conduit à ajouter les relations suivantes aux relations déjà fournies par le client :

Client (idClient int, nomClient char(40), typeClient char, typePaiement char(20), annee int)
Reservation (numResa int)
Simulation (numSimu int, dateDepartSimu date, dateArriveeSimu date, nbPersonnesSimu int)
ResaHotel (idResaHotel int, dateEntree date, nbJoursHotel int, nbPetitDej int)

- Client est la relation qui caractérise le client : elle a un id unique, un nom, un type parmi société, groupe, individuel et une annee d'enregistrement auprès de l'agence, ainsi qu'un type de paiement.
- Reservation est la relation qui décrit une réservation : elle a un numéro de réservation numResa.
- Simulation est la realtion permettant de décrire une simulation : celle-ci a un numéro de réservation numSimu unique, une date de départ, une date d'arrivee, et un nombre de personnes.
- ResaHotel est la relation qui permet de décrire la réservation d'un hôtel de la simulation.

2 Modélisation du problème

2.1 Analyse du problème

L'analyse des relations fournies par le client et des nôtres nous a permis de construire le tableau des dépendances fonctionnelles suivant :

Propriétés élémentaires	Dépendances fonctionnelles	Contraintes de valeurs	Contraintes de multiplicité	Autres contraintes
Schéma existant				
{nomVille, pays}				
{nomLieu, adresseLieu, descriptifLieu, prix}	NomLieu, nomVille, pays --> adresseLieu, descriptifLieu, prix	prix>=0		
{idCircuit, descriptif, villeDepart, paysDepart, villeArrivee, paysArrivee, nbJoursTotal, prixCirucuit}	IdCircuit --> descriptif, villeDepart, paysDepart, villeArrivee, paysArrivee, nbJoursTotal, prixCirucuit	PrixCircuit>=0 nbJoursTotal>=0		
{dateDepartCircuit, nbPersonnes}	IdCircuit, dateDepartCircuit --> nbPersonnes	- NbPersonnes >0 - dateDepartCircuit>aujourd'hui		nbPersonnes>nbPersonnesSimu
{ordre, nbJours}	IdCircuit, ordre --> nomLieu, pays, nomVille, nbJours	- Ordre suite de valeurs entiers consecutifs - NbJours>0		
{nomHotel, adresseHotel, nbChambresHotel, prixChambre, prixPeitDejeurner}	NomHotel, nomVille, pays --> adresseHotel, nbChambre, prixPetitDejeuner	- NbChambres>0 - prixPetitDejeurner>0		NbChambreHotel> nbPersonneSimu
Schéma à définir				
{idSimulation, dateDepart, dateArrivee, nbPersonnesSimu}	IdSimulation --> dateDepart, dateArrivee, nbPersonnesSimu	- DateDepart< dateArrivee - nbPersonnesSimu == nbPersonnes	IdSimulation -|->> nomVille, pays, nomLieu, dateDepartCircuits, idResaHotel	dateArriveeSimu > dateDepartSimu
{idResaHotel, dateEntree, nbJoursResaHotel, nbPetitDej}	IdResaHotel --> dateEntree, nbJoursResaHotel, nbPetitDej	- NbJoursResaHotel > 0 - nbPetitDej >=0 -dateEntree<dateDepart		
{numResa}	NumResa --> idSimu, idClient			
{idClient, nomClient, typeClient, annee}	IdClient --> nomClient, typeClient, annee	maintenant>=Annee >1900		

Figure 1: Tableau des dépendances fonctionnelles

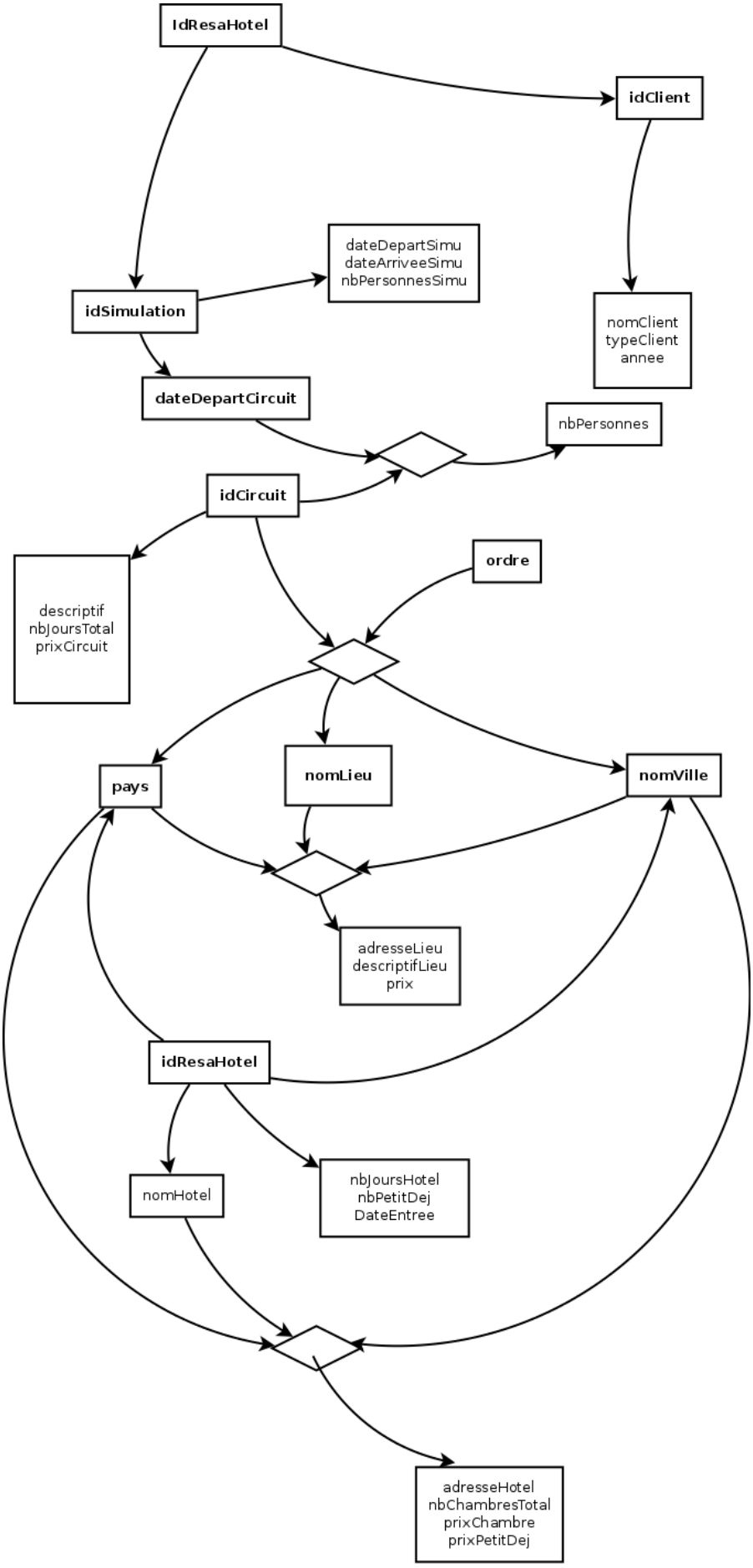

Figure 2: Graphe des dépendances fonctionnelles

2.2 Conception Entités/Associations

L'analyse du tableau de la section précédente nous a permis de construire le schéma entité-association suivant :

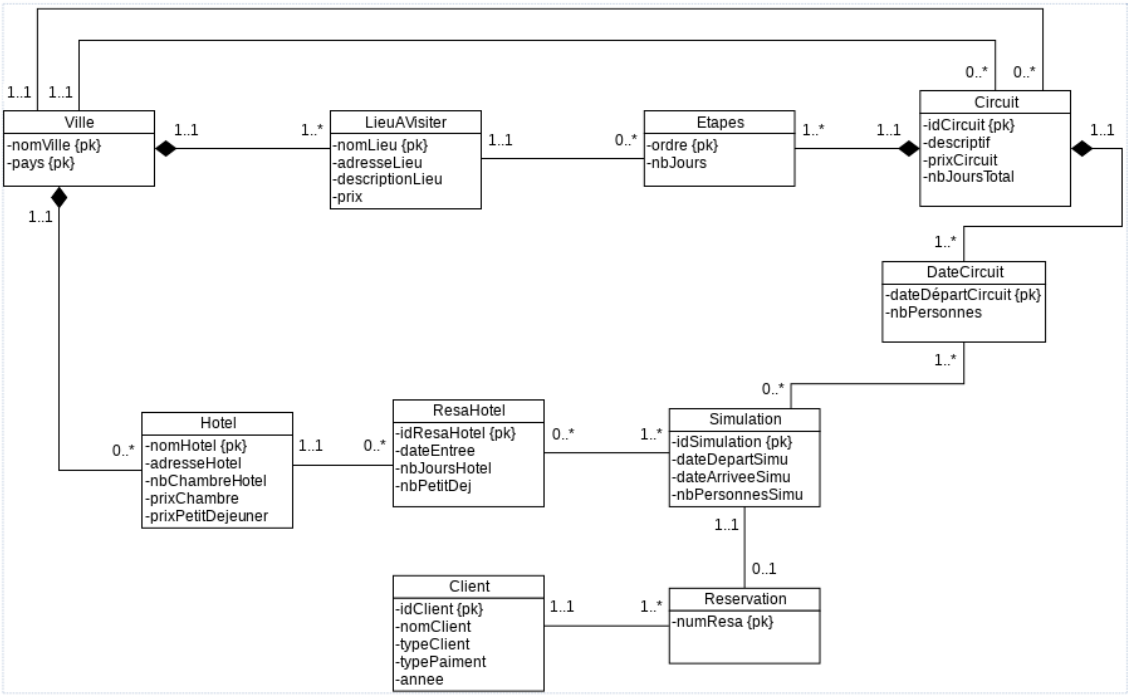

Figure 3: Schéma entités-associations de la base de données

2.3 Traduction en relationnel et forme normale des relations

Afin d'analyser plus facilement les formes normales des différentes relations, nous avons fait le diagramme des dépendances fonctionnelles visible en figure 2.
Nous allons détailler le raisonnement pour trouver la forme normale correspondante à la relation Client, le raisonnement étant exactement le même pour toutes les autres relations. Soit la table `Client` ayant pour clé idClient:

```
Client(idClient {pk} , nomClient, typeClient, annee, typePaiement)
```

D'après le tableau des dépendances fonctionnelles précédent on a :
`idClient` –> `nomClient, typeClient, annee, typePaiement`
On a tous les attributs sont atomiques donc la relation est en FN1.
Les attributs non clés sont pleinement dépendant des clés : la relation est en FN2.
Les attributs non clés sont directement dépendant des clés : la relation est en FN3.

Et ce raisonnement s'applique a toutes les autres relations. celles-ci sont implantées en SQL dans le fichier `peuplage/initialisation.sql`. Celui-ci est issu de l'algorithme de passage du modèle entité association en relationnel vu en cours.

3 Implémentation des transactions

En ce qui concerne les transactions, nous avons utilisé des scripts Python afin d'utiliser la méthode "format" intégrée à la classe string de Python qu'il aurait fallu réimplémenter si on avait utilisé Java. Vous trouverez ces deux scripts dans le dossier peuplage du projet. Notre base de données comporte au total 10 villes, 30 hôtels, 20 lieux à visiter, 90 circuits et date circuit et 360 étapes. Le peuplage des tables s'effectue en éxécutant peuplage.sql.

4 Mode d'emploi de l'application

4.1 Menu principal

Nous avons choisi d'implémenter l'application intégralement sous forme de ligne de commande. Lorsque l'utilisateur lance l'application de réservation AWHY, il arrive sur le menu principal. Celui-ci invite l'utilisateur à choisir parmi 3 modes : CONSULT pour consulter l'intégralité de la brochure de l'agence, SIMU pour effectuer une simulation de voyage, et RESA pour effectuer une réservation.

4.2 Mode consultation (CONSULT)

Lorsque l'utilisateur choisit d'effectuer une consultation de la brochure, cette dernière commence par être intégralement affichée. L'utilisateur peut ainsi avoir une vue d'ensemble sur l'intégralité des offres proposées par l'agence. L'utilisateur peut ensuite affiner sa recherche selon plusieurs critères de son choix, à savoir trier la brochure selon un pays, une ville, un lieu a visiter ou encore trier les circuits selon un prix maximal. L'utilisateur peut bien-sûr revenir en arrière en entrant le caractère "*" pour le critère désiré.
Lorsque l'utilisateur a terminé de consulter la brochure, il peut revenir au menu principal en écrivant "menu".

4.3 Mode simulation (SIMU)

Lorsque l'utilisateur entre dans le mode simulation, l'application lui demande des informations utiles liées au choix de son premier circuit, à savoir : le nombre de personnes à inclure dans la simulation, la date de départ prévue, la date d'arrivée, l'identifiant du circuit selectionné, puis d'autres informations concernant les hôtels de ce circuit.

Lorsque l'utilisateur a terminé de renseigner un circuit, il peut en ajouter d'autres ou bien terminer la simulation, auquel cas l'application lui attribue un identifiant de simulation.

Enfin, il peut valider la simulation pour ensuite passer à l'étape de réservation.

4.4 Mode reservation (RESA)

Lorsque l'utilisateur entre en mode réservation, Un message lui affiche un récapitulatif de sa derniere simulation s'il vient d'en effectuer une, ou sinon il est invité à entrer dans le mode simulation pour effectuer une simulation.

Si l'utilisateur confirme sa simulation, le processus de réservation commence. L'application demande à l'utilisateur s'il possède un compte client. Le cas échéant il passe à l'étape suivante, sinon il est invité à creer un compte client en renseignant quelques informations (nom, type, moyen de paiement).

L'utilisateur doit ensuite confirmer chacune des reservations hotel/circuit qu'il a effectuées. Une fois cette étape terminée, l'application lui donne le prix total de sa réservation.

Enfin, l'application vérifie pour chaque circuit à reserver s'il est disponible ou non. Et pour un circuit donné, si les hotels sont pleins ou non. Si un des circuit est complet ou si un hotel est complet, l'utilisateur est invité à recréer une simulation. Sinon, la réservation est validée.

4.5 Schéma UML de l'application

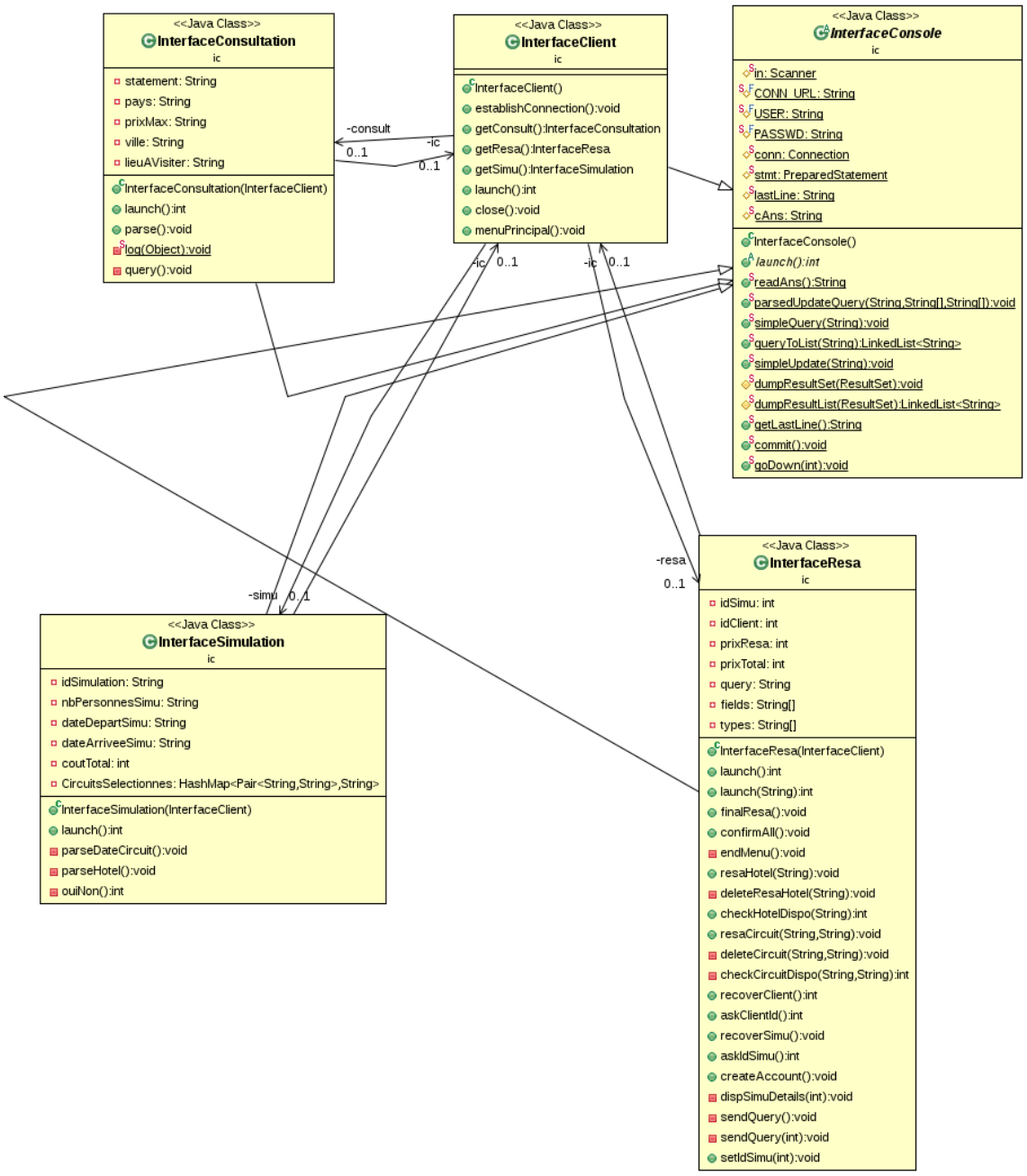

Figure 4: Schema UML de l'application.

5 Gestion des requêtes concurrentes sur la base de données

Nous avons choisi d'établir une connexion au serveur ensioracle en read-commited. Cela permet d'éviter les lectures incorrectes, mais lors d'accès concurrents, les mêmes tables peuvent être modifiées contrairement à serializable.

6 Bilan du projet

6.1 Organisation et répartition des tâches

Tout au long du projet, nous avons essayé de paralléliser au maximum les tâches.

Cependant, durant la première semaine du projet, nous nous sommes chargés d'analyser le sujet et les besoins du client en groupe. Ainsi, nous nous sommes chargés de construire le tableau ré-

capitulatif des dépendances pour les différentes relations, ainsi que le schéma entité-associations. Le schéma a été d'abord construit sur papier, ce qui était plus pratique pour le modifier. Nous l'avons ensuite mis au propre sur un logiciel en ligne de construction de diagramme UML.

Dans un second temps, après l'analyse du problème et sa modélisation, nous avons pu paralléliser un grand nombre de tâches. Aiman et Ahamed se sont chargés d'implémenter les transactions en SQL afin de remplir la base de données avec de nombreux exemples. Tahitoa s'est chargé de penser la hiérarchie du projet en différentes classes. Dans le même temps, Naoto et Jean-Baptiste se sont chargés de traduire le schéma entité-association en relationnel dans initialisation.sql, le fichier permettant de créer toutes les relations de la base de données.

Il a fallu ensuite bien réflechir aux choix de conception - Application graphique ou sous forme de ligne de commande - et aux différents modes de l'applications. Ici tous les membres du groupe ont participé. La conception de l'application a pris du temps d'autant plus qu'il a fallu précisément définir pour chaque mode de l'application les renseignements rentrés par l'utilisateur.

En ce qui concerne la répartition des tâches pour le développement de l'application, Tahitoa s'est chargé d'implémenter le mode réservation. Jean-Baptiste et Naoto se sont chargés respectivement des modes consultation et simulation. Ahamed et Aiman se sont chargés de vérifier que le code fonctionnait en faisant des transactions concurrentes.
La rédaction du présent rapport a été confiée à Jean-Baptiste, les différentes images provenant des différents membres du groupe, comme expliqué plus haut.

6.2 Difficultés rencontrées

Nous n'avons pas eu le temps de finir tous les tets pour vérifier l'intégrité de la base de donnée lors des renseignements fournis par l'utilisateur. Ainsi, il demeure certaines erreurs pour des cas précis, comme lorsque l'on essaye d'effacer des hotels ou des circuits. De plus, dans les modes de simulation et reservation, nous n'avons pas eu le temps de vérifier complètement la cohérence des données rentrées comme la cohérence des différentes dates (exemple : DateCircuit compatible avec DateSimu). Pour finir, lors de la création du client, il manque encore la vérification de certaines entrées. Par exemple, si l'utilisateur rentre un caractère à la place d'une date demandée, l'application renvoit une exception SQL sans être ratrappée.

En outre, de nombreuses étapes de conception énoncées dans le paragraphe précédent se sont davantage entremêlées, car nous revenions itérativement sur le schéma entité-association ou sur les différents fichier sql à mesure que nous détections des erreurs et des nouveaux besoins. D'autre part, certaines parties du sujet étaient peu précises, et nous avons dû changer un grand nombre de fois le déroulement des renseignements demandés à l'utilisateur au sein de l'application.