Rapport Projet Base de Données

Groupe 10

2019 - 2020

L3 MIAGE

Application Air Chance

Dumas Clément
Meziane Farid
Diallo Mohamed Saliou
De Lemos Almeida Pierre

Sommaire

I/- Introduction

L'objectif de ce projet été de mettre en oeuvre les notions vu en cour par la réalisation d'une application Java suivant le protocole JDBC.
Le but de cette application et de pouvoir planifier et gérer les réservation d'un compagnie aérienne. Vous verrez que nous avons séparé notre code en deux fichier java différents : l'un pour la gestion et l'autre pour la réservation.
Pour les technologies utilisées voilà comment nous avons procédés :

Au niveau de l'application, nous avons respecté le cahier des charges et donc nous avons utilisé **Java** et **JDBC.**

Pour la partie base de données, chacuns d'entre nous à coder en local sur son pc et ainsi utiliser du **SQL**, **PL/SQL.**

Pour l'architecture logicielle de l'application, nous nous sommes basés sur le pattern **DAO.**

II/- Modification depuis le rapport intermédiaire

Changement au niveau du MCD : En effet, depuis le rapport intermédiaire nous avons dû modifier notre MCD. Il y avait certains points qui étaient complexe et peu compréhensible. C'est pourquoi nous l'avons repris depuis le début pour ainsi le simplifier et pouvoir réaliser les fonctionnalités de l'application.

Changement de la BD : Comme nous avons modifier le MCD, il était évident de modifier la Base de donnée. Par ailleurs, la Base de donnée, fut modifié de nombreuse fois lors du projet notamment lors de l'ajout des procédure stockée.

III/- Fonctionnalités de l'application

Interface :

L'interface se trouve dans la classe *AirChance.java*, lorsque que vous lancez le programme, on tente dans un premier temps une connexion à la base de données. Si elle fonctionne alors une interface apparaît ou il vous est demandé de saisir un nombre compris entre 1 et 8. Si vous décidez de saisir un autre nombre, automatiquement, vous quitterez le programme et un message de fin apparaîtra. Voici un tableau qui résume ce que vous déclenchez en saisissant un nombre :

Nombre Saisit	**Résultat**
1	Appel à la méthode pour créer un vol
2	Appel à la méthode pour modifier la planification d'un vol
3	Appel à la méthode pour supprimer un vol
4	Appel à la méthode pour confirmer la terminaison d'un vol
5	Appel à la méthode pour ajouter ou supprimer du personnel de vol
6	Appel à la méthode pour afficher la commande d'un client

7	Appel à la méthode pour que réserver un commande d'un client
8	Appel à la méthode pour supprimer une réservation d'un client
AUTRE	Fin du programme

L'interface n'étant pas très développé, lorsque vous aurez saisi un nombre et que la requête sql sera exécuté, le programme se terminera, il faudra lancer une nouvelle fois le programme pour tester d'autres requêtes.

Vous trouverez ci dessous le détail des requêtes que nous vous permettons de formuler.

Requête 1 : Création d'un vol

C'est la méthode *AjoutVol(conn: Connection) : boolean* dans la classe *GestionPlanificationVol.java* qui gère la création d'un vol. Elle renvoie vrai si on a pu ajouter le vol, et faux sinon accompagné de la raison pour laquelle on n'a pas pu créer le vol.

On demande à l'utilisateur de saisir l'horaire de départ (d'abord l'heure, puis la minute puis la seconde, où l'heure est un nombre compris entre 0 et 23, minute et secondes deux nombres compris entre 0 et 59). Ensuite on demande de saisir la date de départ (d'abord l'année, puis le mois (on demande un entier), puis le jour), on vérifie que la date est valide, mais on ne s'interresse pas à savoir si la date est bien dans le futur ou non (l'utilisateur peut très bien saisir un vol prévu au 25 février 1950). après on demande la durée du vol (heure, minute, seconde encore et cette fois l'heure peut être un nombre plus grand que 24). Puis on demande la distance en km (qui peut être un entier ou un double). Ensuite on demande de saisir la ville de départ et d'arrivée. On demande de saisir le nom des villes et on vérifiera qu'elles existent et si c'est le cas on récupérera leur id (Autrement le programme plantera). Pour terminer la saisit, on demandera à l'utilisateur de saisir le numéro de l'avion qui est affréter, ainsi que le prix de base du vol.

Une fois que toute les saisis sont faites, on cherche dans la table vol l'id maximal, on lui ajoute 1 et c'est notre nouvel id pour le vol que l'on va créer. Ensuite on vérifie que la saisi est valide, que l'avion est disponible etc.. (Distance >0, idAvion saisi existe dans la base de données, que l'avion est disponible, que l'avion puisse parcourir la distance souhaitée, que le prix de base soit bien positif strictement et que les villes saisis existent dans notre BDD). Si la saisi est valide alors on exécute la requête créer, autrement et on renvoie vrai, autrement on renvoie faux avec un message d'erreur expliquant les raisons de l'échec.

Pour tester cette méthodes, je propose qu'on rentre des valeurs fausses volontairement, pour vérifier que la méthode renvoie faux, et ensuite on vérifie que la méthode est fausse. On doit vérifier :

1. Que la distance est bonne (>0)
2. Que l'avion est dans la base

3. Que l'avion est disponible (il n'est pas affréter sur un autre vol pendant ce temps la)
4. Que l'avion peut parcourir la distance
5. Que la ville de départ existe dans la base
6. Que la ville d'arrivée existe dans la base
7. Que le prix est bon (>0)

Requête 2 : Modification de la planification d'un vol existant
Dans cette partie, il nous était demandé de gérer la modification d'un vol existant. Il y avait des contraintes qu'il fallait respecter : le remplacement d'un avion devait garantir la compatibilité avec les contraintes du vol précédent. L'avion de remplacement devait être de capacité équivalente sinon le vol était supprimé.
Pour réaliser cette tâche on affiche d'abord la liste des avions, puis on demande à l'utilisateur de choisir lequel des vol il veut modifier.
On peut appliquer 3 types de modification à un vol :
Remplacement d'un avion par un autre
Pour remplacer un avion par un autre il faut respecter la contrainte de compatibilité des avion. En effet l'avion de remplacement doit être de capacité égal que l'avion initialement planifié. On utilise une requête SQL pour afficher les avions qui respectent la contrainte puis on demande à l'utilisateur de choisir un avion. Si il n'y a pas d'avion disponible le vol est supprimé avec la fonctionnalité de la requête 3. Sinon on effectue la mise à jour de la table vol et on revient sur le menu.

Remplacement d'un membre de l'équipage
Pour remplacer un membre de l'équipage, on demande à l'utilisateur si c'est un pilote ou une hôtesse. Si c'est un pilote, on affiche la liste des pilotes initialement planifiés sur le vol pour que l'utilisateur puisse choisir puis la liste des pilotes remplaçant (pour qu'il puisse choisir) et on effectue le remplacement.
Même chose pour les hôtesses.

Modification des données de configuration d'un vol
Pour modifier les données de configuration d'un vol on demande à l'utilisateur de choisir le vol concerné puis la donnée qu'il veut modifier. Ensuite on récupère le changement qu'il faut effectuer et on met à jour la table.

Attention aucune modification n'est prise en compte quand le vol est terminé etat_vol = OK

Requête 3 : Suppression d'un vol
Notre but est de supprimer un vol mais celui-ci doit rester archiver l'on a donc utiliser le champs etat_vol de la table VOL qui permet cela.
Nous faisons donc un update de ce champs.
Maintenant que le vol est supprimé il reste tous les clients ayant fait un réservation sur ce vol à déplacer vers de futurs vols qui n'ont pas encore décollé.
L'on va donc sélectionner chaque client ayant une réservation sur le vol supprimé et le mettre vers un autre vol tout en gardant le nombre de place qu'il a demandé et la classe.

Pour chaque place nous appelons une procédure qui va récupérer les places disponibles et les remplir avec les anciennes places.

Requête 4 : Confirmation de la terminaison d'un vol
Ici l'objectif est de confirmer la terminaison d'un vol, pour cela il faut passer l'état du vol de **att** à **arrive**. Au début on sélectionne dans la liste de vols afficher à l'écran le vol qu'on veut terminer. Puis la mise à jour est effectuée.
Lorsqu'un vol est terminé, l'équipage affecté à ce vol est désaffecté (suppression de l'équipage des table piloter et affecter) et la réservation est supprimé de la table reserve.

Requête 5 : Ajout et suppression d'un personnel de vol
On utilise la méthode la méthode Ajout_Suppr_Personnel(conn : Connection) : void dans la classe GestionPlanificationVol.java pour gèrer cette requête.

Ainsi lors du lancement, un menu avec 4 choix s'affiche :
1. Ajouter un nouveau pilote
2. Suppression d'un pilote
3. Ajouter une nouvelle hôtesse
4. Suppression d'une hôtesse

Fonctionnement de la méthode Ajout_Suppr_Personnel(conn : Connection) :
Pour vous l'expliquer, nous allons prendre en exemple ajout et suppression d'un pilote sachant que pour l'hôtesse c'est le même principe.
Donc lorsque l'utilisateur choisit d'**ajouter un nouveau pilote** à la Base de Données les identifiants, le nom, le prenom, le numero de la rue, la rue, le code postal, la ville et le pays sont demandées.
Une fois les informations saisies par l'utilisateur, on fait appel à executeUpdate qui va permettre grâce à une requête d'insertion (INSERT INTO) de mettre à jour la Base de Données avec les bons éléments saisies par l'utilisateur.
Ainsi, une fois fini il y a une deuxieme requete qui vous affichera le résultat de l'ajout en vous montrant le contenu de la Base de Données de la sorte : id_pilote, nom_pilote, prenom_pilote.

Pour la **suppression la suppression d'un pilote**, on commence par afficher la liste des pilotes disponible de la sorte : id_pilote, nom_pilote, prenom_pilote.
De la sorte, l'utilisateur n'a plus qu'à choisir l'identifiant du pilote dont il souhaite la suppression.
Ainsi, une fois l'id_pilote saisie, on récupère le résultat grâce à la fonction lire.Entier("") de LectureClavier.
Et donc, tout comme pour l'ajout d'un pilote, on fait appel à executeUpdate qui va permettre grâce à une requête de suppression (DELETE) de mettre à jour la Base de Données avec les bons éléments saisies par l'utilisateur. En toute fin, on utilise une requête qui permet d'afficher les pilotes restant dans la Base de Données.
Tout comme nous l'avons dit plus haut, pour l'hôtesse de l'aire c'est sur la même principe que le pilote.

Testons la méthode Ajout_Suppr_Personnel. Pour cela commençons par Ajout d'un personnel. On base les exemples sur pilot mais c'est la même chose pour hôtesse.

1) **Test qui fonctionne** : On entre dans l'ordre identifiants (qui doit être unique), le nom, le prenom, le numero de la rue, la rue, le code postal, la ville et le pays.

2) **Test qui lève une exception** : En reprenant le test qui fonctionne si lors de la saisie de l'identifiant l'utilisateur entre un identifiant qui est déjà présent dans la Base de données alors une fois la saisie terminée une erreur se produira car l'identifiant du pilote (resp. de l'hôtesse) doit être unique.

D'où le fait que l'on affiche une liste du personnel présent au début de l'exécution.
Maintenant parlons des tests lors de la suppression du personnel.

1) **Test qui fonctionne** : Pour supprimer du personnel, l'utilisateur doit saisir un identifiant qui lui est listé dans un affichage avant ça saisie. Ainsi, en choisissant un identifiant présent dans la liste la suppression peut avoir lieu.
2) **Test qui lève une exception** : Si l'utilisateur entre un numéro d'identifiant qui n'est pas présent dans la liste qui s'affiche à lui avant la saisie alors une exception sera levé car il faut que l'identifiant soit présent dans la Base de Donnée pour que la suppression soit réalisé.

D'où le fait que l'on affiche une liste du personnel présent au début de l'exécution.

Requête 6 : Affichage de la commande d'un client

C'est la méthode *AfficheCommandeClient(conn: Connection) : void* dans la classe *GestionReservation.java* qui gère l'affichage de la commande client. On demande à l'utilisateur de saisir le numéro de passeport du client pour qui on veut afficher les commandes. On aurait pu demander l'id du client, mais j'ai fais le choix du numéro du numéro de passeport car chaque numéro de passeport est unique, et qu'il est moins facile de se tromper dans la saisie du numéro de passeport plutôt qu'un identifiant.

On affichera ensuite le numéro de la réservation, l'auteur (nom et prénom de la personne avec le numéro de passeport saisi), la date de la réservation, le numéro de vol, celui de l'avion ainsi que son modèle. Ensuit on affiche la ville de départ, et l'heure et la date, puis la durée de vol et ensuite la ville d'arrivée et la date et l'heure. Enfin on affiche le numéro de la place, ainsi que le prix payé et le numéro de la place.

Pour récupérer toutes les valeurs que l'on veut afficher on utilise la procédure stockée suivante :

```
CREATE Procedure AFFICHAGE_COMMANDE_CLIENT ()
SELECT  R.id_reservation,RESERVATION.date_reservation,
R.numero_vol, V.numero_avion, A.code_modele_avion,
Vd.nom_ville nom_ville_dep ,V.date_vol, V.horaire_vol,V.duree_vol,
Va.nom_ville nom_ville_arr,R.numero_place, R.prixplace_reserve,
V.etat_vol, RESERVATION.id_client
FROM RESERVE R
JOIN RESERVATION ON R.id_reservation=RESERVATION.id_reservation
JOIN VOL V ON R.numero_vol=V.numero_vol
JOIN AVION A ON A.numero_avion=V.numero_avion
JOIN VILLE Vd ON V.id_ville_provenir=Vd.id_ville
JOIN VILLE Va ON V.id_ville_destiner=Va.id_ville  ;
```

Requête 7 : Réservation d'un client sur un vol

Tout d'abord l'on souhaite savoir qu'elle vol veut choisir le client, de ce fait on lui demande la ville de départ et celle de destination, on lui demande aussi en quel classe il souhaite voyager et on lui affiche les différents vols disponibles. On lui demande enfin le nombre de siège qu'il souhaite.

Après cela on a besoin de savoir si ce client et déjà un client ou si il est nouveau, on lui demande son nom et prénom puis si il est client on demande si il veut utiliser ses points de fidélité en les affichant, si il n'est pas client on lui demande toutes les informations nécessaires pour créer un client et on le crée (on part du principe qu'il n'utilise pas ses points de fidélité vu qu'il n'en a pas).

Maintenant nous devons afficher le prix de la place au client et effectuer les réservations. Pour cela on va donc prendre le prix de base du vol et ensuite lui effectué des traitements selon si le vol est plein, la date à laquelle il réserve et selon la classe qu'il a choisi. Puis nous insérons les nouvelles données dans la table réservation puis réserve x fois (x est le nombre de place que le client a choisi).

Requête 8 : Suppression d'une réservation d'un client

Pour gérer cette requête, on utilise la méthode la méthode
SuppressionCommandeClient(conn : Connection) : void
dans la classe GestionPlanificationVol.java.

Fonctionnement de la méthode SuppressionCommandeClient(conn : Connection) :

Pour cette partie j'ai eu recours au procédure stockée (que vous trouverez dans le fichier de la Base de Données sql) pour supprimer la réservation client (CALL supprimer_clients_reservation).

Au début j'ai une requête normal avec un executeQuery qui affiche tous les clients qui on une réservation.

Vu que l'on a une liste des clients avec une réservation, l'utilisateur va pouvoir supprimer la réservation d'un client en donnant son identifiant.

Avec lireEntier de LectureClavier je récupère cet identifiant que je passe en paramètre de la procédure stockée (CALL supprimer_clients_reservation).

Enfin, après l'appel de la procédure et passage de l'identifiant en paramètre de la procédure on exécute la procédure.

Ainsi, après exécution on affiche l'identifiant du client dont la réservation a été supprimer.

Le principe de la procédure stockée est simple :

Je commence par définir un DELIMITEUR autre que le point virgule.

Ensuite je crée la procédure de la sorte : CREATE PROCEDURE supprimer_clients_reservation(IN id_CR INT).

Cette procédure prend un id client en paramètre (en mode IN).

Après une fois l'id_client passer en paramètre je le supprime de la table RESERVATION en vérifiant que l'identifiant correspond à celui déjà présent dans la table (WHERE id_client = id_CR;).

Testons la méthode SuppressionCommandeClient. Pour cela je vais donner une jeu de test.

1) **Test qui fonctionne** : Lors de l'exécution de cette méthode, une liste contenant les clients ayant une réservation s'affiche. Ainsi, l'utilisateur, choisit un identifiant client présent dans cette liste. Une fois l'identifiant choisi, la base de donnée se met à jour et la réservation du client est bien supprimé et une nouvelle liste s'affiche avec les clients restant ayant une réservation.

2) **Test qui lève une exceptions** : Si l'utilisateur entre un identifiant qui n'est pas présent dans la liste qui s'affiche à lui, c'est à dire qui n'est pas contenu dans la base de données une exception est alors levé car l'identifiant doit exister dans la Base et donc la suppressions ne peut avoir lieu.

IV/- Gestion de projet et travail en groupe

Du fait du confinement, nous avons commencé par créer un git, et notre communication était sous forme écrite et oral sur Discord.
Ainsi les quatres membres du groupes pouvais avoir accès à l'avancement en temps réels.

Au niveau de la gestion de projet :
A ce sujet, cela a été très compliqué. En effet, le travail à distance n'est pas quelque chose de facile et il a été dure à des moments de pouvoir être disponible les 4 en même temps du fait des contraintes de chacun.
Ceci explique le fait de la répartition des requêtes à chaque membre de l'équipe.

Au niveau du travail en groupe :
Répartition du travail, chacun d'entre nous à apporter des fonctionnalités à l'application après répartition des tâches.
Pour la répartition des tâches, il y avait 8 requêtes et chacun à fait 2 requêtes :

- Clément à fait la requête 1 et la requête 6.
- Farid à fait la requête 5 et la requête 8.
- Mohamed à fait la requête 2 la requête 4.
- Pierre à fait la requête 3 et la requête 7.

Ensuite, nous procédons à une mise en communs. Ainsi, chacun explique ce qu'il a apporté à l'application au reste de l'équipe.