PROJET SGBD

Bâti Mégane

Bélot Mathieu

Bouton Paul

Donzeau Anaïs

Vincent Kylian

I-Conception de la base de données

1-Analyse du sujet

Attributs	Contraintes
- Catégorie - Idvéhicule ≥0 - NombrePlacesVéhicule >0 - NomStation - AdresseStation - NombrePlacesStation (x5, une par catégorie) ≥0 - DuréeMaximaleUtilisationCatégorie ≥0 - PrixHoraireCatégorie ≥0 - MontantCautionCatégorie ≥0 - DateDépartLocation - DateArrivéeLocation - NumeroCB ≥0 - Nom - Prénom - DateNaissance - Adresse - NumeroForfait ≥0 - NumeroForfaitIllimité ≥0 - NumeroForfaitLimité ≥0 - DuréeForfaitIllimité - DateDébutForfaitIllimité - PrixForfaitIllimité ≥0 - remiseForfaitIllimité ≥0 - PrixForfaitLimité ≥0 - NombreMaximalLocationForfaitLimité ≥0	- Catégorie {« Voiture Electrique », « Vélo », « Vélo Electriques », « Vélo avec remorque », « Petit Utilitaire »} - Un Véhicule appartient forcément à une catégorie - NomStation est unique - Un abonné a 25% de réduction s'il a moins de 25 ans ou plus de 65 ans - Un abonné possède au maximum un forfait par catégorie (il peut ne pas en avoir) - NumeroCB unique - NumeroForfaitUnique - Première Heure d'utilisation gratuite pour toute location - DuréeForfaitLimité {« Jour », « Mois », « Année »} - NuméroForfaitIllimité $\subset$ NuméroForfait - NuméroForfaitLimité $\subset$ NuméroForfait - NuméroFofaitLimité $\cup$ NuméroForfaitIllimité =NuméroForfait - NuméroFofaitLimité $\cap$ NuméroForfaitIllimité = $\emptyset$ - Un véhicule est soit dans une station soit en location - A une date d donnée, un véhicule ne peut être loué qu'une seule fois au maximum (pas de locations simultanées). - Une station doit pouvoir contenir au moins un type de véhicule

2-Dépendances fonctionnelles

Suite à notre phase d'analyse, nous avons identifié les dépendances fonctionnelles suivantes :

IDVehicule → NombrePlacesVéhicule

IDVehicule→Catégorie

NomStation → AdresseStation

NomStation → NombreDePlacesStation (x5 une par catégorie de véhicule)

Catégorie → DuréeMaximaleUtilisationCategorie, PrixHoraireCategorie, MontantcautionCategorie

NumeroCB → Nom, Prenom, DateNaissance, Adresse

NumeroForfait → Catégorie, NumeroCB

NumeroForfaitLimité → NombreMaximalLocationForfaitLimité, PrixForfaitLimité

NuméroForfaitIllimité → DuréeForfaitIllimité, DateDébutForfaitIllimité, PrixForfaitIllimité, remiseForfaitIllimité

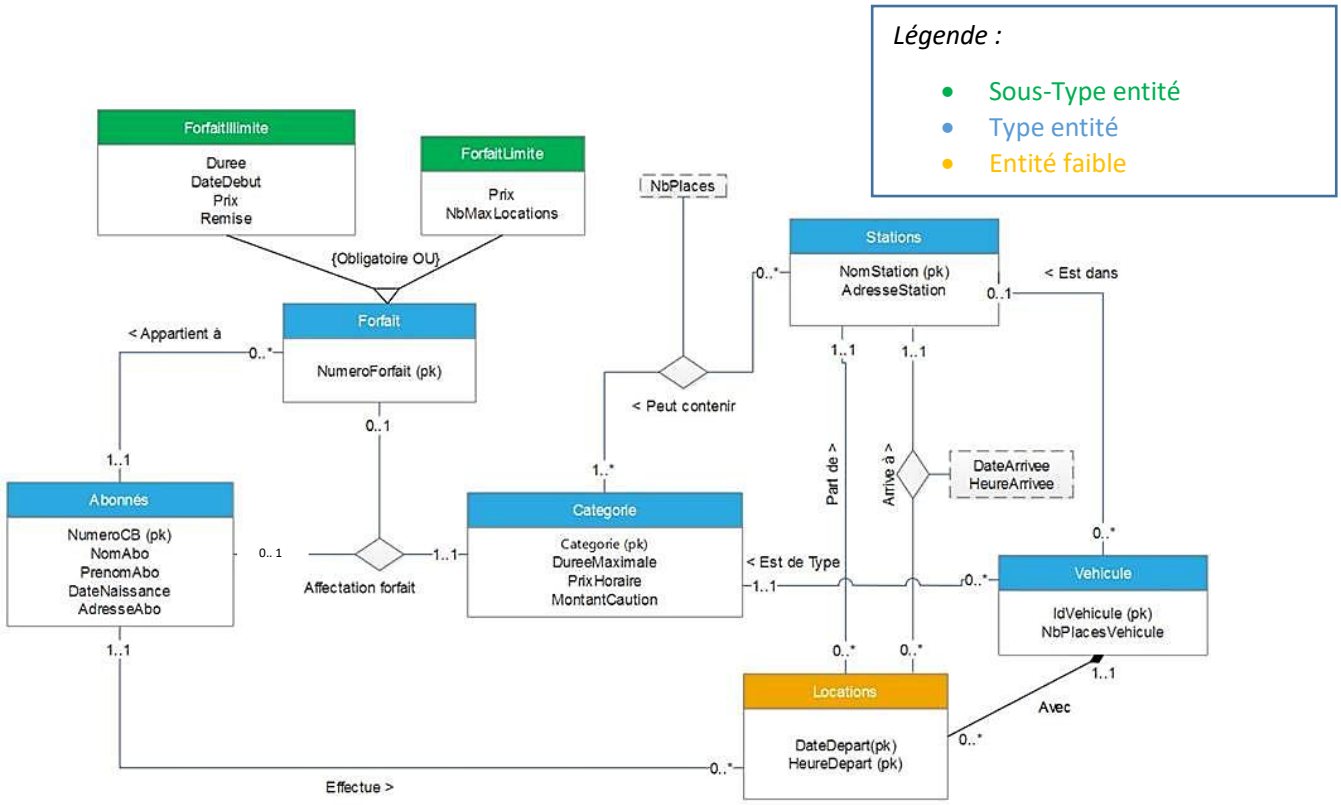
NumeroCB →→NumeroForfait (0,*)

NumeroCB, Categorie →→NumeroForfait(0..1)

IDVehicule→→NomStation (0,1)

3-Schéma Entités/Associations

Un forfait étant soit limité soit illimité, on crée deux sous-types entités pour chacun des deux types de forfait. En ce qui concerne les locations, elle doit posséder comme attribut une date d'arrivée et de départ mais aussi une station d'arrivée et de départ. Cependant, un concept ne pouvant apparaître qu'une seule fois dans notre schéma Entités/Associations (Le concept de Station est ainsi unique), nous avons créé une entité faible. Nous obtenons ainsi le schéma Entités/Associations suivant :



3-Justification des choix de conception

Sous-types entités forfaits limités et illimités

Pour traiter le cas des deux sous-types entités ForfaitIllimité et ForfaitLimité, nous avons examiné plusieurs solutions :

- **Unification** : Nous avons ici estimé que cela engendrerait une perte mémoire trop importante. En effet, chaque forfait aurait eu les attributs d'un forfait limité mais aussi ceux d'un illimité. Ainsi, de nombreux attributs auraient eu une valeur *null*. Nous n'avons donc pas opté pour ce choix de conception (même s'il permet un accès immédiat à chaque forfait à partir de son numéro).
- **Référence** : Le problème de cette solution est qu'il faudrait accéder, dans le pire cas, trois fois à la base de données afin de récupérer les caractéristiques d'un forfait :

    - Un accès à la table forfait pour récupérer son numéro à partir du *NumeroCB*
    - Un voire deux accès pour récupérer le type du forfait (il faut tester l'existence du numéro du forfait dans les tables ForfaitLimite et ForfaitIllimite dans le pire des cas).
- **Duplication** : Nous avons mis en place cette conception pour plusieurs raisons :
    - Surcoût de mémoire, dû à la duplication, faible (on ne duplique que *NumeroCB*)
    - Accès au type de forfait en deux requêtes maximum (on teste l'existence dans les deux tables ForfaitLimité et ForfaitIllimité dans le pire cas).
    - La table forfait est ainsi utilisée pour garantir l'unicité des numéros de forfait
    - Ce choix de conception implique une duplication des insertions de forfaits (une insertion dans la table forfait et une autre dans la table ForfaitLimite ou ForfaitIllimite)

Il est important de noter qu'ici une jointure est impossible. En effet, on ne peut pas joindre les deux tables de forfaits car les numéros de forfaits sont exclusifs.

Traduction en relationnel

En appliquant les règles de passage au relationnel et nos choix de conception, on obtient le schéma relationnel suivant :

- ***Forfait (***NumeroForfait*, NumeroCB****)***
- ***Abonnes(***NumeroCB*, Nom, Prenom, AdressePostale, DateNaissance***)***
- ***Categorie(***Categorie*, DuréeMax, MontantCaution, PrixHoraire***)***
- ***Station(***NomSation*, AdresseStation***)***
- ***Vehicule(***IdVehicule*, NombrePlace, Categorie****)***
- ***ForfaitIllimité(***NumeroForfait**, DateDebut, Prix, Remise, NumeroCB****)***
- ***ForfaitLimité(***NumeroForfait**, Prix, NombreMaxLocation, NumeroCB****)***
- ***Location(***IdVehicule**, DateDebut, DateArrivee, NomStationDepart*, NomStationArrive*, NumeroCB****)***
- ***PeutContenir(***NomStation**, Categorie**, NbPlaces***)***
- ***AffectationForfait(***NumeroForfait**, NumeroCB**, Categorie****)***
- ***EstDans(***IDVehicule**, NomStation****)***

Nous avons donc un schéma relationel Contenant 11 tables toutes en 3FN BCK.

Contraintes non traitées lors du passage au relationnel

Lors du passage au relationnel, des contraintes ne pouvaient pas être vérifiées par un simple script SQL. Pour implémenter celles-ci, nous avons développé un programme Java permettant d'entrer des données dans la base. Lors des insertions, ces contraintes sont vérifiées. Celles-ci sont :

- **Insertion d'un forfait limité ou illimité** : on doit vérifier que l'utilisateur ne possède pas déjà de forfait pour la catégorie visée par ce nouveau forfait. De plus lors de l'insertion de ce forfait

dans la table correspondante (ForfaitLimite ou ForfaitIllimite), il faut aussi l'ajouter dans la table *Forfait* et *Association Forfait*.

- **Pour l'insertion d'un véhicule**, il faut insérer un tuple correspondant dans la table *EstDans* afin de préciser sa localisation dans une station.
- **Pour l'insertion d'une nouvelle location**, il faut vérifier que le véhicule soit disponible à la date de début de la location : il nn'est pas en location à cette date (rendu avant la date de début). Il faut aussi supprimer le véhicule de la table *EstDans*, le véhicule étant maintenant en location
- **Pour mettre à jour une location lors de l'arrivée dans une station** (fin de location et dépôt dans la station) il faut vérifier que celle-ci dispose encore de places disponibles pour la catégorie du véhicule à déposer. Il faut ensuite réinsérer ce dernier dans la table *EstDans* pour indiquer sa localisation dans la station d'arrivée.
- **Enfin pour l'insertion d'une station**, il faut que l'utilisateur rentre au moins une catégorie de véhicules que peut contenir cette dernière.
- **Lors de la suppression d'un forfait**, il faut veiller à bien le supprimer dans les trois tables (Forfait, AffectationForfait et ForfaitIllimité ou ForfaitLimité)

Ce programme d'insertion demande à l'utilisateur de choisir le type de données à insérer et demande ensuite toutes les informations nécessaires à l'insertion dans la base. Si le processus d'insertion est terminé avec succès, le programme le signale à l'utilisateur et les modifications sont validées dans la table. Si un problème survient au cours de l'insertion, l'utilisateur est informé du problème et les éventuelles modifications effectuées dans la base de données sont annulées.

II-Mise en place des fonctionnalités

Facturation

Pour la mise en place de cette fonctionnalité, nous avons fait plusieurs choix concernant son fonctionnement :

- Lors de la création d'une location, la station d'arrivée est à *null*. Cette dernière ne sera renseignée que lorsque que l'abonné arrivera à destination et rendra son véhicule. Ainsi, si une location possède une station d'arrivée à *null*, cela signifie qu'elle n'est pas terminée et on ne facture donc pas l'abonné.

- Pour les forfaits à période illimitée et locations limitées, on considère que si le nombre de de locations maximal vaut 0, alors le forfait n'est plus valable. Ainsi, on décrémentera le nombre de locations maximal après chaque facturation. On suppose donc le fonctionnement suivant :
  - L'utilisateur rend son véhicule
  - On facture l'utilisateur

    - On décrémente son forfait limité si il en possède un pour la catégorie de son véhicule

- En ce qui concerne les remises des forfaits à période limitée, ce sont des nombres compris entre 0 et 1. Par exemple un forfait à période limitée permettant une réduction de 30 % aura une remise égale à 0.7. Cela facilite les calculs de coûts (on multiplie le montant à payer par la remise).

- On considère que pour qu'un forfait à durée limitée soit valable pour une location donnée, il faut que les contraintes suivantes soient vérifiées :
    - DateDébutForfait ≤ DateDébutLocation
    - DateFinForfait ≥ DateFinLocation

Lors d'une facturation, l'utilisateur est invité à renseigner la date de départ de la location ainsi que l'ID du véhicule loué. Le programme Java calcule alors (si la location existe bien et est terminée) la durée de la location, l'âge de l'utilisateur et recherche le coût horaire correspondant au véhicule loué. De plus, il effectue une recherche de forfait de la façon suivante :

- Recherche dans la table AffectationForfait pour savoir si l'utilisateur possède bien un forfait pour la catégorie louée
- Si la requête précédente renvoie un numéro de forfait, on recherche dans la table ForfaitLimite puis ForfaitIllimite si néssecaire.
- Enfin on teste la validité du forfait (période de validité ou nombre de locations restantes)

Un fois cette recherche effectuée, on peut calculer le coût de la location et facturer l'utilisateur.

Temps moyen d'utilisation par véhicule par mois

Cette fonctionnalité affiche un tableau contenant les temps moyens d'utilisation d'un véhicule dans un mois. Par exemple, "2.00 heures" à la ligne "Véhicule 2", colonne "October", signifie qu'un utilisateur moyen loue en moyenne le Véhicule 2 pour une durée de 2 heures pendant le mois d'Octobre.

Celle-ci a été en majeure partie implémentée en Java, contrairement à la fonctionnalité similaire "Temps moyen d'utilisation par catégorie par mois" implémentée en SQL. Il nous a paru intéressant d'approcher les problèmes de deux manières différentes.

D'abord, le programme effectue une requête SQL pour extraire les locations terminées. On considère qu'une location en cours ne doit pas être prise en compte dans la moyenne.

La prochaine étape consiste à remplir une table de hachage, i.e. un tableau de listes.

La i-ème case du tableau contient la liste du véhicule indicesVehicules[i]. IndicesVehicules est un tableau reliant des indices manipulables (de 0 au nombre de véhicules distincts) à IDVehicule. Les listes sont composées d'une Durée (long, en millisecondes) et d'un Mois (int, de 0 à 11).

Ici, le programme va répartir les durées de locations selon le véhicule et le mois. Cependant, lorsqu'une location couvre plusieurs mois, il faut découper les locations en plusieurs durées, une pour chaque mois. Lorsque la location couvre plus de deux mois, il faut compter les mois intermédiaires entièrement.

Exemple : le Véhicule 1 est loué du 31 Janvier 12h00 au 1 Février 09h00. Alors la liste de tableau[1] contiendra les deux éléments : (12 heures, Janvier) et (09 heures, Février).

Enfin, le programme parcourt cette table de hachage, calcule les moyennes dans une matrice, et affiche le résultat.

Catégorie la plus utilisée par tranche d'âge

Pour mettre en place cette fonctionnalité nous avons envisagé deux approches. L'utilisateur peut ainsi, au choix :

- Sélectionner une tranche d'âge manuellement en entrant la borne d'âge inférieure et la borne supérieure. La catégorie la plus utilisée pour cette tranche d'âge est alors affichée.
- Ne pas entrer de tranche d'âge. Le calcul est alors effectué par tranche d'âge de 10 ans pour tous les abonnés présents dans la table (de 0 à la tranche d'âge contenant le plus vieil abonné).

Pour la détermination de la catégorie la plus utilisée, nous avons choisi de ne pas seulement nous baser sur le nombre de locations effectuées mais de considérer aussi leur durée. Ainsi c'est la catégorie avec le temps d'utilisation cumulé pour la tranche d'âge qui est considérée comme la plus utilisée.

Lors de son implémentation nous avons aussi choisi de comptabiliser les locations dans la tranche d'âge qu'avait alors l'abonné au moment de la location et non l'âge actuel de l'abonné. Cela permet un calcul cohérent de l'utilisation des catégories.

Temps moyen d'utilisation par catégorie par mois

Le principe de cette fonctionnalité est le suivant : L'utilisateur entre une catégorie de véhicule, une année et un mois. Le programme lui renvoie le temps moyen qu'un utilisateur a passé dans un véhicule de cette catégorie en jours et en heures. Des exceptions sont levées en cas d'erreur dans les entrées de l'utilisateur.

Lorsque l'utilisateur entre les paramètres cités ci-dessus, le programme vérifie grâce à une requête SQL que la catégorie existe. Ensuite, une seconde requête SQL renvoie (pour une catégorie et un mois donnés) :

- le cumul des temps de location sur ce mois
- le nombre d'utilisateurs concernés

Le temps de location est calculé pour prendre en compte les cas où : la location débute avant le mois choisi par l'utilisateur, elle finit après le mois choisi ou le cas "simple" (elle commence et se termine pendant le mois). Ce calcul est donc de la forme :

plus_ancien(fin_mois, fin_location) - plus_récent(début_mois, début_location)

Le temps moyen de location d'une catégorie est calculé en divisant le premier paramètre renvoyé par la requête SQL par le second. Dans le cas où aucun véhicule de la catégorie n'a été loué sur la période, le programme renverra 0.

La précision du calcul est de l'ordre de la minute car le début et la fin du mois sont calculés de manière artificielle comme étant le premier du mois à l'heure 00:00 et le dernier à l'heure 23h59. Cette précision semble cohérente avec l'ordre de grandeur des temps de location mais peut être améliorée en ajoutant les secondes à la principale requête SQL.

Taux d'occupation des stations sur la journée

Cette fonctionnalité a été écrite en majeur partie en Java. Voici les requêtes SQL utilisées :

- Liste des numéros d'identifiant de tous les véhicules existants (1)
- Liste des noms des stations (2)
- Liste de toutes les locations d'un seul véhicule rangées dans l'ordre anti chronologique (3)
- Liste de toutes les locations ayant la date de départ correspondant à la date demandée partant d'une même station S (4)
- Liste de toutes les locations ayant la date d'arrivée correspondant à la date demandée arrivant à une même station S (5)
- Sélection du nombre de places total dans une certaine station (6)

Fonctionnement du programme :

On notera DAY la date entrée par l'utilisateur.
Le programme fonctionne avec une boucle sur les stations (utilisation requête (2)).

Pour chaque station (notée S) le programme fonctionne en deux temps.

1. Nombre de véhicules à minuit

Dans un premier temps le programme doit déterminer combien de véhicules sont présents le jour demandé à minuit. Ce sera le nombre initial de véhicule dans la station.

Pour cela, le programme va regarder pour chaque véhicule (utilisation requête (1)) la liste des locations effectuées dans l'ordre anti chronologique (utilisation requête (3)).
Le programme peut facilement repérer la dernière location avant la date DAY (première transaction telle que DateDepart soit avant DAY).
Ensuite il suffit d'observer la date d'arrivée de cette location :

- Si elle est antérieure à DAY et que la station d'arrivée correspond à la station S alors nous savons que le véhicule est dans la station à minuit.
- Si elle est postérieure à DAY (ou nulle) et/ou que la station d'arrivée ne correspond pas à S alors le véhicule n'est pas pris en compte dans les véhicules présents à minuit.

Ainsi, après itération sur chaque véhicule nous savons combien de véhicules sont présents dans la station S à minuit.

2. Nombre maximum de véhicules dans la station

Ensuite il faut déterminer quel était le nombre maximum de véhicules présents au cours de la date DAY. Pour cela, grâce aux requêtes (4) et (5), il est possible de se mettre dans le référentiel de la station S. C'est à dire qu'on peut observer location après location dans l'ordre chronologique. Alors que si nous avions simplement sélectionné les n-uplets depuis location ayant (DateArrivee=DAY et StationArrivee=S) ou (DateDepart=DAY et StationDepart=S) alors les locations auraient été traitées dans l'ordre des dates de départ, cela aurait pu fausser les résultats.
Exemple : Départ de la station X à 2h00 → arrivée à la station S à 18h00
Départ de la station S à 8h00 → arrivée à la station X
Dans ce cas on aurait traité d'abord l'arrivée puis le départ. On aurait donc pu considérer un véhicule en trop dans la station.

Ainsi, le programme compte au fur et à mesure le nombre de véhicule dans la station et mémorise le maximum.

Le taux est obtenu simplement en divisant le maximum de véhicule dans la station sur le nombre total de places (requête (6)).
Nous avons choisi d'exprimer ce taux en pourcentage pour que cela soit le mieux exploitable.

Remarques sur le programme :

On parcourt plusieurs fois les locations pour chaque véhicule (partie 1) alors qu'on pourrait parcourir qu'une fois les locations pour chaque véhicule et ajouter sa présence au nombre de véhicule présent à minuit pour la station en question.
Ainsi, il faudrait retirer la boucle sur les stations et utiliser un dictionnaire par exemple (avec nomStation comme clé et un nbVehiculesAMinuit correspondant (ainsi qu'une variable tmp et max pour la partie 2)) .

Et pour la partie 2, il aurait fallu regarder chaque départ et arrivée durant DAY dans l'ordre (à l'aide de requêtes similaires à (4) et (5)) et de modifier la valeur tmp (et éventuellement max) selon la station concernée.

Notre méthode de développement fut de développer la fonctionnalité d'abord pour une station puis de généraliser à toutes. Par manque de temps, nous avons opté pour une solution fonctionnelle et plus rapide à développer mais plus coûteuse.

III-Bilan du projet

Lors de ce projet, nous nous sommes organisés de manière à séparer les tâches. Au début, une partie de notre équipe a travaillé sur l'élaboration du schéma relationnel tandis que certains d'entre nous se sont focalisés sur la compréhension du fonctionnement de JDBC.

Par la suite, nous avons réparti chaque fonctionnalité à réaliser entre les membres de notre groupe (une fonctionnalité par personne)

Enfin, nous avons élaboré ensemble un script de remplissage de la base de données cohérent (pas de locations simultanées pour un même véhicule par exemple) afin de tester nos différents codes.

La difficulté principale que nous avons rencontrée lors de ce projet a été la compréhension du sujet. En effet, plusieurs points étaient peu clairs (l'interprétation des requêtes notamment).

Cependant, nous avons réussi à terminer le projet dans les temps en implémentant toutes les fonctionnalités demandées.