Documentation BD

Thomas Conte, Mathis Gilliot, Mickael Ranaevoarisoa, Thomas Trompette et Nathan Zietek

Analyse du sujet

Nous avons d'abord analysé le problème en énumérant les dépendances fonctionnelles et les contraintes :

Dépendances fonctionnelles	**Contraintes de valeur**	**Contraintes de multiplicité**
id_vehicules → nb_places_veh	nb_places_veh > 0	id_vehicules ->> nom_station(0,1)
id_cat → duree_max, caution, prix_cat	duree_max >0 caution > 0 prix_cat > 0	
nom_station → adresse		nom_station ->> id_cat(1,*)
id_location → date_d, date_a, heure_d, heure_a	date_d < date_a	id_location ->> nom_station(0,1)
num_carte → nom, prenom, adresse_abonne, date_naissance, adr_postale, reduction		num_carte ->> id_location (0,*)
id_ill → duree, date_d, prix_ill, remise	Ext(id_ill) < Ext(id_forfait) duree > 0 prix_ill > 0	
id_lim → nb_location_max	Ext(id_lim) < Ext(id_forfait)	
id_forfait → id_cat, num_carte		
nom_station, id_cat → nb_places_cat		

Nous avons ensuite réalisé le schéma entités/associations, de la même forme que celui tiré de notre base de données. Celui-ci est disponible dans le fichier **diagramme.pdf**.
Nos relations finales sont de la forme suivante :

Vehicule(id_vehicule, nb_place_veh)
Abonne(num_carte, nom_abonne, prenom_abonne, date_naissance, adr_postale, reduction)
Categorie(id_cat, prix_cat, caution, duree_max)
Stations(nom_station, adr_station)
Locations(id_location, date_d, date_a, heure_d, heur_a, id_vehicule, station_d)

Forfait(id_forfait)
Illimite(id_forfait, duree, date_debut, prix_forfait, remise)
Limite(id_forfait, nb_location)
Estgare(id_vehicule, nom_station)
Terminea(id_location, nom_station)
Reserve(id_location, num_carte)
Accueille(id_cat, nom_station, nb_places_station)
Souscrire(num_carte, id_forfait, id_cat)

Nous avons enfin regardé leur forme normale. Tous les attributs qui ne sont pas des clés sont élémentaires. Les relations sont donc sous la forme 2FN.

Requêtes SQL

Nous avons commencé par les requêtes **Temps d'utilisation moyen par véhicule par mois** et **Temps d'utilisation par catégories de véhicule par mois** car elles se ressemblent beaucoup.

Il fallait d'abord connaître le temps d'utilisation pour une location, nous avons alors obtenu la formule suivante : heure_arrivée + 24 * (date_arrivée – date_départ) – heure_départ.
Pour obtenir la moyenne, nous avons utilisé la fonction SQL AVG() qui le fait directement.

Selon la requête, il fallait respectivement grouper le résultat par véhicule ou par catégories de véhicule, ce qui nous donne en SQL un simple GROUP BY id_véhicule et GROUP BY id_catégorie

Enfin, il faut s'assurer que les locations ont eu lieu pour le mois choisi :
WHERE ((date_départ= ?) OR (date_arrivée=?)) AND date_départ=*
Les '?' doivent être remplacés par le mois voulu et '*' par l'année souhaitée. Nous avons choisi de vérifier la date de départ OU la date d'arrivée pour inclure les locations qui commencent un mois et finissent le suivant. Car avec un ET, ces locations ne seraient jamais prises en compte.

Temps d'utilisation moyen par véhicule par mois (exemple novembre 2016):

```
SELECT id_vehicule,
        AVG(heure_a+24*(TO_DATE(date_a,'DD-MM-YYYY')-TO_DATE(date_d,'DD-MM-YYYY'))-heure_d)
FROM Locations
WHERE (TO_CHAR(date_a,'MM')=11 OR TO_CHAR(date_d,'MM')=11) AND
        TO_CHAR(date_d, 'YYYY')=2016
GROUP BY id_vehicule;
```

Temps d'utilisation par catégories de véhicule par mois (exemple janvier 2013) :

```
SELECT id_cat,
        AVG(heure_a+24*(TO_DATE(date_a,'DD-MM-YYYY')-TO_DATE(date_d,'DD-MM-YYYY'))-heure_d)
FROM Vehicules NATURAL JOIN Locations
WHERE TO_CHAR(date_a,'MM')=1 AND TO_CHAR(date_d,'MM')=1
        AND TO_CHAR(date_d, 'YYYY')=2013
GROUP BY id_cat;
```

Pour la requête **Catégories la plus utilisé par tranche d'âge de 10 ans**, il faut récupérer les

locations faites par des utilisateurs dans la tranche d'âge voulue :
WHERE date_naissance <= date_actuelle - ? AND date_naissance>= date_actuelle - ?
Les ' ?' sont remplacés par 10-19, 20-29... selon la tranche d'âge souhaitée.

Il fallait ensuite calculer le temps total d'utilisation par catégories de véhicules, avec la fonction SUM(), nous pouvons obtenir ce total et un GROUP BY id_cat permet de trier par catégories. Pour trouver la plus utilisée, il nous faut le maximum. Nous avons essayé d'utiliser la fonction SQL MAX() mais en l'utilisant, nous perdions la catégorie qui était lié au maximum. Nous avons donc décidé de laisser ce traitement pour la partie en Java.

Catégories la plus utilisé par tranche d'âge de 10 ans (exemple les 20-29 ans) :

```
SELECT id_cat,
        SUM(heure_a+24*(TO_DATE(date_a,'DD-MM-YYYY')-TO_DATE(date_d,'DD-MM-YYYY'))-
heure_d)
        AS Somme
FROM Vehicules NATURAL JOIN Locations NATURAL JOIN RESERVE NATURAL JOIN ABONNES
WHERE TO_CHAR(ABONNES.DATE_NAISSANCE,'YYYY')<=TO_CHAR(sysdate, 'YYYY')-20 AND
        TO_CHAR(ABONNES.DATE_NAISSANCE,'YYYY')>=TO_CHAR(sysdate, 'YYYY')-29
GROUP BY id_cat;
```

Pour la **facturation de la location**, nous avons d'abord étudier les éléments nécessaires pour le calcul du prix:

- le temps de location,
- la catégorie du véhicule loué et ses informations,
- l'âge de l'abonné
- le forfait pour cette catégorie que possède l'abonné

Le temps de location n'est pas compliqué à récupérer, nous l'avons déjà fait pour les requêtes précédentes.
Pour **la catégorie du véhicules et les informations la concerant**, il faut regarder la location, récupérer l'id du véhicule, puis sa catégorie, puis les informations sur la catégories :

```
SELECT id_vehicule, id_cat, duree_max, caution, prix_cat,
        heure_a+24*(TO_DATE(date_a,'DD-MM-YYYY')-TO_DATE(date_d,'DD-MM-YYYY'))-
heure_d
        AS TempsLoc
FROM Categories NATURAL JOIN Vehicules NATURAL JOIN Locations
WHERE (Locations.id_location= ?);
```

(Le ' ?' est changé pour l'id_location que l'on souhaite facturer.)

Pour **l'âge de l'abonné**, on récupère son numéro de carte depuis Reserve pour ensuite récupérer sa date de naissance :

```
SELECT num_carte, TO_CHAR(sysdate,'YYYY-MM-DD')-TO_CHAR(date_naissance, 'YYYY-MM-
DD')
FROM Reserve NATURAL JOIN ABONNES
WHERE id_location= ?;
```

Le ' ?' est changé pour l'id_location que l'on souhaite facturer.

Enfin, on cherche le **type de forfait** de l'abonné, on va utiliser les informations récupérer précédemment sur la catégorie et l'abonné pour le trouver :

```
SELECT *
FROM Illimites
WHERE ID_FORFAIT=(SELECT id_forfait
                  FROM Souscrire
                  WHERE num_carte= ? AND id_cat=*);


SELECT *
FROM Limites
WHERE ID_FORFAIT=(SELECT id_forfait
                  FROM Souscrire
                  WHERE num_carte= ? AND id_cat=*);
```

Si le forfait n'est pas dans Illimites, on le cherche dans Limites. On remplace le '?' par le numéro de carte récupérer avant et le '*' par la catégorie trouver au début. Le reste du traitement sera fait en Java car beaucoup plus simple. Effectivement il faut regarder la durée de la location, si cela dépasse le temps maximum, si l'abonné à moins de 25 ou plus de 65 ans,...

Par manque de temps pour la dernière requête **Taux d'occupation d'une station pour la journée**, nous avons décidé de calculer le taux d'occupation sur l'instant. Grâce à la table EstGare, nous avons une trace des véhicules garés et de la station dans laquelle ils sont garés. Nous allons donc d'abord récupérer le nombre de véhicule garé dans une station :

```
SELECT nom_station, COUNT(*) AS nbgaree
FROM ESTGARE
GROUP BY nom_station
```

Puis nous allons calculer le nombre total de place pour chaque station :

```
SELECT nom_station,SUM(nb_places_station) AS Total
FROM Accueille
GROUP BY nom_station
```

Il nous reste uniquement à calculer le rapport pour obtenir le taux d'occupation

```
SELECT nom_station, nbgare/total*100 AS Taux_occupation
FROM (  SELECT nom_station,SUM(nb_places_station) As total
        FROM Accueille
        GROUP BY nom_station)
NATURAL JOIN
        (SELECT nom_station, COUNT(*) AS nbgare
         FROM ESTGARE
        GROUP BY nom_station)
WHERE nom_station=' ?';
```

La dernière ligne permet de trouver le taux d'occupation pour une station précise en remplaçant le ',' par le nom, elle n'est pas obligatoire.

Mode d’emploi de l’application

Nous avons réalisé trois scripts java. Creation.java permet de créer les entités de la base de données, remplissage.java permet de la peupler et destruction.java permet de tout supprimer.

Le fichier des fonctionnalités se nomme vehlib.java.

Nous avons réalisé un Makefile pour compiler et exécuter les scripts ainsi que les fonctionnalités.

Il suffit de compiler, par exemple avec la commande :

make Ccreation

Puis de l’exécuter avec :

make creation

Bilan

Nous avons su nous répartir les tâches afin de répondre de notre mieux à la question posée. Notre base de données semble cohérente. Nous avons également pu réaliser les fonctionnalités demandées. Celles-ci semblent fonctionner, bien que notre peuplement de la base soit très réduit.