

# Rapport du projet

## Projet de Base de Données

### Objectifs

L'objectif principal était de créer une base de données fonctionnelle pour des stations de ski, capable de gérer certaines contraintes imposées par le client. Pour ce faire, nous sommes partie d'une relation non normalisée, contenant un nombre important d'information sur les stations de ski. Nous avons du d'abord décomposer cette relation et normalisée nos différentes tables, afin de créer notre base. Une fois ceci réaliser, nous devions faire respecter les contraintes du client à la base de donnée. Regardons plus en détail le travail réalisé.

### Créations de la base et normalisation

Le premier objectif était de décomposer cette table en préservant les dépendance fonctionnelles et sans perte d'information. Le but étant d'obtenir plusieurs tables qui respectent tous au minimum la 3ème forme normale.

Nous avons d'abord du imaginer le modèle conceptuel des données (MCD), qui vas nous permettre d'identifier les principales entités à représenter, leurs relations et leurs attributs. Nous obtenons le MCD suivant :

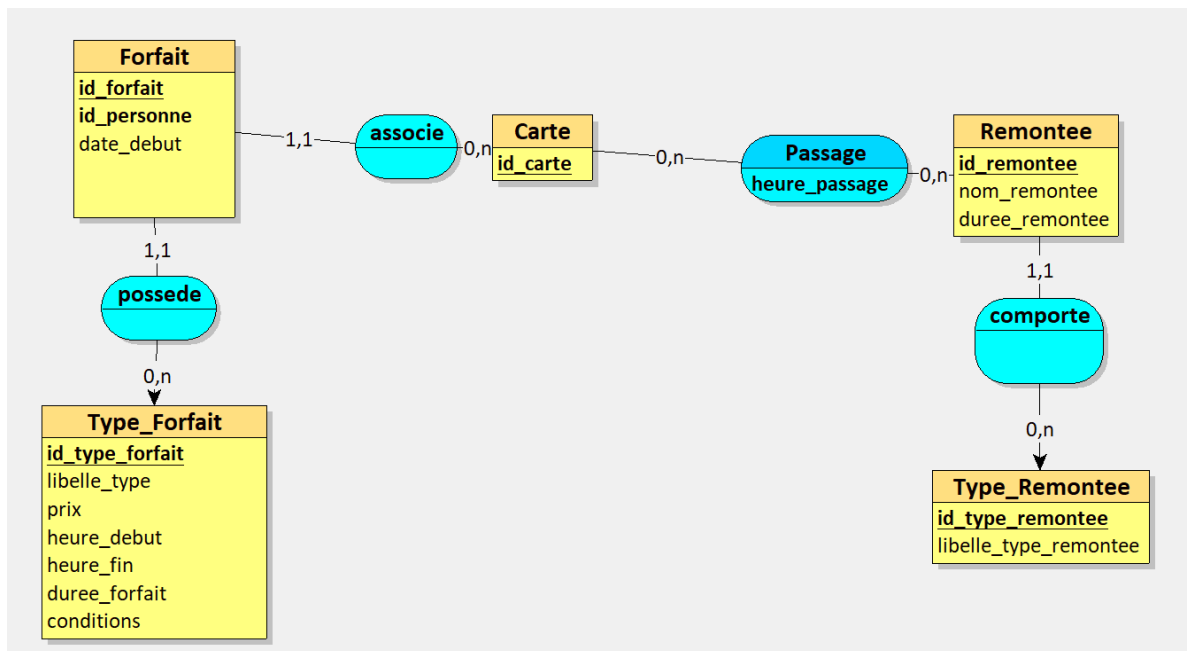


Figure 1: MCD

Une fois le MCD terminé nous avons pu facilement en tirer le modèle logique de données (MLD), afin d'avoir une représentation de toute nos tables, ainsi que leurs relations et leurs attributs. Voici le MLD que nous obtenons :

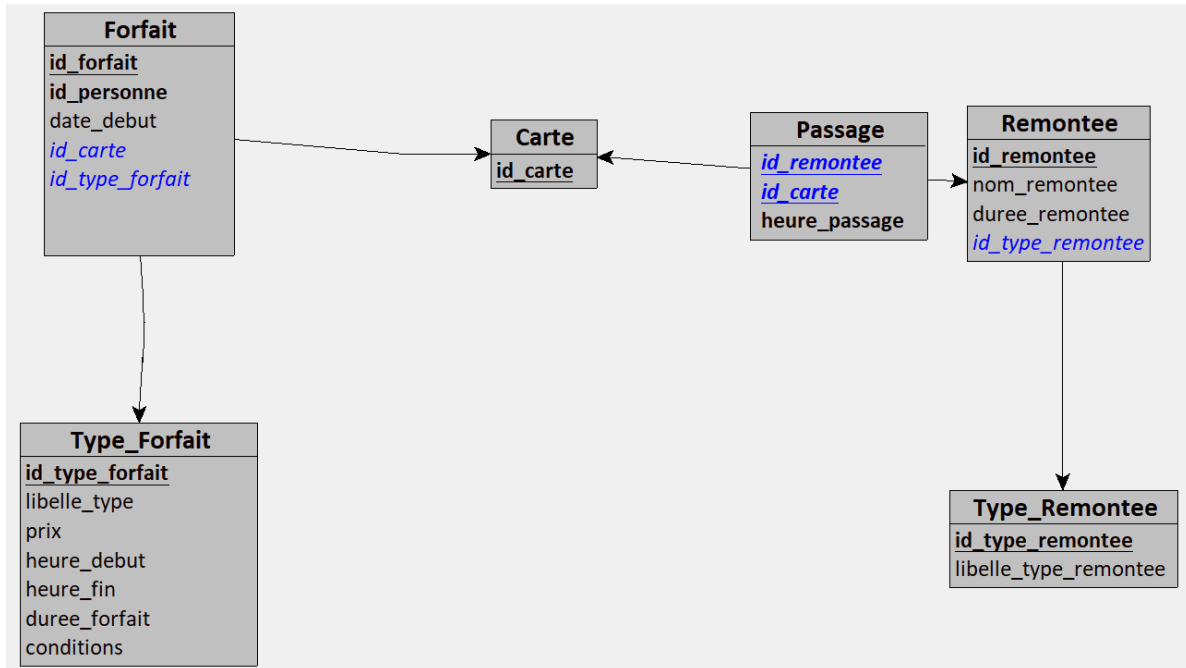


Figure 2: MLD

Nous pouvons voir grâce au MLD que chaque table respecte bien la 3ème forme normale. D'abord nous avons chacune des tables qui admettent une clé primaire, et tous les attributs les attributs contiennent une seule valeur pour chaque tuple. Ensuite, nous retrouvons bien que tous nos attributs non clé ne dépendent pas d'une partie de la clé primaire. Et enfin tous nos attributs n'appartenant pas à une clé ne dépendent pas d'un attribut non clé primaire. Tous ceci nous permet de valider la 3ème forme normale. Pour aller plus loin, nous pouvons admettre que toutes nos tables respectent la forme normale de Boyce-Codd, car tous les attributs non-clé ne sont pas source de dépendance fonctionnelle pour une partie de la clé.

La normalisation est maintenant terminée, nous connaissons les différentes tables qui composeront la base de données des stations de ski. Nous pouvons donc les créer et les alimenter grâce à la table de départ, et passer au deuxième objectif liée aux contraintes du client.

## Contrainte et déclencheur (Triggers)

Dans l'état actuel notre base de données contient certaines failles, qui ne peuvent être gérées directement. Le client nous a dicté ses contraintes qui lui semblent les plus importantes. C'est celle-ci que nous avons traitée puis testée. Voici les contraintes imposées :

- 1- Une carte ne peut pas être associée à deux forfaits qui ont une période de validité commune.
- 2- Une carte ne peut pas être associée à un nouveau forfait si l'ancien n'est pas encore utilisé.

3- On ne peut passer à une remontée mécanique que si le forfait est valide. 4- On ne peut passer à une remontée mécanique que si le forfait n'est pas temporisé.

On peut diviser ces contraintes en deux groupes de deux. La 1 et 2 qui s'appliqueront à notre table forfait, et la 3 et 4 qui s'appliqueront à la table passage. Nous avons donc créés 2 triggers qui font appelés à des fonctions définies au préalable. C'est dans ces fonctions que l'on fait respecter les différentes contraintes. Voyons plus en détails ces fonctions.

### Traitement des contraintes 1 et 2

Tout d'abord avec la fonction pour les deux premières contraintes. L'objectif va être de compter le nombre de forfait qui ne respecte pas nos contraintes par rapport au forfait que nous ajoutons.

```
DECLARE
new_date_debut DATE;
new_duree_forfait INTEGER;
nb_forfaits INTEGER;
```

Figure 3: Declare Trigger 1 & 2

On commence pas initialiser 3 variables:

new\_date\_debut : La date de début du nouveau forfait.

new\_duree\_forfait : La durée du forfait que nous voulons ajouter.

nb\_forfaits : Le nombre de forfait qui ne respecte pas nos contraintes. Nous devons ensuite attribuer des valeur à chacune de ses variables.

```

BEGIN
-- On teste la date de début du nouveau forfait,
--si elle est nulle alors on met la date du jour
IF (NEW.date_debut IS NULL) THEN
new_date_debut = (SELECT CURRENT_DATE);
ELSE
new_date_debut = NEW.date_debut;
END IF;

```

Figure 4: new\_date\_debut

Pour la new\_date\_debut on s'assure que la valeur ne soit pas nulle. Si elle l'est, on lui donne comme valeur la date du jour actuel.

```

-- on récupère la durée du forfait
new_duree_forfait = (SELECT duree_forfait
                     FROM type_forfait
                     WHERE id_type_forfait=NEW.id_type_forfait);

```

Figure 5: new\_duree\_forfait

Pour la new\_duree\_forfait, on doit récupérer la valeur de celle-ci liée au type de forfait qu'on souhaite ajouter. La requête suivante vas donc chercher cette valeur, dans notre table type\_de\_forfait, pour l'id\_type\_forfait liée au forfait que nous ajoutons.

Une fois ces deux variables initialiser nous pouvons chercher le nb\_forfaits.

```

-- requête qui compte le nombre de forfait valides utilisant la carte
nb_forfaits = (SELECT count (id_forfait)
              FROM forfait NATURAL JOIN type_forfait
              WHERE id_carte=NEW.id_carte
              AND ((new_date_debut,new_date_debut + new_duree_forfait) OVERLAPS (date_debut,date_debut+duree_forfait)
                  OR (date_debut is NULL))
              );

```

Figure 6: nb\_forfaits

Pour les cartes qui correspondent à la carte liée au forfait qu'on veut ajouter, on va d'abord chercher si la période de notre nouveau forfait chevauche une période d'un autre forfait déjà présent dans notre table forfait. Ceci va nous permettre de vérifier la première contrainte. Ensuite on cherche si la carte du nouveau forfait que l'on ajoute, correspond à un forfait avec une carte qui n'a pas encore été utilisée. Dans nos données cette information est spécifiée avec une date de début à valeur null. Ici ce sont les deux contraintes qui vont être prises en compte. On compte les forfaits qui respectent ces conditions, ce qui constitue notre variable nb\_forfaits.

```
-- si la carte est déjà utilisée, alors je lève une exception
IF ( nb_forfaits > 0) THEN
RAISE EXCEPTION 'La carte % n'est pas disponible !!!', NEW.id_carte;
END IF;
RETURN NEW;
END
```

Figure 7: Trigger 1 & 2

S'il y a un forfait ou plus qui sont comptés, cela signifie que nos contraintes ne sont pas respectées. Une exception sera levée et signalera que la carte n'est pas disponible, en spécifiant le numéro de la carte. La fonction maintenant terminée il faut vérifier si elle fonctionne.

## Test des contraintes 1 et 2

Comme dit précédemment les deux premières contraintes s'appliquent sur la table forfait. La vérification doit se faire avant l'insertion dans cette table.

Query Query History

1

--TEST TRIGGER 1--

2

3 insert into forfait (id\_forfait,date\_debut,id\_carte,id\_type\_forfait)

4 values (123456789,'2022-10-12',1,7);

5

6 insert into forfait (id\_forfait,date\_debut,id\_carte,id\_type\_forfait)

7 values (12345678,'2022-10-15',1,7);

8

9 -- delete from forfait

10 -- where date\_debut in ('15/10/2022','12/10/2022');

11

12

Data output Messages Notifications

ERROR: ERREUR: La carte 1 nest pas disponible !!!  
CONTEXT: fonction PL/pgSQL carte\_utilisee(), ligne 35 à RAISE

SQL state: P0001

Figure 8: Test trigger 1

On commence par essayer la contrainte 1. Pour cela, on ajoute dans un forfait qui n'est pas présent dans nos données. Ensuite on recommence l'opération en ajoutant un forfait avec le même numéro de carte, et en prenant soin de bien avoir une période qui chevauche le forfait précédemment insérer. Ceci vas donc déclancher une exception informant que la carte n'est pas disponible. La contrainte 1 est bien respectée. Une fois cela réalisé on supprime le forfait utilisé pour le test.

Query

Query History

1

--TEST TRIGGER 2--

2

3

insert into forfait (id\_forfait,date\_debut,id\_carte,id\_type\_forfait)

4

values (123456789,NULL,1,1);

5

6

insert into forfait (id\_forfait,date\_debut,id\_carte,id\_type\_forfait)

7

values (12345678,NOW(),1,1);

8

9

10

-- delete from forfait

11

-- where date\_debut is NULL

12

-- and id\_forfait = 12345678;

Data output

Messages

Notifications

ERROR: ERREUR: La carte 1 nest pas disponible !!!

CONTEXT: fonction PL/pgSQL carte\_utilisee(), ligne 35 à RAISE

SQL state: P0001

Figure 9: Test trigger 2

Pour la contrainte 2 nous réalisons une opération assez similaire. On essaye d'ajouter deux forfaits avec le même numéro de carte, des période qui ne se chevauchent pas, pour éviter toute confusion avec la contrainte 1, et une date nulle pour le premier forfait. Le premier va réaliser l'insertion avec succès, cependant le deuxième forfait va déclencher l'exception liée à la fonction. La seconde contrainte est respectée on peut maintenant supprimer le forfait test, et passer aux contraintes 3 et 4 sur la table passage.

### Traitement des contraintes 3 et 4

Pour ces contraintes nous les traitons ensemble dans une même fonction, mais l'exception déclenchée et les variables utilisées varient d'un trigger à un autre.



```

BEGIN
-- Si l'heure de passage est null mettre la date et heure actuelle
if (new.heure_passage is NULL) then
new_heure_passage = NOW();
else
new_heure_passage=new.heure_passage;
end if;

```

Figure 10: new\_heure\_passage

La variables new\_heure\_passage sera la seule utilisée pour la gestion des deux contraintes. Elle correspond à la date et l'heure de passage dans une remontée pour une carte. Si lors de l'insertion cette valeur est nulle, elle prendra la date et l'heure actuelle.

Nb : Cette contrainte n'est pas forcément nécessaire car l'heure de passage fait partie de la clé primaire de la table passage, et ne sera donc jamais nul.

Maintenant regardons plus en détail le code qui sert à faire respecter la troisième contrainte.

```

DECLARE
-----TRIGGER 3-----
id_forfait_t integer;
date_fin DATE;
heure_fin_t TIME;
date_heure_fin timestamp without time zone;
-----POUR LES DEUX TRIGGERS-----
new_heure_passage timestamp without time zone;

```

Figure 11: Declare trigger 3

On commence par déclarer les variables que nous utiliserons. Il y en a 4 :

```

id_forfait_t
date_fin
heure_fin_t
date_heure_fin

```

```

-- On cherche le forfait le plus récent pour la carte à ajouter
id_forfait_t =(select distinct (id_forfait)
                from forfait
                where id_carte = new.id_carte
                and date_debut in (select distinct max(date_debut)
                                   from forfait
                                   where id_carte = new.id_carte
                                   )
                );

```

Figure 12: id\_forfait\_t

La variable id\_forfait\_t correspond à l'id\_forfait du forfait le plus récent, c'est-à-dire avec la plus grande date de début dans notre table forfait, pour la carte que nous souhaitons ajouter.

```

-- On determine la date de fin du forfait ainsi que l'heure de fin du forfait
date_fin = (select date_debut+duree_forfait
              from type_forfait natural join forfait
              where id_forfait = id_forfait_t);

heure_fin_t = (select heure_fin
               from type_forfait natural join forfait
               where id_forfait = id_forfait_t);

date_heure_fin =date_fin+heure_fin_t;

```

Figure 13: date\_fin, heure\_fin\_t, date\_heure\_fin

On détermine ensuite la date de fin, ainsi que l'heure de fin, pour notre id\_forfait\_t. On va récupérer ces valeurs dans la table type\_forfait. L'heure de fin est donnée, tandis que la date de fin doit être calculé en ajoutant la durée du forfait, à la date de début du forfait correspondant à notre id\_forfait\_t. On concatène les variables date\_fin et heure\_fin\_t pour obtenir la date\_heure\_fin, qui correspond à la date et l'heure d'expiration du forfait liée à la carte que nous essayons de faire passer.

```

-- Si le forfait est expiré on declanche une erreur
IF ( date_heure_fin < new_heure_passage) THEN
RAISE EXCEPTION 'Le forfait est expiré !!!';

END IF;

```

Figure 14: Trigger 3

Si cette date et heure d'expiration est inférieur à la date et l'heure de passage, alors on lève une exception qui spécifiera que le forfait est expiré. La contrainte 3 est maintenant gérée, regardons le code pour la contrainte 4.

```

-----POUR LES DEUX TRIGGERS-----
new_heure_passage timestamp without time zone;
-----TRIGGER 4-----
dernier_passage timestamp without time zone;
duree_remontee_t interval;
moment_interdit timestamp without time zone;

```

Figure 15: Declare trigger 4

Ici seulement 3 variables sont spécifiques pour cette contrainte. Ce sont les suivantes :

```

dernier_passage
duree_remontee_t
moment_interdit

```

```

-----TRIGGER 4-----

--On cherche la dernière fois que la carte est passé dans cette remontée
dernier_passage = (select max(heure_passage)
                  from passage
                  where id_remontee= new.id_remontee
                  and id_carte = new.id_carte);

--On détermine ensuite la période ou le forfait n'a pas le droit d'être utilisé pour cette remontée
duree_remontee_t = (select duree_remontee
                  from remontee
                  where id_remontee= new.id_remontee
                  );

moment_interdit = dernier_passage + duree_remontee_t;

```

Figure 16: Trigger 4 request

On commence par chercher, pour la carte que l'on souhaite ajouter, le dernier passage dans la remontée, que l'on veut ajouter également. On stock le résultat dans la variable `dernier_passage`. On récupère ensuite la durée de la remontée que l'on veut ajouter qui correspond à la variable `duree_remontee_t`. Et on ajoute cette durée au dernier passage pour avoir le moment où le passage sera de nouveau autorisé.

```

--Si la carte repasse dans la même remontée avant la fin de la période une erreur est déclenché
if (new_heure_passage < moment_interdit) then
RAISE EXCEPTION 'Vous venez juste de passer sur cette remontée, attendre le % pour réessayer !!!', moment_interdit;
end if;

```

Figure 17: Trigger 4

Si l'heure de passage est inférieure à ce moment, une exécution est levée signalant que le passage à cette remontée vient juste d'être effectué, précise le moment où le passage peut de nouveau être réalisé. Ce qui fait respecter la contrainte 4. On peut maintenant tester si tout cela fonctionne correctement.

## Test des contraintes 3 et 4

Nos contraintes seront traitées avant l'insertion dans la table `passage`.

Query	Query History
1	<code>insert into</code> passage(id_carte,id_remontee,heure_passage)
2	<code>values</code> (3,4,NOW());
3	-- On test avec une carte qui à une date expiré
4	--ce qui declanche le trigger car la date est expiré
5	
6	
7	
8	
9	
10	
11	
12	

Data output	Messages	Notifications
ERROR: ERREUR: Le forfait est expiré !!!		
CONTEXT: fonction PL/pgSQL jsp_encore(), ligne 48 à RAISE		

SQL state: P0001

Figure 18: Trigger 3 & 4 insert 1

Pour la contrainte 3 le test est très simple. Beaucoup de forfait dans notre base ont une période non valide, il suffit d'ajouter un passage pour une carte et une remontée. Ceci déclanchera l'exception, notre fonction pour le trigger 3 est fonctionnelle.

Query	Query History
6	<b>INSERT into</b> forfait (id_forfait,date_debut,id_carte,id_type_forfait)
7	<b>values</b> (123456789,NOW(),3,7);
8	--On insere un nouveau forfait qui aura une date valide
9	--car il n y en a pas dans la base
10	
11	
12	
13	
14	
15	
16	
17	

Data output	Messages	Notifications
INSERT 0 1		
Query returned successfully in 88 msec.		

Figure 19: Trigger 3 & 4 insert 2

Pour la contrainte 4 nous devons procéder en plusieurs étapes. On va d'abord ajouter un nouveau forfait qui aura une période valide.

Query	Query History
21	<code>insert into passage(id_carte,id_remontee,heure_passage)</code>
22	<code>values (3,4,NOW());</code>
23	<code>-- On test l'ajout dans passage</code>
24	<code>-- Ceci ne declanche pas le trigger !!</code>
25	
26	
27	
28	
29	
30	
31	
32	

Data output	Messages	Notifications
INSERT 0 1		
Query returned successfully in 82 msec.		

Figure 20: Trigger 3 & 4 insert 3

On vas ensuite insérer un nouveau passage, pour la carte et une remontée de ce forfait, avec l'heure et la date du moment comme heure de passage. Ceci ne vas pas déclancher l'exception de la contrainte 4.

```
Query  Query History
27  insert into passage(id_carte,id_remontee,heure_passage)
28  values (3,4,NOW());
29  --On insere la meme carte pour la meme remontee toute de suite apres
30  --Ce qui declanche le trigger car le passage vien d'etre effectué
31
32  --suppressions des données test :
33  -- DELETE from passage
34  -- where passage.id_carte = 3
35  -- and id_remontee=4 ;
36  -- DELETE from forfait where id_forfait = 123456789;
37
38

Data output  Messages  Notifications
ERROR: ERREUR:  Vous venez juste de passer sur cette remontée, attendre le 2022-11-15 17:56:33.732871 pour
réessayer !!
CONTEXT:  fonction PL/pgSQL jsp_encore(), ligne 70 à RAISE

SQL state: P0001
```

Figure 21: Trigger 3 & 4 insert 4

On essaye tout de suite après de refaire la même insertion. Comme l'heure de passage aura changer ceci devrait être possible, mais la contrainte 4 n'étant pas respecté, ceci lève l'exception et indique quand le passage de cette remontée sera de nouveau possible pour cette carte. Nous avons bien réussi à gérer toutes les contraintes imposées par notre client.

## Conclusion

La base des station de ski est maintenant opérationnelle. Nous avons réussi à ciblé toutes les entités, les attributs, et les relations, qui permettent une gestion des données simplifiées. Nous avons ensuite schématisé et crée notre base. Une fois ce travaille réalisé, nous avons fait en sorte de gérer les contraintes imposée par le client, tout en les testants au préalable. Ce projet nous à permis de mettre en oeuvre un grand nombre de nos connaissances en SQL, en passant par la création de base et la gestion des données. Nous avons tout de même fait face à des difficultés, que ce soit par l'utilisation de PgAdmin en début de projet, ou pour l'interprétation des contraintes en langage SQL. Malgré cela, nous avons réussi à aller au bout de nos idées, et obtenons un rendu complet répondant à l'ensembles des demandes du client.