

COMPTE RENDU DU PROJET :

LUXIOURS CAMPING

Projet préparé par :

Boulland-Genet Nicolas

Ben Amira Rawia

L2 informatique

2020/2021

Introduction

Dans ce projet, il nous a demandé de créer un site web qui gère un camping à partir d'une base de données ainsi de pouvoir permettre à des clients de réserver des emplacements, participer à des activités au sein de ce camping, commander des menus etc ..

Pour avoir un site fonctionnel et qui répond aux critères demandés, il faut avoir un minimum de savoir en python, html, et postgres, en postgres il faut savoir créer des tables avec leurs clés primaires et étrangères, des domaines d'intégrité ainsi que créer des triggers et des fonctions qui gèrent les insertions et les suppressions des différentes lignes ou colonnes des tables.

Les langages utilisés afin de nous permettre de réaliser ce projet sont :

Python et html pour la conception du site

Sql (postgres) pour créer la base de données

Notepad pour la rédaction des codes

Presentation du projet

Notre camping est intitulé luxiours_camping , c'est un camping de « luxe » qui contient des emplacements qu'on pas beaucoup l'habitude de voir dans les camping ordinaires , il propose des activités au milieu de la nature animés par des employes , propose aussi des menus à commander dans un petit restaurant , ainsi que d'inscrire les enfants dans un mini_camping qu'on ne géra pas mais juste on pourra calculer le tarif du séjour .

Le tarif des emplacements varie selon la saison durant laquelle le client décide de s'inscrire.

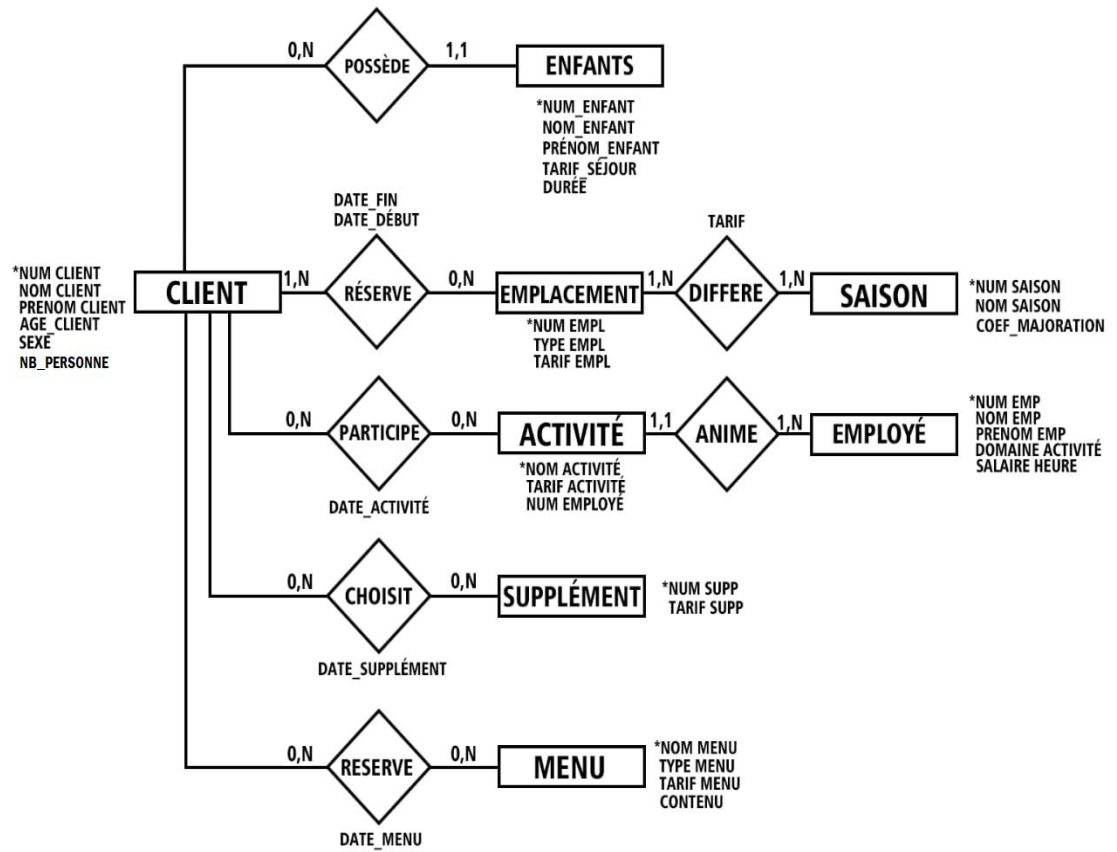
Pour cela on a créé les tables suivantes :

```
postgres=# \d
```

Liste des relations			
Schéma	Nom	Type	Propriétaire
luxiours_camping	activite	table	postgres
luxiours_camping	client	table	postgres
luxiours_camping	emplacement	table	postgres
luxiours_camping	employes	table	postgres
luxiours_camping	enfant	table	postgres
luxiours_camping	menu	table	postgres
luxiours_camping	reservation_activite	table	postgres
luxiours_camping	reservation_menu	table	postgres
luxiours_camping	reservation_supplement	table	postgres
luxiours_camping	saison	table	postgres
luxiours_camping	saison_emplacement	table	postgres
luxiours_camping	sejour	table	postgres
luxiours_camping	supplement	table	postgres

(13 lignes)

Détaillons maintenant la MEA ainsi que les tables et les relations qui les relient :



On remarque de première vue que la table client est liée quasiment à toutes les tables, dans cette partie on va traiter la partie des cardinalités et des relations qui lient les différentes tables :

La relation client-emplacement est de type N-M, en effet un client ne peut devenir réellement client que s'il réserve un emplacement, un client peut réserver plusieurs emplacements mais a des dates différentes c'est pour cela on a la relation reservation_emplacement qui est définie par la date_deb_sejour date_fin_sejour ainsi que le num_client et num_emplacement

Pour les relations reservation_supplement , reservation_menu , reservation_activite sont de type N-M avec pour le client une cardinalité minimale à 0 car un client peut réserver un emplacement et ne jamais commander de plat , jamais participer à des activités etc. ces tables sont définies par des dates , par exemple un client commande le même plat plusieurs fois mais a des dates différentes .

Ici la traduction de la MEA en model relationnel

Model relationnel :

Client(num_client , nom_client , prenom_client , sexe , age)

Enfant(num_enfant , nom_enfant ,prenom_enfant ,sexe ,age
,duree_sejour_enfant , tarif_sejour_enfant

Emplacement (num_emplacement , type_emplacement ,
tarif_emplacement)

Sejour (#num_client,#num_emplacement, date_deb_sejour ,
date_fin_sejour)

Activite (nom_activite, tarif_activite, #num_employe)

Supplement (nom_supplement , tarif_supplement)

Menu (num_menu , type_menu , tarif_menu , contenu)

Reservation_activite (#num_client, #nom_activite , date_activite)

Reservation_supplement (#num_client, #nom_supplement ,
date_supplement)

Reservation_menu (#num_client,#num_menu ,date_menu)

Employes (num_employe , nom_employe ,prenom_employe,
domaine_activite , salaire_heure)

Saison(num_saison ,nom_saison , coefficient_majoration)

Saison_emplacement(#num_saison,#num_emplacement , tarif)

La partie BDD

Table client :

Le script sql qui permet de créer cette table est le suivant :

```
-----  
CREATE TABLE client (  
num_client    INTEGER PRIMARY KEY,  
Nom           VARCHAR(50),  
sexe         dom_sexe ,  
age          dom_age ,  
nb_personne  dom_nb_personne ;  
Prenom       VARCHAR(50) ) ;
```

La table client représente la principale table qui est lié quasiment a toutes les autres tables, en effet un client peut réserver un emplacement, participer à une activité, commander un menu, un supplément ou inscrire son enfant dans un camping lors de son séjour.

Chaque client est identifié par un num_client qui représente la clé primaire de type entier, un nom et un prénom chacun de type varchar de longueur maximum de 50 lettres, son sexe qui est de type dom_sexe et son âge de type dom_age.

On a supposé que chaque emplacement ne peut contenir plus que 5 personnes, d'où la colonne nb_personne qui est de type le domaine qu'on a créé qui vérifie avant chaque insertion que le nombre de personne qu'un client peut inscrire avec lui ne dépasse pas les 5.

Voici les deux scripts pour créer les domaines si dessus :

```
CREATE DOMAIN dom_nb_personne AS INTEGER CHECK(VALUE<5) ;  
CREATE DOMAIN dom_sexe AS CHAR(1) CHECK (VALUE IN ('M','F'));  
CREATE DOMAIN dom_age AS INTEGER CHECK (VALUE > 18);
```

```
postgres=# \dD
```

Liste des domaines						
Schéma	Nom	Type	Collationnement	NULL-able	Par défaut	Vérification
luxiours_camping	dom_age	integer				CHECK (VALUE > 18)
luxiours_camping	dom_age_enfant	integer				CHECK (VALUE <= 18)
luxiours_camping	dom_nb_personne	integer				CHECK (VALUE < 5)
luxiours_camping	dom_sexe	character(1)				CHECK (VALUE = ANY (ARRAY['M'::bpchar, 'F'::bpchar]))

(4 lignes)

Le domaine âge est très important, car un client doit avoir un âge supérieur à 18 ans pour s'inscrire en effet on ne propose pas d'activité pour les enfants de moins de 18 ans, des séances de yoga ou de paintball ne sont pas adéquats a des enfants, c'est pour cela qu'on suppose que si des parents ont des enfants ils peuvent les inscrire dans un autre camping via notre site, un camping dédié aux enfants mais qu'on ne gère pas sur ce site.

Table enfant :

Le script pour créer la table enfant :

```
CREATE TABLE enfant (
num_enfant INTEGER PRIMARY KEY,
Nom_enfant VARCHAR(50),
Prenom_enfant VARCHAR(50),
sexe dom_sexe ,
age dom_age_enfant,
duree_sejour_enfant INTEGER ,
tarif_sejour_enfant INTEGER ,
num_client INTEGER ,
CONSTRAINT fk_num_client FOREIGN KEY (num_client)REFERENCES
client(num_client));
```

La table enfant est une table qui enregistre les informations des enfants des clients qui ont souhaité inscrire leurs enfants dans l'autre camping en

collaboration. Cette table contient num_enfant qui est une clé primaire de type entier permettant d'identifier un enfant, un nom_enfant prenom_enfant de type varchar, sexe de type dom_sexe déjà défini et âge de type dom_age_enfant , duree_sejour_enfant de type entier qui représente le nombre de jour que le parent souhaite inscrire son enfant , tarif séjour enfant de type entier qui représente la somme que le parent doit payer , ce tarif va être calculé automatiquement à partir de la durée du séjour , pour cela on a créé une fonction pour le calcul et un trigger pour déclencher ce calcul à chaque insertion d'un nouvel enfant :

```
CREATE OR REPLACE FUNCTION calcul_tarif_sejour_enfant()  
  
RETURNS trigger  
  
AS $$  
  
DECLARE  
  
BEGIN  
  
UPDATE luxiours_camping.enfant SET tarif_sejour_enfant = duree_sejour_enfant*40  
  
WHERE num_enfant = new.num_enfant;  
  
RETURN new;  
  
END ;  
  
$$  
  
LANGUAGE plpgsql ;
```

Le trigger qui déclenche la fonction ci-dessus :

```
CREATE TRIGGER calcul_sejour_tarif  
  
AFTER INSERT  
  
ON luxiours_camping.enfant  
  
FOR EACH ROW  
  
EXECUTE PROCEDURE calcul_tarif_sejour_enfant();
```

Cette table contient aussi le num_client (numéro du parent) en effet un enfant pour qu'il soit inscrit il faut que son parent le soit aussi

On n'a pas créé une table avec un lien n-m car on suppose que certes un parent peut avoir ou pas un enfant peut l'inscrire ou pas mais aussi un enfant ne peut être inscrit qu'avec un seul parent, si les deux parents sont présents et veulent inscrire leurs enfants ils devront choisir avec quel num_client inscrire leurs enfants.

Le script pour créer le domaine dom_age_enfant :

```
CREATE DOMAIN dom_age_enfant AS INTEGER CHECK (VALUE < 18) ;
```

Un parent ne peut inscrire son enfant que si son âge est inférieur à 18 ans sinon il peut l'inscrire avec lui dans notre camping.

Table activite

Le script pour créer la table

```
CREATE TABLE Activite (
```

```
Nom_activite VARCHAR(50) PRIMARY KEY ,
```

```
tarif_activite integer ;
```

```
ALTER TABLE activite ADD CONSTRAINT fk_num_employes FOREIGN KEY  
(num_employe) REFERENCES employes(num_employe) ;
```

C'est une table qui contient comme clé primaire de type varchar le nom des activités disponibles du camping, le tarif de chaque activité de type entier et comme clé étrangère num_employe qui correspond au numéro des employés qui animent ces activités, ici on ne créera pas une relation avec un lien n-n car on suppose qu'un employé n'anime qu'un seul type d'activité, par exemple un employé x n'anime que le rafting il ne peut pas animer en même temps rafting et paddle

Table employes :

```
CREATE TABLE Employes (
```

```
num_employe          INTEGER PRIMARY KEY,
```

```
Nom_employe VARCHAR(50),
```

```
domaine_activite VARCHAR(50),
```

```
salaire_heure  INTEGER ,
```

```
Prenom_employe VARCHAR(50)) ;
```

C'est une table qui regroupe les employés du camping qui participent à l'animation des activités.

Cette table contient le numéro attribué à chaque employé qui est de type entier clé primaire ainsi que le nom prénom et domaine d'activité de chaque employé en effet un employé ne peut animer qu'un seul type d'activité.

Aussi à chaque employé attribue un salaire par heure

Table reservation_activite

Script pour créer la table reservation_activite

```
CREATE TABLE Reservation_Activite (  
    num_client INTEGER ,  
    nom_activite VARCHAR(50),  
    date_activite DATE ,  
    PRIMARY KEY(num_client,nom_activite,date_activite),  
    CONSTRAINT FK_num_client FOREIGN KEY (num_client) REFERENCES  
    client(num_client),  
    CONSTRAINT FK_Id_activite FOREIGN KEY (nom_activite) REFERENCES  
    activite(nom_activite));
```

Cette table permet de réserver une activité. Elle est définie par le num_client et num_activite qui forment avec la date de l'activité la clé primaire car sinon un client ne pourra plus participer à une activité s'il a déjà participé auparavant

Mais aussi un client ne peut pas participer deux fois le même jour à la même activité car par jour on n'organise qu'une seule fois la même activité, par exemple le 2 septembre 2020 on a une seule séance de rafting, une seule séance de cinéma etc

Pour pouvoir suivre le nombre de réservation de chaque activité par chaque client, on pose la requête suivante :

```
select r.num_client,c.nom,c.prenom,count(r.nom_activite) AS  
nombre_activite_par_client,r.nom_activite
```

```

from client c , reservation_activite r

where c.num_client = r.num_client

group by r.num_client,c.nom,c.prenom,r.nom_activite ;

```

num_client	nom	prenom	nombre_activite_par_client	nom_activite
2	ben amir	rawia	1	cinema en plein air
2	ben amir	rawia	2	paddle
2	ben amir	rawia	1	rafting

(3 lignes)

Ici par exemple le client de numéro 2 a participé à une séance de rafting, 2 de paddle et une de cinéma en plein air. Ceci peut être intéressant si on veut calculer par exemple le tarif global des clients.

Table emplacement

Script pour créer la table emplacement :

```

CREATE TABLE Emplacement(
num_emplacement INTEGER PRIMARY KEY ,
Nom_emplacement VARCHAR(50),
tarif_emplacement INTEGER );

```

La table emplacement regroupe les emplacements disponibles dans notre camping

Chaque emplacement est identifié par un numéro qui représente sa clé primaire, le nom de l'emplacement de type varchar et le tarif emplacement par nuit qui est de type entier pour que le client puisse choisir

Table saison :

Le script pour écrire la table saison

```
CREATE TABLE saison(  
num_saison INTEGER PRIMARY KEY ,  
Nom_saison VARCHAR(50),  
coefficient_majoration FLOAT );
```

Les tarifs des emplacements varient selon la saison, c'est la raison pour laquelle cette table a été créée

Elle contient le numéro de la saison qui est la clé primaire, le nom de la saison et le coefficient de majoration qui est float .

Table sejour :

Le script pour créer la table sejour :

```
CREATE TABLE sejour (  
num_client INTEGER ,  
num_emplacement INTEGER ,  
date_deb_sejour DATE ,  
date_fin_sejour DATE CHECK (date_deb_sejour < date_fin_sejour),  
PRIMARY KEY (num_client , num_emplacement ,date_deb_sejour) ,  
CONSTRAINT fk_num_emplacement FOREIGN KEY (num_emplacement) REFERENCES  
emplacement(num_emplacement),  
CONSTRAINT fk_num_client FOREIGN KEY (num_client) REFERENCES client  
(num_client));
```

la table séjour fait la liaison entre la table client et la table emplacement, c'est un lien n,n qui est non porteur d'information en effet un client peut réserver plusieurs emplacements a différentes date .

Cette table contient deux clés étrangères qui sont le numéro du client et le numéro de l'emplacement, avec la date du début de séjour ils forment la clé primaire en effet un même client ne peut réserver un même emplacement a une date que s'il n'a pas déjà réservé à cette date-là.

Date_deb_sejour et date_fin_sejour sont de type date qui représente le séjour de chaque client, et a vérifier a chaque réservation que la date de début de séjour est inférieur a la date de fin de séjour.

Pour qu'un client puisse réserver un séjour dans un emplacement, il faut que ce dernier soit libre d'où la création d'une fonction qui permet de vérifier ceci avant chaque réservation d'un emplacement :

```
CREATE OR REPLACE FUNCTION inserer_sejour()

RETURNS trigger

AS $$

DECLARE date_ok boolean := false ;

BEGIN

IF (new.num_emplacement not in (select num_emplacement from sejour) )

THEN

RETURN NEW ;

END IF ;

IF ( ((new.date_fin_sejour < ALL (select date_deb_sejour from luxiours_camping.sejour where
new.num_emplacement = num_emplacement)) AND (new.date_deb_sejour <ALL(select
date_deb_sejour from luxiours_camping.sejour where new.num_emplacement =
num_emplacement)))OR((new.date_fin_sejour > ALL (select date_deb_sejour from
luxiours_camping.sejour where new.num_emplacement = num_emplacement )) AND
(new.date_deb_sejour >ALL(select date_deb_sejour from luxiours_camping.sejour where
new.num_emplacement = num_emplacement))))

THEN

RETURN new ;

ELSE

RETURN NULL ;

END IF;

END ;

$$ LANGUAGE PLPGSQL;
```

Le trigger qui déclenche la fonction inserer_sejour :

```
CREATE trigger verif_inser_sejour  
BEFORE INSERT OR UPDATE ON sejour  
FOR EACH ROW  
EXECUTE PROCEDURE inserer_sejour() ;
```

Pour que cette fonction soit exécuter a chaque réservation on a créé un trigger qui se déclenche AVANT l'insertion ou les modifications la table séjour.

Trouvez ici un exemple d'exécution :

```
postgres=# select * from sejour ;  
 num_client | num_emplacement | date_deb_sejour | date_fin_sejour  
-----+-----+-----+-----  
(0 ligne)  
  
postgres=# INSERT INTO sejour VALUES (1,2,'2020-11-10','2020-11-20');  
INSERT 0 1  
postgres=# INSERT INTO sejour VALUES (1,1,'2020-11-10','2020-11-20');  
INSERT 0 1  
postgres=# INSERT INTO sejour VALUES (2,3,'2020-11-10','2020-11-20');  
INSERT 0 1  
postgres=# INSERT INTO sejour VALUES (2,2,'2020-11-10','2020-11-20');  
INSERT 0 0  
postgres=# select * from sejour ;  
 num_client | num_emplacement | date_deb_sejour | date_fin_sejour  
-----+-----+-----+-----  
          1 |                2 | 2020-11-10      | 2020-11-20  
          1 |                1 | 2020-11-10      | 2020-11-20  
          2 |                3 | 2020-11-10      | 2020-11-20  
(3 lignes)  
  
postgres=# INSERT INTO sejour VALUES (2,2,'2020-11-10','2020-11-20');  
INSERT 0 0  
postgres=# INSERT INTO sejour VALUES (1,1,'2020-11-10','2020-11-20');  
INSERT 0 0
```

Ici on part d'une table séjour vide, on réserve des emplacements a des dates donnees, mais quand on veut réserver l'emplacement 2 a la date 10/11/2020 on a INSERT 00 car cet emplacement a été déjà réservé auparavant, pareil pour l'emplacement 1..

Table saison emplacement :

Le script pour avoir la table saison_emplacement :

```
CREATE TABLE saison_emplacement (  
num_saison INTEGER ,  
num_emplacement INTEGER ,  
PRIMARY KEY(num_saison , num_emplacement) ,  
tarif FLOAT ,
```

```
CONSTRAINT fk_num_saison FOREIGN KEY (num_saison) REFERENCES  
saison(num_saison) ,
```

```
CONSTRAINT fk_num_emplacement FOREIGN KEY ( num_emplacement ) REFERENCES  
emplacement(num_emplacement));
```

La table saison_emplacement a été créée pour calculer automatiquement le nouveau tarif selon la saison, en effet cette table contient comme clé étrangère le numéro de la saison et le numéro de l'emplacement qui forment ensemble la clé primaire de la table ainsi que le tarif qui sera calculé à partir de cette requête :

```
UPDATE saison_emplacement set tarif = (SELECT coefficient_majoration FROM saison where  
num_saison = saison_emplacement.num_saison )*(select tarif_emplacement from emplacement  
where num_emplacement = saison_emplacement.num_emplacement );
```

Table menu :

Le script pour créer la table menu

```
CREATE TABLE menu (  
num_menu INTEGER PRIMARY KEY,  
type_menu VARCHAR (50),  
contenu VARCHAR (255),  
tarif_menu INTEGER ) ;
```

Dans la table menu vous allez trouver l'identifiant de chaque menu ainsi que son type (petit déjeuner, goûte etc..) son contenu et son tarif.

Table reservation menu :

Le script pour écrire cette table est :

```
CREATE TABLE Reservation_Menu (  
num_client INTEGER ,  
num_menu INTEGER ,  
date_menu DATE,  
PRIMARY KEY(num_client,num_menu,date_menu),  
CONSTRAINT FK_num_client FOREIGN KEY (num_client) REFERENCES  
client(num_client),
```


CONSTRAINT FK_num_menu FOREIGN KEY (num_menu) REFERENCES menu(num_menu));

Dans cette table on a comme clé primaire les deux clés étrangères num_client et num_menu ainsi que la date de réservation du menu qui est de type date, c'est une table similaire à celle de reservation_menu.

Pour pouvoir trouver les réservations des suppléments par chaque client on a la requête suivante :

```
select r.num_client, c.nom, c.prenom, r.num_menu ,  
count(r.num_menu) AS nombre_menu_par_client  
from client c , reservation_menu r  
where c.num_client = r.num_client  
group by r.num_client,c.nom,c.prenom,r.num_menu ;
```

Table supplement :

Le script pour créer la table supplément :

```
CREATE TABLE supplement (  
nom_supp VARCHAR (50) PRIMARY KEY ,  
tarif_supp INTEGER) ;
```

C'est une table qui contient la liste des suppléments qu'offre le camping avec pour chaque supplément le numéro qui est la clé primaire et le tarif qui est un entier.

Table reservation supplement :

Le script pour avoir la table :

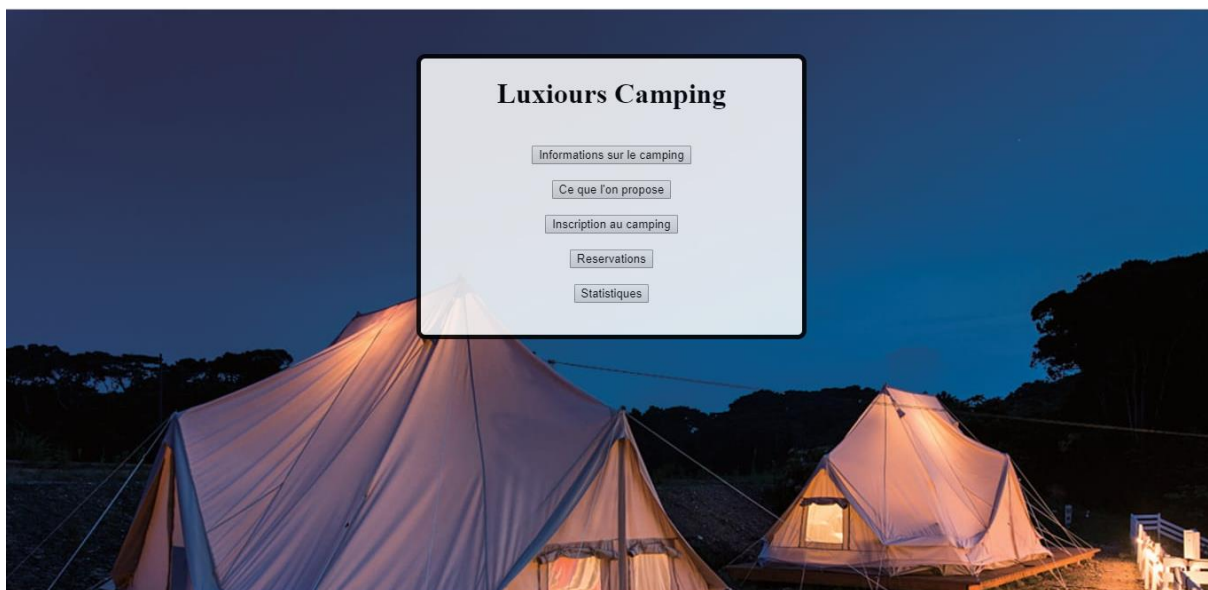
```
CREATE TABLE Reservation_supplement (  
    num_client INTEGER ,  
    num_supp INTEGER ,  
    date_supplement DATE,  
    PRIMARY KEY(num_client,num_supp,date_supplement),  
    CONSTRAINT FK_num_client FOREIGN KEY (num_client) REFERENCES  
    client(num_client),  
    CONSTRAINT FK_num_supp FOREIGN KEY (num_supp) REFERENCES  
    menu(num_supp));
```

Cette table ci-dessus permet à la clientèle de bénéficier d'un supplément qu'offre notre camping, avec comme colonne le numéro du client qui est une clé étrangère et le numéro du supplément qui l'est aussi et fortement avec la date la clé primaire. On a fait ainsi pour ne pas avoir beaucoup de client sur le même supplément.

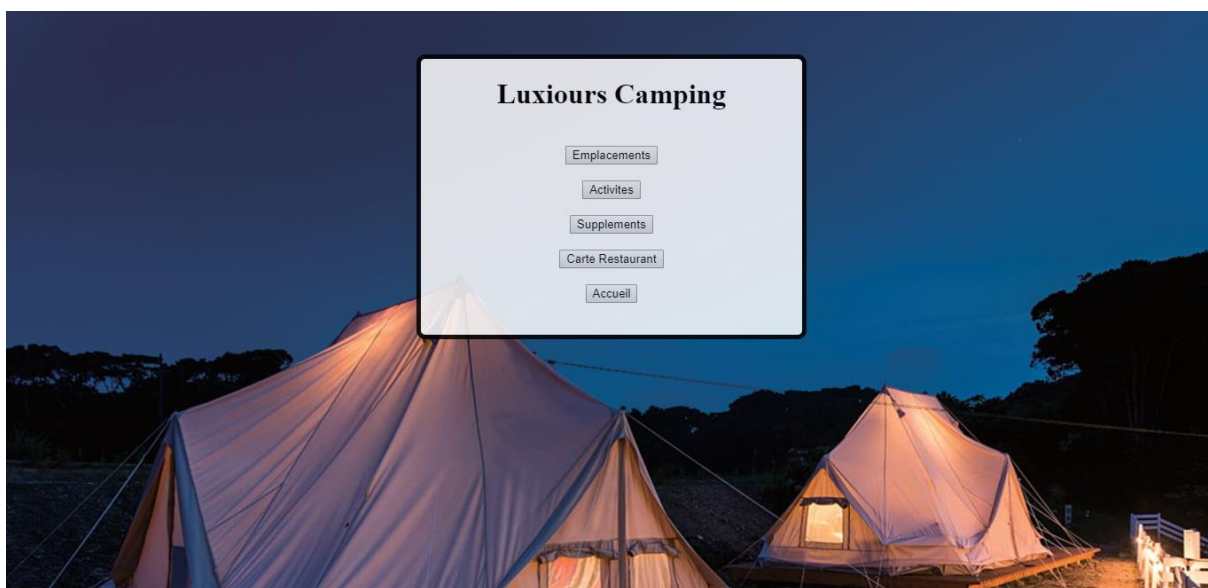
Pour pouvoir afficher pour chaque supplément le nombre de fois qu'un client a réservé on a la requête suivante :

```
select  
r.num_client,c.nom,c.prenom,r.nom_supp ,count(r.nom_supp) AS  
nombre_supplement_par_client  
from client c , reservation_supplement r  
where c.num_client = r.num_client  
group by r.num_client,c.nom,c.prenom,r.nom_supp ;
```

La partie conception du site web



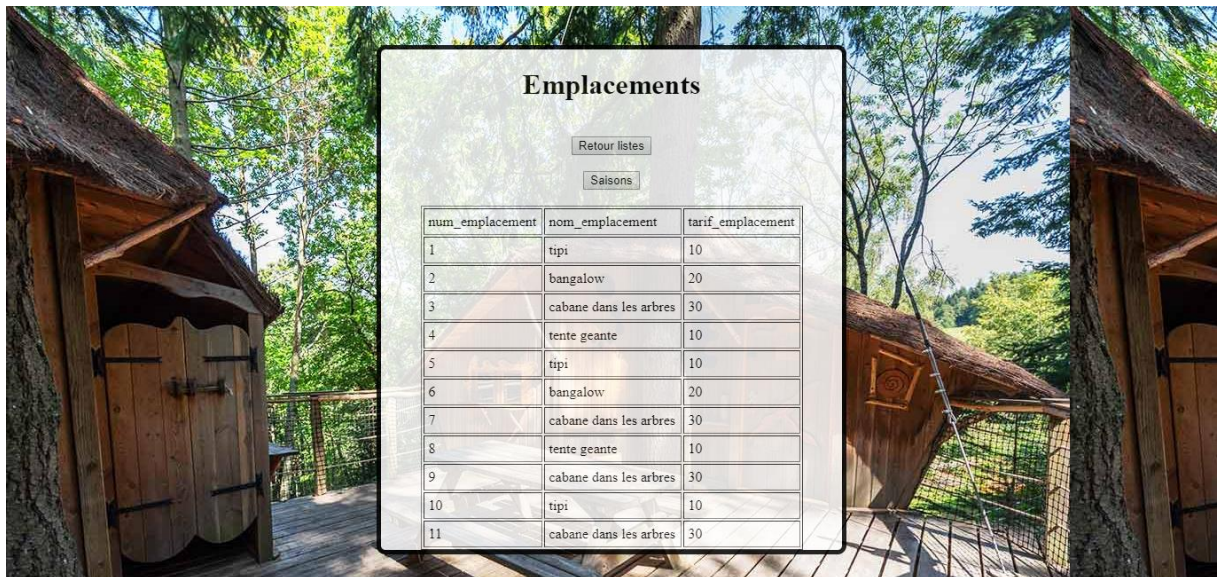
Voici la page d'accueil de notre site, elle est composée de 5 boutons envoyant vers d'autres pages, cliquons pour commencer sur « ce que l'on propose »



Nous arrivons donc sur cette page, qui nous permet de nouveau d'aller vers 4 nouvelles pages, ainsi qu'un dernier bouton permettant de revenir vers l'accueil.

Ces pages nous permettent de voir ce que le camping propose comme emplacements (tentes, mobil home, etc ...), ses activités, etc...

Allons sur emplacement :



Emplacements

[Retour listes](#)

[Saisons](#)

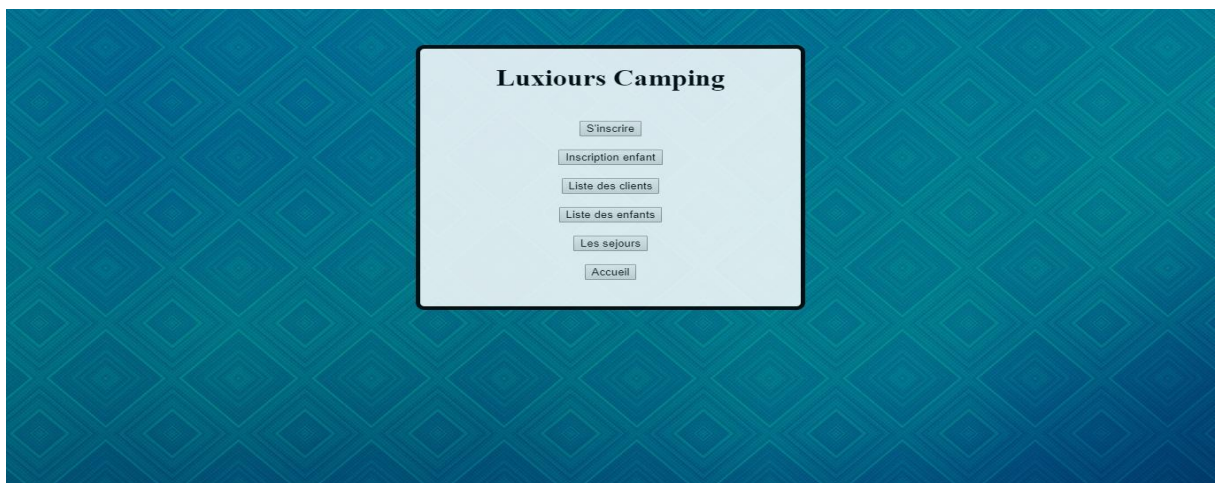
num_emplacement	nom_emplacement	tarif_emplacement
1	tipi	10
2	bangalow	20
3	cabane dans les arbres	30
4	tente geante	10
5	tipi	10
6	bangalow	20
7	cabane dans les arbres	30
8	tente geante	10
9	cabane dans les arbres	30
10	tipi	10
11	cabane dans les arbres	30

Voici donc la liste des emplacements que notre camping propose.

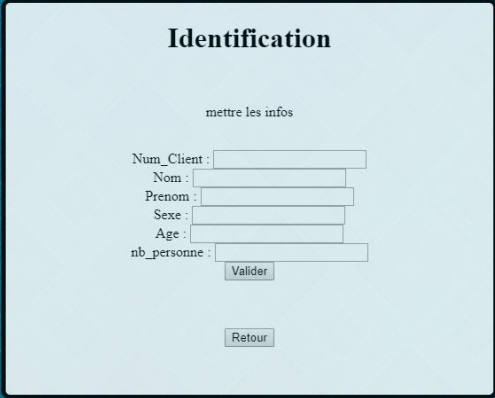
Cette page est donc reliée à notre base de données et nous affiche la liste sous forme de tableau.

Toutes les autres pages présentes dans la page « Ce que l'on propose » vu un peu plus haut nous permettent d'accéder à une page comme celle-ci reliée à la table de données adéquate (Activités montrera la liste des activités etc...)

Sur notre page d'accueil nous avons également un bouton nous redirigeant vers une page d'inscription :



Cette page permet de nous inscrire ou d'inscrire par exemple son enfant de cette manière :



The image shows a web form titled "Identification" centered on a blue background with a repeating diamond pattern. The form itself is a light blue rectangle. Inside the form, the text "mettre les infos" is centered. Below it, there are six input fields, each preceded by a label: "Num_Client :", "Nom :", "Prenom :", "Sexe :", "Age :", and "nb_personne :". The "Nom" and "Prenom" fields are stacked vertically. Below the input fields are two buttons: "Valider" and "Retour".

Cette page permet d'entrer toute nos informations afin de s'inscrire, une fois l'inscription réussie un message de confirmation s'affiche comme ceci :



The image shows a confirmation message box titled "Enregistrement valide !" centered on the same blue background with a repeating diamond pattern. The message box is a light blue rectangle. Inside, the text "votre inscription a bien fonctionne" is centered. Below the text is a button labeled "Liste des clients".

Cette page nous permet d'accéder directement à la liste des clients inscrits :

Liste des clients

[Retour](#)

num_client	nom	prenom	sexe	age	nb_personne
1	Boulland	Nicolas	M	20	1

Voici la page listant les personnes inscrites, celle-ci s'actualise à chaque inscription.

Pour les inscriptions enfants le format est le même, il faut en revanche avoir un adulte préalablement inscrit, car pour inscrire un enfant il faut pouvoir entrer le numéro de client du « parent » de celui-ci.

Inscription enfant

mettre les infos

Num_enfant :

Nom_enfant :

Prenom_enfant :

Sexe :

Age :

duree_sejour_enfant :

Num_client :

[Valider](#)

[Retour](#)

Voici donc le formulaire permettant d'inscrire son enfant, une fois inscrit voici la liste des enfants inscrits :

Liste des enfants							
Retour							
num_enfant	nom_enfant	prenom_enfant	sexe	age	duree_sejour_enfant	tarif_sejour_enfant	num_client
1	Boulland	Pierre	M	10	4	160	1

L'enfant est donc bien inscrit avec comme « parent » le numéro de client 1, donc « Nicolas Boulland »

De plus, on peut voir que le tarif est automatiquement calculé lorsqu'on a mis le nombre de jour durant lesquels l'enfant sera présent

Nous avons également mis en place une liste de statistiques, des tableaux montrant par exemple toute les personnes ayant réservé une activité :



nombre des activites par client				
Retour				
num_client	nom	prenom	nombre d activite par client	nom_activite
1	Boulland	Nicolas	1	paintball
1	Boulland	Nicolas	2	paddle

Cette page rassemble donc toutes les personnes ayant réservé une activité, le tableau et calcule le nombre de fois qu'un client a participé à une activité donnée.

Liaison entre site et sql avec python :

L'index :

Lors de l'ouverture du site, nous allons nous retrouver sur la page Index.py, qui est la page principale du site. Cette page ne va servir qu'à nous renvoyer sur un tas d'autres pages, qui elles, seront reliés à des bases de données et autre.

La page index est donc comme vu ci-dessus, composée de boutons, renvoyant vers d'autres pages.

Cette page est donc constituée uniquement de HTML, et pour faire du HTML sur python, il faut créer une variable dans laquelle tout sera en HTML et que nous « activerons » avec un print.

```
Page = """
<!doctype html>
```



```









    <html lang="fr">
    .
    .
    </html>"""
print(Page)

```

Pour un minimum de présentation, les pages ont tous un fond, avec un autre fond blanc plus petit dans lequel toute les informations importantes sont situés, et le tout est centré sur la page.

Passons maintenant au plus important :

Chaque bouton présent sert a nous renvoyer vers une nouvelle page. Toute les pages nécessaires au site sont au préalable créées dans notre dossier camping en python.

 env	01/12/2020 16:04	Dossier de fichiers	
 images	04/12/2020 15:46	Dossier de fichiers	
 activite.py	04/12/2020 15:18	Python File	1 Ko
 emplacements.py	04/12/2020 15:29	Python File	1 Ko
 identification.py	04/12/2020 15:46	Python File	2 Ko
 index.py	04/12/2020 15:42	Python File	2 Ko
 informations.py	04/12/2020 15:15	Python File	1 Ko
 server.py	01/12/2020 16:18	Python File	1 Ko

En HTML, les boutons de redirection se font de cette manière :

```

<form action="page.py" method="">
    <input type="submit" value="Nom page" />

```

Il faudra ensuite remplacer le « page.py » par le nom exact de la page vers laquelle on souhaite être rediriger, ainsi que « Nom page » par le nom qu'on souhaite voir être écrit sur le bouton.

Par exemple

```

<form action="emplacement.py" method="">

```

```
<input type="submit" value="Liste des  
emplacements" />
```

servira a nous renvoyer vers la page Liste des emplacements que nous avons vu un peu plus haut.

Liste des emplacements :

Cette page ne sert qu'à afficher la liste des différents emplacements sur le camping. Elle est donc liée a la base de données.

Dans un premier temps il nous faut donc, sur python, nous relier à la base de donnée de cette manière :

```
5  hostname = 'localhost'  
6  username = 'postgres'  
7  password = 'password'  
8  database = 'postgres'
```

Les variables pour les identifiants

```
Requete = "SELECT num_emplacement,nom_emplacement,tarif_emplacement FROM emplacement"
```

La requête SQL

```
conn = psycopg2.connect( host=hostname, user=username, password=password, dbname=database )  
cur = conn.cursor()  
cur.execute( Requete )
```

La connexion.

Nous avons décidé de les faire apparaître dans un tableau que nous créons de cette manière :

```

28         <table border="1" cellpadding="5">
29             <tr>
30                 <td>num_emplacement</td>
31                 <td>nom_emplacement</td>
32                 <td>tarif_emplacement</td>
33             </tr>""")
34
35     for num_emplacement,nom_emplacement,tarif_emplacement in cur.fetchall() :
36         print(        ""<tr>
37             <td>"" , num_emplacement , ""</td>
38             <td>"" , nom_emplacement , ""</td>
39             <td>"" , tarif_emplacement | , ""</td>
40             </tr>""")
41     print( ""</table>
42

```

la ligne 38 sert à chercher les données de la table SQL dans
« cur.fetchall() »

lorsque nous exécutons la requete, toute les données des tables SQL sont envoyés dans cur.fetchall() qui regroupe tout un tas d'informations, avec le for, on extrait donc uniquement les données que l'on veut, puis on les affiche dans notre tableau.