CAUCHEPIN Yann
GRASMICK Morgane
GIS 3

Tuteur : Mr. Flissi

Projet 2018:

Base de Données

Système d'information d'une compagnie aérienne

Sommaire :

Introduction

Nous sommes étudiants en formation Génie Informatique et Statistique de
Polytech Lille. Nous devons réaliser un projet de base de données qui porte sur la gestion d'une compagnie aérienne.

Problématique

Une compagnie aérienne souhaite mettre en place un outil pour la gestion d'une flottille aérienne d'un aéroport, des vols et de l'équipage, des passagers et de la billetterie et enfin de la maintenance des avions.

L'objectif de notre projet est de créer un site internet afin de gérer les vols de la compagnie, la liste des passagers ainsi que les membres de l'équipage. L'application doit donc permettre d'organiser un nouveau vol et gérer la réservation des billets sur ce vol.
Nous devons également prendre en compte les maintenances et le personnel de maintenance qui s'assure de la fonctionnalité de l'avion. Le site doit également avoir accès à l'historique de tous les entretiens réalisés sur les avions et permettre d'ajouter les commentaires d'un nouvel entretien.

Analyse du sujet

La compagnie aérienne possède plusieurs avions.

Pour chaque vol, il est indispensable de connaître les informations de la ligne telles que la ville, les dates les heures de départs et d'arrivés ainsi que l'avion qui parcours la ligne de vol. De même, il est nécessaire de suivre les informations liées aux avions comme la taille de l'avion, c'est à dire son nombre de sièges. Ce sont des données essentielles pour créer des vols.

Il faut aussi que la compagnie puisse supprimer un vol en cas de problème et que cela supprime ainsi les billets correspondants à ce vol.

Ensuite, il faut s'assurer que l'avion qu'on sélectionne pour un vol ne soit pas déjà choisi pour un autre vol au même moment.

De plus, il est important de gérer l'entretien régulier des avions. Il faut pouvoir obtenir l'historique de la maintenance, c'est à dire à quelle date elle a eu lieu et quelles actions ont été réalisées. Les maintenances sont effectuées par les employés de la compagnie et un employé est désigné responsable de la maintenance. Il faut donc prévoir d'ajouter une maintenance pour un avion précis et aussi d'ajouter l'équipe de maintenance correspondante. Les membres de la compagnie pouvant être désignés dans l'équipe de maintenance correspondent aux mécaniciens. Il faut définir un délai de maintenance pour chaque avion, par exemple, organiser une maintenance tous les trois mois.

Pour la réservation, il est important que le client s'identifie avec son numéro unique. Il existe deux possibilités : la personne est déjà cliente ou la personne effectue une réservation pour la première fois. Cela devra être pris en compte. Une fois que le client a un numéro de client, il pourra réserver un vol. A noter que si le vol est complet, plus aucune réservation ne pourra être prise en compte. C'est à dire que les vols complets ne seront pas affichés lorsque le client fait sa recherche.

Nous ne prenons pas en compte la suppression des billets si un client ne veut plus prendre le vol qu'il aura préalablement réservé.

Les clients doivent avoir un billet avec la date de réservation et un numéro de place. Cette réservation sera enregistré par un employé d'accueil ou directement sur l'application.

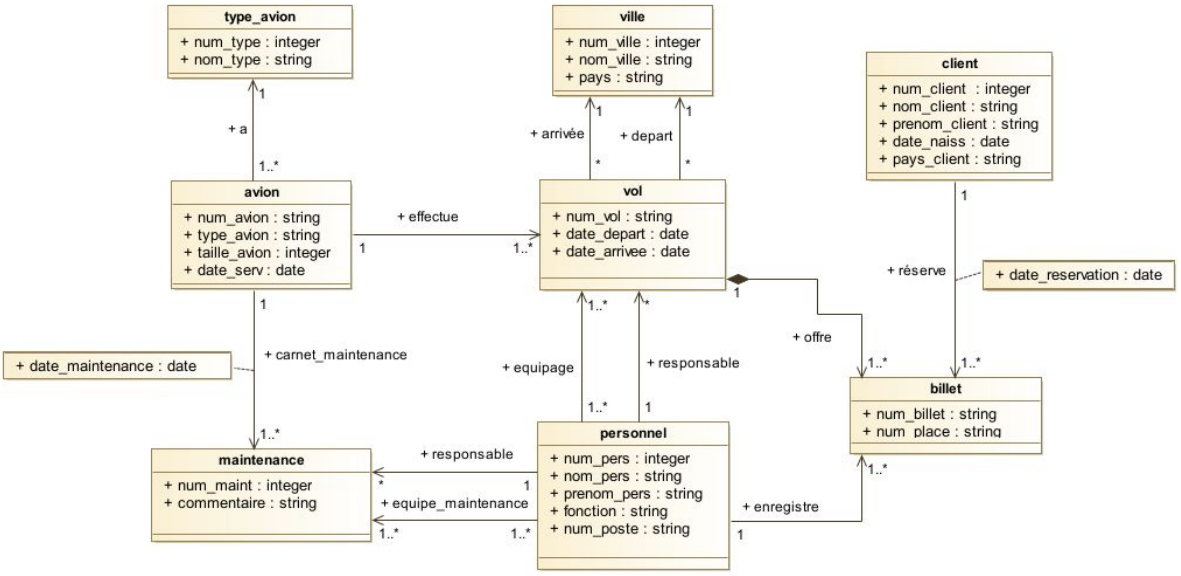
La compagnie a ses propres employés pour constituer l'équipage. Un vol nécessite plusieurs membres du personnel tel qu'un pilote, un copilote, des hôtesses et des stewards. Un employé sera désigné responsable de l'équipage pour un vol en particulier.

Concernant le personnel, il faut prévoir d'ajouter un membre, de pouvoir modifier ses informations ou encore de le supprimer de la liste du personnel.

Par manque de temps, l'application ne prendra toutefois pas en charge certains aspects du projet :

- les problèmes concernant les retards et les annulations des vols.
- si un employé de la compagnie veut effectuer une réservation, il sera alors considéré comme un client standard : il peut donc y avoir des doublons entre les membres du personnel et les clients.
- la mise en place des maintenances régulières des avions.
- les clients ne pourront pas choisir leur place parmi toutes celles disponibles.
- il n'y a pas de système de connection du client avec un mot de passe.
- un unique client n'est pas censé pouvoir réserver plusieurs sièges d'un même vol à son nom en même temps : pour cela il devra faire plusieurs fois la réservation.
- les différentes fonctions du personnel sont prédéfinies.
- les vols, les avions et et les réservations ne pourront pas être supprimés ou modifiés.

Modèle conceptuel



Nous devons connaître quel avion assure chaque vol. Nous avons donc créé la classe **avion**, qui comporte toutes les informations relatives à un avion telles que son numéro unique (num_avion) de type serial qui correspond à un integer. Ce type permet de d'attribuer un entier, qui s'incrémente automatiquement lors de l'ajout d'un nouvel avion dans la base données. Ensuite un avion est aussi décrit par le nombre de sièges à bord (taille_avion) et sa date de mise en service (date_serv) de type date.

Nous avons créé la table **type_avion** afin d'y répertorier les informations concernant l'avion : num_avion, de type serial et son nom de type varchar (ex:A380).

Relation avion-type_avion : un avion a seulement un type et la compagnie peut avoir plusieurs avions du même type.

Aussi, nous avons créé la table **vol** car nous devons stocker les informations du vol. Le numéro unique du vol (num_vol) est de type serial pour faciliter l'incrémentation des vols. Les autres informations figurant dans la table sont la date de départ/d'arrivée. A noter que les dates sont de type **timestamp**, type qui permet à la fois d'avoir la date et l'heure (format : « AAAA/MM/JJ HH:mm:ss »)

Relation avion-vol : un avion peut effectuer plusieurs vols mais un vol est réalisé par un seul avion.

Une table **ville** a été créé pour faire correspondre une ville à un numéro unique afin d'éviter les doublons notamment si une ville a le même nom dans deux pays différents.

Relation ville-vol : un vol décolle d'une ville et une ville peut être la ville de départ de plusieurs vols. Un vol atterrit dans une seule ville et une ville peut être la ville d'arrivée de plusieurs vols.

De plus, un vol est caractérisé par son équipage et ses passagers. Concernant l'équipage, il s'agit des employés de la compagnie aérienne dont la fonction est soit pilote, co-pilote, hôtesse ou steward.

Nous avons donc créé une classe **personnel** qui fait référence à tous les employés de la compagnie aérienne et qui comporte les attributs suivants : numéro unique de l'employé (num_pers), son nom, son prénom, sa fonction et son numéro de poste, c'est-à-dire son numéro de téléphone qui est de type varchar(10) pour afficher le premier '0'.

Relation vol-personnel : plusieurs membres de la compagnie assurent un vol, et un vol est assuré par plusieurs employés. Un vol a pour responsable un seul employé de l'équipage mais un employé de l'équipage peut être responsable plusieurs fois d'un vol. Toutefois, un employé n'appartenant pas à l'équipage ne peut pas être responsable d'un vol.

Ensuite, nous avons créé une classe **client** pour stocker les informations relatives aux clients. Dans cette table figure le numéro unique d'un client (num_client) ainsi que son nom, prénom, date de naissance et son pays de naissance.

Nous avons également créé une table **billet**, qui correspond au billet de réservation. Ce billet a un numéro unique (num_billet) de type serial et un numéro de place, de type integer. Nous n'avons pas géré les numéros comme dans un vrai vol, à savoir "13A" par exemple.

Relation client-billet : un client peut réserver plusieurs billets mais un billet ne correspond qu'à un seul client.

Relation personnel-billet : un employé peut enregistrer plusieurs billets, mais un billet n'est enregistré que par un employé.

Relation vol-billet : un vol offre plusieurs billets car il a plusieurs places disponibles, mais un billet ne correspond qu'à un vol en particulier. Le billet ne peut exister sans le vol car il dépend de celui-ci, nous avons donc rajouté une composition entre vol et billet.

Nous nous intéressons à présent à la gestion de la maintenance :

Nous avons créé la classe **maintenance** caractérisée par un numéro unique (num_maint) et les commentaires qui correspondent aux actions d'entretiens réalisées durant cette maintenance.

Relation avion-maintenance : un avion a plusieurs entretiens prévus mais une maintenance ne correspond qu'à un seul avion.

La maintenance est effectuée par des employés de la compagnie :

Relation personnel-maintenance : une maintenance est réalisée par plusieurs employés et un employé peut intervenir sur plusieurs maintenances.
Aussi, une maintenance est effectué avec un seul responsable et un employé peut être responsable de plusieurs maintenances. Toutefois, un employé n'appartenant pas à l'équipe de maintenance ne peut être responsable d'une maintenance.

Schéma relationnel

- type_avion (num_type, nom_type)

On considère num_type comme clé primaire de la classe **type_avion** car c'est un numéro unique. On y ajoute le nom du type de l'avion.

- avion (num_avion, taille_avion, date_serv, #num_type)

On considère num_avion comme clé primaire car c'est un numéro unique qui permet d'identifier un avion précis de la compagnie.

Comme l'association "a" entre **avion** et **type_avion** est de cardinalités * → 1, on ajoute num_type en clé étrangère à la classe avion.

- maintenance (num_maint, commentaire, date_maintenance, #num_pers, #num_avion)

Pour la classe maintenance, on a choisi de définir un numéro unique num_maint. Ensuite, comme l'association "responsable" entre **personnel** et **maintenance** est de cardinalités 1 → *, on ajoute num_pers comme clé étrangère à la classe qui a pour cardinalité * : il s'agit ici de la classe maintenance. Ce numéro de personnel correspond à l'employé qui sera l'unique responsable de la maintenance.

De même, dans l'association "carnet_maintenance" entre **avion** et **maintenance** est de cardinalités 1 → 1..*, on ajoute num_avion comme clé étrangère à **maintenance**. On y ajoute aussi l'attribut date_maintenance qui figure sur l'association "carnet_maintenance".

- equipe_maintenance (#num_pers, #num_maint)

Il existe une deuxième association entre les classes **personnel** et **maintenance**, de cardinalités 1..* → 1..* donc cela génère la création d'une nouvelle classe nommée **equipe_maintenance**. La clé primaire de cette classe correspond aux clés primaires des deux classes concernées : num_pers et num_maint. Cette classe permettra de connaître l'ensemble des employés qui ont travaillé sur cette maintenance.

- personnel (num_pers, nom_pers, prenom_pers, fonction, num_poste)

On considère num_pers comme clé primaire de la classe **personnel** car c'est un numéro unique. On y ajoute les attributs de la classe.

- ville (num_ville, nom_ville, pays)

On considère num_ville comme clé primaire de la classe **ville** car c'est un numéro unique. On y ajoute le nom de la ville.

- vol (num_vol, date_depart, date_arrivee, #num_avion, #responsable, #ville_depart, #ville_arrivee)

Pour la classe **vol**, on a choisi de définir un numéro unique num_vol.
De plus, l'association "effectue" entre **avion** et **vol** est de cardinalités 1 → 1..*, on ajoute num_avion comme clé étrangère à la classe **vol**. Ce numéro d'avion correspond à l'appareil qui sera utilisé pour ce vol.

L'association "départ" entre **vol** et **ville** est de cardinalités * → 1, alors on ajoute num_ville dans la classe vol comme clé étrangère sous le nom de ville_depart.
L'association "arrivée" entre **vol** et **ville** est de cardinalités * → 1, alors on ajoute num_ville dans la classe vol comme clé étrangère sous le nom de ville_arrivee.

Enfin, l'association "responsable" entre **personnel** et **vol** est de cardinalités 1 → *, on ajoute num_pers comme clé étrangère à la classe **vol** pour connaître quel employé est responsable de ce vol. Ici, on a la cardinalité * pour personnel car tous les employés de la compagnie ne peuvent pas être responsable d'un vol, c'est à dire que c'est de cardinalité 0..*, ce qui équivaut à écrire *.

- equipage (#num_vol, #num_pers)

Il existe une deuxième association entre les classes **personnel** et **vol**, de cardinalités 1..* → 1..*. Cela génère la création d'une nouvelle classe nommée **équipage**. La clé primaire de cette classe correspond aux clés primaires des deux classes concernées : num_vol et num_pers.

- billet (#num_vol, num_billet, num_place, date_réservation, #num_pers, #num_client)

Pour la classe **billet**, on a choisi de définir un numéro unique num_billet.
L'association "offre" entre **billet** et **vol** est une composition donc on ajoute num_vol à la clé primaire de la classe **billet**.

L'association "enregistre" entre **personnel** et **billet** est de cardinalités 1 → 0..*, on ajoute num_pers comme clé étrangère à la classe **billet**. Enfin, l'association "réserve" entre **client** et **billet** est de cardinalités 1 → 1..*, on ajoute donc num_client comme clé étrangère

à la classe **billet**. Il existe un attribut date_reservation entre billet et client. Comme l'association est de cardinalités 1 → 1..*, cet attribut est également ajouté à la table **billet**.

- client (num_client, nom_client, prenom_client, date_naiss, pays_client)

On considère num_client comme clé primaire de la classe **client** car c'est un numéro unique. On y ajoute les attributs de la classe.

Création des tables

Nous avons tenu compte des **clés primaires** des tables qui seront donc les attributs présents entre les parenthèses de PRIMARY KEY.

Ensuite, concernant les **clés étrangères**, nous utilisons l'ajout d'une contrainte : références *nom_table_origine (nom_attribut)*.

De plus, pour les clés étrangères importantes, nous utilisons 'on delete **cascade'** qui supprime toutes les lignes qui contiennent la référence de la valeur supprimée.

Ensuite pour les clés étrangères moins importantes, nous utilisons 'on delete **set null**' qui permet de mettre NULL aux valeurs dont la référence est supprimée.

```
--création des tables

CREATE TABLE type_avion (
      num_type serial NOT NULL,
      nom_type varchar(10) NOT NULL,
      PRIMARY KEY(num_type)
      );


CREATE TABLE avion (
  num_avion serial NOT NULL, -- le serial permet de numéroter automatiquement
  n_type integer REFERENCES type_avion(num_type) on delete SET NULL,
  taille_avion integer NOT NULL,
  date_serv date,
  PRIMARY KEY (num_avion)
);

CREATE TABLE personnel (
  num_pers serial NOT NULL,
  nom_pers varchar(35) NOT NULL,
  prenom_pers varchar(35) NOT NULL,
  fonction varchar(35) NOT NULL,
  num_poste varchar(10),
  PRIMARY KEY (num_pers)
);

CREATE TABLE maintenance (
  num_maint serial NOT NULL,
  n_avion integer NOT NULL REFERENCES avion(num_avion)on delete cascade,
  date_maint date NOT NULL,
  commentaire varchar(200),
  n_pers integer REFERENCES personnel(num_pers) on delete SET NULL,
  PRIMARY KEY (num_maint)
);


CREATE TABLE equipe_maintenance (
  n_pers integer NOT NULL REFERENCES personnel(num_pers) on delete cascade,
  n_maint integer NOT NULL REFERENCES maintenance(num_maint) on delete cascade,
  PRIMARY KEY (n_pers,n_maint)
);


CREATE TABLE ville(
  num_ville serial NOT NULL,
  nom_ville varchar(35) NOT NULL,
  PRIMARY KEY (num_ville)
);
```

```sql
CREATE TABLE vol (
  num_vol serial NOT NULL,
  date_depart timestamp NOT NULL,
  CHECK (date_depart>NOW()),
  date_arrivee timestamp NOT NULL,
  CHECK (date_depart>NOW()),
  responsable integer REFERENCES personnel(num_pers) on delete SET NULL,
  n_avion integer NOT NULL REFERENCES avion(num_avion)on delete cascade,
  n_ville_depart integer NOT NULL REFERENCES ville(num_ville)on delete cascade,
  n_ville_arrivee integer NOT NULL REFERENCES ville(num_ville)on delete cascade,
  PRIMARY KEY (num_vol)
);


CREATE TABLE equipage (
  n_pers integer NOT NULL REFERENCES personnel(num_pers) on delete cascade,
  n_vol integer NOT NULL REFERENCES vol(num_vol) on delete cascade,
  PRIMARY KEY (n_pers,n_vol)
);


CREATE TABLE client (
  num_client serial NOT NULL,
  nom_client varchar(35) NOT NULL,
  prenom_client varchar(35),
  date_naiss date,
  pays_client varchar(35),
  PRIMARY KEY (num_client)
);


CREATE TABLE billet (
  num_billet serial NOT NULL,
  num_place integer NOT NULL,
  date_reservation date,
  n_client integer NOT NULL REFERENCES client(num_client)on delete cascade,
  n_pers integer REFERENCES personnel(num_pers) on delete SET NULL,
  n_vol integer NOT NULL REFERENCES vol(num_vol) on delete cascade,
  PRIMARY KEY (num_billet,n_vol)
);
```

Application

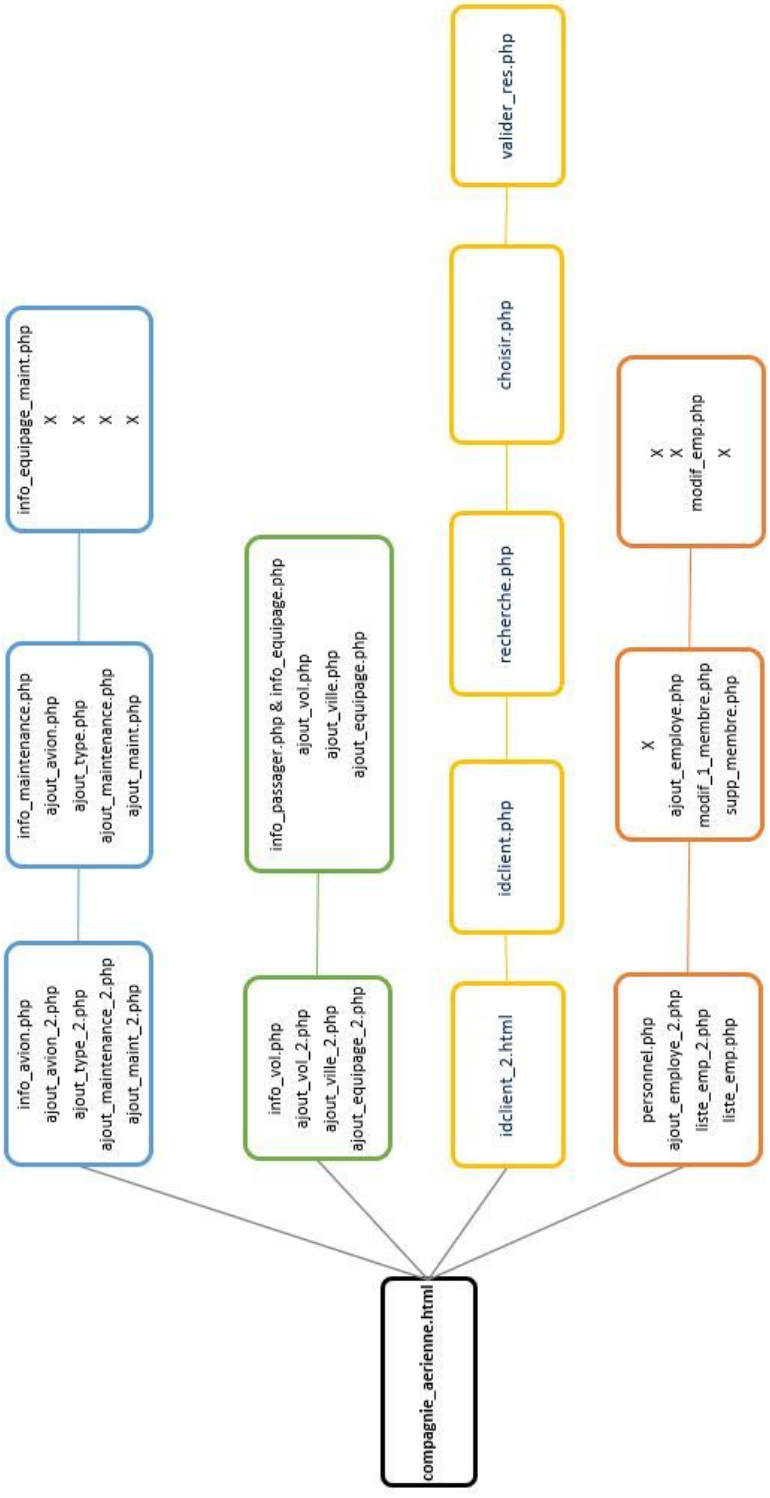
ADRESSE DU SITE :

http://houplin.studserv.deule.net/~mgrasmic/compagnie_aerienne.html

Le site peut être utilisé par les membres du personnel de la compagnie aérienne et par les clients qui souhaitent réserver un vol auprès de la compagnie.

La page compagnie aérienne est organisée en un tableau de 4 colonnes : avions, vols, réservation et personnel. Dans chacune de ces colonnes, il est possible de cliquer sur des liens pour obtenir les informations qui nous intéresses ou pour effectuer une action.

Organisation des fichiers



Concernant la flotille d’avions

Liste des avions

La liste des avions

Num	Type d avion	Date mise en service	Taille	maintenance
1	A380	2015-12-08	200	Consulter les maintenances
2	A380	2005-12-08	200	Consulter les maintenances
3	A220	2005-12-08	180	Consulter les maintenances
4	A220	2018-12-28	250	Consulter les maintenances

Ce lien nous permet d’afficher la flottille des avions de la compagnie sous forme de tableau. On identifie chaque appareil par son numéro, son type d’avion, sa date de mise en service et sa taille.
Il est alors possible d’afficher la liste des maintenances pour chacun des appareils en cliquant sur un second lien, effectuant une requête SQL. Le résultat sera alors transmis sous une forme de tableau avec le numéro et la date de maintenance, le nom du responsable ainsi que le commentaire de l’opération.

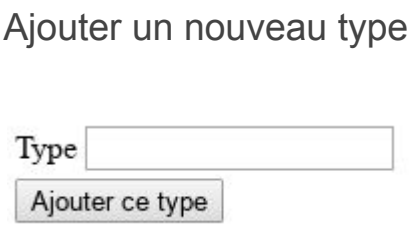
Dans le tableau des maintenances pour un avion particulier, la dernière colonne contient un lien pour chaque maintenance. Ce lien correspond à la liste du personnel qui a travaillé sur la maintenance sélectionnée.

Ajouter un avion

Type
Taille
Date service jj/mm/aaaa
Ajouter cet avion

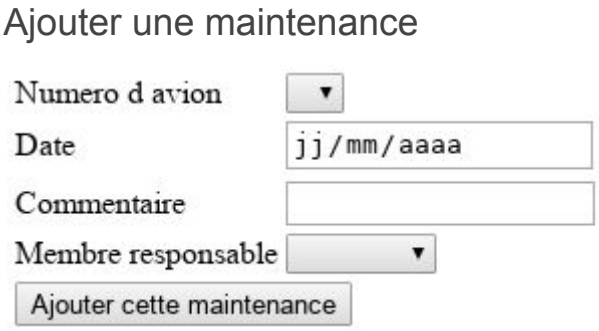
Ce lien nous permet d’afficher un formulaire pour ajouter un appareil à la flotille d’avions. On doit donc renseigner le type de l’appareil parmi une liste déroulante basé sur le résultat d’une requête SQL. Le numéro de l’avion sera automatiquement ajouté de manière à poursuivre la suite logique des numéros de la base.
Rmq : Pour ajouter un avion d’un nouveau type, il nécessaire de rentrer préalablement le nouveau type dans la base.

Ensuite, il est demandé de renseigner la taille l'avion, c'est-à-dire le nombre de sièges et sa date de mise en service à partir d'un calendrier.



Ce lien nous permet d'afficher un formulaire pour ajouter un type d'avion. On doit donc renseigner le nom du type d'avion. Le numéro du type de l'avion sera automatiquement ajouté de manière à poursuivre la suite logique des numéros de la base.

Ce lien nous permet d'ajouter un employé à une maintenance. On doit donc renseigner le numéro de maintenance et le nom de l'employé parmi deux listes déroulantes basées sur les résultats de requêtes SQL.
Rmq : Pour attribuer un employé à une maintenance, il est nécessaire de rentrer préalablement la maintenance et l'employé dans la base de donnée.
Rmq : Seuls les employés attribués aux maintenances (par exemples les mécaniciens) sont disponibles pour être ajouter à une maintenance.



Ce lien nous permet d'afficher un formulaire pour ajouter une maintenance. On doit donc renseigner le numéro de l'avion et le nom du responsable parmi deux listes déroulantes, basées sur les résultats de requêtes SQL. Il est également demandé de renseigner la date et l'énoncé de la maintenance. Le numéro de la maintenance sera automatiquement ajouté de manière à poursuivre la suite logique des numéros de la base.

Rmq : Seuls les employés attribués aux maintenances sont disponibles pour être responsable d'une maintenance.

Concernant les vols

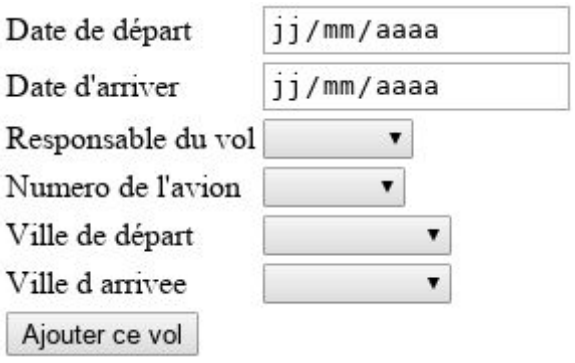
Liste des vols

La liste des vols

Numero Vol	Ville Depart	Ville d'arrivee	Date Depart	Date Arrivee	Numero Avion	Responsable	Equipage	Passagers
1	Prague	Lille	2019-05-11 12:50:00	2019-05-11 15:15:00	1	JOSTE	Consulter equipage	Consulter la liste
2	Seville	Lille	2019-11-11 08:50:00	2019-05-11 10:20:00	1	LALA	Consulter equipage	Consulter la liste
3	Prague	Toulouse	2019-12-06 11:50:00	2019-12-06 15:15:00	1	LABOIS	Consulter equipage	Consulter la liste
4	Toulouse	Montreal	2019-02-11 05:50:00	2019-02-11 15:45:00	2	JOSTE	Consulter equipage	Consulter la liste
5	Montreal	Lille	2018-12-22 00:00:00	2018-12-29 00:00:00	1	BOISSON	Consulter equipage	Consulter la liste

Ce lien nous permet d'afficher la liste des vols sous forme de tableau. On identifie chaque vol par son numéro, sa ville de départ et celle d'arrivée, sa date de départ et celle d'arrivée ainsi que le numéro de l'avion attribué au vol et le responsable du vol.
Il est alors possible d'afficher la liste de l'équipage et des passagers pour chaque vol en cliquant sur deux liens distincts, effectuant chacun une requête SQL.

Ajouter un vol



Ce lien nous permet d'afficher un formulaire pour ajouter un vol. On doit donc renseigner le responsable du vol, le numéro de l'avion, la ville de départ et l'arrivée, basés sur les résultats de requêtes SQL. Le numéro du vol sera automatiquement ajouté de manière à poursuivre la suite logique des numéros de la base.
Il est également demandé de renseigner les dates de départ et d'arrivée du vol.

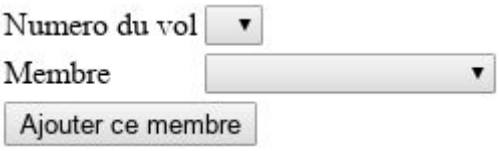
Rmq : Si la ville n'est pas proposée dans la liste déroulante, cela signifie qu'elle n'existe pas au préalable dans la base. Il faut alors l'ajouter dans la table ville.

Rmq : Seuls les employés attribués aux vols (par exemple pilote ou hotesse) sont disponibles pour être responsable d'un vol.

Ajouter une ville

Ce lien nous permet d'afficher un formulaire pour ajouter une ville. On doit donc renseigner le nom de la ville. Le numéro de la ville sera automatiquement ajouté de manière à poursuivre la suite logique des numéros de la base.

Ajouter un membre dans l'équipage



Ce lien nous permet d'ajouter un employé à un vol. On doit donc renseigner le numéro du vol et le nom de l'employé parmi deux listes déroulantes basées sur les résultats de requêtes SQL.
Rmq : Pour attribuer un employé à un vol, il est nécessaire de rentrer préalablement le vol et l'employé dans la base de donnée.
Rmq : Seuls les employés attribués aux vols sont disponibles pour être attribués à un vol.

Concernant les réservations

Effectuer une réservation

L'identification

Cette étape correspond à un logIN.
Le client doit saisir ses informations personnelles. Il doit donc remplir les champs du formulaire.

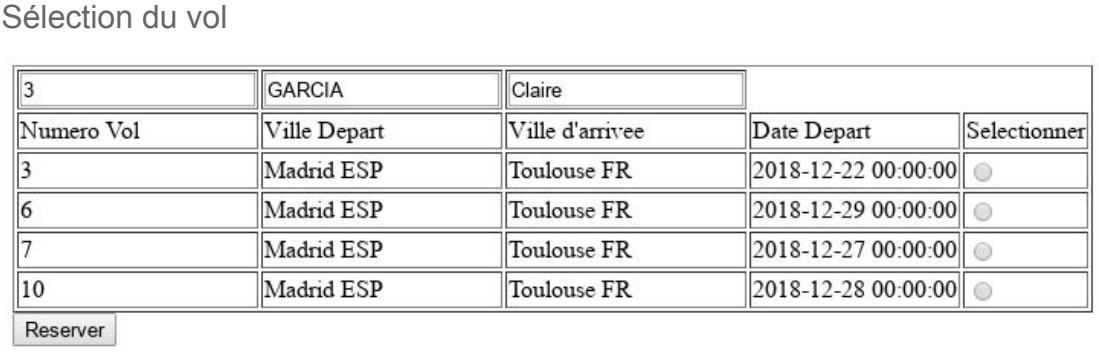
Une fois identifié, la page suivante est affichée au client :

Il s'agit ici de faire valider par le client qu'il s'agit bien de lui.

Une fois que le client a validé son identification, il choisit sa ville de départ et sa ville d'arrivée parmi une liste déroulante de villes, basées sur les résultats de requêtes SQL. Ce qui signifie que la liste déroulante n'affiche que les villes qui sont déjà dans la base de données de la compagnie. Et plus précisément : les villes de départs correspondent seulement aux villes qui existent dans la colonne ville de départ de l'ensemble des vols. Il en est de même pour les villes d'arrivées.

Ensuite, le client doit saisir une date dans le calendrier, vérifiant la condition que celle-ci doit être ultérieure à la date actuelle.

L'application effectue alors une requête SQL complexe (voir dans la partie "Difficultés rencontrées") qui cherche les vols libres. C'est-à-dire que le résultat de la requête affichera l'ensemble des vols pour lesquels les avions ont encore des sièges non réservés.



Ce lien nous permet d'afficher la liste des vols correspondant à la recherche du client. On identifie chaque vol par son numéro, sa ville de départ et celle d'arrivée ainsi que la date précise du vol.

Il est alors possible de sélectionner un seul et unique vol parmi la liste. En cliquant sur le bouton réserver, le billet est automatique réservé.

À noter que nous n'avons pas géré la manière dont le client va récupérer son billet pour ensuite prendre son vol.

Concernant le personnel

Liste du personnel

Numero	Nom	Prenom	Fonction	Numero de poste
1	SACHA	Mathieu	Pilote	0665889201
2	JOSTE	Anna	Hotesse	0621457895
3	FIRMIN	Thomas	Stewart	0645783225
4	LABOIS	Michel	Pilote	0625458798
5	BOISSON	Benjamin	Co-pilote	0656892154
6	FONTEY	Mathieu	Mecanicien	0665889211
7	JOSTE	Tom	Mecanicien	0621457898
8	MONTO	Charlotte	Pilote	0641783225

Ce lien nous permet d'afficher la liste du personnel sous forme de tableau. On identifie chaque personne par son numéro, son nom et son prénom, sa fonction et son numéro de poste. Il est considéré que les fonctions des membres de la compagnie sont restreints à : Pilote, Co-Pilote, Hôtesse, Steward et Mécanicien.

Ajouter un membre

Ce lien nous permet d'afficher un formulaire pour ajouter un employé à la compagnie. On doit donc renseigner son nom, son prénom et son numéro de poste. Il est également demandé de renseigner sa fonction parmi une liste déroulante basée sur le résultat d'une requête SQL. Le numéro du personnel sera automatiquement ajouté de manière à poursuivre la suite logique des numéros de la base.

Modifier un employé

Numero	Nom	Prenom	Fonction	Modifier
3	FIRMIN	Thomas	Stewart	◉
4	LABOIS	Michel	Pilote	○
5	BOISSON	Benjamin	Co-pilote	○
7	JOSTE	Tom	Mecanicien	○
8	MONTO	Charlotte	Pilote	○
1	LALA	LILI	Pilote	○
2	JOSTE	EDDY	Mecanicien	○

Modifier

Ce lien nous permet d'afficher la liste du personnel sous forme de tableau. On identifie chaque personne par son numéro, son nom et son prénom ainsi que sa fonction.

Il est aussi possible de modifier un employé grâce à une case à cocher dans la dernière colonne. Un seul employé peut être modifié à la fois. En appuyant sur le lien 'Modifier', les informations de la base de données liées à cet employé pourront être modifiées.

Num 7

Nom JOSTE

Prenom Tom

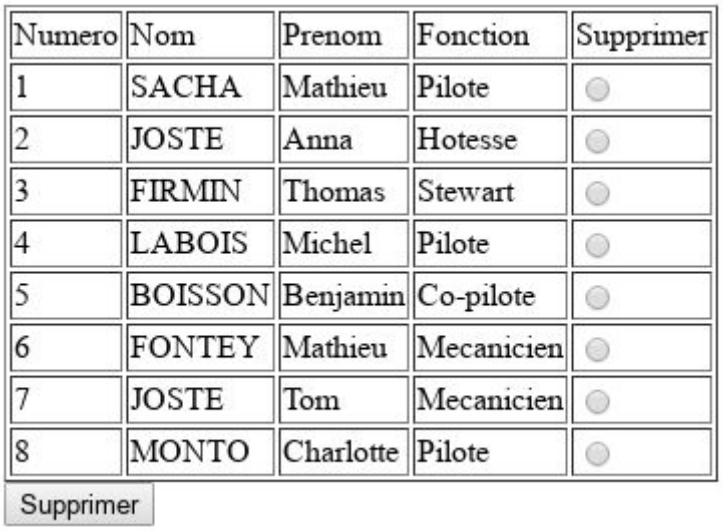
Fonction

Numero de telephone 0621457898

Modifier ce membre

Ce lien nous permet donc d'afficher un formulaire pré-rempli par les données actuelles de l'employé. Il est alors possible de modifier ces données en les modifiants directement dans les champs.

Supprimer un membre



Numero	Nom	Prenom	Fonction	Supprimer
1	SACHA	Mathieu	Pilote	
2	JOSTE	Anna	Hotesse	
3	FIRMIN	Thomas	Stewart	
4	LABOIS	Michel	Pilote	
5	BOISSON	Benjamin	Co-pilote	
6	FONTEY	Mathieu	Mecanicien	
7	JOSTE	Tom	Mecanicien	
8	MONTO	Charlotte	Pilote	

Supprimer

Ce lien nous permet d'afficher la liste du personnel sous forme de tableau. On identifie chaque personne par son numéro, son nom et son prénom ainsi que sa fonction.

Il est aussi possible de supprimer un employé grâce à la case à cocher dans la dernière colonne. Un seul employé peut être supprimé à la fois. En appuyant sur le bouton 'Supprimer', cet employé sera supprimé de la base de données.

Rmq : Il aurait été judicieux d'ajouter un attribut dans la table personnel, de type booléen, qui aurait pris la valeur actif ou non actif. Cela aurait précisé l'état de l'employé au sein de la compagnie et aurait permis notamment de garder un historique des anciens employés.

Difficultés rencontrées

Requêtes difficiles

1) Nous allons étudier les requête et le code complet nécessaire pour effectuer une réservation.

Tout d'abord, le client s'identifie : il rentre donc les valeurs dans les inputs : numéro de client, nom et prénom.

Le bouton 'Identifier' vérifie à l'aide d'une requête SQL si la personne est enregistrée dans la base, c'est-à-dire si son numéro id ET son nom ET son prénom existent déjà dans la base :

S'il existe, alors le client passe directement à la réservation. Son numéro id, son nom et son prénom suffiront à le connecter.

S'il n'existe pas, alors le client devra compléter tous les champs du formulaire pour qu'il soit automatique ajouté à la base de donnée.

Page idclient_2.html :

```
<form action='idclient.php' method='POST'>
    <table border=0>
        <tr>
        <td> Numero id </td>
                    <td> <input type='text' name='numclient'></td>
        </tr>
```

Ici, *action='idclient.php'* permet d'aller vers cette page.
Dans le code, nous avons récupérer la valeur du numéro du client de cette manière :

```
$numclient= $_POST['numclient'];
```

En effet, le **$_POST** permet de récupérer la valeur qui se trouve dans l'input du fichier *idclient_2.html* de nom 'numclient'.

Ensuite, nous faisons une requête qui permet de savoir si le client existe déjà dans la base ou non :

```
$client ="select count(*) AS ex from
(SELECT num_client, nom_client, prenom_client from client
WHERE num_client =".$numclient." and  nom_client='".$nomclient."'
and  prenom_client = '".$pclient."') AS N; " ;
```

La sous-requête permet de sélectionner le client dans la base dont le numéro, le nom et le prénom correspondent à ceux que le client vient de saisie dans les cases. Ensuite, le count permet de compter le nombre de ligne. Dans ce cas, nous aurons une ligne si le client existe et zéro ligne s'il n'existe pas.

Cette requête retournera alors 1 si le client existe et 0 si le client n'existe pas.
Le résultat de cette requête est stocké dans la variable $result.

```
$result = pg_query($connect, $client) ;
```

Ce code php permet d'exécuter la requête avec $connect qui permet de connecter le code php à notre base de donnée.
Ensuite si le client n'existe pas , alors il faut l'insérer :

```
$ajout="INSERT INTO client VALUES (default,'".$nomclient."',
'".$pclient."','".$ddn."','".$pays."');" ;
```

Une fois que nous avons vérifié l'existence ou ajouté le client dans la base, le client est redirigé vers une autre page où l'on copie son numéro de client, son nom et son prénom.

Sur cette page, il y a un bouton 'Reservation' qui permet de continuer la réservation. Une fois encore, on continue de copier les informations concernant le client pour ne pas les perdre.

Enfin on arrive au formulaire pour faire la recherche du vol :

```
$villeDEP="SELECT DISTINCT ville.nom_ville FROM ville
            INNER JOIN vol ON ville.num_ville = vol.n_ville_depart
            WHERE ville.num_ville IN (SELECT distinct vol.n_ville_depart from vol) ; " ;
```

Dans cette requête on sélectionne seulement les villes qui sont définies comme villes départs.
Il en est de même pour la ville d'arrivée.

A présent, on cherche à afficher seulement les vols qui :

- ont les **même villes de départ et d'arrivée** que celle saisies par le client dans le formulaire
- sont prévus pour une **date supérieur** à la date sélectionné par le client
- **ne sont pas complets**

Pour le dernier point, nous avons compté le nombre de personnes ayant déjà réservées le vol, c'est à dire compter le nombre de billets pour chaque vol. Enfin, nous avons comparé ce nombre de billets à la taille de l'avion. Tant que le nombre de billets est inférieur au nombre de sièges (taille) alors le vol sera affiché. Sinon, même si les trois premières conditions sont respectées mais que le vol est complet alors il n'apparaitra pas dans le résultat de cette requête.

```
$requet_vol = "(SELECT DISTINCT vol.num_vol, v1.nom_ville AS villeDep, v2.nom_ville AS
villeArr, vol.date_depart FROM vol
INNER JOIN ville v1 ON v1.num_ville = vol.n_ville_depart
INNER JOIN ville v2 ON v2.num_ville = vol.n_ville_arrivee
INNER JOIN avion ON vol.n_avion=avion.num_avion
```

```
WHERE date_depart > '".$datevol."' AND v1.nom_ville ='".$villedepart."' AND
v2.nom_ville ='".$villearrivee."' AND vol.num_vol IN (SELECT vol.num_vol from billet
INNER JOIN vol on billet.n_vol = vol.num_vol
GROUP BY vol.num_vol
HAVING count(billet.num_billet) <= avion.taille_avion))
UNION(SELECT vol.num_vol, v1.nom_ville AS villeDep, v2.nom_ville AS villeArr,
vol.date_depart FROM vol
INNER JOIN ville v1 ON v1.num_ville = vol.n_ville_depart
INNER JOIN ville v2 ON v2.num_ville = vol.n_ville_arrivee
WHERE date_depart > '".$datevol."' AND v1.nom_ville = '".$villedepart."' AND
v2.nom_ville ='".$villearrivee."' AND num_vol NOT IN (SELECT distinct billet.n_vol
from billet));" ;
```

La sous requête en vert permet d'afficher les vols qui ont un nombre de billets inférieur à la taille de l'avion : **count(billet.num_billet) <= avion.taille_avion**

Ainsi, on sélectionne en premier les vols non complets. Puis dans cette liste, on sélectionne les vols qui correspondent aux critères de la demande du client.

Cependant nous avons rencontré un problème car les vols qui n'avaient encore aucune réservation de s'afficher pas avec cette requête.
Nous avons donc ajouté un requête.
Cette requête nécessite une sous requête : **SELECT distinct billet.n_vol from billet** qui permet d'afficher tous les billets.

Ensuite dans la requête principale, on cherche les vols dont le numéro n'apparaît pas dans cette sous requête : c'est à dire les vols qui n'ont encore aucune réservation puisqu'aucun billet ne correspond à ce vol. Nous avons ajouté dans la requête principale les conditions de date et de villes.

Enfin, pour avoir la liste entière des vols disponibles pour la recherche du client, nous avons fait **l'union** de ces deux requêtes.

2) Pour la modification :

Nous prenons comme exemple la modification d'un membre du personnel.

- `$query = "Select * from personnel where num_pers=".$recup;`

Cette requête permet de récupérer les valeurs pour un employé dont le numéro correspond au numéro de la ligne que nous avons sélectionné dans la page *liste_emp_2.php.*

- `$result = pg_query($connect, $query) ;`

Connexion avec la base de données.

- $pers = pg_fetch_array($result) ;

Résultat de la requête :

$nom = $pers['nom_pers'] ;
$prenom = $pers['prenom_pers'] ;
$fonction = $pers['fonction'] ;
$poste = $pers['num_poste'] ;

Dans ces variables, on récupère la valeur de la requête pour les colonnes nom_pers, prenom_pers, fonction et num_poste de la table **personnel**.

Ici, le formulaire de la page modif_1_membre.php :

```
echo"<h2> Ajouter un membre du personnel </h2>
        <form action='modif_emp.php' method=POST>
        <table border=0>
        <tr>
        <td> Num </td>
                <td> <input type='text'name='nump' value='".$recup."'></td>
        </tr>
        <tr>
```

On affiche les valeurs déjà existantes dans la base de données.

Dans la page *modif_emp.php* on récupère les valeurs présentes et modifiées du formulaire.

```
$num = $_POST['nump'];
$nompers = $_POST['nomp'];
$prenompers = $_POST['prenomp'];
$fonction = $_POST['fonct'];
$numposte = $_POST['nposte'];


$query  = "UPDATE personnel SET
num_pers=".$num.", nom_pers='".$nompers."',
prenom_pers='".$prenompers."', fonction='".$fonction."',
num_poste='".$numposte."' where num_pers=".$num."; " ;
```

Nous avons donc récupérer les valeurs du formulaire et nous modifions les informations d'un employé grâce au UPDATE.

3) Pour la suppression :

Dans la page , on récupère le numéro de l'employé à supprimer.

```
$recup = $_POST['supp'];

$query  = "DELETE from personnel where num_pers=".$recup;
```

L'employé est alors supprimé de la base de données.

À noter que sa suppression n'implique en aucun cas la suppression de ligne contenant son numéro d'employé grâce au **delete set null** dans la création des tables.

Bilan personnel

Morgane Grasmick

Premièrement, lors de mon DUT, j'avais étudié le langage HTML et CSS. Nous n'avons jamais travaillé ces langages avec le php. J'étais donc curieuse de découvrir le langage php afin de faire un lien entre les pages internet et les bases de données, ce qui devient de suite, plus intéressant.
Ainsi, ce projet m'a permis de développer mes connaissances pour créer un site internet et d'assurer par la suite la connexion avec une base de données.

Deuxièmement, ce projet m'a permis d'appliquer et d'approfondir ce que nous avions étudié en sql.

Enfin, je suis contente de l'entente avec mon binôme tout au long de ce projet. Dès le début, il y a eu une bonne communication et cela nous a permis de mener à bien notre projet. Nous avons, pour la plupart du temps, travaillé ensemble, ce qui nous a fait sûrement perdre du temps mais cela nous a aussi permis de nous corriger l'un l'autre.

Pour conclure, j'ai apprécié travailler sur ce projet.

Yann Cauchepin

J'ai vraiment apprécié réaliser ce projet, notamment en travaillant avec Morgane. Le sujet est intéressant et le travail en binôme est très ludique.
Et d'un point de vue collectif, le projet a permis de partager et de débattre sur beaucoup de connaissances, et également d'améliorer la cohésion de la classe.
Toutefois, n'ayant jamais suivi de cours de base de donnée avant GIS, je trouve que le php n'est pas suffisamment enseigné dans ce semestre.