MOD 4.6 - BE

Gestion Compagnie Aérienne

Clémence LÉVECQUE
Guillaume LESAINE

Rapport soumis dans le cadre du MOD 4.6 - Bases de données
École Centrale de Lyon

Décembre 2018

ÉCOLE
CENTRALE LYON

Table des matières

Table des figures

Liste des tableaux

I Introduction

I.1 Contexte

L'application permet de gérer les aspects internes d'une compagnie aérienne. Elle permet d'intéragir avec les entités suivantes :

- Employés
- Vols
- Départs
- Billets
- Passagers

Les fonctionnalités proposées par l'application permettent de :

- Créer des employés, vols et départs
- Visuliser l'activité des pilotes
- Réserver des billets pour des passagers
- Gérer les cinq éléments précédents en permettant de les supprimer.

L'application repose sur une base de données pré-existante contenant les données nécessaires à la création des éléments cités. Certains de ces données comme les aéroports ou les liaisons sont fixes, l'utilisateur ne peut les modifier.

Par ailleurs, pour simplifier l'application et limiter l'effet du temps que nous ne pouvons prendre en compte, nous nous limitons (en particulier au niveau de la programmation de vol) au cas de la gestion de la compagnie au mois de décembre 2018.

I.2 Guide de lecture

Guide de lecture du document Le rapport doit servir à la prise en main de l'application ainsi qu'à l'évaluation en profondeur du travail réalisé. Dans ce but, nous proposons au lecteur, en fonction de ses attentes, de suivre le chemin de lecture suivant qui lui convient.

- **Utilisateur** - Vous souhaitez savoir comment utiliser l'application. Après avoir lu la partie **I. Introduction**, allez directement à la partie **III.3 Parcours utilisateur** et suivez les instructions.

- **Évaluateur** - Vous souhaitez comprendre les dessous de l'application pour mesurer la qualité du travail réalisé. Prenez en main l'application comme un utilisateur (cf chemin de lecture précédent). Lisez ensuite de manière linéaire le rapport. **II. Base de données** présente les choix de construction de la base de données. **III. Application** présente la composition de l'application (**III.1**), ainsi que ses fonctionnalités (**III.2**). Les mesures prises pour assurer l'intégrité de la base de données et une bonne utilisabilité de l'application (**III.2.5** + **III.2.6**) y sont aussi détaillées.

I.3 Ressources

L'application a été déployée, elle peut être consultée avec le lien suivant : Air Centrale

Le projet est hébergé sur GitHub et il est disponible au lien suivant : GitHub

II Base de données

II.1 Schéma de structure

La base de données créée, dont un schéma peut être observé en Figure 1, est composée de 9 tables qui peuvent être séparées comme suit :

— Les tables **navigants** et **employes** qui renseignent toutes les informations nécessaires concernant les salariés de la compagnie aérienne.

— Les tables **passagers** et **billets** dans lesquelles sont stockées les informations concernant les clients et leurs réservations.

— Les tables **liaisons** et **aeroports** qui désignent les liaisons que la compagnie aérienne est autorisée à effectuer.

— La table **appareils** contient la liste de tous les avions de la compagnie aérienne.

— Enfin, les tables **vols** et **departs** sont les plus importantes dans notre application : elles contiennent toutes les données sur les départs, quels pilotes y sont assignés, quel appareil et sur quelle liaison par exemple.

II.2 Choix de construction

II.2.1 Construction des tables

Nous allons ici détailler certains choix de construction pour la base de données.

Tables *vols* et *departs* Vu l'importance de ces tables dans toutes les requêtes qui sont effectuées, il était important de bien séparer les données. Il a été décidé que la table **departs** renseignerait des informations potentiellement plus fluctuentes que les données de la table **vols**.

En effet, on peut aisément concevoir qu'une compagnie aérienne ait des liaisons sur lesquelles des vols sont prévus de manière hebdomadaire. Ces données vont être régulièrement réutilisées sans être modifiées. Cependant, les pilotes, les membres d'équipage ou même les vols assignés à chacun des vols varient selon les disponibilités.

Tables *liaisons* et *aeroports* Il peut sembler plus complexe d'ajouter une table intermédiaire **liaisons** mais ce choix a été fait pour éviter de dupliquer les données. On peut imaginer une table dans laquelle serait renseignées les liaisons. Pour chacune, noms et localisation des aéroports seraient stockés dans des colonnes supplémentaires. Or dans ce cas, le nom d'un aéroport et toutes les informations liées seront répliquées pour chaque liaison.

Pour cela, la table **aeroports** contient des informations statiques sur des aéroports, comme leur noms, codes IATA ou pays. La table **liaisons** contient seulement les liaisons que la compagnie est autorisée à effectuer.

Dans la table **aeroports** il a été décidé que le code IATA ne serait pas la *primary key* puisqu'il existe des cas - bien que rares - où ce code peut être identique pour des aéroports différents. Il était donc préférable d'avoir une clé primaire qui s'auto-incrémente puisqu'elle doit être unique.

Tables *employes* et *naviguants* La première idée était de créer quatre tables : la première contenant tous les employés, la seconde contenant uniquement les pilotes avec leurs temps de vol et leurs numéros de licence, la troisième avec les personnels navigants et leurs temps de vol et enfin une dernières table pour les personnels au sol. Seulement une telle construction semblait très complexe pour l'utilisation qu'il en serait faite.

Puisque l'application s'oriente principalement sur le personnel navigant, la solution imaginée a été de ne créer que deux tables, la première pour tous les employés et la seconde pour tout le personnel navigant et d'ajouter dans la table **employes** une variable *type* qui peut prendre les valeurs "navigant" ou "au_sol". Par conséquent, pour sélectionner un employé au sol il suffit de rajouter en plus des conditions voulues la condition :

```
WHERE type = 'au_sol'
```

et de même pour le personnel navigant. Cependant, il ne s'agit en aucun cas d'une clé avec la table **navigants**.

Dans la table **navigants**, les pilotes et les membres d'équipage sont regroupés, on a de la même manière ajouté un champ *fonction* pour les différencier. De plus, seuls les pilotes auront le champ *num_licence_pilote* rempli.

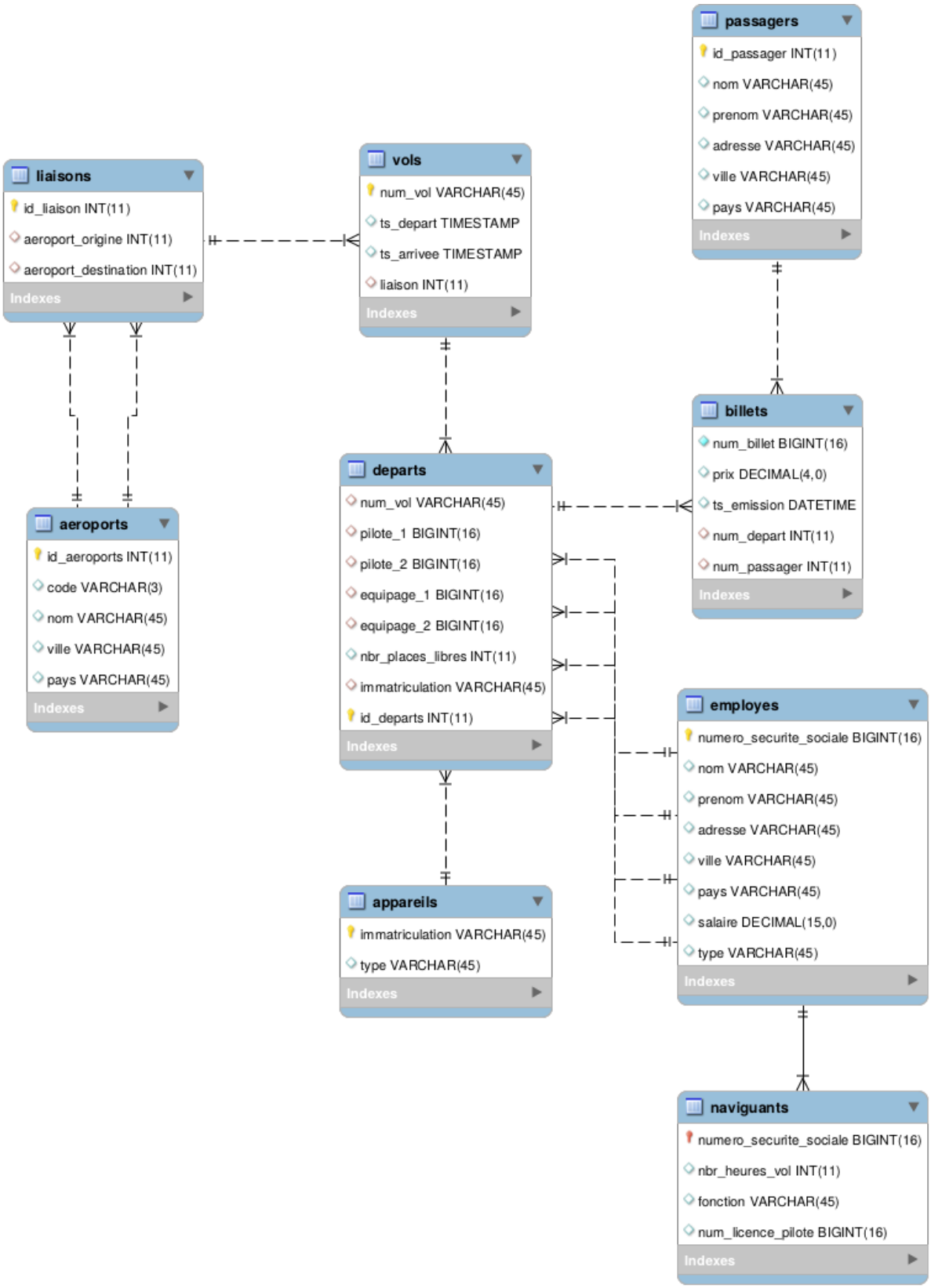

Figure 1 – Schéma complet de la base de données

II.2.2 Format de quelques colonnes

Formats de temps Le format `DATE` ne prend que des dates en compte de format `YY:MM:DD` sauf que pour planifier des départs il est plus utile d'avoir aussi l'heure du départ. On pourrait imaginer faire plusieurs variables : *date_depart*, *date_arrivee*, *heure_depart* et *heure_arrivee* mais cela fait beaucoup de variables à gérer. On préfère donc ne faire que deux variables, *ts_depart* et *ts_arrivee* de format `DATETIME`. Pourquoi ce format plutôt que `TIMESTAMP` ? `TIMESTAMP` désigne une différence de temps entre la date voulue et le 1er Janvier 1970, donc très utile pour une chronologie d'événements par exemple. Cependant, ici on travaille avec des horaires et des dates précis donc on préfère le format `DATETIME`.

Formats de clés primaires Certaines des clés primaires de la base ne sont pas des entiers. Ce n'est pas essentiel du moment qu'elles sont uniques. Par conséquent, le numéro de sécurité sociale a un format `BIGINT(16)` puisque certains numéros de sécurité sociale peuvent aller jusqu'à 16 caractères (13 caractères la majorité du temps). Les numéros de vol sont des `VARCHAR` puisque souvent ils sont composés de chiffres et de lettres.

II.3 Explications et exemples de requêtes

Nous allons détailler ici quelques unes des requêtes utilisées afin de mieux comprendre les conditions qui ont été choisies.

Sélection des personnels en cours de vol Pour sélectionner tous le personnel navigant actuellement en vol, il va falloir travailler sur la table **departs**. On veut pouvoir sélectionner chaque personne avec le vol auquel elle est associée puis vérifier si le vol a lieu maintenant. Cependant la table est construite comme suit en Figure 1.

num_vol	pilote_1	pilote_2	equipage_1	equipage_2	...
AFR348	191851249560845	163577693139738	174718143121594	292635691720225	...
AFR7644	275964135451744	182942912944414	293109716948355	282123825733664	...
DAL405	191851249560845	189145542576748	293109716948355	293109716948355	...

Table 1 – Tables departs (5 premières colonnes)

Par conséquent, pour récupérer toutes les personnes assignées à un vol il faut faire des `UNION`, ce qui rend la requête un peu lourde mais est simplement une sélection dans chaque colonne de toutes les lignes.

```
SELECT naviguant
FROM (
        (SELECT pilote_1 AS naviguant, num_vol
                FROM departs)
        UNION DISTINCT
        (SELECT pilote_2 AS naviguant, num_vol
                FROM departs)
        UNION DISTINCT
        (SELECT equipage_1 AS naviguant, num_vol
                FROM departs)
        UNION DISTINCT
        (SELECT equipage_2 AS naviguant, num_vol
                FROM departs)
) AS naviguants_vols
LEFT JOIN vols ON naviguants_vols.num_vol = vols.num_vol
WHERE (ts_depart < NOW() AND NOW() < ts_arrivee);
```

On voit clairement les quatre sous-tables qui sont rassemblées en une seule grâce aux `UNION`. En SQL, chaque nouvelle sélection doit être renommée c'est pourquoi celle-ci a été renommée **naviguants_vols** et correspond à tous les naviguants assignés sur un vol. C'est sur cette nouvelle sous-table que la suite de la requête est effectuée. On `JOIN` cette dernière avec la table **vols** pour récupérer les horaires des vols et on sélectionne tous les vols dont l'heure de départ est plus petite que l'heure actuelle (`NOW()`) et l'heure d'arrivée est plus grande. On obtient bien tous les naviguants en vol actuellement.

Sélection des prochains vols pour chaque personnel Une autre requête particulière est la sélection des vols à venir (ou passés, la requête est similaire) pour chacun des membres navigants.

```
SELECT naviguant,
       MIN(ts_depart) as min_ts_depart
FROM (
        (SELECT pilote_1 AS naviguant, num_vol
                FROM departs)
        UNION
        (SELECT pilote_2 AS naviguant, num_vol
                FROM departs)
        UNION
        (SELECT equipage_1 AS naviguant, num_vol
                FROM departs)
        UNION
        (SELECT equipage_2 AS naviguant, num_vol
                FROM departs)
) AS naviguants_vols
LEFT JOIN vols
ON naviguants_vols.num_vol = vols.num_vol
WHERE ts_depart > NOW()
GROUP BY naviguant
```

Comme dans la requête précédente, on va sélectionner tous les personnels navigants mais sans l'option `DISTINCT` puisque cette fois-ci on ne veut pas juste les numéros de sécurité sociale de chaque personnel mais aussi tous les vols auxquels ils sont associés et il peut y en avoir plusieurs. On a donc une liste de tous les personnels (avec duplicats) et de tous les vols qui leur correspondent. On va regrouper les personnels grâce à un `GROUPBY` et sélectionner l'heure du prochain vol (son horaire doit être plus grand que l'heure actuelle mais le plus petit parmi tous ces prochains horaires).

Pour sélectionner le dernier départ on réalise la même fonction mais on va calculer `MAX(ts_arrivee)` et la condition sera `ts_arrivee < NOW()` : parmi tous les vols qui sont déjà finis on va sélectionner le plus récent.

Sélection des personnels disponibles pour un certain vol Les requêtes précédemment exposées vont être très utiles pour une interrogation qui a orienté toute l'application :

"Je dois m'assurer que chaque vol décolle bien avec deux pilotes et deux membres de l'équipage. Je créé un nouveau vol XXX. Comment assigner les pilotes et les membres de l'équipage sur ce vol en étant sure que ceux-ci soient totalement disponibles ?"

Certaines conditions pour qu'un personnel navigant soit considéré comme disponible ont été décidées. On imagine chercher un pilote pour un vol qu'on note `NV_VOL`.

- La personne doit être dans le pays de départ ou que son dernier vol l'ait amené dans le pays duquel repartira le `NV_VOL`.
- La personne ne doit pas être en vol pendant cette période, c'est-à-dire pas de superposition de vols avec le `NV_VOL`.
- Si la destination du `NV_VOL` est différente que le prochain vol que la personne doit effectuer, elle doit avoir le temps de rentrer. Sinon, il faut simplement qu'elle ait le temps d'arriver avant le départ de `NV_VOL`.
- Son nombre d'heures de vol doit être inférieur à 95 dans les deux cas précédents (aller ou aller-retour).
- Si le dernier vol de la personne remonte à plus de deux jours, on considère qu'elle est rentrée dans son pays d'origine.

III Application

III.1 Choix de conception

L'application créée est une application web. Elle permet la gestion d'une compagnie aérienne du point de vue d'un employé interne. Nous n'avons considéré qu'un seul type d'utilisateur tout puissant.

Les choix de développement suivants ont été faits :

Rôle	Outil	Raison
Serveur	Flask (Python)	**Flask** est un framework Python simple et léger permettant d'écrire en Python et d'utiliser SQL.
Front-end	• HTML + CSS • Bootstrap • JavaScript	**Bootstrap** est une boite à outils, qui améliore l'esthétique des interface et qui permet une bonne compatibilité avec Flask. **JavaScript** améliore l'intéraction entre l'utilisateur et l'application et en facilite l'usage.
Base de données	• MySQL • MySQLWorkbench	**MySQL** permet d'utiliser SQL et suffit pour mettre en oeuvre les applications du cours. **MySQLWorkbench** permet de gérer visuellement la base de données.
Déploiment	Heroku	**Heroku** (Salesforce) propose un service d'hébergement d'application et de base de données. Le déploiment a été réalisé par soucis d'accessibilité. Les données sont hébergées aux États-Unis mais cela ne pose pas problème car elles n'ont aucune valeur.

Table 2 – Choix de développement

III.2 Fonctionnalités

Les fonctionnalités offertes par l'application sont regroupables par quatre fonctions décrites dans le tableau suivant.

Fonction	Rôle	Fonctionnalités
Créer	Permettre à l'utilisateur d'enrichir la base de données avec des éléments visant à l'organisation interne de la compagnie aérienne.	• Créer un employé navigant ou au sol. • Créer un vol. • Créer un départ associant des employés disponibles à un vol.
Visualiser	Informer l'utilisateur sur l'activité des employés navigants.	• Visualiser les vols en cours, futurs et passés d'un employé navigant.
Réserver	Permettre à un utilisateur de réserver un billet pour un client.	• Enregistrer un passager • Associer un passager à une place sur un départ
Gérer	Permettre à l'utilisateur de contrôler le contenu de la base de données et d'en supprimer les éléments que l'application permet de créer.	• Supprimer un employé • Supprimer un vol • Supprimer un départ • Supprimer un passager • Supprimer un billet

Table 3 – Fonctionnalités de l'application

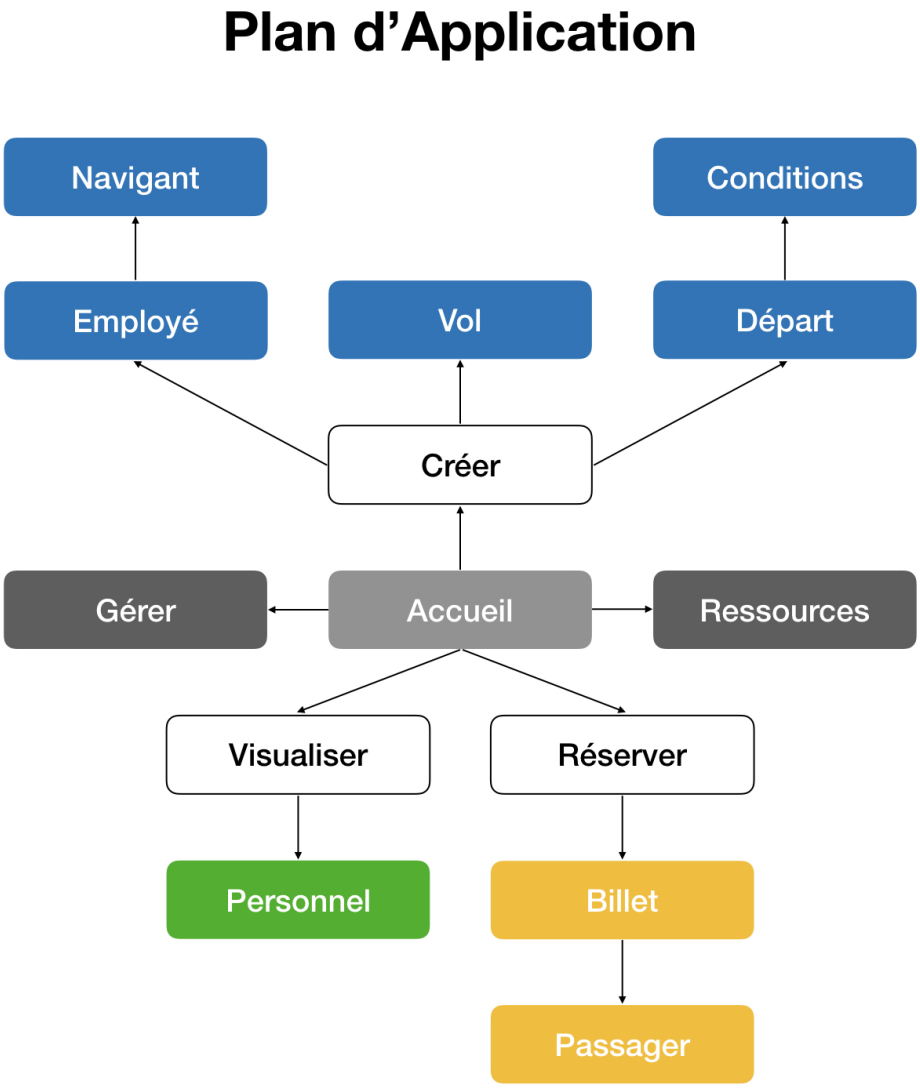

Figure 2 – Plan d'application

Dans le but de détailler les fonctionnalités et de donner une compréhension des requêtes SQL à l'oeuvre, nous fixons la convention suivante. Pour chaque page nous distinguons des groupes de requêtes symbolisés par $\star$ dans les paragraphes suivants. À chaque groupe est associé un qualificatif entre crochets [ ].

Groupe	Objectif
[Validation]	Vérification d'un fait dans la base de donnée nécessaire au respect de règles MySQL ou de la logique de l'application.
[Insertion]	Ajout d'une instance à une table SQL.
[Affichage]	Prise d'information dans la base de données pour donner des éléments de choix explicites à l'utilisateur.
[Modification]	Mise à jour d'une valeur dans la base de donnée.
[Suppression]	Suppression d'une instance d'une table SQL.

Table 4 – Groupes de requêtes

III.2.1 Créer

- **/creer/employe** - Formulaire de création d'employé
  - ⋆ [Validation] - Vérification de l'unicité du numéro de sécurité sociale entré par l'utilisateur.
  - ⋆ [Insertion] - Ajout d'une instance dans la table employé.
- **/creer/employe/naviguant/(x)** - Formulaire de caractérisation d'un employé navigant portant le numéro de sécurité sociale "x". Cette page suit la soumission du formulaire de la page /creation/employee si l'employé est navigant.
  - ⋆ [Validation] - Vérification de l'unicité du numéro de licence entré par l'utilisateur si l'employé est pilote.
  - ⋆ [Insertion] - Ajout d'une instance dans la table navigants.
- **/creer/vol** - Formulaire de création d'un vol
  - ⋆ [Affichage] - Sélection des liaisons afin de les proposer à l'utilisateur. La requête comprend des jointures afin de proposer à l'utilisateur des codes explicites d'aéroports provenant de la table aéroports.
  - ⋆ [Validation] - Vérification de l'unicité du numéro de vol et validité des temps entrés par l'utilisateur.
  - ⋆ [Insertion] - Ajout d'une instance dans la table vols.
- **/creer/depart** - Page de création d'un départ. Une liste de vols est proposée à l'utilisateur.
  - ⋆ [Affichage] - Sélection des vols existants afin de les proposer à l'utilisateur. La requête comprend des jointures afin de proposer des informations explicites sur aéroports provenants de la table aéroports.
- **/creer/depart/conditions/(x)** - Page de caractérisation du départ portant le numéro de vol "x". Cette page suit la sélection d'un vol de la page /creation/depart.
  - ⋆ [Affichage] - Sélection des pilotes et des appareils disponibles pour le départ en question. Ces requêtes sont particulièrement complexes et ont été évoquées en partie **II.3**.
  - ⋆ [Insertion] - Ajout d'une instance dans la table départs.
  - ⋆ [Modification] - Modification du nombre d'heures de vol du personnel navigant concerné.

III.2.2 Visualiser

La fonction Visualiser permet à l'utilisateur de connaitre l'activité d'un employé navigant en terme de départs. Cette activité peut être passée, présente ou future. Une unique page existe dans cette fonction, elle est rendue intéractive par du JavaScript et des jQuery pour permettre un accès simplifié à l'ensemble de l'information. Aucune soumission à la base de données n'est réalisée sur cette fonction.

- **/visualiser/personnel** - Page de sélection d'un départ amenant à une visualisation de l'activité d'un pilote.
  - ⋆ [Affichage] - Obtenir l'information descriptive des employés permettant leur sélection. Cette requête nécessite de joindre l'ensemble des informations utiles à l'identification d'employés par l'utilisateur, contenu dans les tables employés et navigants.
  - ⋆ [Affichage] - Obtenir pour chaque employé les vols passés, présents et futurs. Cette requête nécessite de joindre l'ensemble des informations nécessaires à l'identification des départs. À ce titre, des jointures et la commande SQL "NOW()" ont été utilisées.

III.2.3 Réserver

La fonction Réserver permet à l'utilisateur de réserver un billet pour un passager à partir d'un départ. Cette réservation doit être répercutée sur le nombre de places libres du départ sélectionné. Deux pages existent dans cette fonction.

- **/reserver/billet** - Page de sélection du départ voulu par le passager.
  - ⋆ [Affichage] - Obtenir l'information descriptive des départs permettant au passager de faire son choix.
- **/reserver/billet/passager/(x)** - Page de renseignement de l'information du passager souhaitant réserver un billet pour le départ numéro "x".

- ⋆ [Affichage] - Obtenir l'information descriptive du départ choisi par le passager pour rappel.
- ⋆ [Validation + Insertion] - Vérification de la pré-existence du passager dans la base de donnés. Si le passager n'a jamais voyagé avec la compagnie, ses informations sont ajoutées à la base de données.
- ⋆ [Insertion // Modification] - Ajout du billet réservé par le passager dans la table Billets. En parallèle, le nombre de places disponibles dans la table des Départs est décrémenté.

III.2.4 Gérer

La fonction Gérer permet à l'utilisateur de prendre connaissance du contenu de la base de données concernant les tables employés, vols, départs, billets et passagers. L'utilisateur a aussi la possiblité de supprimer du contenu. Une unique page existe dans cette fonction, elle est rendue intéractive par du JavaScript et du jQuery pour améliorer l'alternance entre les différentes entités de la base de données.

- **/gerer** - Page de sélection des entités de la base de données à supprimer.
  - ⋆ [Affichage] - Obtenir l'information descriptive des entités de la base de données permettant une identification claire en vue de la suppression. Des jointures permettent d'avoir l'ensemble des informations nécessaires.
  - ⋆ [Validation + Suppression // Modification] - Les contrôles de non-ruptrue de relation de clés-étrangères sont basés sur des requêtes de vérifications de liens entre les entités (Validation). L'interface permet ensuite de supprimer les instances sélectionnées par l'utilisateur (Suppression). Les suppressions d'une table peuvent avoir des répercutions sur les autres tables (Modification). La stratégie adoptée lors d'une requête de suppression pour chaque entité est la suivante :
    - — **Billets** - Suppression du billet concerné dans la table Billets et incrément de 1 du nombre de places disponibles dans la table Départs pour le départ lié.
    - — **Départs** - Suppression du départ concerné dans la table Départs et suppression automatique de tous les billets liés au départ dans la table Billets. Modification du nombre d'heures de vols pour les employés navigants concernés par le départ.
    - — **Passagers** - Suppression du passager dans la table Passagers et suppression de l'ensemble des billets correspondants entrainant des incréments de places disponibles dans la table Départs.
    - — **Vols** - Si aucun départ n'est lié au vol, le vol est supprimé. Sinon, le vol n'est pas supprimé et l'utilisateur est notifié du départ lié à supprimer.
    - — **Employés** - Si aucun départ n'est lié à l'employé, l'employé est supprimé. Sinon, l'employé n'est pas supprimé et l'utilisateur est notifié du départ lié à supprimer.

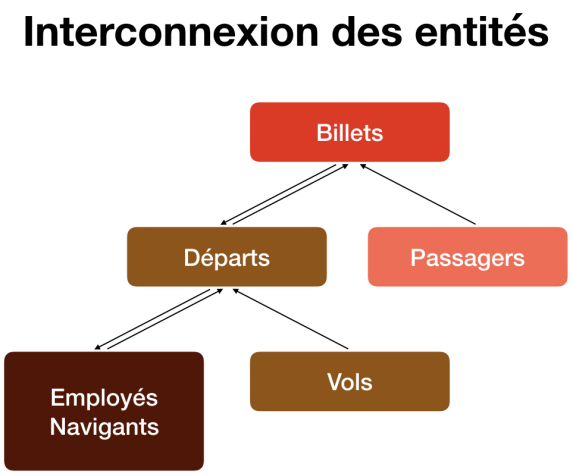

Figure 3 – Interconnexion des entités de la base de données

Une des limites de l'application présentée est la nécessité d'annuler un départ et les billets associés si un employé est supprimé. Une interface permettant d'interchanger des pilotes serait nécessaire.

III.2.5 Contrôle d'intégrité

Un élément primordial de la gestion d'une base de données supportant une application est d'assurer son intégrité au fil des utilisations. Pour cela, il est nécessaire de contrôler l'activité de l'utilisateur et de l'aligner sur la logique de l'application. À cette fin, un ensemble de dispositifs a été mis en place pour accompagner et contraindre (avec bienveillance) l'utilisateur.

Contraintes invisibles d'utilisation

- **SelectField** - Proposition de choix à l'utilsateur plutôt qu'entrée de valeur entrainant des inconsistences.

Figure 4 – SelectField contre EntryField

- **RadioBoxes** - Limitation de la sélection à un élément plutôt que choix multiples.

Figure 5 – RadioBoxes contre CheckBoxes

Messages d'avertissement

- **Messages Flash** - Envoi de message en tête de page afin d'informer l'utilisateur. Un message rouge correspond à une action illicite de l'utilisateur l'empêchant d'endommager la base de données.

Figure 6 – Avertissement de clé primaire déjà existante

- **Contrôle de saisie** - Apparition d'information pour accompagner l'utilisateur dans la soumission d'un formulaire et lui indiquer le contenu exigé des champs de saisie.

Figure 7 – Avertissement du format attendu de l'entrée

Processus d'accompagnement

- **Contrôle de suppression** - L'application contraint la suppression d'entités pour respecter les relations de clés étrangères. L'utilisateur est alors informé du caractère problématique de ce qu'il est en train de faire et il est accompagné dans la réalisation de son action de manière licite.

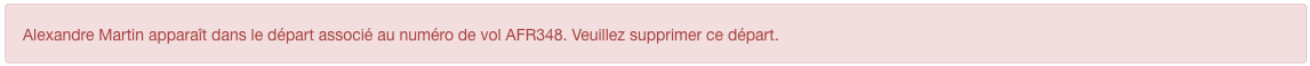

Figure 8 – Avertissement de liaisons clés-étrangères et conseil de résolution de conflit

III.2.6 Feedback

Pour finir, afin d'améliorer la compréhension qu'un utilisateur peut avoir de ses actions sur la base de données, un dispositif de feedback a été mis en place.

Le dispositif repose essentiellement sur des Messages Flash informant l'utilisateur de la bonne ou non réalisation de ses actions sur la base de données. Un message bleu signifie une confirmation de soumission ou une information d'accompagnement. Un message rouge correspond à une action illicite de l'utilisateur l'empêchant d'endommager la base de données, comme expliqué précédemment.

Figure 9 – Confirmation d'exécution à l'utilisateur

Toujours dans un soucis de rapprochement entre l'utilisateur et la base de données, la fonction Gérer a été créée. Elle informe sur le contenu de la base, en plus de permettre la suppression d'entités.

III.3 Parcours utilisateur

La liste d'action suivante est une proposition d'utilisation de l'application. Les actions s'enchaînent et doivent permettre à un utilisateur de comprendre la logique de l'application ainsi que ses fonctionnalités. Pour vous connecter à l'application, utiliser le lien suivant Air Centrale.

Nous conseillons fortement à un nouvel utilisateur de suivre ces étapes pour apprivoiser l'application. Les étapes **III.3.1** à **III.3.7** sont très guidées. Une fois celles-ci réalisées, réaliser la partie **III.3.8**.

III.3.1 Créer - Ajout d'un nouveau pilote

Dans l'ensemble des scénarios suivants, les champs dont les valeurs ne sont pas spécifiés peuvent être complétés de manière libre. Par ailleurs, la localisation des employés à une grande importance pour les processus, respecter donc leur pays de vie.

Objectif	Créer un nouvel employé pilote appelé Franck Debouck
Page(s) utilisée(s)	Création d'un employé et caractérisation comme navigant
Action	• Sélectionner la fonctionnalité Créer - Employé • Entrer les information du pilote, Franck Debouck habite en France, à Écully • Entrer un numéro de licence quelconque
Constat	• Vérifier que le pilote a bien été créé avec la fonction Gérer

Table 5 – Scénario d'ajout d'un nouveau pilote

III.3.2 Créer - Ajout d'un nouveau vol

Objectif	Créer un nouveau vol Paris - Madrid
Page(s) utilisée(s)	Création d'un vol
Action	• Sélectionner la fonctionnalité Créer - Vol • Donner un numéro quelconque au vol • Choisissez la liaison CDG - MAD • La date du vol est le 15/12/2018
Constat	• Vérifier que le vol a bien été créé avec la fonction Gérer

Table 6 – Scénario d'ajout d'un nouveau vol

III.3.3 Créer - Ajout d'un nouveau départ

Objectif	Créer un nouveau départ pour Franck Debouck
Page(s) utilisée(s)	Création d'un départ
Action	• Sélectionner la fonctionnalité Créer - Départ • Choisissez le vol précédemment créé • Choisir parmi les employés, le pilote Franck Debouck • Il n'est pas obligatoire de choisir d'autres employés • Choisir un appareil quelconque
Constat	• Vérifier que le départ a bien été créé avec la fonction Gérer • Observer le nombre de places disponibles avant ajout d'un passager • Observer que les heures de vol de Franck sont égales au temps de vol entré lors de la création du vol

Table 7 – Scénario d'ajout d'un nouveau départ

III.3.4 Visualiser - Vue du programme de vol d'un pilote

Objectif	S'informer sur le programme de vol de Franck Debouck
Page(s) utilisée(s)	Visualisation du personnel
Action	• Sélectionner la fonctionnalité Visualiser - Personnel • Choisissez le pilote Franck Debouck

Table 8 – Scénario de visualisation de l'activité d'un pilote

III.3.5 Réserver - Enregistrement d'une réversation de billet

Objectif	Réserver un billet pour Zinedine Zidane sur le vol piloté par Franck Debouck
Page(s) utilisée(s)	Réservation de billet
Action	• Sélectionner la fonctionnalité Réserver - Billet • Choisir le départ précédemment créé • Entrer les informations du passager, Zinedine habite à Madrid
Constat	• Vérifier l'enregistrement du passager et du billet dans la fonction Gérer • Observer le nombre de places désormais disponibles dans le départ

Table 9 – Scénario de réservation de billet

III.3.6 Gérer - Suppression d'entités

Objectif	Supprimer les entités précédemment créées
Page(s) utilisée(s)	Gestion de la base de données
Action	• Sélectionner la fonctionnalité Gérer • Supprimer le pilote déjà créé • Supprimer le vol déjà créé • Supprimer le départ déjà créé • Supprimer le billet déjà créé • Supprimer le passager déjà créé
Constat	• L'application contrôle la suppression des entités liées par des clés étrangères

Table 10 – Scénario de gestion de la base de données

III.3.7 Visualiser - Relève d'activité

Objectif	Visualiser d'activités de pilotes
Page(s) utilisée(s)	Visualisation du personnel
Action	• Sélectionner la fonctionnalité Visualiser - Personnel • Observer l'activité de Vladimir POUTANE • Observer l'activité de Maria AGRADA

Table 11 – Scénario de visualisation d'activité d'employés existants

III.3.8 Problème ouvert

Une nouvelle hôtesse vivant en france arrive dans l'entreprise. Le service RH souhaite programmer son premier itinéraire. Le service imagine de lui faire faire une boucle Paris - New-York - Montréal - Paris.

L'hôtesse doit donc partir des aéroports suivants aux dates associées : **CDG à Paris le 10/12/2018**, **JFK à New-York le 12/12/2018**, **YUL à Montréal le 15/12/2018**.

Des vols ont déjà été entrés dans la base. Créer les départs qui lui permettront d'effectuer ce circuit à partir des vols déjà créés. Observer les répercussions sur la base de données, avec la fonction Gérer.

IV Bibliographie

Les bases de l'application sont liées au tutoriel suivant : Tutoriel Flask + Bootstrap de Miguel Grinberg

Plus généralement, le développement de l'application a nécessité les sites suivants :

- Bootstrap
- Stack Overflow
- W3Schools
- Flask Documentation