Équipe Atla

ERiP
Recrutement des étudiants

Rapport sur la partie base de données

Décembre 2019

Sommaire

Introduction

ERiP est un logiciel de pilotage de formations universitaires. Notre équipe se charge du développement de la composante *Recrutement des élèves* qui comprend la gestion d'une base de données. Nous allons présenter dans ce rapport les choix de modélisation et d'implémentation de cette base de données.

Présentation de la composante

> "Le logiciel doit pouvoir superviser le recrutement des étudiants, qui peuvent être recrutés sur plusieurs concours : un concours destiné aux élèves en classes préparatoires, et un autre aux élèves titulaires d'un diplôme ou d'une équivalence (dits "sur titres"). Le concours CPGE est géré par le SCEI qui transmet les informations nécessaires à l'école une fois les candidats sélectionnés. Le concours sur titres est organisé par l'école et piloté à l'aide du présent logiciel. Il doit notamment permettre aux candidats de déposer les pièces constituant leur dossier, à l'école de déclarer admissible certains candidats, d'organiser des entretiens pour les évaluer puis de déclarer admis certains candidats. Le suivi et l'intégration des candidats doivent également être possibles. L'objet du recrutement est de permettre ensuite l'accueil des étudiants pour l'intégration de l'école à la rentrée."
>
> Extrait de l'appel d'offre du logiciel ERiP

Cet extrait et un rendez-vous avec la MOA nous ont permis de modéliser complètement notre composante. L'étude fonctionnelle a déjà été présentée dans la réponse à appel d'offre, mais nous la rappelons et la précisons dans la partie suivante.

La base de données doit assurer le bon fonctionnement de ces différentes fonctionnalités. Nous avons utilisé l'outil MySQL Workbench pour l'implémenter.

Identification des cas d'utilisation

L'étude de la composante *Recrutement des élèves* du logiciel ERiP nous a permis la constitution de ce diagramme des cas d'utilisation :

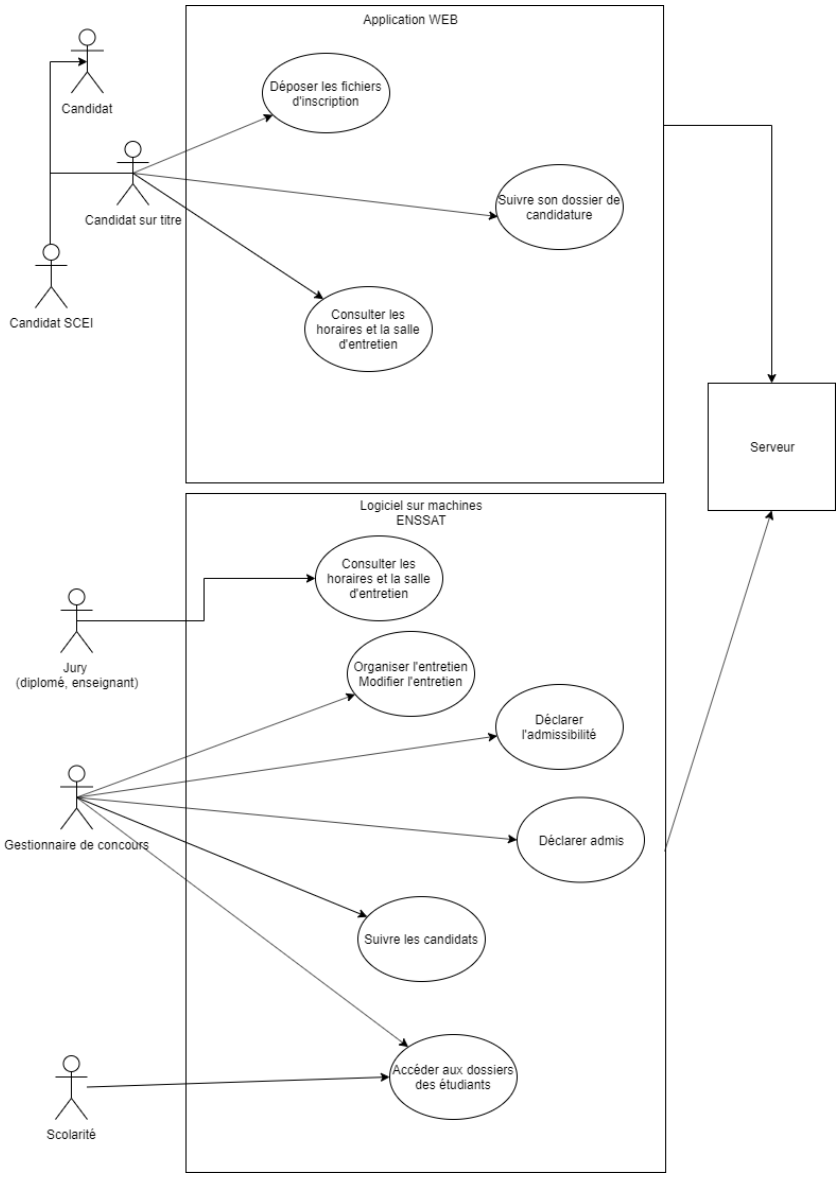

FIGURE 1 – Diagramme des cas d'utilisation

Dans les prochains paragraphes, nous allons expliciter chaque acteur et chaque fonctionnalité.

Acteurs

Afin de réaliser une base de données en accord avec nos besoins, nous avons identifié différents rôles d'utilisation du système :

Candidat

On distingue deux types de candidats :

- Candidat sur titre : candidat devant créer une candidature et éventuellement participer à un entretien.
- Candidat SCEI : candidat admis et intégré plus tard dans la base.

Jury

Le jury peut être un enseignant, un diplômé ou même quelqu'un de l'extérieur.

Gestionnaire de Concours

Il est en charge du bon déroulement des phases d'admission de la candidature sur titre qui sont le traitement du dossier, l'entretien le résultat. Il gère aussi la gestion des états d'admissibilité à chaque phase.

Scolarité

La scolarité doit avoir accès au données des candidats admis afin de pouvoir les inscrire.

Fonctionnalités

Déposer les fichiers d'inscription

Le candidat doit déposer son dossier pour s'inscrire au concours sur titre. Il doit déposer des pièces justificatives stockées sur le serveur.

Suivre le dossier de candidature

Le candidat a le droit de consulter son dossier et son statut dans les étapes d'admission.

Consulter les informations de l'entretien - Candidat

Le candidat doit pouvoir consulter les informations nécessaires pour passer son entretien du concours sur titre.

Consulter les informations de l'entretien - Jury

Le jury doit pouvoir consulter les informations nécessaires pour encadrer les entretiens du concours.

Organiser ou modifier l'entretien

Le gestionnaire de concours doit avoir accès, doit pouvoir modifier, doit pouvoir créer des entretiens du concours sur titre.Le jury peut attribuer une note à l'issu d'un entretien.

Déclarer l'admissibilité

Le gestionnaire peut déclarer et modifier l'admissibilité d'un candidat.

Suivre les candidats

Le gestionnaire et le jury a accès au dossier complet des candidats.

Accéder aux dossiers des étudiants

A la fin du concours, les candidats admis vont pouvoir s'inscrire dans l'établissement et donc les dossiers qui servait à l'inscription au concours sont transmis à la scolarité.

Modélisation

L'identification des cas d'utilisation nous permet de dégager la structure de notre base de données à l'aide du schéma entité-association et relationnel.

Schéma entité-association

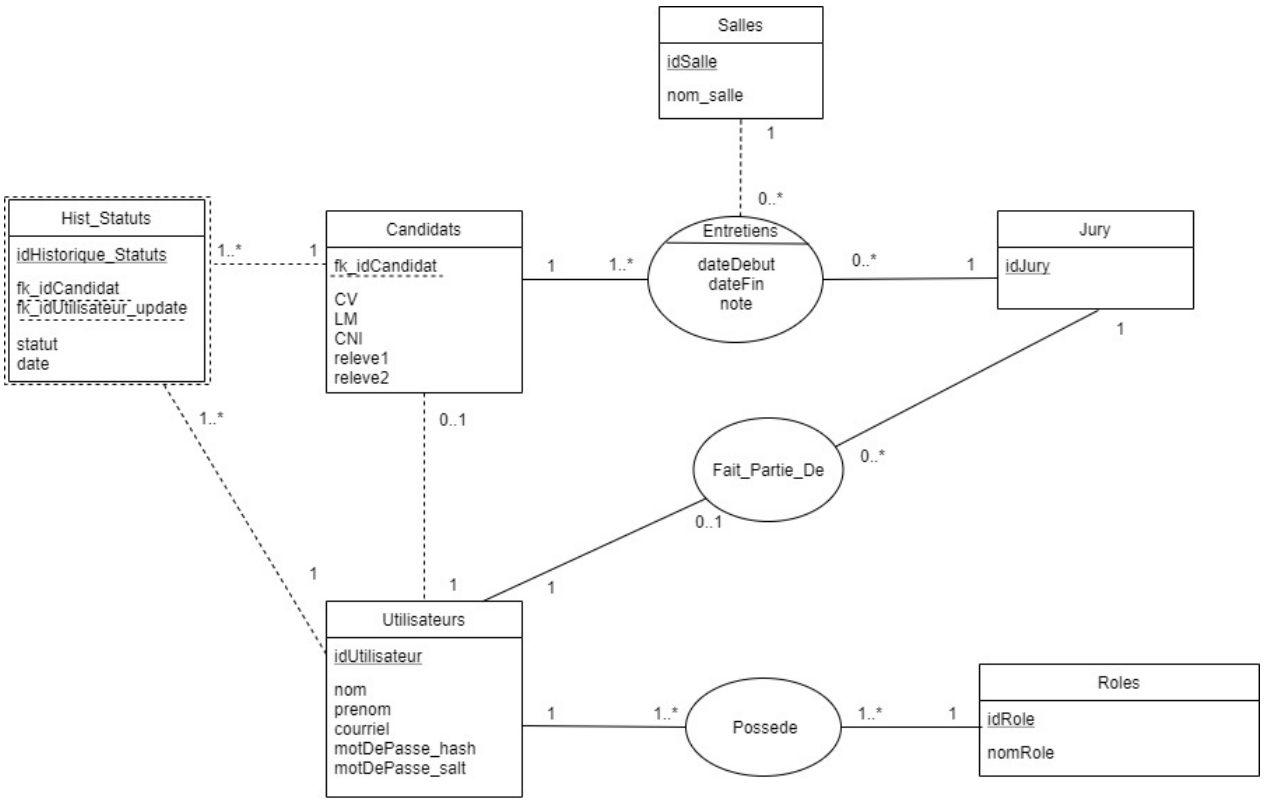

FIGURE 2 – Schéma entité-association

Le schéma entité-association permet de spécifier les tables nécessaires à la BDD.

Schéma relationnel

S'ensuit donc le schéma relationnel de la FIGURE 3 généré sur MySQL Workbench.

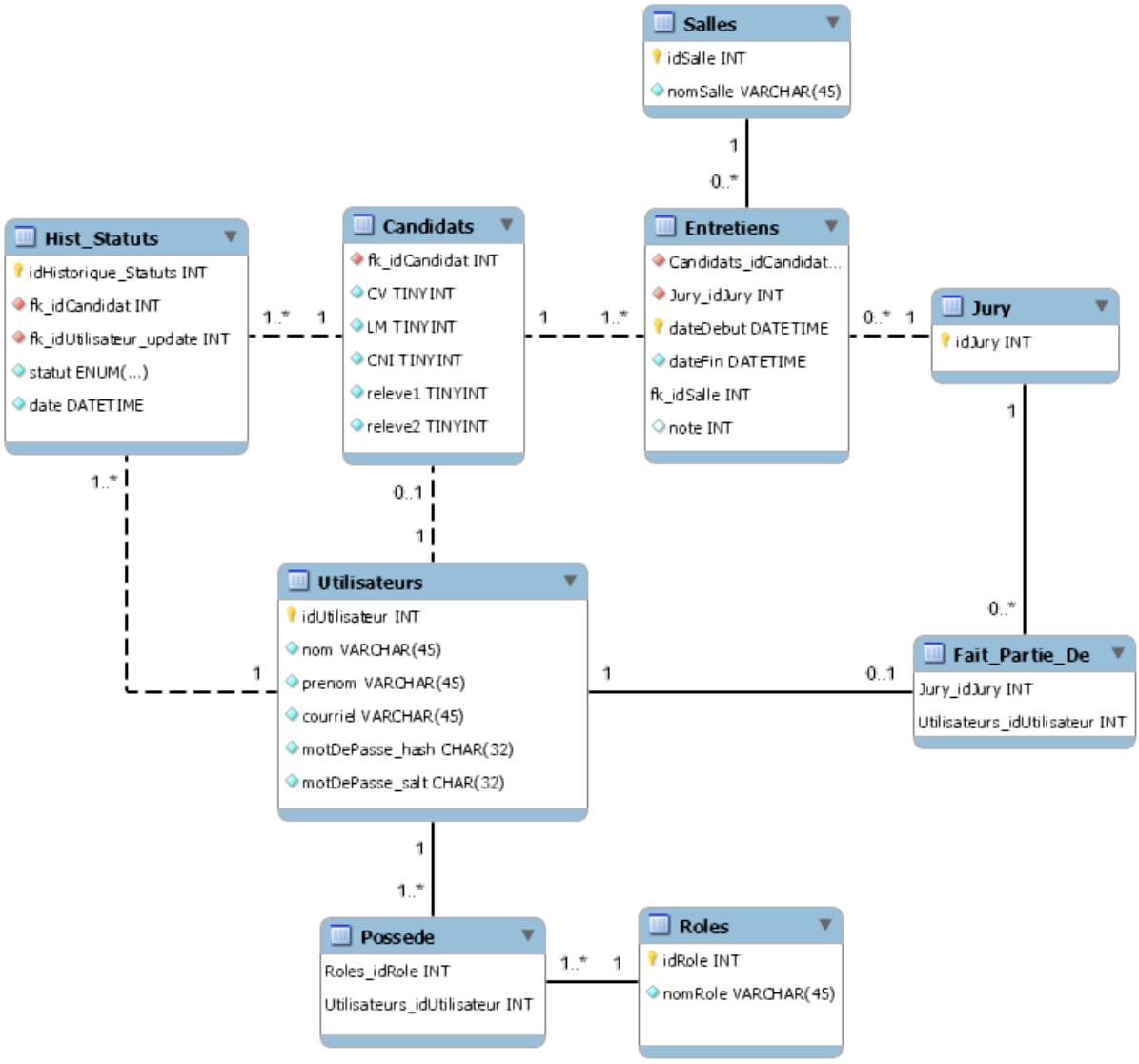

FIGURE 3 – Schéma relationnel

Justification des choix de modélisation

Pour la **table Utilisateurs**, nous avons décidé de factoriser les informations communes à chaque utilisateur indépendemment de leur statut. Cependant, les candidats et les jurys ont besoin de leur propre table car ils partagent la relation Entretien. On utilise donc une **table Candidats** et une **table Jurys** qui héritent de la table Utilisateurs. La table Candidats permet de connaître l'avancement du dossier de l'étudiant et donc de vérifier si le dossier est complet.

Les autres utilisateurs se voient attribuer un rôle (d'où l'implémentation de la **table Rôles**). Cela permet d'attribuer différents rôles à un même utilisateur sans pour autant créer une table pour chaque rôle. On accédera alors à l'ensemble des utilisateurs possédant un rôle donné en créant des vues.

La **table Possede** permet de faire le lien entre la table Utilisateurs et la table Roles : un utilisateur possède un ou plusieurs rôles.

La **table Fait_Partie_De** permet de faire le lien entre la table utilisateurs et la table Jury. En effet, un Jury est composé d'un ou plusieurs utilisateurs.

La **table Hist_Statuts** offre la possibilité de garder un historique des différents status que le Candidat a pu acquérir au cours de son parcours d'admission. Ainsi, l'attribut de type énuméré "statut" contiendra les informations relatives à l'historique du parcours du candidat (inscrit, admissible, admis, intégré, refusé).

La **table Entretiens** permet d'associer un candidat, un jury et une salle pour créer un entretien. Cet entretien possède une date-heure de début et de fin ainsi qu'une note qui est attribuée après l'entretien.

Enfin, la **table Salles** est créée de manière à attribuer une salle à chaque entretien.

Limites du schéma

- L'insertion d'un candidat dans la base de donnée engendre une redondance de données puisque il est stocké dans la table Candidat et en même temps on lui donne le rôle candidat.
- Nous avons fait le choix de limiter le nombre de jury qu'un utilisateur peut composé à un pour éviter qu'il participe à des entretiens qui se chevauchent.

Réalisation

Pour mettre en œuvre cette base de donnée, nous avons utilisé Workbench 8.0 avec MySQL 5.7. Nous avons crée les tables en accord avec notre modélisation et nous y avons ajouté les contraintes conservant l'intégrité des données (les triggers). Ensuite, nous avons créé les procédures essentielles à l'utilisation de cette base par l'application. Les triggers et les procédures ont été testés progressivement pendant l'implémentation grace à une génération de données. Ils ont été modifié et validé par la suite grace à des fichiers test.

Triggers

Chaque table a besoin de contraintes pour avoir de la cohérence dans les données. En effet, ces contraintes interviennent à des moments particuliers, avant et après l'insertion, avant et après la mise à jour et avant la suppression des données. C'est ainsi que nous allons les présenter.

Table Candidat

- **Avant l'insertion**
  Un utilisateur ne peut pas déjà avoir un rôle avant d'être inséré dans la table.
- **Après l'insertion**
  L'utilisateur reçoit en parallèle le rôle candidat, en étant inséré dans la table Possède.
- **Avant la mise à jour**
  La modification d'un identifiant dans la table candidat est impossible.
- **Après la suppression**
  Il faut supprimer l'instance correspondante dans la table Possède aussi.

Table Entretien

- **Avant l'insertion**
  Il faut que :
  - La date de début d'un entretien soit avant celle de fin.
  - La date de début et de fin soit des dates futures.
  - La note soit entre 0 et 20.
  - L'horaire d'entretien soit entre 8h et 18h.
  - Un candidat ou un jury n'ait pas d'entretiens qui se chevauchent.
  - Une salle n'ait pas d'entretiens qui se chevauchent.
- **Avant l'insertion**
  On procède aux mêmes vérifications qu'avant insertion.

Table Fait_Partie_De

- **Avant l'insertion**
  On vérifie que l'utilisateur inseré en tant que jury ne soit pas un candidat ou étudiant. Le nombre d'utilisateurs qui consitue un jury ne doit pas dépasser 5. Un jurée n'appartient qu'à un jury.
- **Avant la mise à jour**
  On vérifie toujours que l'utilisateur jurée mis à jour ne soit pas toujours pas un candidat ou un étudiant. Le nombre de personne dans un jury ne doit toujours pas dépasser 5.

Table Hist_Statuts

- **Avant l'insertion**
  - On vérifie que l'utilisateur ayant effectué le changement de statut possède le rôle "Gestionnaire de concours" ou "Scolarité".
  - Un candidat doit changer de statut dans l'ordre (inscrit -> admissible -> admis), il ne peut pas sauter une étape.

Table Jury

Il n'y a pas de contraintes d'intégrité.

Table Possede

- **Avant l'insertion**
  On vérifie si l'utilisateur possède déjà le rôle candidat ou étudiant : il ne peut pas obtenir un autre rôle.

Table Roles

Il n'y a pas de contraintes d'intégrité.

Table Salles

Il n'y a pas de contraintes d'intégrité.

Table Utilisateurs

- **Avant l'insertion**
  On vérifie que le courriel soit d'une forme correcte.
- **Avant la mise à jour**
  On vérifie que le courriel soit d'une forme correcte.

Fonctions

Nous nous sommes servis de diverses fonctions dans nos procédures. Les voici.

- **estRole (idUtilisateur, nomRole)**
  Cette fonction retourne un booléen indiquant si l'utilisateur correspondant à l'identifiant idUtilisateur possède bien le rôle correspondant à nomRole. Pour cela, la fonction utilise la vue UtilisateursEtRoles.
- **idRoleParNomRole (nomRole)**
  Cette fonction retourne l'identifiant du rôle correspondant à nomRole. Pour cela, elle vérifie d'abord que le rôle nomRole existe bien dans la table Roles, puis elle retourne son identifiant.
- **idSalleParNomSalle (nomSalle)**
  Cette fonction retourne l'identifiant de la salle correspondant à nomSalle. Pour cela, elle vérifie d'abord que la salle nomSalle existe bien dans la table Salles, puis elle retourne son identifiant.
- **idUtilisateurParCourriel (courrielUtilisateur)**
  Cette fonction retourne l'identifiant de l'utilisateur ayant pour courriel courrielUtilisateur. Pour cela, elle vérifie d'abord que le courriel existe bien dans la table Utilisateurs, puis elle retourne l'identifiant de l'utilisateur correspondant.
- **randString (taille)**
  Cette fonction retourne une chaine de caractères aléatoires de longueur taille.

Procédures stockées

Les procédures sont nécessaires lors de l'utilisation de la base via l'application. Généralement, on a besoin de fonction d'affichage, d'ajout, de mise à jour et de suppression. Voici un panel de fonctions essentielles.

Utilitaire

- **raiseError (msg)**
  Cette procédure affiche le message passé en argument lorsqu'une erreur est repérée dans la procédure ou la fonction qui utilise raiseError.

Procédures d'affichage

- **afficheCandidats ()**
  Cette procédure affiche les données de l'ensemble des candidats présents dans la base de donnée. Elle affiche donc l'identifiant du candidat ainsi que quatre booléens qui correspondent à la présence du CV, de la lettre de motivation et de deux relevés de notes du candidat (0 si les documents sont absents, 1 s'ils sont présents).
- **afficheCandidatsParStatut (statutCandidat)**
  Cette procédure prend en entrée un statut de candidat parmi 'inscrit', 'admissible, 'admis', 'refuse'

ou 'integre'. Elle affiche alors les données de tous les candidats ayant eu ce statut à un moment de la procédure.

- **afficheUtilisateurs()**
  Cette procédure affiche les données de tous les utilisateurs de la base de données, à savoir l'identifiant, le nom, le prénom, le courriel, le mot de passe haché et le sel de chaque utilisateur.
- **afficheUtilisateursEtRoles()**
  Cette procédure affiche l'identifiant, le nom, le prénom, le courriel et le nom du rôle de chaque utilisateur de la base de données.
- **afficheUtilisateursSelonRole()**
  Cette procédure prend en entrée le nom d'un rôle de la table Roles et affiche les données de tous les utilisateurs de la base de données ayant ce rôle.
- **afficheStatutSelonCandidat(courriel)**
  Cette procédure permet d'afficher l'ensemble des statuts qu'a eu le candidat ainsi que les différentes informations de ces statuts, à savoir la date de changement et l'identifiant de la personne ayant fait le changement ainsi que le nom et le prénom du candidat que nous avons pu récupéré grâce à une jointure avec la table Candidats puis avec la table Utilisateurs.
- **afficheEntretienSelonCandidat(courriel)**
  Cette procédure permet l'affichage des entretiens à venir du candidat dont on a indiqué le courriel en argument. En effectuant des jointures entre les tables Entretiens, Candidats, Salles et Utilisateur, cette procédure affiche l'identifiant, le nom et le prénim du candidat, l'identifiant du jury qui va l'évaluer, les dates et heures de début et de fin de l'entretien et la salle où aura lieu l'entretien.
- **afficheEntretienSelonJury(idJury)**
  Cette procédure affiche les entretiens à venir du jury dont on a indiqué l'identifiant en argument. En effectuant des jointures entre les tables Candidats, Entretiens, Salles et Utilisateurs, cette procédure affiche l'identifiant du jury, l'identifiant, le nom et le prénom du candidat, la date et l'heure de début et de fin de l'entretien ainsi que la salle dans laquelle se déroulera l'entretien.

Procédures d'ajout

- **ajoutCandidat (courriel)**
  Cette procédure permet l'ajout d'un candidat à partir de son courriel lorsque ce candidat est dans la table Utilisateurs. Elle prend en entrée le courriel d'un candidat puis elle récupère l'identifiant correspondant au courriel grâce à la fonction idUtilisateurParCourriel. Enfin, elle insère le bon utilisateur dans la table Candidats.
- **ajoutEntretien (courriel, idJury, nomSalle, dateDebut, dateFin)**
  Cette procédure permet la création d'un nouvel entretien à partir d'un courriel de candidat, d'un identifiant de jury, d'un nom de salle, d'une heure de début et d'une heure de fin. On utilise les fonctions idUtilisateurParCourriel et idSalleParNom pour récupérer les identifiants du candidat et de la salle afin d'insérer ces données dans la table Entretiens.
- **ajoutHistStatut (courrielCandidat, courrielGestionnaire, nouveauStatut)**
  Cette procédure permet l'ajout d'un nouveau statut de candidat dans la table Hist_Statuts repré-

sentant l'historique des statuts des candidats à partir du courriel du candidat, du courriel du gestionnaire qui ajoute le statut du candidat et le nouveau statut du candidat.
On utilise alors la fonction idUtilisateurParCourriel pour récupérer les identifiants du candidat et du gestionnaire de concours concernés afin d'insérer ces données dans la table Hist_Statuts en prenant pour date, la date et l'heure actuelle.

- **ajoutJury ()**
  Cette procédure permet d'ajouter un jury dans la table Jury. Le jury ajouté n'aura pas de membre.
- **ajoutRole (nomRole)**
  Cette procédure permet d'ajouter un role dans la table Roles à partir de son nom.
- **ajoutRoleToUtilisateur (courriel, nomRole)**
  Cette procédure permet d'ajouter un role à un utilisateur à partir de son courriel et du nom du rôle. On utilise ensuite les fonctions idUtilisateurParCourriel et idRoleParNomRole pour récupérer les identfiants de l'utilisateur et du rôle que l'on veut lui attribuer. On insère ensuite ces données dans la table Possede
- **ajoutSalle (nomSalle)**
  Cette procédure permet d'ajouter une salle dans la table Salles en prenant en entrée le nom de la salle.
- **ajoutUtilisateur (nom, prenom, courriel, mdp)**
  Cette procédure permet d'ajouter un utilisateur dans la table Utilisateurs en prenant en entrée le nom, le prénom, le courriel et le mot de passe en clair de l'utilisateur.
  On récupère crée le sel grâce à la fonction RandString. On hache ensuite le mot de passe grâce à la fonction md5, après l'avoir concaténé au sel avec la fonction CONCAT. Enfin, on ajoute ensuite ces données dans la table Utilisateurs.
- **ajoutUtilisateurToJury (courriel, idJury)**
  Cette procédure permet d'ajouter un utilisateur à un jury.
  On utilise la fonction idUtilisateurParCourriel pour récupérer l'identifiant de l'utilisateur que l'on veut ajouter au jury.
  On insère ensuite ces données dans la table Fait_Partie_De.

Procédures de modification

- **changerDatesHeuresEntretien (ancienneDateHeureDebut, nom_Salle, nouvelleDateHeureDebut, nouvelleDateHeureFin)**
  Cette procédure permet de modifier la date et l'heure de début et de fin d'un entretien en passant en paramètres l'ancienne date de début et la salle de l'entretien que l'on veut modifier.
- **changerMdpUtilisateur (courriel, mdp)**
  Cette procédure permet de modifier le mot de passe d'un utilisateur en prenant en entrée son courriel et le nouveau mot de passe qu'il veut enregistrer.
  On récupère le sel du précédent mot de passe afin de le réutiliser pour le nouveau mot de passe. On hache ensuite le nouveau mot de passe grâce à la fonction md5, après l'avoir concaténé au sel récupéré avec la fonction CONCAT. Enfin, on modifie le mot de passe dans la table Utilisateurs.

- **changerNomUtilisateur (mail, nouveauNomUtilisateur)**
  Cette procédure permet de modifier le nom de famille d'un utilisateur en prenant en entrée le mail et le nouveau nom de l'utilisateur.
- **changerPrenomUtilisateur (mail, nouveauPrenomUtilisateur)**
  Cette procédure permet de modifier le prénom d'un utilisateur en prenant en entrée le mail et le nouveau prénom de l'utilisateur.
- **changerNomPrenomUtilisateur (mail, nouveauNomUtilisateur, nouveauPrenomUtilisateur)**
  Cette procédure permet de modifier le nom et le prénom d'un utilisateur en prenant en entrée le mail, le nouveau nom et le nouveau prénom de l'utilisateur. Pour cela, nous avons utilisé les fonctions changerNomUtilisateur et changerPrenomUtilisateur.
- **changerNomRole (idRole, nouveauNomRole)**
  Cette procédure permet de modifier le nom d'un rôle en fonction de son identifiant.
- **changerNomSalle (idSalle, nouveauNomSalle)**
  Cette procédure permet de modifier le nom d'une salle en fonction de son identifiant.
- **changerNoteEntretien (dateHeure, nomSalle, nouvelleNote)**
  Cette procédure permet de modifier la note reçue par un candidat lors d'un entretien. Elle prend en arguments la date et l'heure de l'entretien concerné ainsi que le nom de la salle où se déroulait l'entretien et la nouvelle note que l'on veut attribuer au candidat.
  On utilise la fonction idSalleParNomSalle pour retrouver l'identifiant de la salle grâce à son nom.
  On modifie alors ces données dans la table Entretiens.
- **changerSalleEntretien (dateHeure, nomSalle, nouvelleSalle)**
  Cette procédure permet de modifier la salle d'un entretien. Elle prend en arguments la date et l'heure de l'entretien concerné ainsi que le nom de la salle actuelle et le nom de la nouvelle salle.
  On utilise alors la fonction idSalleParNomSalle pour récupérer les identifiants correspondant aux noms de l'ancienne et la nouvelle salle.
  On modifie ensuite ces données dans la table Entretiens.
- **changerUtilisateurDeJury (courriel, ancienIdJury, nouvelIdJury)**
  Cette procédure permet de changer un utilisateur de jury. Elle récupère l'identifiant de l'utilisateur grâce à la fonction idUtilisateurParCourriel avec le courriel de l'utilisateur comme argument. On modifie ensuite la table Fait_Partie_De en remplaçant ancienIdJury par nouvelIdJury.
- **changerUtilisateurDeRole (courriel, ancienRole, nouveauRole)**
  Cette procédure permet de changer un utilisateur de rôle. Elle récupère l'identifiant de l'utilisateur grâce à la fonction idUtilisateurParCourriel avec le courriel de l'utilisateur comme argument.
  Elle utilise ensuite la fonction idRoleParNomRole pour récupérer les identifiants de l'ancien rôle et du nouveau rôle.
  On modifie ensuite la table Possede en remplaçant l'identifiant de ancienRole par l'identifiant de nouveau rôle.
- **changerDocument(courriel, cv, lm, cni, releve1, releve2)**
  Cette procédure permet de modifier le statut des pièces du dossier d'un candidat. Elle prend en entrée le courriel du candidat dont on veut modifier le dossier ainsi que cinq booléens représentant

la nouvelle situation (0 s'ils sont absents, 1 sinon) des cinq pièces obligatoires à l'inscription : le CV, la lettre de motivation, la copie de la carte d'identité et deux relevés de notes.
On récupère ensuite l'identifiant du candidat grâce à la fonction idUtilisateurParCourriel, puis on modifie ces données dans la table Candidat.

Procédures de suppression

- **supprimerCandidat (courriel)**
  Cette procédure supprime un candidat de la table Candidat. Elle récupère l'identifiant du cnadidat à supprimer grâce à la fonction idUtilisateurParCourriel.
  Après avoir vérifié que l'utilisateur sélectionné est bien un candidat, on le supprime de la table Candidat.
- **supprimerEntretien (dateHeure, nomSalle)**
  Cette procédure supprime un entretien - défini par sa date et son heure ainsi que par sa salle - de la table Entretien en récupérant l'identifiant de la salle grâce à la fonction idSalleParRole.
- **supprimerHistStatuts (courriel)**
  Cette procédure supprime l'historique des statuts d'un candidat de la table Hist-Statuts. Ce candidat est indentifié grâce à son courriel que l'on entre en argument de la fonction. On trouve ensuite son identifiant grâce à la fonction idUtilisateurParCourriel.
- **supprimerJury (numJury)**
  Cette procédure supprime un jury identifié par son numéro de jury de la table Jury.
- **supprimerRole (nomRole)**
  Cette procédure supprime un rôle identifié par son nom de la table Roles.
- **supprimerRoleUtilisateur (courriel, nomRole)**
  Cette procédure supprime l'attribution d'un rôle identifié par son nom à un utilisateur identifié par nom courriel.
  On trouve l'identifiant de l'utilisateur dont on doit supprimer le rôle grâce à la procédure idUtilisateurParCourriel. On trouve ensuite l'identifiant du rôle grâce à la procédure idRoleParNomRole.
  Enfin, on effectue la suppression de ces données de la table Possede.
- **supprimerSalle (nomSalle)**
  Cette procédure supprime une salle identifiée par son nom de la table Salles.
- **supprimerUtilisateur (courriel)**
  Cette procédure supprime un utilisateur identifié par son courriel de la table Utilisateurs.
- **supprimerUtilisateurDeJury (courriel, idJury)**
  Cette procédure supprime l'attibution d'un utilisateur identifié par son courriel à un jury identifié par son identifiant.
  On récupère l'identifiant de l'utilisateur à partir de son courriel.
  On supprime ensuite ces données de la table Fait_Partie_De.

Génération de données

Nous avons utilisé un script python pour générer un set de données de base. Cela nous permet de créer des utilisateurs factices afin de remplir les tables Utilisateurs et Candidats et faciliter les tests par la suite. Les autres tables ont quant à elles été remplies à la main pour avoir une cohérence des données.

Tests mis en place

Pour les tests nous avons choisi de proposer deux scripts différents. Le premier permettant de vérifier que la base de données et les fonctions mises en place permettent de répondre aux fonctionnalités énoncées précédemment. Il suffit de suivre pas à pas le scénario du fichier. Un deuxième fichier script a pour but de montrer l'efficacité des triggers afin de protéger l'intégrité de la base de données.

Ces deux fichiers sont présents dans le fichier Workbench chacun étant nommés respectivement : tests_fonctionnalités.sql et tests_triggers.sql

Conclusion

Nous avons mis en place une base de données fonctionnelle et satisfaisante pour continuer notre développement de la composante *Recrutement des élèves*. Lors de la modélisation, nos choix structurels ont été pensés pour laisser le plus de liberté même si certaines limites existent encore. Lors de la réalisation, le remplissage des tables nous a permis de bien nous rendre compte des défauts de conception, que nous avons pu corriger. De plus, nous avons pu travailler efficacement car le test des procédures était plus concret.