UE-L204 MINI-PROJET

Rapport intermédiaire

Sylvain *Chambon*,
Jade *Faroux*,
Jeanne *Salvadori*,
Zoé *Van De Moortele*

4 décembre 2025

1 La méthode de travail

1.1 Outils utilisés

- Teams pour la communication et le partage de fichiers ;
- GitHub pour la mise en commun du code sur un dépôt et pouvoir mieux gérer les modifications via le système de branche ;
- Forum de groupe de l'UE : communication et retour sur l'avancement du projet ;
- Word pour une rédaction commune et LaTeXpour la finalisation des rapports ;
- un système d'intelligence artificielle générationnelle (Gemini) pour générer les entrées dans les tables.

1.2 Étude préliminaire

1.3 Scénario

Créer un site universitaire avec différents niveaux d'accès : étudiant et professeur, éventuellement un administrateur.

Les professeurs peuvent :

- Créer, modifier, supprimer des cours.
- Voir la liste d'étudiant inscrit.
- Gérer les inscriptions (accepter/refuser les étudiants dans leurs cours).
- Modifier leur profil (informations personnelles).

Les étudiants peuvent :

- Consulter l'ensemble des cours disponibles.
- Rechercher des cours (nom, professeur, domaine).
- S'inscrire à des cours.
- Modifier leur profil (informations personnelles).

Si assez de temps, mettre en place un système de note (prof donne des notes et les étudiants peuvent y accéder), et d'emploi du temps.

1.4 Objectifs adaptés à notre scénario

1.4.1 Page de connexion

- Formulaire d'entrée où l'utilisateur entre son login et mot de passe.
- Système vérifie si cet utilisateur existe dans la BDD.
- Le mot de passe doit être crypté.

- Si correct, accès au portail étudiant ou professeur.
- Si incorrect, message d'erreur.
- Changement de mot de passe lors de la première connexion (sécurité).

1.4.2 Page de recherche

- Champ de recherche de cours (par nom, professeur ou domaine).
- Doit interroger la BDD avec les critères de recherche.
- Afficher les résultats sous forme de tableau, ou message d'erreur si aucun résultat.

1.4.3 Page d'ajout de contenu

- Les professeurs ont une page dédiée pour ajouter, modifier ou supprimer leur cours.
- Les professeurs et étudiants peuvent modifier leurs informations personnelles (mail, tel, mot de passe...).
- Toutes ces modifications doivent être enregistrées dans la BDD.

1.4.4 Vérification des données

Avant d'enregistrer quoi que ce soit dans la BDD, tests de sécurité à effectuer systématiquement :

- Avant d'enregistrer quoi que ce soit dans la BDD, tests de sécurité à effectuer systématiquement :
- Vérifier que les champs obligatoires soient remplis
- Les formats des entrées doivent être corrects
- Pas de caractères dangereux (balise html)
- Messages d'erreur si une de ces vérifications est non conforme

2 Avancement du projet

2.1 Base de données

Nous avons modélisé notre base de données dans diagrams.net. Nous avons pensé à définir deux utilisateurs pour notre base de données : l'enseignant et l'étudiant.

En structurant notre base, nous nous sommes aperçu que ces deux entités partageaient beaucoup d'attributs donc nous avons préférer définir l'entité plus générale d'`utilisateur` et définir les entités `enseignant` et `etudiant` comme des héritages de l'entité `utilisateur` avec certains attributs spécifiques.

Dans cette base de données, nous aurons évidemment aussi une entité `cours`.

Nous aurons enfin des tables de liaison :

- une table permettant d'associer des enseignants avec des cours (entité `enseigne`);
- un autre permttant d'associer des étudiants avec des cours (entité `etudiant`);
- une dernière permettant de définir des contraintes sur les inscriptions dans certains cours (entité `prerequis`).

2.1.1 Entité *utilisateur*

L'entité `utilisateur` aura les attributs (voir requête nº 1) :

- `login` : identifiant de connexion;
- `mot_de_passe` : mot de passe pour la connextion : il sera encrypté par l'application (ce n'est pas de la responsabilité de la BBD);
- `mot_de_passe_provisoire` : booléen servant de drapeau pour savoir si le mot de passe initial a été changé ou pas;
- `nom`, `prenom`, `email` : pour renseigner des élément lié à la personne et son contact;
- `role` : trois rôles sont défini ici pour pouvoir tester les droits relatifs à chaque utilisateur de la BDD;
- `date_creation` pour stocker la date à laquelle l'utilisateur a été saisi dans la BDD;
- `actif` drapeau pour connaître si la personne est encore en activité dans la faculté.

Requête SQL nº 1 : Création de la table `utilisateur`

```sql
CREATE TABLE utilisateur (
  id INT PRIMARY KEY AUTO_INCREMENT,
  login VARCHAR(255) UNIQUE NOT NULL,
  mot_de_passe VARCHAR(255) NOT NULL,
  mot_de_passe_provisoire BOOLEAN DEFAULT TRUE,
  nom VARCHAR(100) NOT NULL,
  prenom VARCHAR(100) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  role ENUM('enseignant', 'etudiant', 'admin') NOT NULL,
  date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  actif BOOLEAN DEFAULT TRUE
);
```

On se rend compte que tous ces attributs serons partagés à la fois par les enseignants et par les étudiants.

Un ajout d'un utilisateur dans la BDD peut donc se déclarer ainsi (voir exemple de requête nº 2)

Requête SQL nº 2 : Création d'un utilisateur dans la table

```
(1, 'turing', '<mot-de-passe>', 'Turing', 'Alan', 'alan.turing@univ.fr', '
    enseignant'),
```

2.1.2 Entité *enseignant*

L'entité enseignant hérite de de l'entité utilisateur avec quelques spécificités :

- bureau : localisation de la salle de travail de l'enseignant ;
- telephone : numéro ;
- specialite : domaine d'expertise ;
- statut : sous quel titre l'enseignant a-t-il été embauché.

On constate la présence d'une clé étrangère afin de lier l'entité enseignant avec une entité utilisateur existante (héritage).

Requête SQL nº 3 : Création de la table enseignant

```
CREATE TABLE enseignant (
  id INT PRIMARY KEY AUTO_INCREMENT,
  utilisateur_id INT UNIQUE NOT NULL,
  bureau VARCHAR(50),
  telephone VARCHAR(20),
  specialite VARCHAR(255),
  statut ENUM('titulaire', 'vacataire', 'contractuel') DEFAULT 'titulaire',
  FOREIGN KEY (utilisateur_id) REFERENCES utilisateur(id) ON DELETE CASCADE
);
```

On ajoutera un enseignant ainsi dans la base (voir exemple de requête nº 4)

Requête SQL nº 4 : Création d'un enseignant dans la table

```
INSERT INTO enseignant (utilisateur_id, bureau, telephone, specialite, statut)
    VALUES
(1, 'B101', '0102030401', 'Intelligence Artificielle', 'titulaire');
```

2.1.3 Entité *etudiant*

L'entité etudiant hérite de de l'entité utilisateur avec quelques spécificités :

- le classique numero_etudiant, comme référence nationale ;
- niveau qui référence si l'étudiant est en licence ou master et en quelle année ;
- date_inscription qui peut-être de la date_creation si l'étudiant n'a pas validé ses frais de scolarité par exemple.

On constate ici aussi la présence d'une clé étrangère afin de lier l'entité étudiant avec une entité utilisateur existante (héritage, comme pour l'entité enseignant).

Requête SQL nº 5 : Création de la table `etudiant`

```sql
CREATE TABLE etudiant (
  id INT PRIMARY KEY AUTO_INCREMENT,
  utilisateur_id INT UNIQUE NOT NULL,
  numero_etudiant VARCHAR(20) UNIQUE NOT NULL,
  niveau ENUM('L1', 'L2', 'L3', 'M1', 'M2') NOT NULL,
  date_inscription DATE NOT NULL,
  FOREIGN KEY (utilisateur_id) REFERENCES utilisateur(id) ON DELETE CASCADE
);
```

Pour ajouter un étudiant dans la base, on pourra procéder ainsi (voir exemple de requête nº 6)

Requête SQL nº 6 : Création d'un étudiant dans la table

```sql
INSERT INTO etudiant (utilisateur_id, numero_etudiant, niveau, date_inscription)
    VALUES
(12, '20250001', 'L3', '2024-09-01');
```

2.1.4 Entité *cours*

L'entité cours a les attributs suivants :

- code joue le rôle d'identifiant visuel et sera pratique pour les recherches ;
- nom intitulé du cours ;
- credits pour garder le nombre de crédits ECTS ;
- description pour donner le détail sdu contenu du cours ou un syllabus ;
- capacite_max pour gérer le nombre d'étudiants qui peuvent s'inscrire ;
- annee_universitaire indique quand le cours est proposé ;
- actif en un drapeau booléen permettant de savoir si le cours est proposé en enseignement ou pas.

Requête SQL nº 7 : Création de la table `cours`

```sql
CREATE TABLE cours (
  id INT PRIMARY KEY AUTO_INCREMENT,
  code VARCHAR(20) UNIQUE NOT NULL,
  nom VARCHAR(255) NOT NULL,
  credits INT NOT NULL CHECK (credits > 0),
  description TEXT,
  capacite_max INT DEFAULT 30,
  annee_universitaire VARCHAR(9) NOT NULL, -- "2025-2026"
  actif BOOLEAN DEFAULT TRUE
);
```

La création d'un cours peut s'effectuer à l'aide de la requête suivante :

Requête SQL n° 8 : Création d'un cours dans la table

```sql
INSERT INTO cours (code, nom, credits, description, annee_universitaire) VALUES
('INFO-L101', 'Introduction à l''Algorithmique', 6, 'Logique, pseudo-code,
    variables et boucles', '2025-2026');
```

2.1.5 Tables de liaison

À ce stade, nous n'avons généré dans la base qu'une seule table de liaison : celles concernant les prérequis.

Pour les autres tables de liaison (`enseigne` et `inscription`), nous pensons qu'il faudra plutôt les faire depuis l'interface.

Un prérequis pour un cours fonctionne ainsi : on réunit deux ID de cours, la première référençant un Cours A, la deuxièmes référençant un Cours B nécessaire pour suivre le Cours A. Ainsi, cette relation est bien une relation *many-to-many* :

- le Cours A ayant besoin de plusieurs cours pour être suivi ;
- le Cours A pouvant être nécessaire à d'autres cours.

Voici comment nous avons donc défini cette table :

Requête SQL n° 9 : Table de liaison `prerequis`

```sql
CREATE TABLE prerequis (
  cours_id INT NOT NULL,
  prerequis_cours_id INT NOT NULL,
  PRIMARY KEY (cours_id, prerequis_cours_id),
  FOREIGN KEY (cours_id) REFERENCES cours(id) ON DELETE CASCADE,
  FOREIGN KEY (prerequis_cours_id) REFERENCES cours(id) ON DELETE CASCADE,
  CHECK (cours_id ≠ prerequis_cours_id)
);
```

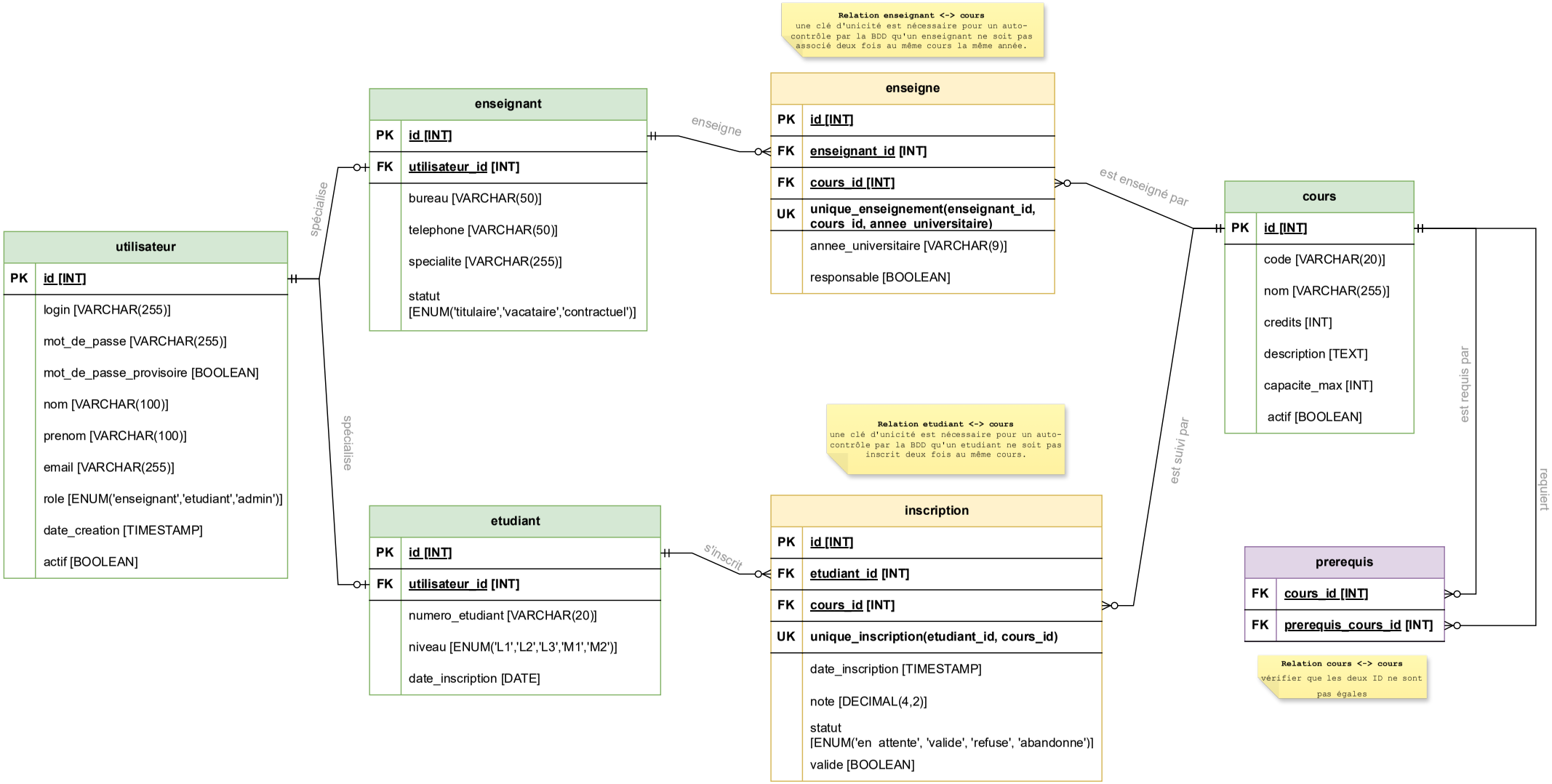
La dernière ligne n° 7 permet d'interdire un auto-référencement.

Pour déclarer une telle relation, nous pouvons faire (voir exemple de requête ci-dessous n° 10) :

Requête SQL n° 10 : Création d'une relation de prérequis

```sql
INSERT INTO prerequis (cours_id, prerequis_cours_id) VALUES
((SELECT id FROM cours WHERE code='INFO-L201'), (SELECT id FROM cours WHERE code
    ='INFO-L101'));
```

On remarque que plutôt que d'utiliser directement les ID des cours, on utilise l'attribut `code` sur une condition dans une clause qui va nous permettre de retrouver l'ID du cours. Cette méthode est plus robuste car si l'ID d'un cours venait à changer alors on ne perdrait pas la relation de prérequis.


Relation enseignant <-> cours
une clé d'unicité est nécessaire pour un auto-contrôle par la BDD qu'un enseignant ne soit pas associé deux fois au même cours la même année.
enseignant
PK id [INT]
FK utilisateur_id [INT]
bureau [VARCHAR(50)]
telephone [VARCHAR(50)]
specialite [VARCHAR(255)]
statut [ENUM('titulaire','vacataire','contractuel')]
enseigne
PK id [INT]
FK enseignant_id [INT]
FK cours_id [INT]
UK unique_enseignement(enseignant_id, cours_id, annee_universitaire)
annee_universitaire [VARCHAR(9)]
responsable [BOOLEAN]
est enseigné par
cours
PK id [INT]
code [VARCHAR(20)]
nom [VARCHAR(255)]
credits [INT]
description [TEXT]
capacite_max [INT]
actif [BOOLEAN]
est requis par
requiert
utilisateur
PK id [INT]
login [VARCHAR(255)]
mot_de_passe [VARCHAR(255)]
mot_de_passe_provisoire [BOOLEAN]
nom [VARCHAR(100)]
prenom [VARCHAR(100)]
email [VARCHAR(255)]
role [ENUM('enseignant','etudiant','admin')]
date_creation [TIMESTAMP]
actif [BOOLEAN]
spécialise
spécialise
Relation etudiant <-> cours
une clé d'unicité est nécessaire pour un auto-contrôle par la BDD qu'un etudiant ne soit pas inscrit deux fois au même cours.
etudiant
PK id [INT]
FK utilisateur_id [INT]
numero_etudiant [VARCHAR(20)]
niveau [ENUM('L1','L2','L3','M1','M2')]
date_inscription [DATE]
s'inscrit
inscription
PK id [INT]
FK etudiant_id [INT]
FK cours_id [INT]
UK unique_inscription(etudiant_id, cours_id)
date_inscription [TIMESTAMP]
note [DECIMAL(4,2)]
statut [ENUM('en attente', 'valide', 'refuse', 'abandonne')]
valide [BOOLEAN]
est suivi par
prerequis
FK cours_id [INT]
FK prerequis_cours_id [INT]
Relation cours <-> cours
vérifier que les deux ID ne sont pas égales

3 Difficultés rencontrées / pas encore résolues

4 Projections pour la 2e semaine