UNIVERSITÉ LAVAL

Projet de session:

Plate-forme d'apprentissage en ligne gratuite

Remise 15 avril 2025

Présenté Richard Khoury

Par

Équipe 62

Matricules	Noms
537 006 598	Miguel Dumetz
537 172 828	William Beaulieu
536 941 994	Dangar Othniel Djilouba
537 167 091	Saïkou Oumar Balde

Informations Importante

Dans le cadre de notre projet nous avons utilisé des migrations pour faciliter la collaboration sur notre base de données. Dans notre projet, il y a un fichier README.md qui explique en détail le fonctionnement de l'application. Nous avons choisi de conserver l'historique complet des versions de la base de données. Toutefois, lors de l'exécution des migrations, c'est la version 008 qui sera appliquée.

Introduction

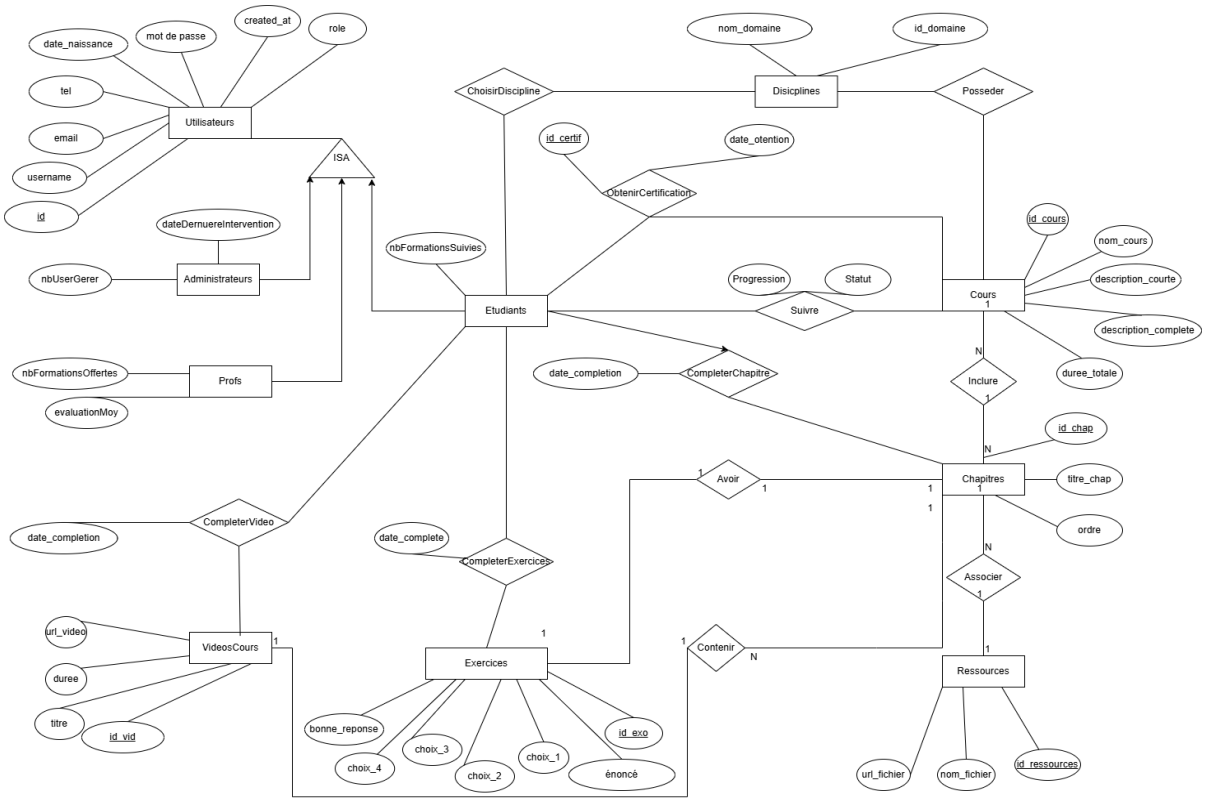
L'application que nous avons décidé de développer est une plateforme d'apprentissage en ligne qui permet à des étudiants de s'inscrire à des cours divers et de pouvoir à l'aide de vidéos apprendre la matière pour ensuite répondre à des quiz interactifs. Les utilisateurs peuvent se créer un compte pour accéder aux cours et ils ont aussi accès à un mode sombre dans l'application s'ils le souhaitent. Plusieurs fonctionnalités d'un tel projet sont en jeux et vont impacter de multiples aspects du système. Le rapport en parlera et parlera aussi plus en détail de comment la base de données contenant les informations nécessaires au projet fonctionne.

Partie 1. Base de données

1.1. Modélisation des données

Introduction du modèle entité-relation

Le modèle entité-relation présenté ci-dessus illustre la structure conceptuelle d'une plateforme d'apprentissage en ligne. Il met en évidence les principales entités comme Utilisateurs, Étudiants, Profs, Cours, Chapitres, Ressources, Exercices, et VidéosCours, ainsi que leurs relations. Le modèle prend aussi en compte la spécialisation des utilisateurs (ISA) en Administrateurs, Étudiants et Profs, avec des attributs spécifiques à chaque rôle. Il reflète les interactions entre étudiants et contenus pédagogiques, telles que suivre un cours, compléter un chapitre ou une vidéo, ou répondre à un exercice. Des entités comme Certifications et Disciplines permettent de suivre la progression et les compétences acquises. Le modèle facilite ainsi la conception d'une base de données relationnelle bien structurée, normalisée, évolutive et adaptée à une application d'e-learning complète.



1.2. Création de la base de données

Cette section décrit la base de données conçue pour le projet et son fonctionnement. Ces données ont été tirées des données réelles sur YouTube pour les différents Cours.

1.2.1 Structure des Entités

1.2.1.1. Utilisateurs et Étudiants

- La table users contient les informations de base des utilisateurs (nom, email, mot de passe, etc.).
- La table étudiants est une spécialisation de users qui stocke uniquement les étudiants inscrits.
- Professeur et Administrateur également deux types d'utilisateurs qui auront accès à la plateforme : l'un pour ajouter des cours et suivre ses étudiants puis l'autre pour administrer le site web en tant que propriétaire. Présentement, la logique de ces tables n'a pas encore été fait, nous sommes encore sur l'étudiant qui est plus important. Même si les rôles de professeur et d'administrateur ne soient pas encore utilisés dans la version actuelle de l'application, nous avons tout de même choisi de les implémenter dès maintenant. Cette décision vise à faciliter une éventuelle expansion de notre plateforme

1.2.1.2 Discipline et Cours

- Discipline représente les domaines de formations proposées (ex: Bases de données, sécurité).
- Cours regroupe les formations disponibles dans chaque discipline.

- Chapitres liste les chapitres de chaque cours.
- VideosCours et Exercice sont liés aux chapitres pour structurer les leçons.

1.2.1.3 Suivre

- Suivre stocke la liste des étudiants inscrits à chaque cours et leurs progressions dans chacun des cours.
- Videos_complete et Exercices_complete suivent l'achèvement des vidéos et exercices.
- Chapitres_complete regroupe les chapitres terminés par un étudiant.
- Cours_complete indique lorsqu'un étudiant termine un cours.

1.2.1.4. Certifications

- Certifications permet de stocker la liste d'étudiants ayant terminé un cours et leur délivre un certificat de formation. Elle est mise à jour lorsqu'un étudiant termine un cours.

1.2.3. Contraintes d'intégrité de la base de données

La base de données implémente plusieurs contraintes d'intégrité essentielles afin d'assurer la cohérence, la validité et la fiabilité des données manipulées par les utilisateurs et les déclencheurs automatiques. Ces contraintes sont répertoriées comme suit:

1. 2.3.1. Clés primaires:

Chaque table de la base dispose d'une clé primaire assurant l'unicité de ses enregistrements :
- users(id_user)
- discipline(id_d)
- cours(id_cours)
- chapitres(id_chap)
- videoscours(id_vid)
- exercice(id_exo)
- suivre(id_user, id_cours) — clé composée
- videos_complete(id_user, id_vid) — clé composée
- exercices_complete(id_user, id_exo) — clé composée
- chapitres_complete(id_user, id_chap) — clé composée
- cours_complete(id_user, id_cours) — clé composée
- certifications(id_user, id_cours) — clé composéé

1.2.3.2. Clés étrangères:

Des contraintes de clés étrangères ont été définies pour assurer l'intégrité référentielle entre les entités :
- cours(id_d) REFERENCES discipline(id_d)

- chapitres(id_cours) REFERENCES cours(id_cours)
- videoscours(id_chap) REFERENCES chapitres(id_chap)
- exercice(id_chap) REFERENCES chapitres(id_chap)
- suivre(id_user) REFERENCES users(id_user)
- suivre(id_cours) REFERENCES cours(id_cours)
- videos_complete(id_user) REFERENCES users(id_user)
- videos_complete(id_vid) REFERENCES videoscours(id_vid)
- exercices_complete(id_user) REFERENCES users(id_user)
- exercices_complete(id_exo) REFERENCES exercice(id_exo)
- chapitres_complete(id_user) REFERENCES users(id_user)
- chapitres_complete(id_chap) REFERENCES chapitres(id_chap)
- cours_complete(id_user) REFERENCES users(id_user)
- cours_complete(id_cours) REFERENCES cours(id_cours)
- certifications(id_user) REFERENCES users(id_user)
- certifications(id_cours) REFERENCES cours(id_cours)

1.2.3.3. Contraintes NOT NULL

Les colonnes suivantes sont définies comme obligatoires pour garantir la complétude des données :

- users.nom, users.email, users.password
- discipline.nom_discipline
- cours.nom_cours
- chapitres.titre_chap
- exercice.question, exercice.bonne_reponse
- videoscours.titre_vid, videoscours.lien

1.2.3.4. Contraintes UNIQUE

Certaines colonnes et combinaisons de colonnes sont définies comme uniques pour éviter les doublons :

- users.email (chaque utilisateur a un email unique)
- combinaison (id_user, id_cours) dans suivre
- combinaison (id_user, id_vid) dans videos_complete
- combinaison (id_user, id_exo) dans exercices_complete
- combinaison (id_user, id_chap) dans chapitres_complete
- combinaison (id_user, id_cours) dans cours_complete et certifications

Partie 2. Requêtes, routines et sécurité.

2.1. Création des requêtes et routines

Une grande importance a été placer sur la création des requêtes et routines. Lorsqu'elles sont bien implémentées, elles permettent d'optimiser la performance et la rapidité de l'application.

Dans cette étape du projet, nous avons créé toutes les requêtes nécessaires pour notre application-web. Pour mieux démontrer le fonctionnement et la formulation de ces requêtes, nous allons présenter deux d'entre elles. L'une de ces requêtes permet d'automatiser la complétion d'un cours. Ci-dessous se trouve la gâchette insert_into_cours_complete, qui illustre cette logique.

```
TRIGGER insert_into_cours_complete AFTER INSERT ON chapitres_complete FOR EACH ROW BEGIN DECLARE
course_id INT; DECLARE total_chapitres INT; DECLARE chapitres_termines INT;

SELECT id_cours INTO course_id
FROM chapitres
WHERE id_chap = NEW.id_chap;



SELECT COUNT(*) INTO total_chapitres
FROM chapitres
WHERE id_cours = course_id;

SELECT COUNT(*) INTO chapitres_termines
FROM chapitres_complete cc
JOIN chapitres c ON cc.id_chap = c.id_chap
WHERE cc.id_user = NEW.id_user AND c.id_cours = course_id;


IF chapitres_termines = total_chapitres THEN
   IF NOT EXISTS (
      SELECT 1 FROM cours_complete
      WHERE id_user = NEW.id_user AND id_cours = course_id
   ) THEN
      INSERT INTO cours_complete (id_user, id_cours)
      VALUES (NEW.id_user, course_id);
   END IF;


   DELETE FROM suivre
   WHERE id_user = NEW.id_user AND id_cours = course_id;
END IF;
```

Cette gâchette est exécutée chaque fois qu'un chapitre est insérer dans la table chapitres_completer afin de vérifier si tous les chapitres d'un cours ont été complétés. Lors de son exécution, la gâchette trouve l'id du cours présent. Ensuite, elle compte le nombre total de chapitre dans ce cours et le nombre de chapitre compléter. Avec ces informations, on peut vérifier si ces deux valeurs sont égales. Si c'est le cas, on peut conclure que le cours est terminé. Dans ce cas on ajoute le cours dans la table cours_completer puis on supprime le cours de la table suivre. Nous avons choisi cette approche pour sauver de l'espace occupé dans la table suivre. Si un cours compléter y restait, au fil du temps, il aurait beaucoup de tuple inutile dans cette table ce qui ralentirait les opérations de recherche. Ceci était juste un exemple d'une routine parmi plusieurs dans notre application. Pour la deuxième requête, se trouve un exemple d'une requête permettant de compléter un chapitre.

```
@progression_bp.route('/complete_chapitre', methods=['POST'])
```

```
def complete_chapitre():

data = request.get_json() id_user = data['id_user'] id_chap = data['id_chap'] id_cours = data['id_cours']

connection = get_db_connection()
cursor = connection.cursor()


cursor.execute("""
   INSERT INTO chapitres_complete (id_user, id_chap)
   SELECT %s, %s
   WHERE NOT EXISTS (
      SELECT 1 FROM chapitres_complete WHERE id_user = %s AND id_chap = %s
   )
""", (id_user, id_chap, id_user, id_chap))


cursor.execute("""
   SELECT COUNT(*) FROM chapitres WHERE id_cours = %s
""", (id_cours,))
total_chapitres = cursor.fetchone()[0]


cursor.execute("""
   SELECT COUNT(*) FROM chapitres_complete c
   JOIN chapitres ch ON c.id_chap = ch.id_chap
   WHERE c.id_user = %s AND ch.id_cours = %s
""", (id_user, id_cours))
completed = cursor.fetchone()[0]


if total_chapitres == 0:
   progression_percent = 0
else:
   progression_percent = round((completed / total_chapitres) * 100)
   progression_percent = min(progression_percent, 100)


cursor.execute("""
   UPDATE suivre SET progression = %s
   WHERE id_user = %s AND id_cours = %s
""", (progression_percent, id_user, id_cours))


connection.commit()
cursor.close()
connection.close()
return '', 204
```

Dans notre application, nous avons décidé que cette requête est appelée lorsqu'un utilisateur donne la bonne réponse à un exercice d'un chapitre. Cette requête ajoute non-seulement le chapitres dans la table chapitres compléter, mais elle également à jour la progression du cours. Elle fonctionne de la manière suivante. Elle commence par ajouter le chapitre à la table chapitre_complete. Après, elle trouve le nombre de chapitres total et le nombre de chapitre compléter. Pour ajuster la progression, on divise le nombre de chapitre compléter par le nombre total de chapitre puis on met à jour la progression actuelle avec cette valeur. Nous avons choisi cette approche car elle nous permet d'ajouter autant de chapitres que nécessaire à un cours, tout en garantissant que la progression affichée reste toujours précise et à jour.

2.2. Sécurité

Pour la sécurité, nous avons implémenter le hachage des mots de passes avec Werkzeug Security. Lors de la création de compte, on place dans la base de données une version hachées du mot de passe choisi. Ensuite, lors de la connexion, on vérifie le hash du mot-de-passe entrée pour confirmer la connexion. Alors on ne stock jamais la valeur actuelle du mot de passe pour empêcher le piratage des comptes.

Partie 3. Interface utilisateur

3.1. Fonctionnement

Exigences et impact sur les fonctionnalités:

1. Création et gestion de compte: Chaque utilisateur peut se créer un compte et s'y connecter. Ceci nécessite une bonne gestion de la sécurité des comptes utilisateurs.
2. Accéder aux cours: Chaque utilisateur peut s'inscrire aux cours et compléter les exercices associés (quiz et vidéos). Notre système doit donc pouvoir suivre la progression de l'utilisateur.
3. Afficher statistiques: Notre système doit pouvoir montrer aux utilisateurs combien de personnes sont inscrites pour chaque cours pour qu'ils sachent les cours les plus populaires.
4. Filtrage des cours: Notre système doit pouvoir filtrer les cours selon différents paramètres (longueur et type).
5. Mode sombre: Le système doit laisser l'utilisateur passer en mode sombre s'il le désire. On doit donc ajuster l'interface pour qu'elle soit lisible en mode clair et sombre.

3.2. Architecture

Responsabilités des niveaux du système:

1. Frontend:

   a) Interface: Présentation des cours, vidéos, exercices et page profile

   b) Gestion du mode sombre: Basculer du mode sombre au mode clair

   c) Création et gestion de compte: Formulaire d'inscription et de connexion

2. Backend:

   a) Filtres des cours: Options diverses pour filtrer les cours selon les préférences

   b) Statistiques des cours: Nombre d'inscription pour chaque cours et mise à jour en temps réel

   c) Vérification des informations de connexion avant d'envoyer dans la base donnée

   d) Gestion des cours: Traitement de toutes les données qui ont rapport aux cours à partir de la base de données

3. Base de données:

   a) Stockage des utilisateurs: Stockage dans la base de données des informations relatives aux utilisateurs

   b) Stockage des cours: Stockage de toutes les informations qui concernent les cours

   c) Stockage des cours suivis: Stockage des cours suivis par chaque utilisateur

Partie 4. Normalisation et Indexation

4.1. Normalisation de la BD

Dans le cadre de notre projet (plate-forme d'apprentissage en ligne gratuite), nous avons procédé à l'analyse et à la normalisation des différentes tables du système. L'objectif principal était de structurer les données de manière logique, cohérente et sans redondances, tout en respectant les différentes formes normales (1FN, 2FN, 3FN et FNBC).

Chaque table a été analysée individuellement afin de vérifier si elle respectait bien les règles de normalisation. Cela nous a permis d'identifier les dépendances fonctionnelles, de repérer d'éventuelles anomalies, et de proposer des ajustements lorsque c'était nécessaire. Par exemple, la table certifications violait la 3FN à cause d'une dépendance transitive, ce qui nous a amenés à retirer l'attribut *id_domaine*. De même, la table videosCours présentait une redondance à cause de l'attribut *id_cours*, qui pouvait être déduit via *id_chap*. Là encore, nous avons ajusté pour respecter la FNBC.

Un autre exemple marquant concerne la table exercice, qui contenait initialement des données répétitives dans plusieurs colonnes (notamment pour les choix de réponse). Pour respecter les formes normales, nous avons séparé cette table en deux : exercice (contenant l'énoncé) et choix_exercice (contenant les choix de réponse associés), ce qui a permis de mieux organiser les données et d'améliorer la flexibilité du système.

En somme, ce travail de normalisation nous a permis de concevoir une base de données plus propre, plus évolutive et plus facile à maintenir. Dans ce document, nous détaillons l'analyse de chaque table, en expliquant pour chacune d'elles si elle respecte les formes normales, et les modifications apportées si nécessaire.

1. Table users

- **1FN** : Valeurs atomiques, pas de répétitions.
- **2FN** : Clé primaire simple (id), dépendances totales.
- **3FN** : Aucune dépendance transitive.
- **FNBC** : id, username, email sont des clés candidates.

2. Table administrateur

- **1FN à FNBC** : Respect de toutes les formes normales grâce à la clé simple (id_admin) et l'absence de dépendances transitives.

3. Table professeur

- **1FN à FNBC** : Attributs atomiques, dépendance directe à la clé primaire (id_prof), pas de dépendances transitives.

4. Table etudiants

- **1FN à FNBC** : Données atomiques, dépendance directe à la clé (id_user), aucune redondance.

5. Table discipline

- **1FN à FNBC** : Structure simple (2 attributs), dépendance directe à id_domaine.

6. Table certifications

- **Violation 3FN ET FNBC** : id_domaine dépend transitivement de id_certif.
- **Solution** : Supprimer id_domaine et passer par une jointure via id_cours.

7. Table chapitres

- **1FN à FNBC** : Clé simple (id_chap), pas de dépendances entre colonnes non-clés.

8. Table chapitres_complete

- **1FN à FNBC** : Clé composée (id_user, id_chap), date_completion dépend de cette combinaison.

9. Table choixDiscipline

- **1FN à FNBC** : Clé composée (id_user, id_domaine), aucune autre colonne donc aucune dépendance partielle ou transitive.

10. Table cours

- **1FN à FNBC** : Tous les attributs dépendent directement de la clé primaire (id_cours), pas de redondance.

11. Table cours_complete

- **1FN à FNBC** : Clé composée (id_user, id_cours), dépendance directe de date_completion.

12. Table exercice

- **Problème initial** : Données répétitives.
- **Solution** : Séparer en deux tables (exercice, choix_exercice) pour respecter les 3FN + FNBC.

13. Table exercice_complete

- **1FN à FNBC** : Clé composée (id_user, id_exo), pas de dépendances non conformes.

14. Table ressources

- **1FN à FNBC** : Clé simple (id_ressource), dépendance directe sans redondance.

15. Table suivre

- **1FN à FNBC** : Clé composée (id_user, id_cours), attributs (statut, progression) dépendent de la clé.

16. Table video_complete

- **1FN à FNBC** : Clé composée (id_user, id_vid), dépendance fonctionnelle respectée.

17. Table videoCours

- **Problème** : Violation de la FNBC à cause de la redondance id_cours (transitif via id_chap).
- **Solution** : Supprimer id_cours pour éviter la redondance et respecter la FNBC.

Ce travail de normalisation nous a permis de mieux structurer notre base de données et d'éviter les problèmes liés aux redondances, anomalies ou incohérences. En appliquant les formes normales (1FN, 2FN, 3FN et FNBC), nous avons pu valider la qualité de la majorité des tables comme **users**, **cours**, **chapitres**, **discipline**, etc. Certaines tables, comme **certifications** ou **videosCours**, contenaient des dépendances transitives ou redondantes qu'on a corrigées pour respecter la 3FN et la FNBC. On a aussi réorganisé la table **exercice** en deux entités distinctes : **exercice** (l'énoncé) et **choix_exercice** (les choix de réponse), pour une meilleure flexibilité. Globalement, ce travail nous a permis de mieux comprendre les dépendances fonctionnelles et l'importance de la normalisation dans la conception d'une base de données bien optimisée et évolutive. Cependant nous avons appris cette matière après avoir déjà implémenter les requêtes nécessaires pour le fonctionnement de notre site web. Apporter des modifications aux tables problématiques afin de les normaliser aurait entraîné un retard dans l'avancement de notre projet. Malgré cela, nous avons quand même inclus dans notre remise une version corrigée et normalisée des tables concernées.

4.2. Indexation de la BD

Dans le cadre de notre système d'apprentissage, nous avons défini des index afin d'optimiser l'exécution des requêtes les plus fréquentes.

Le système repose sur des requêtes fréquentes portant sur les relations entre utilisateurs, cours, vidéos, exercices, chapitres et certificats. Sans index, ces requêtes deviendraient de plus en plus lentes à mesure que les données augmentent.

Table	Colonnes indexées	Justification
Suivre	Id_user, id_cours	Pour vérifier rapidement la progression d'un cours
Videos_complete	Id_user, id_vid	Pour valider une vidéo terminée par un user
Exercices_complete	Id_user, id_exo	Pour vérifier qu'un exercice a été complété
Chapitres_complete	Id_user, id_chap	Pour savoir si un chapitre a été entièrement validé
Cours_complete	Id_user, id_cours	Pour identifier les cours complétés
Certifications	Id_user, id_cours	Pour retrouver les certificats délivrés

Conclusion

En résumé, l'application devra offrir une plateforme simple et efficace pour l'apprentissage en ligne, tout en permettant une gestion claire des utilisateurs et des cours. La répartition des responsabilités entre le client, le serveur et la base de données assure une séparation claire des tâches et une meilleure évolutivité du système.