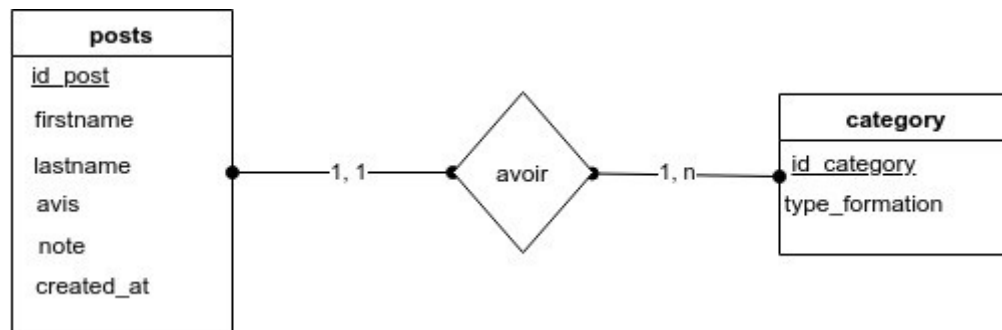




❖ Rapport de réalisation et de fonctionnement des solutions mises en place

I-Partie UML / MERISE :

La partie UML / MERISE consiste à concevoir le diagramme MCD du formulaire pour pouvoir établir la base de donnée MySQL. Le diagramme se présente comme suit après analyse :



Le diagramme MCD est constitué de 2 Entités bien distinctes relié par une association directe qui s'appelle « **avoir** »:

Caractéristiques de chaque Entité :

post :

Id_post qui est la clé primaire de la table,
firname,
lastname,
avis,
note,
created_at qui est la date de création du post

category :

Id_category qui est la clé primaire de la table,
type_formation

Cardinalités :

(1, n) => Une catégorie peut avoir un ou plusieurs post,

(1, 1) => Un post peut avoir une et une seule catégorie.

Conversion du diagramme MCD en MLD :

post (id_post, firstname, lastname, avis, note, created_at, #id_category)

category (id_category, type_formation)

II-Partie EER / MYSQL:

Création du diagramme EER sur MySQL Workbench :

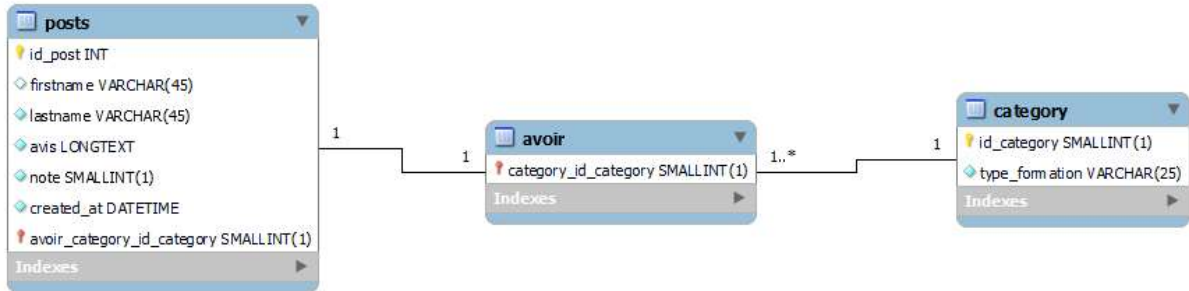


Figure 2: Diagramme EER

Création de la base de donnée “postavis”:

Veuillez charger le fichier “dump.sql” qui se trouve à la racine du dossier;

Explication sur la création de la base de données:

- ❖ Création de la table “**category**” en premier car la table “**post**” contient une clé étrangère qui empêche sa création

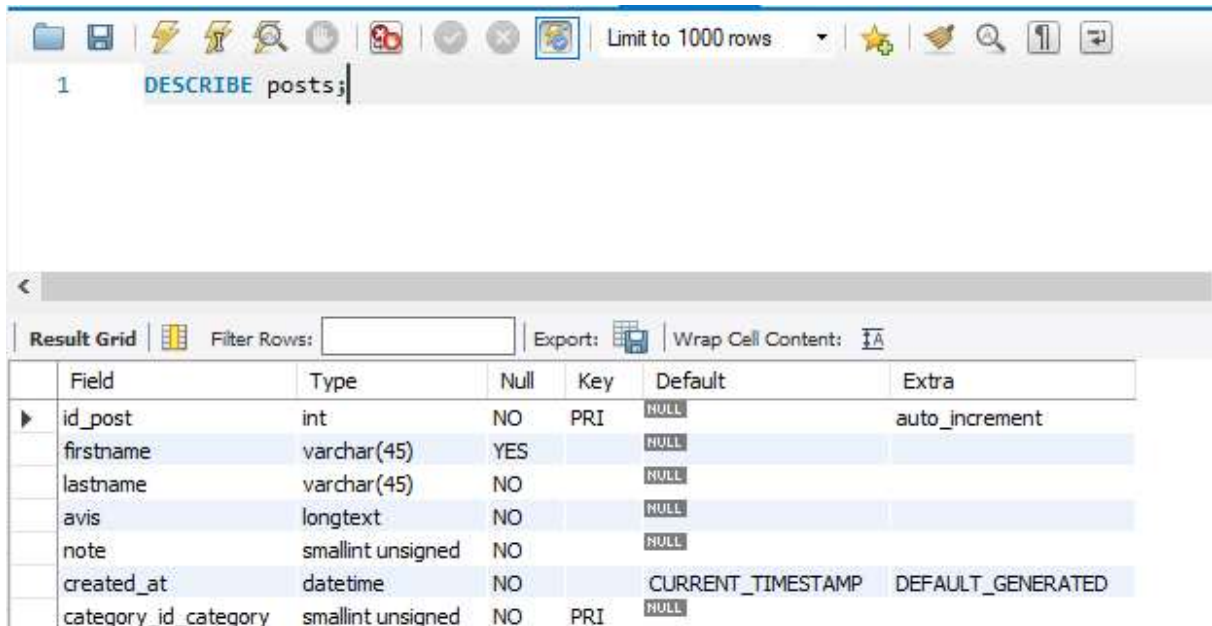
Structure de la table: DESCRIBE category;



Result Grid						
Filter Rows: <input type="text"/>						
Export: <input type="button" value="Export"/> Wrap Cell Contents: <input type="checkbox"/>						
	Field	Type	Null	Key	Default	Extra
▶	id_category	smallint unsigned	NO	PRI	NULL	
	type_formation	varchar(25)	NO		NULL	

❖ Création de la table “posts”

Structure de la table: DESCRIBE posts;



The screenshot shows a database management interface. At the top, there is a toolbar with various icons and a text input field containing the SQL command `DESCRIBE posts;`. Below the command, a "Result Grid" is displayed, showing the structure of the 'posts' table. The grid has columns for Field, Type, Null, Key, Default, and Extra. The rows represent the table's columns: id_post (int, NO, PRI, NULL, auto_increment), firstname (varchar(45), YES, NULL), lastname (varchar(45), NO, NULL), avis (longtext, NO, NULL), note (smallint unsigned, NO, NULL), created_at (datetime, NO, CURRENT_TIMESTAMP, DEFAULT_GENERATED), and category_id_category (smallint unsigned, NO, PRI, NULL).

Field	Type	Null	Key	Default	Extra
id_post	int	NO	PRI	NULL	auto_increment
firstname	varchar(45)	YES		NULL	
lastname	varchar(45)	NO		NULL	
avis	longtext	NO		NULL	
note	smallint unsigned	NO		NULL	
created_at	datetime	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
category_id_category	smallint unsigned	NO	PRI	NULL	

❖ Contrainte de clé étrangère :

En cas de modification ou de suppression d’une catégorie directement dans la base de données, la contrainte utilisée est la suppression ou modification en cascade

Remarque : dès fois, il se pourrait qu’un utilisateur ne possède pas de prénom alors j’en ai conclu que le champ « **firstname** » pourra être **null** par contre un utilisateur devrait toujours avoir un « **lastname** ».

Et aussi, j’ai rajouté un champ date de création (**created_at**) qui aura une valeur par défaut la date courante au moment de l’enregistrement du post

Insertion de donnée dans la base de données :

La table « **category** » est remplie au préalable à partir des données qui sont associés au champ du formulaire de la balise select avec l’identifiant *formation*, chaque type de formation est associée à une valeur d’identifiant *id_category* qui est la clé primaire de la table. Pour pouvoir associer les deux tables, c’est cette clé qui est repris en tant que clé étrangère de la table « **posts** » par l’option *value* du formulaire.

Des données ont été insérées à la table post pour pouvoir effectuer des tests sur le rendement visuel de la partie front, elles sont présentes dans la base elle-même.

III-Partie NODEJS:

📁 Correction des fichiers **HTML** et **CSS** pour un meilleur rendu et ajout de touches personnelles

📁 Création de l'API:

Création du point d'entrée App.js dans un répertoire;

Initialisation du projet (npm init) ;

Installation des modules nécessaires (npm list):

- Dépendances directes:
 - dotenv,
 - express,
 - ejs,
 - mysql,
 - express-session
- Dépendances de développement:
 - nodemon

Création des diverses routes du site dans le dossier ./routes/routes.js ;

Ajout des liens dynamiques sur la partie front.

Pour pouvoir créer les diverses routes, j'ai utilisé le module ejs de npm en utilisant la méthode `render()` donc j'ai dû convertir tous les fichiers **HTML** en **EJS** sauf la page `signup.html` (je ne sais pas à quoi elle sert) et en configurant `"app.set('view engine','ejs')"`, la raison est qu'elle permet une meilleure manipulation du DOM avec ses templates surtout lorsqu'on travaille avec une base de donnée

📁 Configuration des variables d'environnement :

Elles sont dans le fichier `.env` directement sous la racine du dossier

📁 Gestion de la soumission du formulaire (page contact) :

`Router.post('/contact')`

Gestion des erreurs et conditions de soumissions:

Si le champ `lastname`, `avis` et `note` sont nuls, le middleware `flash` à l'aide du module `express-session` s'occupe de renvoyer l'erreur sur le front à l'aide de la méthode `req.flash` et un message est envoyé à l'utilisateur, un fichier `semantic.min.css` a été rajouté spécialement à la page contact pour améliorer le rendu visuel des erreurs locales ;

Sinon si la condition est respectée, un message de succès est affiché à l'utilisateur.

Configuration de la base de données:

Le module mysql de npm a été installé en utilisant un require et la méthode **create pool** pour connecter la base de données et recevoir la soumission du formulaire

Soumission du formulaire vers la base de données:

La classe Objet **Avis** permet d'insérer les données collectées sur le formulaire vers la base de données, elle prend en paramètre les champs du formulaire initial, une requête d'insertion (*insertquery* qui se trouve dans la feuille **query**) et un callback. Elle renvoie une erreur si quelque chose s'est mal produite ou insère les résultats dans la base de données en cas de succès.

Elle est ensuite invoquée dans le cas success de la soumission du formulaire en recevant les arguments du *req.body*.

Affichage des requêtes vers le DOM:

La fonction **SelectAllElements** a été créée pour récupérer les données de la base de données vers les diverses routes de l'API, elle retourne une promesse et est utilisée de façon asynchrone et rejette une erreur si quelque chose s'est mal produite ou renvoie les résultats sous forme d'un tableau d'objets.

Elle en prend en paramètre une requête **query**. La liste des tous les requêtes étant stockée sous forme d'Array dans une feuille qui s'appelle **query** et est exportée.

Elle est ensuite appelée sur les diverses routes des diverses pages en la stockant dans une variable avec la méthode *async await* et les résultats sont passés en second paramètre de la méthode *render()* pour pouvoir les afficher sur le **DOM** à l'aide des templates engine d'ejs et il suffit d'itérer sur les éléments après.

Remarque générale: site pas du tout responsive (problème non corrigé)

*****o*****