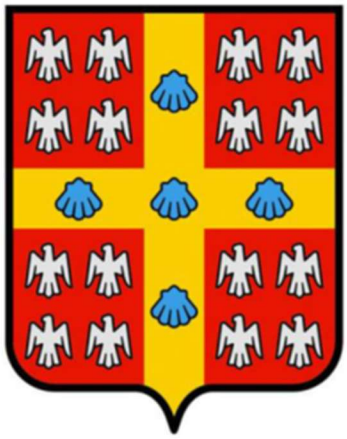


UNIVERSITÉ LAVAL

Projet : Concessionnaire

Présenté à Monsieur Richard Khoury

Par l'équipe **01**

Nom	**NI**
Kamel, Richard	(537 035 274)
Dari, Merwane	(536 900 247)

Dans le cadre du cours GLO-2005 : Modèles et langages des bases de données pour l'ingénierie

Faculté science et génie

Table des matières

1. Énonciation du problème et des exigences 3

2. Modélisation des données 5

3. Création de la base de données 7

4. Création des requêtes et des routines 7

5. Indexation et optimisation du système 9

6. Normalisation des relations 10

7. Sécurité de la base de données 11

8. Implémentation de la logique d'affaire 11

9. Implémentation de l'interface utilisateur 11

10. Tests du système 12

11. Accessibilité du système 13

12. Gestion de l'équipe et organisation du travail 14

1. Énonciation du problème et des exigences

Description de l'application et du contexte

Pour ce projet, nous avons choisi de développer une application web destinée à la gestion des activités internes d'un concessionnaire automobile. L'objectif principal de cette application est d'offrir un outil centralisé et fonctionnel permettant de suivre efficacement les différents volets des opérations courantes, notamment la gestion de l'inventaire des véhicules, le suivi des clients, l'enregistrement des ventes, la tenue à jour des entretiens effectués sur les véhicules ainsi que la gestion des fournisseurs de pièces détachées. Le système est conçu pour être utilisé uniquement par les employés du concessionnaire, incluant les vendeurs, les mécaniciens et le personnel administratif. Il ne s'agit pas d'une plateforme ouverte au public, mais plutôt d'un outil interne simplifiant la gestion quotidienne de l'entreprise.

Dans cette optique, l'application sera déployée dans un environnement local ou interne, sans qu'aucun client externe n'y ait accès. Elle a été pensée pour simuler le fonctionnement réel d'un garage, en se basant sur des données fictives mais cohérentes et crédibles. Ces données permettront de tester et démontrer les fonctionnalités attendues du système dans des conditions proches de la réalité, tout en évitant la complexité associée à l'utilisation de données réelles ou sensibles.

Groupes d'utilisateurs et besoins

L'application est conçue pour être utilisée exclusivement par le personnel du concessionnaire, réparti en trois grands types d'utilisateurs : les vendeurs, les mécaniciens et les administrateurs. Les vendeurs s'en serviront pour consulter l'inventaire des véhicules disponibles, ajouter de nouveaux clients dans le système, enregistrer les ventes réalisées et, au besoin, générer des informations relatives à la transaction, comme des documents de facturation. Les mécaniciens auront accès à l'historique des entretiens pour chaque véhicule et pourront y ajouter de nouvelles interventions, tout en précisant les employés responsables ou les fournisseurs impliqués. De leur côté, les administrateurs bénéficieront d'un accès étendu à l'ensemble du système, leur permettant de gérer les comptes employés, d'apporter des modifications aux données sur les véhicules, et de tenir à jour les informations concernant les fournisseurs. Chaque utilisateur interagit donc avec le système en fonction de son rôle, et l'ensemble des fonctionnalités développées vise à refléter cette organisation de manière claire, intuitive et adaptée aux besoins réels du garage.

Répartition des responsabilités dans l'architecture trois-niveaux

Niveau	Responsabilités
Client (Frontend)	Interface web (HTML/CSS/JS) pour naviguer dans les véhicules, ventes, clients, entretiens, etc. Affichage des données et formulaires d'entrée. Validation simple des champs (JS).
Serveur d'application (Flask)	Traitement des requêtes, logique d'affaires (ex. validation de disponibilité avant-vente), appels à la BD, gestion des erreurs.
Base de données (MySQL)	Stockage structuré des données dans les 6 tables (Vehicules, Clients, Ventes, Employées, Entretien, Fournisseurs). Contraintes d'intégrité, clés primaires/étrangères, index.

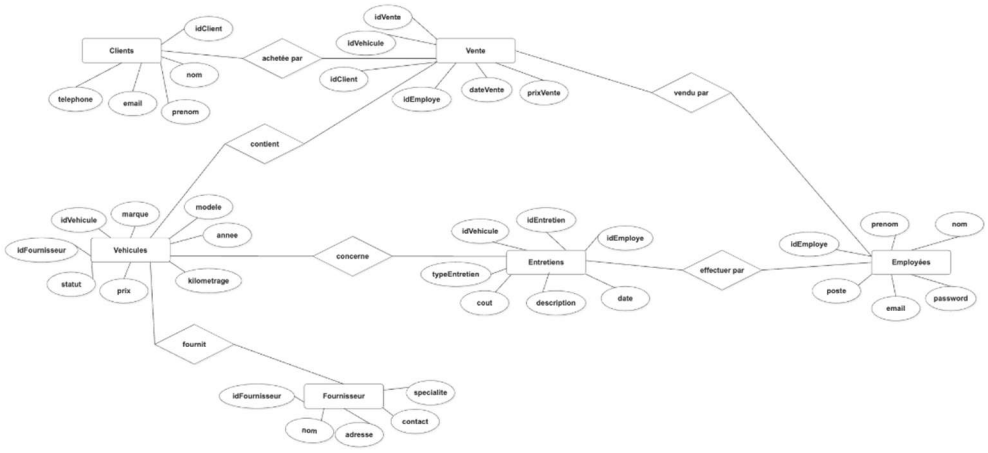
Communication entre les niveaux

Client → Serveur : envoie des données saisies dans des formulaires, ce qui inclurait une nouvelle vente, un ajout d'entretien, etc.

Serveur → BD : Requêtes SQL en fonction des actions des utilisateurs.

BD → Serveur → Client : renvoie les résultats des requêtes (ex. véhicules disponibles, historique d'un client), affichés dans l'interface.

2. Modélisation des données



Modèle relationnel

Fournisseur

- **idFournisseur** : int, **PK**
- nom : varchar(100)
- adresse : varchar(255)
- contact : varchar(100)
- specialite : varchar(100)

Vehicule

- **idVehicule** : int, **PK**
- marque : varchar(50)
- modele : varchar(50)
- annee : int
- kilometrage : int
- prix : decimal(10,2)
- statut : varchar(15)
- **idFournisseur** : int, **FK** → Fournisseur(idFournisseur)
  (ON DELETE SET NULL, ON UPDATE CASCADE)

Client

- **idClient** : int, **PK**
- nom : varchar(100)
- prenom : varchar(100)
- email : varchar(100)
- telephone : varchar(20)

Employe

- **idEmploye** : int, **PK**
- nom : varchar(50)
- prenom : varchar(50)
- poste : varchar(30)
- email : varchar(100)
- password : varchar(255)

Vente

- **idVente** : int, **PK**
- **idVehicule** : int, **FK** → Vehicule(idVehicule)
- **idClient** : int, **FK** → Client(idClient)
- **idEmploye** : int, **FK** → Employe(idEmploye)
- dateVente : date
- prixVente : decimal(10,2)

  (ON DELETE RESTRICT, ON UPDATE CASCADE pour chaque FK)

Entretien

- **idEntretien** : int, **PK**
- **idVehicule** : int, **FK** → Vehicule(idVehicule)
- **idEmploye** : int, **FK** (nullable) → Employe(idEmploye)
- date : date
- typeEntretien : varchar(100)
- cout : decimal(10,2)
- description : text

  (ON DELETE CASCADE pour Vehicule, ON DELETE SET NULL pour Employe)

3. Création de la base de données

La base de données que nous avons conçue vise à modéliser les opérations internes d'un concessionnaire automobile. Elle contient un total de six tables : *Client, Employe, Fournisseur, Vehicule, Vente et Entretien*. Chacune représentant une entité clé du système. Pour le projet, nous avons choisi de remplir la base avec un volume de données réaliste mais gérable : 100 clients, 20 employés, 10 fournisseurs, 100 véhicules, 60 ventes et 60 entretiens.

La création de la base a été faite dans MySQL sous le nom Concessionnaire, à l'aide d'un script SQL contenant toutes les instructions nécessaires. Celui-ci commence par supprimer toute version existante de la base, puis crée les différentes tables avec leurs clés primaires, leurs clés étrangères et les types de données appropriés. Chaque table respecte la structure issue du modèle entité-relation développé à l'étape précédente.

Le peuplement de la base a été effectué à l'aide d'un script Python. Celui-ci utilise la bibliothèque Faker, configurée pour générer des données réalistes en français canadien (noms, adresses, courriels, téléphones), ainsi que le module mysql-connector-python pour insérer automatiquement les tuples dans la base MySQL. Les données sont fictives, mais l'ensemble du jeu de données a été conçu pour rester logique et cohérent. Par exemple, seuls les employés ayant le poste « vendeur » sont assignés aux ventes, et seuls les mécaniciens sont associés aux entretiens. De plus, une liste de 60 véhicules a été sélectionnée aléatoirement pour représenter les ventes, et ces mêmes véhicules ont été marqués avec le statut "vendu" dans la table Vehicule.

Une fois les données insérées, plusieurs requêtes de validation ont été exécutées pour s'assurer qu'il n'existait aucune incohérence entre les tables. Parmi ces vérifications : s'assurer que tous les identifiants référencés existent bien, que chaque véhicule marqué "vendu" est bel et bien présent dans la table Vente, et que les employés assignés à une vente ou un entretien avaient bien le poste correspondant. Tous les résultats obtenus ont confirmé que la base est intègre et conforme à la structure définie. Elle est maintenant prête à être utilisée pour les prochaines étapes du projet.

4. Création des requêtes et des routines

Pour rendre notre système réellement utilisable, nous avons développé plusieurs requêtes SQL avancées, ainsi que des routines sous forme de procédures, fonctions et gâchettes. L'objectif était de répondre aux besoins concrets d'un concessionnaire automobile.

Requêtes

Nous avons créé cinq requêtes SQL qui illustrent différents concepts clés :

- Une première utilise des jointures explicites pour afficher toutes les ventes, avec les informations sur le client, l'employé et le véhicule.
- Une autre effectue une agrégation pour calculer le total des ventes réalisées par chaque employé.
- Une requête imbriquée permet d'identifier tous les véhicules qui sont encore disponibles et n'ont jamais été vendus.
- Une autre, utilisant HAVING, sert à détecter les clients qui ont effectué plus d'une transaction.
- Enfin, une requête de type division relationnelle (via NOT EXISTS) permet d'obtenir la liste des mécaniciens qui n'ont jamais été assignés à un entretien.

Routines

- *DerniersEntretiensVehicule* affiche les cinq derniers entretiens effectués sur un véhicule. Elle est particulièrement utile pour les mécaniciens ou les administrateurs qui veulent rapidement consulter l'historique d'un véhicule. ENTRETIEN
- *VentesParEmploye* permet d'afficher toutes les ventes faites par un employé sur une période donnée. Elle peut par exemple être utilisée pour générer des statistiques de performance. EMPLOYE

Fonctions

- *PrixMoyenVendu* retourne le **prix moyen des véhicules vendus**, ce qui est pratique pour analyser les tendances de prix. VENTE
- *NbVehiculesParFournisseur* **le nombre de véhicules fournis par un fournisseur**, utile pour suivre les relations avec les fournisseurs. FOURNISSEUR

Triggers

- *VerifierVehiculeAvantVente* empêche qu'on puisse vendre un véhicule qui est déjà marqué comme "vendu", ce qui évite les erreurs de double-vente. VEHICULE
- *MajStatutVehiculeApresVente* met automatiquement à jour le statut du véhicule à "vendu" dès qu'une vente est enregistrée. Cela permet de garder les données cohérentes sans avoir à le faire manuellement. VEHICULE

5. Indexation et optimisation du système

Pour optimiser l'exécution des requêtes dans notre base de données, nous avons mis en place une liste d'index adaptés à l'utilisation réelle prévue du système. Ces index ont été choisis en fonction des colonnes les plus souvent utilisées dans les requêtes SQL complexes (filtres, jointures, regroupements), tout en évitant de créer des doublons avec les clés primaires ou étrangères déjà indexées automatiquement par MySQL.

- idx_dateVente sur Vente(dateVente) : Améliore les recherches et tris sur les ventes effectuées à des dates précises.
- idx_typeEntretien sur Entretien(typeEntretien) : Accélère les filtres sur le type d'entretien demandé, comme les diagnostics ou les vidanges.
- idx_client_vehicule sur Vente(idClient, idVehicule) : Optimise les requêtes combinant clients et véhicules dans le contexte des ventes.
- idx_idFournisseur sur Vehicule(idFournisseur) : Facilite les regroupements ou filtres par fournisseur dans les analyses d'inventaire.
- idx_posteEmploye sur Employe(poste) : Permet de distinguer rapidement les employés selon leur rôle dans le système.

6. Normalisation des relations

Relation	Forme normale atteinte	Justification
Client	FNBC	La clé primaire **idClient** détermine tous les attributs. Aucun attribut ne dépend d'un autre attribut non-clé.
Vehicule	FNBC	**idVehicule** est la clé. Tous les attributs (marque, modèle, année, etc.) dépendent directement de cette clé sans dépendance transitive.
Employe	FNBC	**idEmploye** détermine nom, prenom, poste et email, sans dépendance partielle ou transitive.
Fournisseur	FNBC	Tous les attributs dépendent uniquement de **idFournisseur**, aucune redondance ni transitivité.
Vente	FNBC	Clé primaire simple **idVente**. Les autres attributs dépendent directement de cette clé (aucune transitive entre **idClient**, **idVehicule**, etc.).
Entretien	FNBC	Même structure que Vente, avec une clé simple **idEntretien** déterminant tous les autres champs.

7. Sécurité de la BD

Nous avons utilisé des mots de passe pour sécuriser l'accès des utilisateurs à l'application. Ces mots de passe sont obligatoires lors de la connexion au système, afin de garantir que seules les personnes autorisées puissent accéder aux fonctionnalités du site. Bien sûr, sans un identifiant valide, autrement dit un identifiant présent dans notre base de données, personne ne peut se connecter au serveur.

8. Implémentation de la logique d'affaire.

Pour cette partie, nous avons utilisé Flask afin de mettre en place la logique qui se trouve entre l'interface web et la base de données. Concrètement, cette couche permet de recevoir les informations envoyées par l'utilisateur (comme ses identifiants de connexion), de les vérifier, puis d'interagir avec la base de données pour valider ou refuser l'accès.

Nous avons développé une page de connexion fonctionnelle qui permet aux employés de se connecter au système. Lorsqu'un utilisateur remplit le formulaire et envoie ses informations, notre application vérifie que le courriel existe et que le mot de passe fourni correspond bien à celui enregistré (de façon sécurisée, sans jamais stocker le mot de passe directement). Si l'information est valide, l'utilisateur reçoit un message de bienvenue personnalisé. Sinon, un message d'erreur s'affiche.

Nous avons aussi prévu des vérifications côté serveur pour éviter que le système plante en cas de problème (par exemple, si la base de données ne répond pas). On s'est assurés que le code est bien commenté pour qu'il soit facile à comprendre ou modifier plus tard.

En résumé, cette étape nous a permis de poser les bases du fonctionnement interne du système, en traitant les données entrées par l'utilisateur de façon sécurisée et logique. Tout fonctionne bien, et cette partie est maintenant prête pour être connectée aux autres fonctionnalités du site.

9. Implémentation de l'interface utilisateur.

Pour cette dernière partie du projet, nous avons développé une interface web fonctionnelle permettant à un employé de se connecter et de consulter ou modifier les données de la base en fonction de ses besoins. Le tout a été fait avec Flask en Python pour le serveur, et du HTML, CSS, et JavaScript pour le frontend.

Dès la connexion, l'utilisateur arrive sur une page de login simple. Si les identifiants sont valides, il est redirigé vers un tableau de bord lui donnant accès aux différentes sections : véhicules, clients,

ventes et entretiens. En cas d'erreur, un message clair s'affiche sur la page même, sans redirection, grâce à une combinaison de Flask et de JavaScript.

Chaque page (ex. vehicules.html, clients.html, etc.) est construite pour afficher des données dynamiques récupérées de la base MySQL, et est reliée à son propre fichier .js pour gérer certaines interactions côté client. Tous les fichiers JavaScript ont été centralisés sous static/js et un fichier app.js commun a été ajouté pour gérer des comportements globaux comme l'affichage de messages.

Enfin, la page entretiens.html permet d'ajouter un nouvel entretien. Lorsqu'un ID de véhicule invalide est entré, un message d'erreur clair s'affiche sans rediriger, grâce à une gestion des erreurs bien intégrée dans Flask, combinée à un peu de JavaScript pour l'affichage. Ce genre de gestion d'erreurs améliore beaucoup l'expérience utilisateur.

Toutes les routes Flask ont été définies dans serveur.py, et chaque route est bien associée à un fichier .html. La structure du projet suit une logique claire : un dossier static pour tout ce qui est JS et CSS, un dossier templates pour le HTML, et un backend en Python. Le tout est simple à maintenir et fonctionne efficacement.

10. Implémentation de l'interface utilisateur.

Le site Web qu'on a conçu a pour objectif de simplifier la gestion interne d'un concessionnaire automobile. Il permet aux employés — qu'ils soient vendeurs, mécaniciens ou administrateurs de consulter, ajouter et visualiser des données importantes liées à leur travail, le tout à partir d'un portail central.

Lorsqu'un employé veut utiliser le système, il doit d'abord se connecter à l'aide de ses identifiants. Une fois connecté, il est dirigé vers un tableau de bord qui donne accès aux différentes sections : véhicules, clients, ventes et entretiens.

Chaque section a sa propre page et une interface simple. Par exemple, la page des véhicules permet d'afficher l'inventaire et d'effectuer des recherches. La page des clients montre leurs informations, et celle des ventes affiche les transactions entre clients, employés et véhicules.

La section entretiens a demandé plus d'attention, car elle combine l'ajout de nouveaux entretiens et l'affichage des anciens. L'utilisateur peut remplir un formulaire avec l'ID du véhicule, le type d'entretien, la date et le coût. Si tout est valide, l'entrée est ajoutée à la base de données et visible dans un tableau. En cas d'erreur, un message s'affiche directement dans la page (par exemple, si l'ID du véhicule n'existe pas), ce qui rend le système plus facile à utiliser.

Le visuel du site est géré avec un fichier CSS pour garder une mise en page propre, et plusieurs fichiers JavaScript (comme app.js ou entretiens.js) sont utilisés pour ajouter de l'interactivité.

Bref, le site est fonctionnel, facile à naviguer, et il permet aux employés d'effectuer leur travail efficacement avec un accès rapide à toutes les données importantes.

11. Accessibilité du système.

Le site web développé pour le concessionnaire automobile a été conçu dans le but d'être à la fois accessible, convivial et robuste. Dès le départ, nous avons mis en place une interface claire et simple d'utilisation, permettant à tout employé (peu importe son rôle) de naviguer efficacement à travers les différentes sections comme les véhicules, les clients, les ventes ou les entretiens.

Accessibilité : Le site est accessible depuis un navigateur web standard, ce qui signifie qu'il ne nécessite pas de logiciel externe ou de configuration complexe. L'utilisateur n'a qu'à se connecter avec ses identifiants à partir de la page de connexion. Une fois authentifié, il accède aux fonctionnalités qui lui sont pertinentes. L'interface a été pensée pour minimiser les obstacles, autant sur ordinateur que sur d'autres types d'écrans, bien qu'elle soit optimisée pour une utilisation en interne sur un poste de travail.

Portabilité : Le projet repose sur des technologies bien connues comme HTML, CSS, JavaScript (dans le dossier static/js) et Flask côté serveur. Cela le rend facilement déployable sur n'importe quelle machine supportant Python et MySQL. Le fichier serveur.py est modulaire et peut être ajusté rapidement si le projet devait être migré vers un autre environnement ou serveur.

Convivialité : Nous avons accordé une attention particulière à la facilité d'utilisation. Par exemple, chaque section du site a une mise en page cohérente, avec des boutons visibles, des champs bien étiquetés, et des formulaires simples. Le fichier styles.css assure une apparence uniforme, professionnelle et claire. De plus, les fichiers JavaScript comme login.js ou entretiens.js permettent d'afficher des messages d'erreur directement sur la page sans rechargement, améliorant ainsi l'expérience utilisateur.

Robustesse : Le système est capable de gérer des erreurs de saisie, comme un ID de véhicule invalide dans le formulaire d'entretien. Ces cas sont détectés côté serveur (serveur.py) et un message clair est affiché à l'utilisateur via JavaScript. Cela évite l'insertion de données incohérentes dans la base. La logique serveur est également encapsulée dans des blocs try-except, ce qui permet d'éviter les plantages du système en cas d'imprévu.

En résumé, le système développé respecte les grands principes de qualité logicielle. Il est simple à utiliser, facile à maintenir et assez flexible pour être amélioré ou étendu si nécessaire.

12. Gestion de l’équipe et organisation du travail.

Initialement, notre projet devait être réalisé par une équipe de quatre membres. Toutefois, deux personnes ont quitté le cours en cours de session, ce qui nous a obligés à revoir l’organisation du travail. Malgré cette difficulté, nous avons maintenu une bonne coordination en partageant les responsabilités équitablement entre les deux membres restants.

Nous avons pris soin de planifier les tâches de façon claire dès le début, en se répartissant les différentes sections du projet selon nos forces. Par exemple, pendant que l’un travaillait sur les pages HTML et la structure visuelle, l’autre se chargeait du backend avec Flask et la connexion à la base de données. Les communications ont été constantes afin de rester à jour sur l’avancement et les intégrations à faire.

Cette situation a demandé plus de flexibilité et d’organisation de notre part, surtout pour assurer que toutes les fonctionnalités du système fonctionnent bien ensemble. Finalement, malgré la réduction de l’équipe, nous avons réussi à livrer un projet fonctionnel et complet, en assurant la cohérence entre toutes les parties.