BORDEAUX INP Enseirb-Matmeca

Entretien de Véhicules

20 Octobre 2023 - 06 Décembre 2023

Rapport de Projet de Système de Gestion de Bases des Données

RODRIGUEZ Esteban, PERIN Côme, L'HONORE Antoine, REINERT Mathis

Encadrants :

Mohamed Mosbah

Sylvain Lombardy

Table des matières

1 Introduction

Ce projet s'inscrit dans le cadre de la matière "Système de Gestion de Bases de Données (IT203)" de notre cursus. Il consiste en la réalisation d'une base de données capable de gérer l'entretien de véhicules au sein d'un garage. De plus, afin de percevoir l'évolution de notre base de donnée, nous avons eu l'occasion de développer une page Web. Cette page nous permet aussi d'interagir avec cette base de données (ajout/suppression/modification de données).

Le but de ce projet est d'implémenter une base de données qui pourrait faire l'objet d'un travail d'un développeur professionnel. Effectivement, un garage pourrait commanditer cette base de données avec son interface web afin de suivre les statistiques de l'entreprise de manière simple et rapide.

Nous nous sommes donc mis dans la peau d'une équipe de développeurs devant implémenter une base de données pour un client ayant besoin d'informatiser les informations nécessaires au bon fonctionnement de son garage.

2 Modélisation des données

L'objectif est donc de créer une base de données qui contient toutes les données que doit avoir un garage. Nous allons donc devoir commencer par créer un schéma conceptuel avant d'implémenter notre base.

Le schéma conceptuel (schéma entité-association) que nous avons créé est, nous le concevons, plutôt grand et complexe. Ainsi, afin de l'expliquer, nous allons nous concentrer sur des fragments de ce schéma en expliquant les raisons de nos choix (section 2.1 à 2.3).

Puis, nous verrons notre schéma conceptuel en entier (section 2.4).

Nous comparerons ensuite notre schéma conceptuel final avec notre schéma conceptuel initial en expliquant les changements que nous avons effectués. (section 2.5)

Nous finirons par une revue des opérations que nous avons prévues sur notre base de données.

2.1 Modélisation des clients et des voitures

Pour commencer, nous nous intéressons à la représentation des clients.

Un client est enregistré avec ses informations personnelles : identité civile (nom et prénom) et contact (mail et téléphone). Le choix de la clé primaire s'est porté sur le numéro de sécurité sociale, étant donné son unicité. Les clients sont inscrits dans l'entité *Personnes* de notre schéma conceptuel.

Le sujet de ce projet nous demandait de renseigner l'adresse des clients. Cependant, l'adresse d'un lieu nous est aussi utile pour renseigner l'adresse d'un garage (nous le verrons plus tard, section 2.2). En conséquence, nous avons décidé de stocker les renseignements de l'adresse des clients dans une entité à part (nommée *adresses*) et non dans l'entité du client (nommé *Personnes*). Cela nous permettra de réutiliser l'entité *"adresses"* plus tard.

Un client possède forcément une ou plusieurs voitures (pour être considéré comme client pour notre garage). Pour identifier chaque voiture de manière unique, nous avons désigné le numéro d'immatriculation d'une voiture comme clé primaire de l'entité "*voiture*". Cette entité contient aussi des caractéristiques d'une voiture (puissance fiscale, cylindre et couleur).

Un garagiste va effectuer des travaux sur une ou plusieurs pièces d'un véhicule. Ainsi, il faut pouvoir identifier les pièces d'une voiture. C'est ainsi que nous avons créé une entité "*pièces*" qui composent un type de voiture. Ce type de voiture est représenté par notre entité "*modeles_ voitures*". Et cette entité est reliée à la voiture du client que nous avons définie plus tôt.

La figure ci-dessous est le fragment de notre schéma global qui représente les clients ainsi que leurs véhicules dans notre base de données.

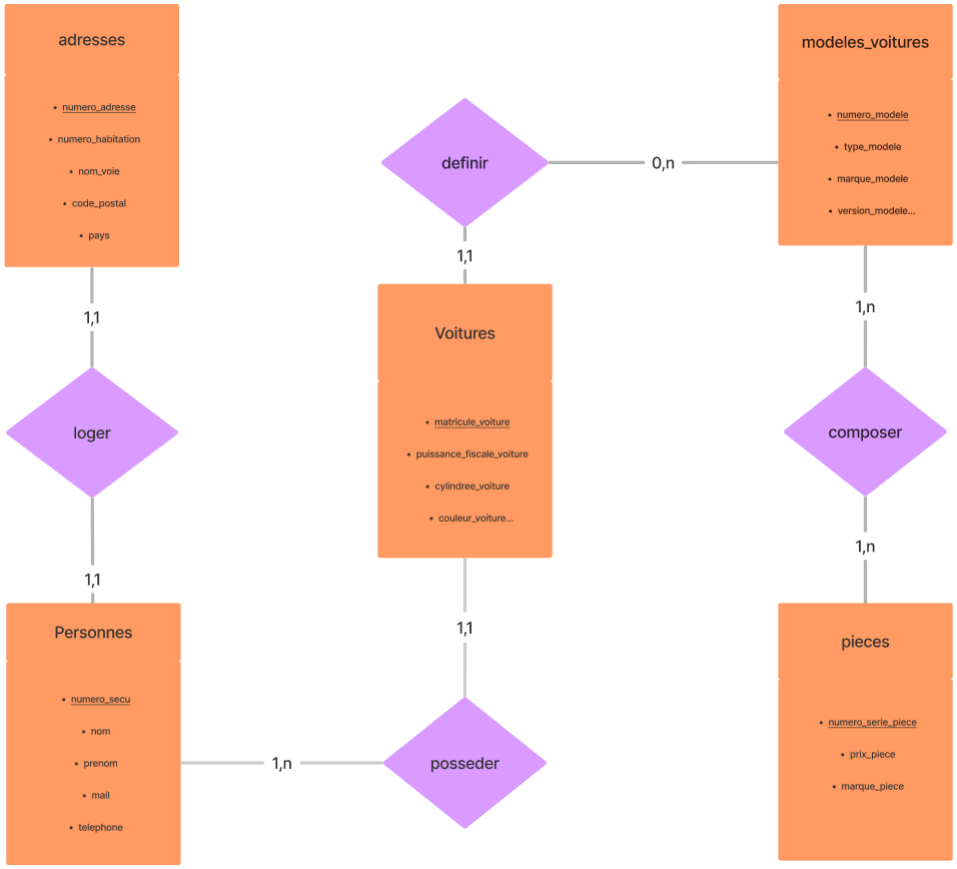

FIGURE 2.1 – Schéma conceptuel des clients et des voitures

2.2 Modélisation d'un garage

Vient désormais la définition des garages.

Un garage contient les informations nécessaires à notre base de données (dénomination, forme juridique et date de création). Afin de désigner un garage de manière unique, nous avons désigné comme clé primaire le numéro du SIREN.

Ce qui nous intéresse avec la définition des garages, c'est de lier ces garages avec des interventions. Nous créons donc l'entité *interventions*. Cette entité est identifiée avec la clé primaire *"numero_intervention"* et contient des attributs essentiels à notre étude (dates de début et de fin, le kilométrage et l'information si cette intervention est finie ou non).

L'entité *garage* est reliée à l'entité *intervention* grâce à l'association *intervenir*. Une intervention est propre à un garage, donc nous avons une cardinalité (1,1) entre l'entité *interventions* et l'association *intervenir*. Cependant, un garage est concerné par aucune ou plusieurs interventions : la cardinalité entre l'entité *garage* et l'association *intervenir* est donc (0,n).

Afin de situer un garage, nous relions l'entité *"garages"* avec l'entité *"adresses"*. Cette entité préexistait pour définir l'adresse des clients (section 2.1) et nous la réutilisons.

La figure ci-dessous représente le fragment de notre schéma global qui représente un garage dans notre base de données.

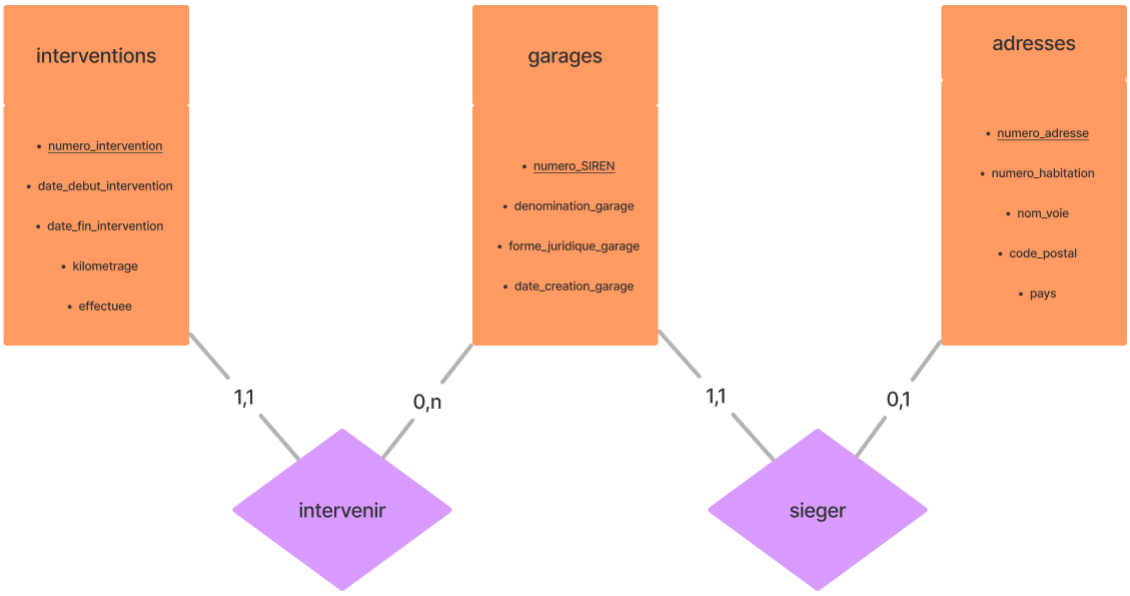

FIGURE 2.2 – Schéma conceptuel d'un garage

2.3 Modélisation d'une intervention

Nous arrivons à la partie centrale de notre schéma conceptuel : les interventions. En effet, elle traduit une partie importante de notre base de donnée.

L'entité *interventions* a été définie plus tôt (section 2.2) puisqu'une intervention est liée à un garage.
Afin d'éviter tout type de redondance, nous lions l'intervention à une voiture (qui identifiera le client).

Afin de présenter une intervention à un potentiel client, notre garagiste doit prévoir un devis sur une intervention. C'est pourquoi nous avons créé une entité *devis*. Cette entité est identifiée avec un numéro comme clé primaire et elle contient des informations nécessaires telles que la date et le montant à proposer aux clients.
L'entité *devis* est liée à l'entité *interventions* grâce à l'association *prevoir*.
Un devis concerne une et une seule intervention, c'est pour cela que nous avons une cardinalité (1,1) entre l'entité *devis* et l'association *prevoir*. Cependant, il se peut que le devis ne plaise pas au client, en conséquence notre garagiste devra refaire d'autres devis qui concerne une même intervention. De plus, le devis est optionnel, il se peut qu'une intervention subvienne sans devis au préalable. C'est pour cela que nous avons mis une cardinalité (0,n) entre l'entité *intervention* et l'association *prevoir*.

Si le client du garagiste accepte le devis, l'intervention est effectuée, il faut donc affecter une ou plusieurs factures à cette ou ces interventions. L'entité *factures* contient les mêmes informations que l'entité *devis* (numéro comme clé primaire, date et montant). Cependant, il fallait bien différencier un devis d'une facture : deux entités différentes étaient donc nécessaires à notre schéma conceptuel.
L'entité *intervention* est quant à elle liée avec l'entité *factures* avec l'association *generer*. Tout comme avec l'entité *devis*, une facture ne concerne qu'une et une seule intervention : d'où la cardinalité (1,1) entre l'entité *factures* et l'association *generer*. Cependant, il est obligatoire d'avoir une facture pour une intervention. De plus, une intervention peut nécessiter plusieurs factures. Ainsi, nous avons mis une cardinalité (1,n) entre l'entité *intervention* et l'association *generer*.

Afin de préciser les interventions, nous sommes partis du principe qu'une intervention est une succession d'actions.
Nous avons décidé de définir une action avec un numéro (clé primaire) est des informations sur cette action (nom, durée et tarif).
Pour qu'une intervention soit désignée comme tel, il faut qu'il y ait au moins une action dans cette intervention. C'est pour cela que nous avons une cardinalité (1,n) entre l'entité *interventions* et l'association *contenir*. Cependant, une action

peut certes être contenue dans plusieurs interventions, mais elle peut n'être définie dans aucune (dans le cas où notre garage n'a pas eu encore à traiter ce genre d'actions). Ainsi, nous avons une cardinalité (0,n) entre l'association *contenir* et l'entité *actions*.

Cependant, des imprévus peuvent arriver au cours d'une intervention. Nous avons tenu compte de cette possibilité et c'est pour cela que nous avons créé une autre relation entre les entités, *interventions* et *actions*. Ce deuxième lien se fait grâce à l'association *survenir*. Nous avons aux deux bords de cette association binaire des cardinalités (0, n) puisqu'une intervention peut se dérouler sans le moindre problème (pour le plus grand plaisir de notre garagiste) et notre garagiste n'a peut-être jamais eu à régler un certain problème en particulier.

Voici ci-dessous (figure 2.3) le fragment de notre schéma conceptuel qui représente les interventions dans notre base de données.

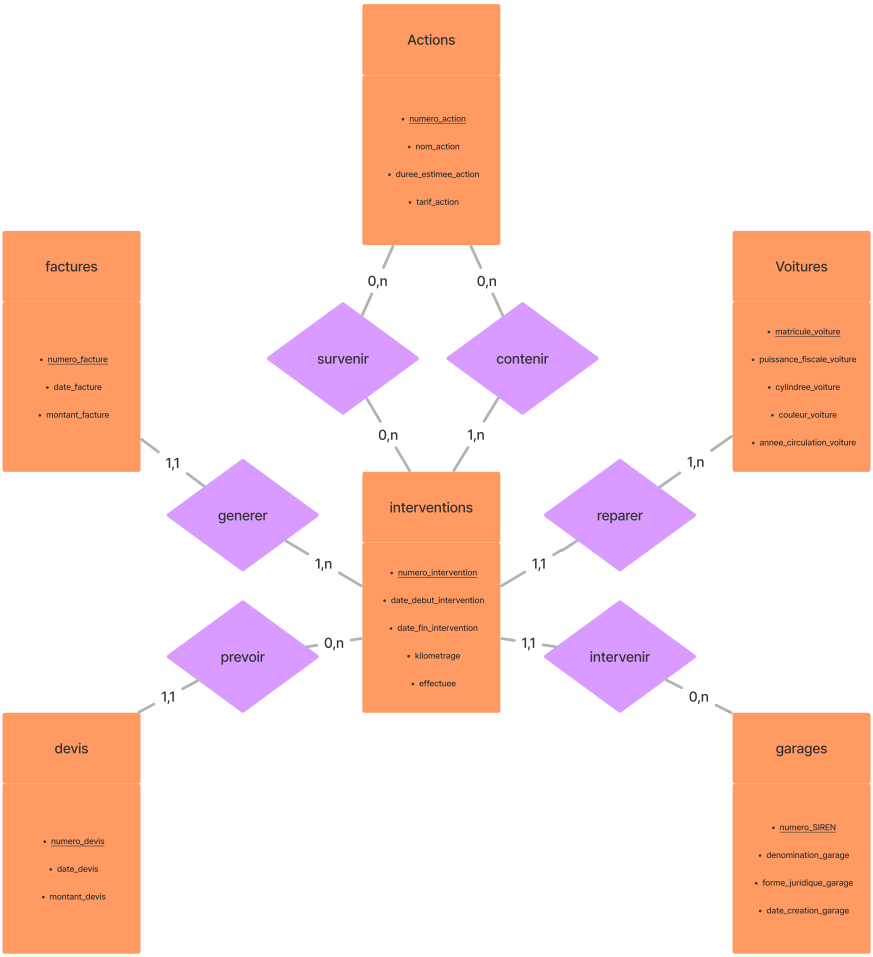

FIGURE 2.3 – Schéma conceptuel d'une intervention

2.4 Notre schéma conceptuel global

Nous avons à présent vu les 3 fragments importants de notre schéma conceptuel. Voici à présent notre schéma entité-association dans la globalité. (figure 2.4).

Nous reconnaissons bien dans ce schéma les 3 fragments vus plus haut : le fragment des clients et des voitures, le fragment des garages ainsi que le fragment des interventions.

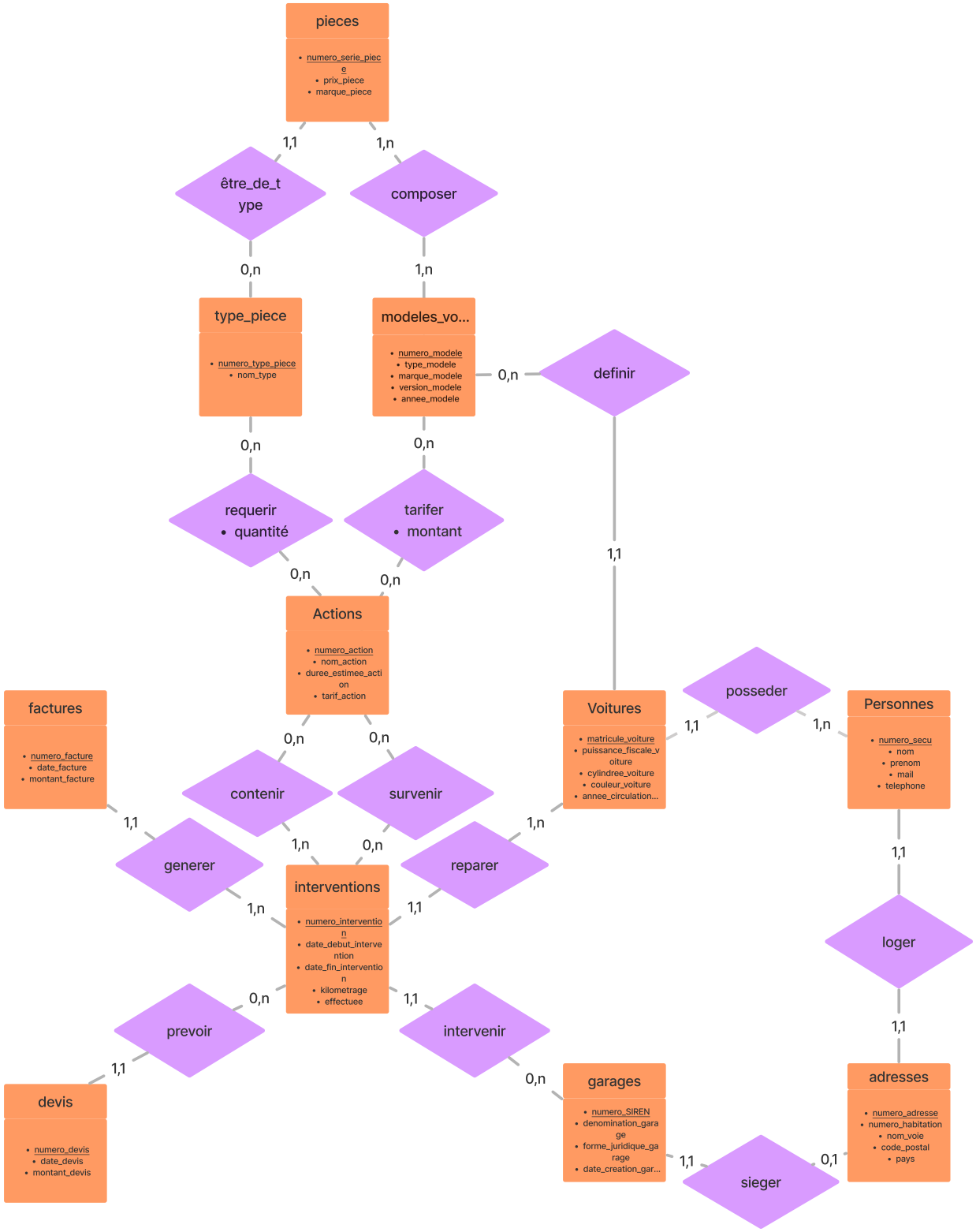

FIGURE 2.4 – Schéma conceptuel final de notre base de données

2.5 Comparaisons avec notre schéma conceptuel initial

Au début du projet, nous avons dû produire une première version du schéma conceptuel que voici :

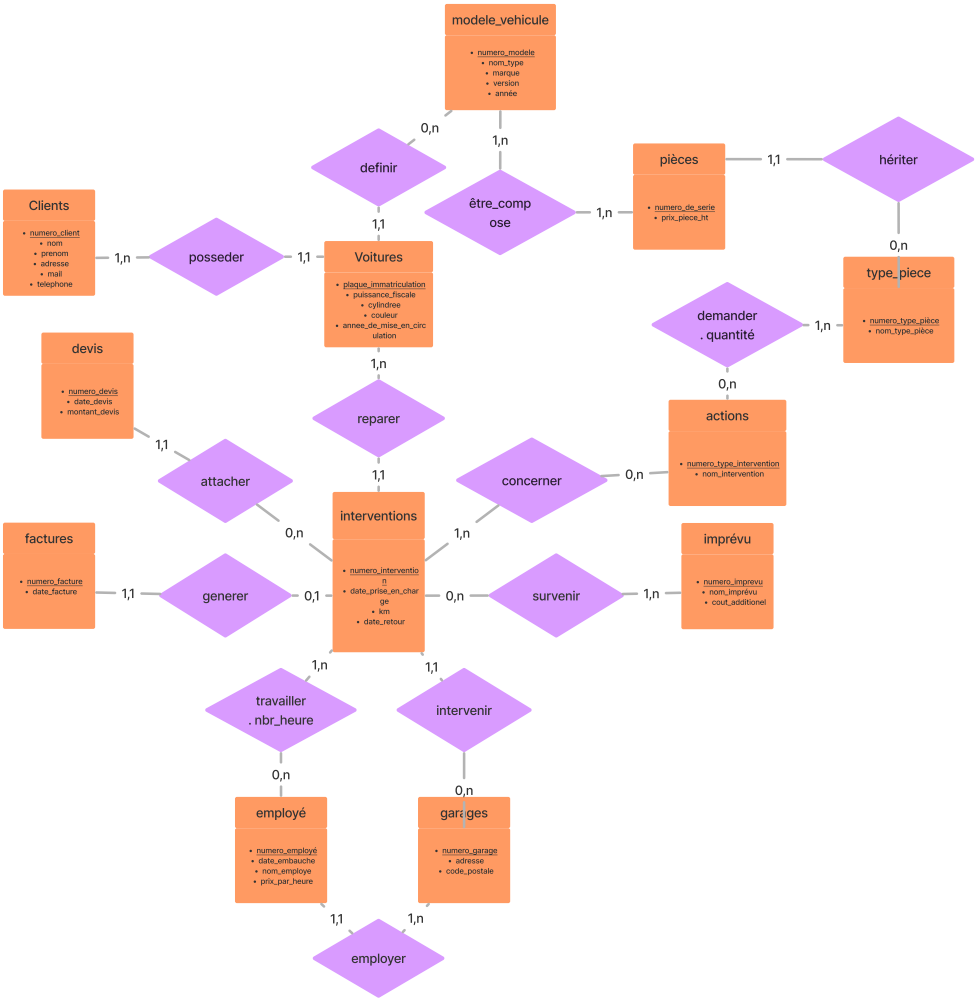

FIGURE 2.5 – Schéma conceptuel initial de notre base de données

Cette première version a été effectuée à partir de la lecture du sujet et des dialogues entre les membres du groupe pour comparer les visions et interprétations de chacun. Il subira des modifications par la suite pour ces principales raisons :

— Réinterprétation du sujet (souvent pour supprimer de la redondance)
— besoin en termes de requêtes.

Néanmoins, certaines entités comme *interventions* resteront centrales par la suite. Avec le recul nécessaire, il s'est avéré que, bien qu'inadapté pour mener à bien le projet dans sa globalité, cette première "ébauche" était un bon point de départ. On remarquera cependant qu'il est préférable d'être sûr des entités considérées, car une fois la base de tests implémentée, les changements d'entités peuvent devenir lourds à effectuer.

Quelques modifications principales sont notables :

— Suppression des employés
— Fusion des actions et imprévus.
— Création d'une table *adresses*

2.6 Opérations prévues sur notre base de données

Le rôle d'une équipe de développeur professionnel est aussi de communiquer avec le client de l'avancée du projet.

Ainsi, après avoir défini le modèle conceptuel de notre base de données et avoir reçu l'approbation du client, il nous faut aussi tenir compte des opérations que ce dernier veut effectuer sur la base de données que nous allons lui fournir.

À ce jour, le client nous a exprimé l'envie avec la base de données de pouvoir :
— Lister les interventions prévues dans les deux mois à venir.
— Lister des clients avec le nombre de véhicules qu'ils ont confiés au garage.
— Lister des modèles de véhicule pris en charge lors de l'année écoulée.
— Obtenir la liste des clients, avec le total des sommes facturées à chacun.
— Obtenir le nombre d'heures facturées par mois.
— Obtenir la liste des types de véhicules, avec le type d'intervention majoritaire pratiqué sur ces véhicules.

En plus de ces demandes spécifiques, le client souhaite également pouvoir ajouter, supprimer ou d'effectuer des modifications sur la base pour en assurer son bon fonctionnement.

Afin de répondre à ces demandes, nous avons prévu la définition de fonctions SQL directement dans notre base de données tel que `get_client_informations(int4)` par exemple, nous permettant de consulter les informations relatives au client tel que son adresse ou son numéro de téléphone. Cependant, cette fonction permet également de calculer des informations plus précise tel que la somme totale facturée à celui-ci. Ces fonctions permettent d'éviter la duplication de code au sein du client et permettent un meilleur maintien de code.

3 Schéma Relationnel

À présent, nous disposons d'un schéma conceptuel qui nous permet d'avancer dans notre travail afin d'obtenir une base de données implémentée en une base SQL.

Afin d'implémenter cette base, il nous faut passer de notre schéma conceptuel à un schéma relationnel.

Pour expliquer ce passage, nous commencerons par étudier les 4 règles que nous avons utilisées avec des exemples précis sur notre base. Suite à cela, nous verrons notre schéma relationnel final. Nous verrons ensuite les contraintes d'intégrité ainsi que les dépendances fonctionnelles de notre base de données et nous finirons par voir notre schéma relationnel en 3ème forme normale.

Rq : il existe une 5ème règle pour traduire un schéma conceptuel en un schéma relationnel qui concerne les associations non-binaire. Nous avons décidé d'omettre cette dernière règle dans notre rapport puisque notre schéma conceptuel ne comporte que des associations binaires.

3.1 Règle n°1 : Traduction d'une entité

Pour obtenir un schéma relationnel à partir d'un schéma conceptuel, il faut commencer par traduire les entités : Les entités deviennent des tables.

Les attributs deviennent des colonnes de cette table et l'identifiant (attributs soulignés dans le schéma conceptuel) devient la clé primaire de la table.

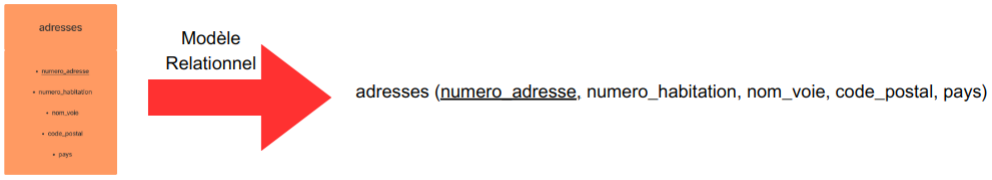

FIGURE 3.1 – Exemple de traduction d'une entité dans notre cas

3.2 Règle n°2 : Traduction d'une association binaire "1 :n"

Une association binaire de type "1 :n" est une association binaire qui a des cardinalités dont les maximums sont "1" d'une part et "n" de l'autre.

Une telle association dans un schéma conceptuel disparait dans le schéma relationnel.

Cependant, il y a l'apparition d'une clé étrangère dans l'entité du côté où la cardinalité est de maximum "1". Cette clé étrangères est précédée d'un "#" et référence la clé primaire de l'autre table.

Rq : Dans notre étude nous n'avons pas eu ce cas, cependant s'il y avait un attribut dans l'association, cet attribut se retrouverait dans la table qui a un lien maximum "1" avec l'association.

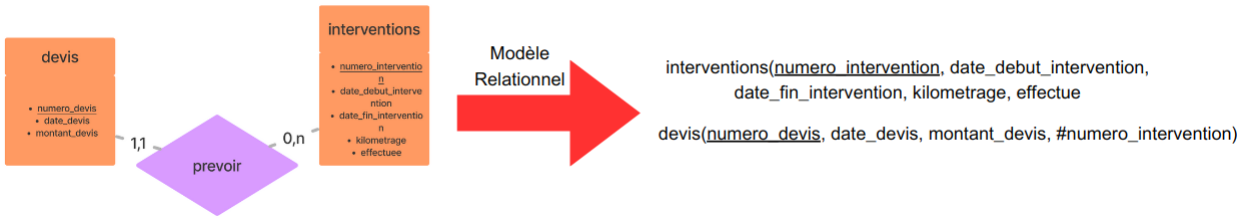

FIGURE 3.2 – Exemple de la traduction d'une association binaire "1 :n" dans notre cas

3.3 Règle n°3 : Traduction d'une association binaire "n :m"

Une association binaire de type "n :m" est une association dont les cardinalités dont les maximums sont tous deux de "n".

Une telle association dans un schéma conceptuel se traduit par une nouvelle table dont la clé primaire est composée de deux clés étrangères qui référencent les deux clés primaires des deux tables en lien avec l'association initiale.

Les attributs présents dans l'association deviennent des attributs de la table formée.

Il n'y a pas de clé étrangère dans les deux tables des deux entités.

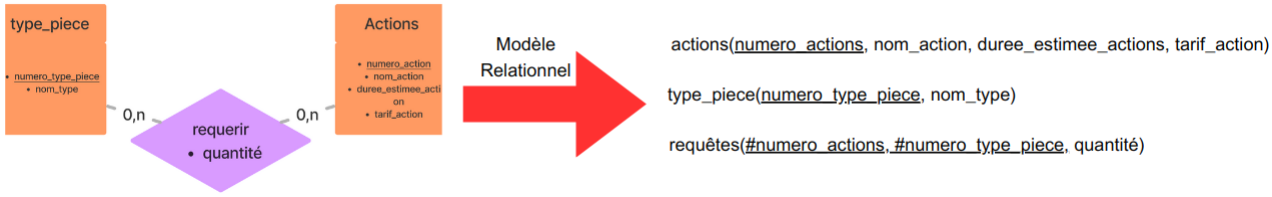

FIGURE 3.3 – Exemple de la traduction d'une association binaire "n :m" dans notre cas

3.4 Règle n°4 : Traduction d'une association binaire "1 :1"

Une association binaire de type "1 :1" est une association binaire qui a des cardinalités dont les maximums sont "1".

Une telle association est traitée comme une association binaire de type "1 :n" (règle n°2 section 3.2).

La seule différence est que la clé étrangère se voit imposer une contrainte d'unicité.

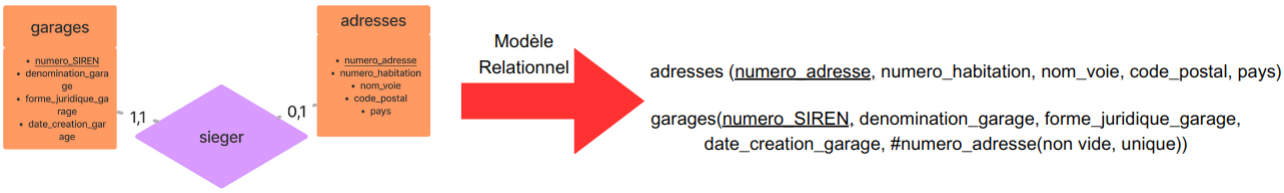

FIGURE 3.4 – Exemple de la traduction d'une association binaire "1 :1" dans notre cas

3.5 Notre schéma relationnel global

Après avoir vu les 4 règles que nous avons dû utiliser lors de la transformation de notre schéma conceptuel en schéma relationnel, voici le schéma que nous avons obtenu :

- interventions(numero_intervention, #matricule_voiture, #numero_SIREN, date_debut_intervention, date_fin_intervention, kilometrage, effectuee,)
- factures(numero_facture, #numero_intervention, date_facture, montant_facture)
- devis(numero_devis, #numero_intervention, date_devis, montant_devis)
- actions(numero_action, nom_action, duree_estime_action, tarif_action)
- actions_contenues(#numero_action, #numero_intervention)
- actions_survenues(#numero_action, #numero_intervention)
- types_pieces(numero_type_piece, nom_type_piece)
- pieces(numero_serie_piece, #numero_type_piece, prix_piece, marque_piece)
- requete(#numero_action, #numero_type_piece, quantite)
- modeles_voiture(numero_modele, type_modele, marque_modele, version_modele, annee_modele)
- composition(#numero_serie_piece, #numero_modele)
- voitures(matricule_voiture, #numero_modele, #numero_securite_sociale, puissance_fiscale_voiture, cylindree_voiture, couleur_voiture, annee_circulation_voiture)
- adresses(numero_adresse, numero_habitation, nom_voie, code_postal, pays)
- personnes(numero_securite_sociale, #numero_adresse, nom_personne, prenom_personne, mail_personne, telephone_personne)
- garages(numero_SIREN, #numero_adresse, denomination_garage, forme_juridique_garage, date_creation_garage)
- tarif(#numero_action, #numero_modele, montant)

3.6 Expressions des dépendances fonctionnelles

Afin de modéliser les contraintes d'intégrités de notre base de données ainsi que les dépendances fonctionnelles, il nous faut réaliser un graphe de couverture minimale.

Un graphe de couverture minimale est un réseau que l'on obtient en représentant toutes les attributs ainsi que les dépendances fonctionnelles élémentaires entre ces attributs.

Les noms des entités sont omis et les dépendances fonctionnelles sont représentée par des flèches.

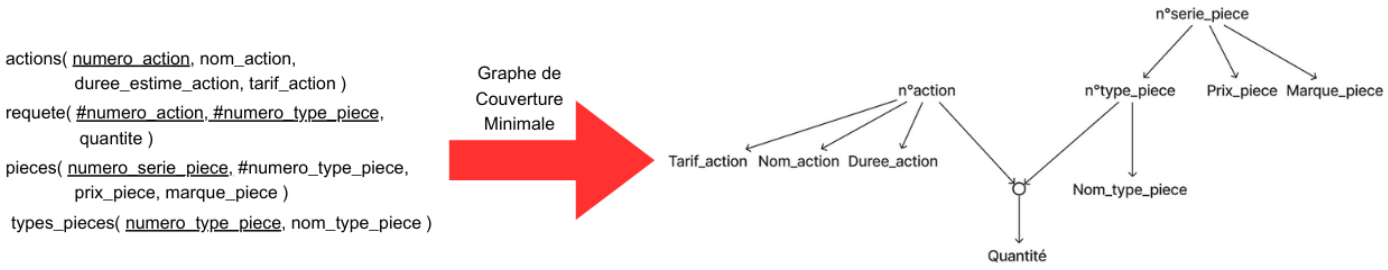

FIGURE 3.5 – Exemple d'un graphe de couverture minimale à partir d'un fragment de notre base de données

Dans l'exemple fournis ci-dessus (Figure 3.5), le noms des tables *actions*, *requetes*, *pieces* et *type_ pieces* n'apparaissent pas dans le graphe de couverture minimale.

Ce sont les clés primaires des tables qui définissent les attributs.

La table *requetes* est quand a elle transformée par un cercle qui est désignée par deux flèches qui partent de respectivement *n°action* et *n°type_piece* puisque ces deux clés étrangères forment la clé primaire de la table *requêtes*.

3.7 Contraintes d'intégrités et dépendances fonctionnelles dans notre base de données

Avec le schéma relationnel (fourni section 3.5), nous obtenons le graphe de couverture minimale suivant :

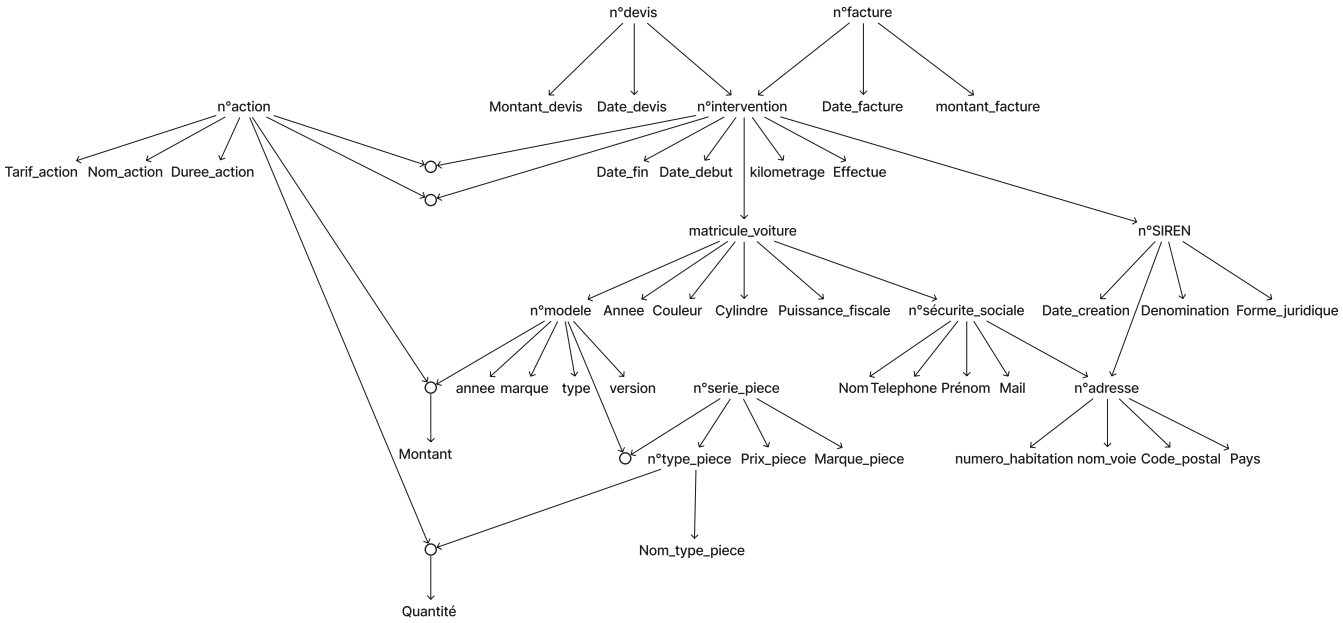

FIGURE 3.6 – Graphe de couverture minimale de notre base de données

3.8 Schéma relationnel en 3ème forme normale

Pour valider la structure de notre modèle relationnel, il faut nous assurer que ce dernier respecte bien les trois premières formes normales.

Une forme normale est un ensemble de règles qui vérifie la bonne structure de notre modèle relationnel.

Les formes normales sont classées par ordre de contraintes :

$$(n+1)^{\text{e}}\ forme\ normale = n^{\text{e}}\ forme\ normale + autres\ contraintes$$

Cela signifie qu'une forme normale est validée si le schéma relationnel valide la forme normale précédente ainsi que de nouvelles contraintes.

Par exemple, la 2^{e} forme normale est valide si la 1^{er} l'est et si des contraintes sont rajoutées.

3.8.1 1er forme normale

Soit la première forme normale. Elle stipule :

"Un attribut ne peut prendre qu'une valeur, et non pas un ensemble ou une liste de valeurs"

Cela signifie que les attributs ont une valeur dite *atomique* (non sécable).

Tout les attributs des tables de notre base de données sont *atomique*. En effet, aucun ne peut prendre plus d'une valeur.

Ainsi, la 1er forme normale est respectée dans notre schéma relationnel.

3.8.2 2e forme normale

Pour respecter la 2eforme normale, il faut respecter la 1er, ainsi, tout les attributs présents dans nos tables doivent être atomiques.

De plus, la 2eforme normale applique la contrainte suivante :

"La clé primaire d'une relation peut être composée de plusieurs champs mais les autres champs de la relation doivent dépendre de la clé en entier"

Ainsi, pour respecter la 2eforme normale, il faut que tout les attributs d'une table dépendent de l'entiereté de la clé primaire de cette table.

Il est évident que les tables ayant des clé primaires composé que d'un seul champ respecte cette contraintes.

Cependant, il faut le vérifier pour les autres tables qui possèdent des clés primaires dites "*composites*" (composée de plus d'un champ).

Nous possèdons dans notre schéma relationel (défini section 3.5), 5 tables qui possèdent des clés primaires composites.

- `actions_contenues( #numero_action, #numero_intervention )`
- `actions_survenues( #numero_action, #numero_intervention )`
- `composition( #numero_serie_piece, #numero_modele )`

Ces trois tables ont des clés primaires composites mais ne possèdents cependant pas d'autres attributs.

Elles respectent par conséquent la 2eforme normale, puisqu'aucun de leurs attributs ne dépend pas de la clé primaire dans son entièreté.

- `tarif( #numero_action, #numero_modele, montant )`
- `requete( #numero_action, #numero_type_piece, quantite )`

En ce qui concerne ces deux tables, elles possèdent des clés primaires composites ainsi qu'un autre attribut.

Pour la table *tarif*, l'attribut *montant* dépend bien de l'action concernée donc du champ *numero_ action* ainsi que du modèle de voiture concerné donc du champ *numero_ type_ piece*. Ainsi, l'attribut *montant* dépend bien de l'entièreté de la clé primaire composite de sa table.

Pour la table *requete*, l'attribut *quantite* dépend bien de l'action concernée donc du champ *numero_ action* ainsi que du type de pièce nécessaire donc du champ *numero_ type_ piece*. Ainsi, l'attribut *quantite* dépend bien de l'entièreté de la clé primaire composite de sa table.

Le schéma relationnel que nous avons proposé respecte donc bien la 2eforme normale.

3.8.3 3e forme normale

Pour respecter la 3eforme normale, il faut respecter les deux premières. Tout les attributs des tables doivent donc être atomique et ils doivent tous dépendre de l'entièreté de la clé primaire de leur table.

De plus, la 3eforme normale ajoute la contrainte suivante :

"Tous les attributs d'une entité doivent dépendre directement de son identifiant et d'aucun autre attribut"

Ainsi, pour respecter la 3eforme normale, il ne faut pas avoir de dépendances transitives entre les attributs d'une même table.

Lorsque nous avons construit notre schéma conceptuel ainsi que notre schéma relationnel, nous avions déjà l'objectif de respecter les 3 premières formes normales pour notre base de données. Ainsi, il n'y a aucune dépendance transitive entre les attributs de toutes les tables de notre schéma relationnel, la 3eforme normale est donc respectée.

4 Implémentation

Le but de notre projet est de créer une base de données SQL pour un client (garagiste qui souhaite informatiser toutes les informations nécessaires au bon fonctionnement de son entreprise).

Nous avons commencé par créer le schéma conceptuel (section 2), cela nous a permis en nous basant sur ce schéma d'obtenir le schéma relationnel (section 3).
À présent, nous allons nous intéresser à l'implémentation de cette base. En effet, après avoir beaucoup travaillé sur l'aspect théorique de notre base de donnée, il faut désormais concrétiser notre travail en implémentant notre base.

Nous allons tout d'abord commencer par voir comment nous avons créé cette base de donnée, puis nous verrons quelques commandes SQL que notre client pourra effectuer.

4.1 Création de la base de données et scripts d'opérations sur cette base

Avant d'effectuer des commandes SQL que notre client souhaite effectuer, nous devons créer la base de données.

Nous avons créé la base de données avec le système de gestion MySQL. Cette création est située dans le fichier *create.slq* (dans le dossier *sql/* de notre dépôt).

4.1.1 Scripts de création

Voici un exemple de création de table que nous avons implémenté :

```
1    CREATE TABLE pieces (
2         numero_serie_piece INTEGER NOT NULL,
3         numero_type_piece INTEGER NOT NULL,
4         prix_piece FLOAT,
5         marque_piece VARCHAR(50) NOT NULL,
6         PRIMARY KEY (numero_serie_piece)
7    );
```

Le mot clé *CREATE TABLE* nous permet la création de la table *piece* dans notre base de données.
Suite à cela, les attributs de notre table sont implémentés aux lignes [2-6], qui sont encadrés par des accolades qui délimitent la table.
On commence par donner le nom de l'attribut et on définit ensuite les contraintes (notés en bleu dans l'exemple). Par exemple, la ligne 91 définit l'attribut *prix_piece* qui a la contrainte d'être un flottant (*FLOAT*).

Pour définir la clé primaire de la table, nous avons utilisé le mot clé *PRIMARY KEY* que l'on utilise avant le champ que nous voulons définir comme clé primaire. Ici à la ligne 6, le champ *numero_serie_piece* est défini comme clé primaire de la table *pieces*.
Il est à noter que le champ de la clé primaire doit auparavant être défini comme attribut avant d'être défini comme clé primaire : *numero_serie_piece* est défini comme clé primaire à la ligne 6 de notre exemple, mais avait été déclaré comme attribut à la ligne 2.

Pour définir les clés étrangères (ce qui va être utile, car c'est le ce qui va permettre un lien entre nos tables) nous les avons définis après les créations de tables.

L'exemple ci-dessous nous montre l'affectation d'une clé étrangère à la table *piece* (dont la création est montré plus ci-dessus).

```
1    ALTER TABLE pieces ADD CONSTRAINT fk_pieces_types_pieces
FOREIGN KEY (numero_type_piece) REFERENCES types_pieces(numero_type_piece);
```

Le mot clé *ALTER TABLE* nous permet de modifier la table, le mot clé *ADD CONSTRAINT* quant à lui nous permet de nommer la contrainte que nous sommes en train d'ajouter.

Ainsi le début de la ligne de notre exemple "*ALTER TABLE pieces ADD CONSTRAINT fk_pieces_types_pieces*" nous montre que la modification va s'effectuer sur la table *piece* et va s'appeler *fk_pieces_types_pieces*.

Le mot clé *FOREIGN KEY* permet de définir une clé étrangère et le mot clé *REFERENCES* permet de faire une référence à un champ d'une table déjà existante.

La fin de la ligne de notre exemple :
"*FOREIGN KEY (numero_type_piece) REFERENCES types_pieces(numero_type_piece)*" nous indique que la modification est une définition de clé étrangère qui référence le champ *numero_type_piece* de la table *type_piece*.

4.1.2 Scripts de suppression

Le problème principal lors de la suppression de tables est la présence potentielle de liens entre les tables créés par les clés étrangères.

Afin de contrer ce problème, nous avons créé le fichier *drop.sql* dans le fichier *sql* de notre dépôt.

Ce fichier nous permet d'introduire le mot clé "*CASCADE*" que nous avons créé pour avoir un déclencheur de destructions de nos tables en évitant le problème de liens induit par les clés étrangères.

Si l'on utilise le mot *"CASCADE"* dans une ligne, le fichier *sql/drop.sql* le détectera et effectuera une suppression des tables de notre base de données.

4.1.3 Scripts d'insertion

Nous avons créé un script d'insertion permettant de copier des valeurs provenant de fichiers csv sur notre serveur. Cependant, les numéros de séquences permettant l'auto-incrémentation des *ID* de certaines ne se mettent pas à jour lors de l'insertion par *copy*. Nous avons donc dû rajouter dans ce script la ligne suivante :

```
SELECT setval('interventions_numero_intervention_seq', (SELECT max(numero_intervention) FROM int
```

par exemple ici pour la table *interventions* afin de pouvoir insérer par suite des lignes sans avoir d'erreurs.

4.2 Implémentation de quelques requêtes SQL

Pour répondre au besoin de manipulation de la base de donnée, notre système comporte différents types de requêtes implémentées. Ces requêtes sont implémentées dans les fichiers *sql/select.sql* et *sql/update.sql* Il y a tous d'abord les requêtes de consultation suivantes :

- Liste des interventions prévues dans les deux mois.
- Liste des modèles de véhicule pris en charge lors de l'année écoulée.
- Liste des clients ainsi liste de toutes les informations relatives à un client puis toutes les informations relatives aux véhicules de celui-ci.
- Liste de toutes les tables.
- Liste des interventions par véhicules et par client.
- Liste des actions et de leurs informations par intervention.

On a ensuite des requêtes de statistiques :

- Le nombre d'heures facturées par mois ainsi qu'une comparaison avec le mois précédent (en pourcentage).
- Le total facturé par mois ainsi qu'une comparaison avec le mois précédent (en pourcentage).
- La liste des types de véhicules, avec le type d'intervention majoritaire pratiqué sur ces véhicules.
- La liste des clients, avec le total des sommes facturées à chacun.

Pour finir, des requêtes de mise à jour :

- Ajout de donnée dans la base pour toutes les tables et toutes les colonnes.
- Modification des informations clients. (mise à jour du mail, du téléphone, etc.)
- Ajout d'interventions programmées et d'actions pour à réaliser durant celle-ci pour un véhicule donné.

5 Utilisation

5.1 Description de l'environnement d'exécution

La base de donnée est hébergée sur le serveur `server.rodriguez-esteban.com` et nous y accédons grâce à *Postgresql*. L'application est disponible sur *https ://esrodriguez.zzz.bordeaux-inp.fr*.

5.2 Notice d'utilisation

Pour déployer la base de donnée, il suffit de charger les fichiers suivants depuis `psql` :

1. `sql/create.sql`
2. `sql/select.sql`
3. `sql/update.sql`
4. `sql/insert.sql`

La base est alors prête à l'emploi avec son lot de données de tests.

La suppression de la base s'effectue en chargeant le fichier `sql/drop.sql`.

Il faudra, pour se connecter à la base, remplir les credentials de la base de donnée dans le fichier `src/auth_sample.php` et le renommer en `auth.php`.

5.3 Description des interfaces

La page d'accueil permet de choisir quel garage représente l'utilisateur de l'application (en réalité, ce choix ne change rien pour la suite, car nous n'avons pas encore mis en place le backend nécessaire, mais le faire fonctionner ne devrait pas être très contraignant).

Sur la page d'accueil, se trouvent les accès à quelques requêtes spécifiées par le sujet.

La barre de navigation permet de **Parcourir** les tables de la base de donnée, et de les modifier directement depuis l'application. Elle propose aussi un accès aux clients : les informations personnelles sont consultables et modifiables. De plus, les interventions (et actions contenues) sont modifiables pour chaque voiture détenue par un client.

Les résultats des requêtes sont mis en forme grâce au frontend mais souvent la table résultante brute est accessible (le bouton `liste des clients` dans l'onglet **Clients** par exemple.).

6 Conclusion

Nous sommes donc parvenus à construire une base de données pour l'entretien de véhicules d'un garage, ainsi qu'une page web permettant de visualiser cette base et de faire des opérations dessus.

Nous avons pu mettre en œuvre nos compétences acquises durant le module de "Systèmes de Gestion de Bases de Données" (IT203), notamment lors de la création d'un schéma conceptuel ainsi que de la formation d'un schéma relationnel en 3^{e}formes normales.

Ce projet nous a beaucoup appris, tant sur l'implémentation d'une base de données que sur la création d'une interface Web.