# révision-1147

Joanie Birtz

December 2025

## Exercice 1 : Gestion d'un tableau de voitures en PHP (Tableaux + Fonctions)

Dans cet exercice, vous devez manipuler un tableau de voitures en utilisant les tableaux associatifs, les fonctions, et quelques fonctions utilitaires de PHP. Chaque voiture possède les informations suivantes :

- **id** : identifiant unique (int)
- **marque** : fabricant (string)
- **modele** : modèle (string)
- **prix** : prix en dollars (float)
- **electrique** : voiture électrique ou non (bool)

### Objectifs

Vous devez compléter les fonctions suivantes :

1. **ajouterVoiture(...)** Ajouter une nouvelle voiture dans le tableau et retourner le tableau mis à jour.
2. **trouverVoitureParId(...)** Retourner la voiture qui possède l'id donné, ou `null` si elle n'existe pas.
3. **filtrerVoituresElectriques(...)** Retourner un nouveau tableau contenant uniquement les voitures électriques.
4. **trierVoituresParPrix(...)** Retourner les voitures triées selon leur prix (ordre croissant) en utilisant `usort()` et une fonction anonyme.
5. **printVoiture(...)** Afficher une liste de voitures dans un format clair. Gérer également le cas où la voiture est `null`.

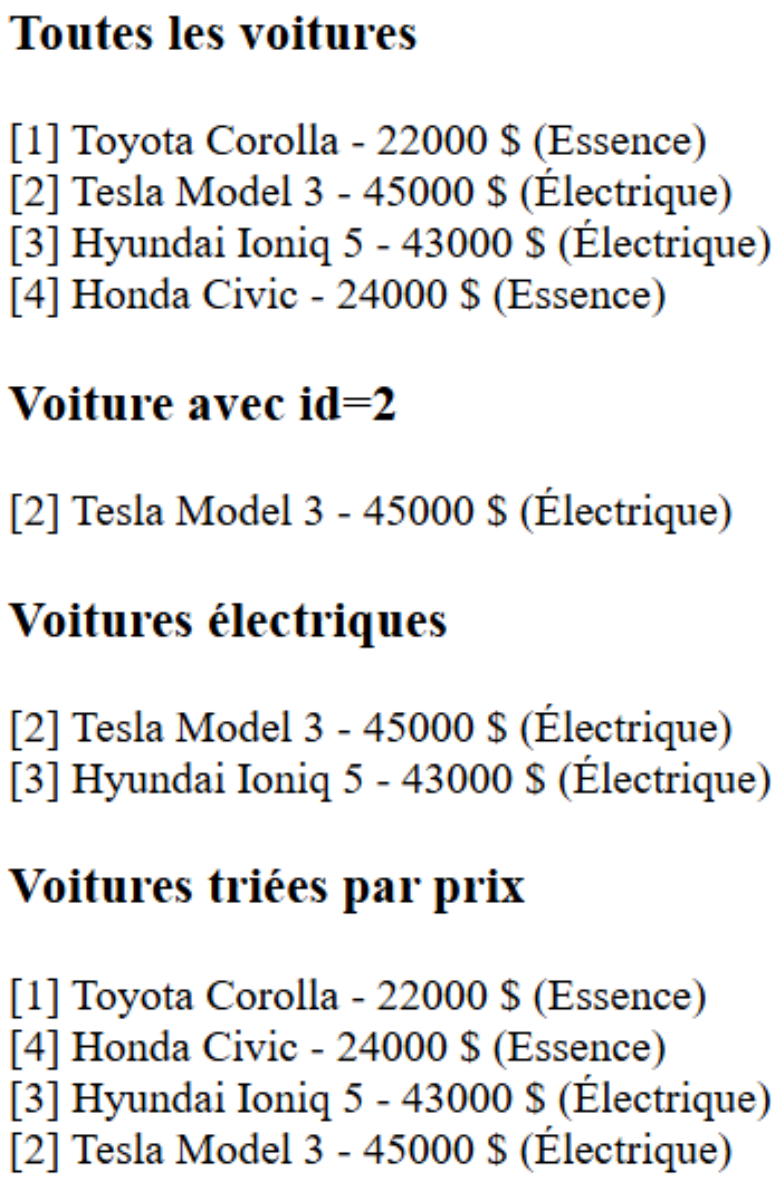


Figure 1: Résultats exercice 1

## Exercice 2 : Projet complet MVC–DAO – Gestion de voitures (inspiré de `refuge_animaux`)

Dans cet exercice, vous devez réaliser une petite application Web complète de gestion de voitures, en réutilisant la même structure que le projet `refuge_animaux_mvc_full_demo`

vu en cours.

L'application doit permettre à un administrateur de :

- lister les voitures présentes dans la base de données ;
- ajouter une nouvelle voiture ;
- modifier une voiture existante ;
- supprimer une voiture.

## Données et base de données

On utilise une base de données **MySQL/MariaDB** nommée `garage_voitures` qui contient une table `voiture` :

- `id_voiture` (INT, PK, AUTO_INCREMENT)
- `marque` (VARCHAR(100), NOT NULL)
- `modele` (VARCHAR(100), NOT NULL)
- `description` (TEXT, optionnel)
- `type_carburant` (ENUM(`ESSENCE`, `DIESEL`, `HYBRIDE`, `ELECTRIQUE`))
- `prix` (DECIMAL(10,2), NOT NULL)
- `date_arrivee` (DATE, NOT NULL)
- `photo` (VARCHAR(255), optionnel)

Un script SQL `voiture_table.sql` est fourni : vous devez simplement l'importer dans `phpMyAdmin` (ou un autre outil) pour créer la base et insérer quelques exemples de voitures.

## Structure du projet (même logique que `refuge_animaux`)

Vous devez reprendre la même organisation de fichiers/dossiers que le projet `refuge_animaux_mvc_full_demo`, en adaptant l'entité `animal` vers `voiture` :

- **client/** : partie front-end (HTML, CSS, JavaScript ES6 modules)
  - `client/public/pages/admin_voitures.html` : page d'administration (formulaire + tableau)
  - `client/public/voitures/RequetesVoitures.js` : fonctions d'appel à l'API (fetch vers `route.php`)
  - `client/public/voitures/VuesVoitures.js` : logique d'affichage (remplir le tableau, gérer le formulaire, boutons Modifier/Supprimer)
- **serveur/** : partie back-end (PHP, MVC, DAO, PDO)

- `serveur/includes/bd/Connexion.php` : classe Singleton de connexion PDO (réutiliser celle de `refuge_animaux`)
- `serveur/route.php` : routeur central, redirige les requêtes vers le bon contrôleur selon `entite` et `action`
- `serveur/Voiture/ModelVoiture.php` : classe modèle `ModelVoiture` (représente une ligne de la table `voiture`)
- `serveur/Voiture/VoitureDAO.php` : classe `VoitureDAO` (CRUD complet via PDO)
- `serveur/Voiture/ControleurVoiture.php` : contrôleur `ControleurVoiture` qui reçoit les actions (`liste`, `sauver`, `supprimer`), appelle le DAO et retourne du JSON
- `serveur/Voiture/photos/` : dossier où vous pourriez éventuellement stocker les photos de voitures (optionnel pour cette révision)

## Fonctionnalités à implémenter

### Côté serveur (PHP)

1. **Connexion PDO** Réutiliser la classe `Connexion` du projet `refuge_animaux` pour se connecter à la BD `garage_voitures`.
2. **DAO** `VoitureDAO` Implémenter les méthodes suivantes avec des requêtes préparées (`prepare()`, `bindValue()`, `execute()`) :
   - `lister()` : retourne un tableau de `ModelVoiture` (toutes les lignes de la table)
   - `trouver($id)` : retourne une voiture par son id ou `null`
   - `creer(ModelVoiture $v)` : insère une nouvelle voiture
   - `mettreAJour(ModelVoiture $v)` : met à jour une voiture existante
   - `supprimer(int $id)` : supprime une voiture
3. **Contrôleur** `ControleurVoiture` Le contrôleur doit :
   - lire les paramètres `_GET` / `_POST` (`entite`, `action`, données du formulaire) ;
   - appeler les méthodes du `VoitureDAO` correspondantes ;
   - retourner une réponse JSON au client, par exemple :

```
{
  "succes": true,
  "message": "Voiture ajoutée avec succès",
  "data": [ ... ]
}
```

**Côté client (JavaScript)**

1. **Requêtes API** (`RequetesVoitures.js`)
   - `apiListerVoitures()` : GET vers `route.php?entite=voiture&action=liste`
   - `apiSauverVoiture(formData, isEdition)` : POST vers `route.php` avec `entite=voiture`, `action=sauver`
   - `apiSupprimerVoiture(id)` : POST vers `route.php` avec `entite=voiture`, `action=supprimer`
2. **Vue et interactions** (`VuesVoitures.js`)
   - charger la liste des voitures au chargement de la page et remplir le tableau HTML ;
   - gérer la soumission du formulaire (`submit`) pour créer ou modifier une voiture ;
   - gérer les boutons **Modifier** et **Supprimer** dans chaque ligne du tableau ;
   - afficher des messages de succès / erreur dans une zone prévue (par exemple un `<div>` Bootstrap).

## Consignes

- Vous pouvez **copier la structure du projet** `refuge_animaux` et simplement renommer l'entité `animal` en `voiture`, puis adapter les champs.
- Respectez le découpage MVC–DAO : pas de requêtes SQL dans le JavaScript, pas d'affichage HTML dans le DAO.
- Testez vos requêtes (ajout, modification, suppression) et vérifiez que la base de données est bien mise à jour.