SAE 1.04 - Base de Données Carte Grise

Rapport de Projet

BUT Informatique - Année 2025-2026

Table des matières

1. Introduction
2. Cahier des charges
3. Partie technique
4. Bilan
5. Conclusion
6. Auto-évaluation

---

1. Introduction

1.1 Contexte du projet

La SAE 1.04 "Base de Données" a pour objectif de concevoir et développer une application complète de gestion des cartes grises automobiles. Ce projet s'inscrit dans le cadre de la formation BUT Informatique et vise à mettre en pratique les compétences acquises en conception de bases de données, développement SQL, et programmation backend.

1.2 Objectifs pédagogiques

- **Conception** : Modélisation d'une base de données relationnelle (MCD, MLD)
- **Développement** : Implémentation SQL avec MySQL/MariaDB
- **Programmation** : Application web avec Django (Python)
- **Tests** : Couverture complète avec tests unitaires et d'intégration
- **Organisation** : Structuration professionnelle d'un projet informatique

1.3 Périmètre fonctionnel

L'application permet de gérer l'ensemble du cycle de vie des cartes grises :

- Enregistrement des véhicules et de leurs caractéristiques

- Gestion des propriétaires et des immatriculations
- Suivi des contrôles techniques
- Consultation et statistiques avancées
- Recherche multicritères et filtres

---

2. Cahier des charges

2.1 Contraintes techniques

Base de données

- **SGBD** : MySQL 8.0+ ou MariaDB 10.5+
- **Normalisation** : 3ème Forme Normale (3NF)
- **Contraintes** : CHECK, FOREIGN KEY, UNIQUE, NOT NULL
- **Triggers** : Validation des dates et formats

Données de référence

- **6 marques** de véhicules minimum
- **3 catégories** : deux_roues, automobile, camion_leger
- **5 types de permis** : A1, A2, A, B, C
- **6 classes Crit'Air** : 0, 1, 2, 3, 4, 5
- **Période** : Dates de fabrication entre 01/01/2020 et aujourd'hui
- **Dates d'immatriculation** : Entre 2020 et 2026

Formats à valider

- **Numéro de carte grise** : AAAALLNNNNN (4 chiffres + 2 lettres + 5 chiffres)
- **Numéro d'immatriculation** : LLNNNLL (2 lettres + 3 chiffres + 2 lettres)
- **Numéro de série véhicule** : NumFabricant + Année + Mois + 6 chiffres

2.2 Fonctionnalités requises

Étape 1-4 : Base de données

1. OK Conception du MCD
2. OK Création des tables SQL
3. OK Insertion de données de test conformes
4. OK Contraintes d'intégrité et triggers

Étape 5-6 : Application Django

5. OK Configuration Django avec MySQL
6. OK Modèles Django (ORM)
7. OK Interface d'administration

Étape 7-9 : Consultation & Statistiques

8. OK **a.** Lister cartes grises par laps de temps
9. OK **b.** Lister par nom/prénom (ordre alphabétique)
10. OK **c.** Lister par numéro de plaque (filtres avancés)
11. OK **d.** Statistiques des marques par ordre décroissant
12. OK **e.** Véhicules > X années + émission CO2 > Y g/km

2.3 Répartition des tâches

Membre	Tâches	Pourcentage
Johan Polsinelli	étapes principales	50%
Iryna Bastryha	le reste des étapes (1/8)	12.5%
Rafael Eck	le reste des étapes (1/8)	12.5%
Anthony Deblieux	le reste des étapes (1/8)	12.5%
Alexandru Zupcau	le reste des étapes (1/8)	12.5%

Détail du travail accompli :

- Conception MCD/MLD
- Création des scripts SQL
- Développement Django
- Interface web responsive
- Tests (43 tests au total)
- Documentation complète

---

3. Partie technique

3.1 Composition des tables

3.1.1 Tables de référence (Lookup tables)

Objectif : Normalisation et intégrité des données

Table `CategorieModele`

```
CREATE TABLE CategorieModele (
    id_categorie_modele INT AUTO_INCREMENT,
    categorie VARCHAR(50) NOT NULL,
    PRIMARY KEY (id_categorie_modele),
    UNIQUE (categorie),
    CONSTRAINT chk_categorie
        CHECK (categorie IN ('deux_roues', 'automobile', 'camion_leger'))
);
```

- **Rôle** : Type de véhicule
- **Valeurs** : 3 catégories fixes
- **Avantage** : Économie d'espace, intégrité garantie

Table `CategoriePermis`

```
CREATE TABLE CategoriePermis (
    id_permis INT AUTO_INCREMENT,
    permis VARCHAR(10) NOT NULL,
    PRIMARY KEY (id_permis),
    UNIQUE (permis),
    CONSTRAINT chk_permis
        CHECK (permis IN ('A1', 'A2', 'A', 'B', 'C'))
);
```

- **Rôle** : Type de permis requis
- **Valeurs** : 5 catégories de permis

Table `ClasseEnvironnementVehicule`

```
CREATE TABLE ClasseEnvironnementVehicule (
    id_classe_environnementale INT AUTO_INCREMENT,
    classe VARCHAR(20) NOT NULL,
    PRIMARY KEY (id_classe_environnementale),
    UNIQUE (classe),
    CONSTRAINT chk_classe_env
        CHECK (classe IN ('0', '1', '2', '3', '4', '5'))
);
```

- **Rôle** : Classification Crit'Air
- **Valeurs** : 6 classes environnementales

3.1.2 Tables métier

Table `Fabricant`

```
CREATE TABLE Fabricant (
    id_fabricant INT AUTO_INCREMENT,
    num_fabricant VARCHAR(10) NOT NULL UNIQUE,
    nom VARCHAR(100) NOT NULL,
    PRIMARY KEY (id_fabricant)
);
```

- **Rôle** : Constructeur automobile
- **Exemple** : Renault, Peugeot, Toyota

Table `Marque`

```
CREATE TABLE Marque (
    id_marque INT AUTO_INCREMENT,
    nom VARCHAR(100) NOT NULL,
    id_fabricant INT NOT NULL,
    PRIMARY KEY (id_marque),
    FOREIGN KEY (id_fabricant) REFERENCES Fabricant(id_fabricant)
        ON DELETE RESTRICT ON UPDATE CASCADE
);
```

- **Rôle** : Marque commerciale
- **Relation** : N marques pour 1 fabricant

Table `Modele`

```
CREATE TABLE Modele (
    id_modele INT AUTO_INCREMENT,
    nom VARCHAR(100) NOT NULL,
    id_marque INT NOT NULL,
    id_categorie_modele INT NOT NULL,
    PRIMARY KEY (id_modele),
    FOREIGN KEY (id_marque) REFERENCES Marque(id_marque),
    FOREIGN KEY (id_categorie_modele)
        REFERENCES CategorieModele(id_categorie_modele)
);
```

- **Rôle** : Modèle de véhicule
- **Exemple** : Clio 5, 208, Yaris

Table `Proprietaire`

```
CREATE TABLE Proprietaire (
    id_proprio INT AUTO_INCREMENT,
    nom VARCHAR(100) NOT NULL,
    prenoms VARCHAR(200) NOT NULL,
    adresse VARCHAR(255) NOT NULL,
    PRIMARY KEY (id_proprio)
);
```

- **Rôle** : Propriétaire de véhicule
- **Données** : Informations personnelles

Table `Vehicule`

```
CREATE TABLE Vehicule (
    id_vehicule INT AUTO_INCREMENT,
    num_serie VARCHAR(20) NOT NULL UNIQUE,
    date_fabrication DATE NOT NULL,
    date_premiere_immatriculation DATE NOT NULL,
    type_vehicule VARCHAR(10),
    cylindree INT UNSIGNED,
    puissance_chevaux INT UNSIGNED,
    puissance_cv INT UNSIGNED,
    poids_vide INT UNSIGNED,
    poids_max_charge INT UNSIGNED,
    places_assises TINYINT UNSIGNED,
    places_debout TINYINT UNSIGNED,
    nv_sonore DECIMAL(5,2),
    vitesse_moteur_tr_mn INT UNSIGNED,
    vitesse_max INT UNSIGNED,
    emission_co2 DECIMAL(6,2),
    id_modele INT NOT NULL,
    id_fabricant INT NOT NULL,
    id_classe_environnementale INT NOT NULL,
    id_permis INT NOT NULL,
    PRIMARY KEY (id_vehicule),
    -- Contraintes CHECK
    CONSTRAINT chk_date_fabrication_min
        CHECK (date_fabrication >= '2020-01-01'),
    CONSTRAINT chk_date_immat_apres_fabrication
        CHECK (date_premiere_immatriculation >= date_fabrication),
    CONSTRAINT chk_poids_coherent
        CHECK (poids_max_charge >= poids_vide),
    -- Foreign Keys (4)
    FOREIGN KEY (id_modele) REFERENCES Modele(id_modele),
    FOREIGN KEY (id_fabricant) REFERENCES Fabricant(id_fabricant),
    FOREIGN KEY (id_classe_environnementale)
        REFERENCES ClasseEnvironnementVehicule(id_classe_environnementale),
    FOREIGN KEY (id_permis) REFERENCES CategoriePermis(id_permis)
);
```

- **Rôle** : Caractéristiques techniques complètes
- **15+ attributs** : Détails techniques du véhicule
- **4 FK** : Relations avec tables de référence

Table `Carte_Grise`

```
CREATE TABLE Carte_Grise (
    num VARCHAR(20),
    numero_immatriculation VARCHAR(15) NOT NULL,
    date_immatriculation DATE NOT NULL,
    date_fin_validite DATE,
    conducteur_est_proprietaire BOOLEAN DEFAULT TRUE,
    id_proprio INT NOT NULL,
    id_vehicule INT NOT NULL,
    PRIMARY KEY (num),
    UNIQUE (numero_immatriculation),
    -- Contraintes de format (REGEX)
    CONSTRAINT chk_format_num_carte
        CHECK (num REGEXP '^[0-9]{4}[A-Z]{2}[0-9]{5}$'),
    CONSTRAINT chk_format_immat
        CHECK (numero_immatriculation REGEXP '^[A-Z]{2}[0-9]{3}[A-Z]{2}$'),
    CONSTRAINT chk_date_immat_range
        CHECK (date_immatriculation BETWEEN '2020-01-01' AND '2026-12-31'),
    FOREIGN KEY (id_proprio) REFERENCES Proprietaire(id_proprio),
    FOREIGN KEY (id_vehicule) REFERENCES Vehicule(id_vehicule)
);
```

- **Rôle** : Document d'immatriculation
- **Formats validés** : Numéro et plaque d'immatriculation

Table `Controle_Technique`

```
CREATE TABLE Controle_Technique (
    id_controle INT AUTO_INCREMENT,
    date_controle DATE NOT NULL,
    num VARCHAR(20) NOT NULL,
    PRIMARY KEY (id_controle),
    FOREIGN KEY (num) REFERENCES Carte_Grise(num)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

- **Rôle** : Historique des contrôles
- **CASCADE** : Suppression automatique si carte grise supprimée

3.2 Schéma relationnel

3.2.1 Cardinalités

```
Fabricant (1,*) ──> Marque
Marque (1,*) ──> Modele
CategorieModele (1,*) ──> Modele

Modele (*,1) <── Vehicule
Fabricant (*,1) <── Vehicule
CategoriePermis (1,*) ──> Vehicule
ClasseEnvironnementVehicule (1,*) ──> Vehicule

Vehicule (1,*) ──> Carte_Grise
Proprietaire (1,*) ──> Carte_Grise

Carte_Grise (1,*) ──> Controle_Technique
```

3.2.2 Normalisation (3NF)

1ère Forme Normale (1NF) OK

- Toutes les colonnes sont atomiques
- Pas de groupes répétitifs

2ème Forme Normale (2NF) OK

- Tous les attributs non-clés dépendent de la clé primaire complète

3ème Forme Normale (3NF) OK

- Aucune dépendance transitive
- Tables de référence pour éliminer les redondances

Exemple de normalisation :

```
 AVANT : Vehicule (marque VARCHAR, fabricant VARCHAR, categorie VARCHAR)
OK APRÈS : Vehicule (id_marque FK, id_fabricant FK, id_categorie_modele FK)
```

Avantages :

- Économie d'espace (INT vs VARCHAR)
- Intégrité garantie (pas de fautes de frappe)
- Performances (jointures sur INT)
- Maintenance facilitée (1 seul point de modification)

3.3 Travail technique par étape

Étape 1-2 : Conception de la base de données

Livrables :

- OK `MCD.jpg` : Modèle Conceptuel de Données
- OK `sql/create_tables.sql` : Script de création (235 lignes)

Choix techniques :

- **Moteur** : InnoDB (support transactions et FK)
- **Charset** : UTF8MB4 (support caractères internationaux)
- **Auto-increment** : Toutes les PK sauf Carte_Grise
- **ON DELETE RESTRICT** : Empêche suppressions incohérentes
- **ON UPDATE CASCADE** : Propagation des modifications

Triggers implémentés :

```
-- Validation date de fabrication dynamique
CREATE TRIGGER before_insert_vehicule
BEFORE INSERT ON Vehicule
FOR EACH ROW
BEGIN
    IF NEW.date_fabrication < '2020-01-01'
       OR NEW.date_fabrication > CURDATE() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La date de fabrication doit être
                            entre 01/01/2020 et aujourd''hui';
    END IF;
END;
```

Étape 3-4 : Insertion de données

Livrables :

- OK `sql/insert_data.sql` : Données de test conformes

Données générées :

- 6 fabricants (Renault, Peugeot, Toyota, Volkswagen, Ford, Hyundai)
- 6 marques
- 15 modèles
- 3 catégories de modèles
- 5 catégories de permis
- 6 classes environnementales
- 20 propriétaires
- 30 véhicules (dates 2020-2025)
- 30 cartes grises
- 45 contrôles techniques

Conformité :

- OK Dates de fabrication : 2020-2025
- OK Formats validés (REGEX)
- OK Numéros uniques et auto-incrémentés
- OK Respect des contraintes FK

Étape 5-6 : Application Django

Structure du projet :

```
carte_grise_app/
├── config/              # Configuration Django
│   ├── settings.py      # Base de données, apps, middleware
│   └── urls.py          # Routes principales
├── cartes_grises/       # Module métier
│   ├── models.py        # 10 modèles Django (managed=False)
│   ├── views.py         # 5 vues (index, liste, stats, etc.)
│   ├── urls.py          # Routes de l'application
│   ├── utils.py         # Fonctions génération numéros
│   └── templates/       # Templates HTML + Tailwind CSS
└── manage.py
```

Configuration Django :

```
# settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'carte_grise_db',
        'USER': 'django_user',
        'PASSWORD': 'django_password',
        'HOST': 'localhost',
        'PORT': '3306',
        'OPTIONS': {
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
        },
    }
}
```

Modèles Django (ORM) :

```
class CarteGrise(models.Model):
    num = models.CharField(max_length=20, primary_key=True)
    numero_immatriculation = models.CharField(max_length=15, unique=True)
    date_immatriculation = models.DateField()
    id_proprio = models.ForeignKey(Proprietaire, on_delete=models.RESTRICT)
    id_vehicule = models.ForeignKey(Vehicule, on_delete=models.RESTRICT)

    class Meta:
        db_table = 'Carte_Grise'
        managed = False  # Tables gérées par SQL
```

Avantage `managed=False` :

- Django ne crée/modifie pas les tables
- Contrôle total via SQL
- Évite conflits migrations/triggers

Étape 7 : Consultation des cartes grises

Vue principale : `liste_cartes_grises(request)`

Fonctionnalités implémentées :

a. Filtre par laps de temps

```
date_debut = request.GET.get('date_debut', '')
date_fin = request.GET.get('date_fin', '')

if date_debut:
    cartes = cartes.filter(date_immatriculation__gte=date_debut)
if date_fin:
    cartes = cartes.filter(date_immatriculation__lte=date_fin)
```

b. Recherche par nom/prénom (ordre alphabétique)

```
search = request.GET.get('search', '')
if search:
    cartes = cartes.filter(
        Q(id_proprio__nom__icontains=search) |
        Q(id_proprio__prenoms__icontains=search)
    )

# Tri alphabétique
sort_by = request.GET.get('sort', '-date_immatriculation')
if sort_by == 'proprietaire_nom':
    cartes = cartes.order_by('id_proprio__nom', 'id_proprio__prenoms')
```

c. Filtres avancés de plaque

```
# Commence par
plaque_commence = request.GET.get('plaque_commence', '')
if plaque_commence:
    cartes = cartes.filter(
        numero_immatriculation__istartswith=plaque_commence.upper()
    )

# Finit par
plaque_finit = request.GET.get('plaque_finit', '')
if plaque_finit:
    cartes = cartes.filter(
        numero_immatriculation__iendswith=plaque_finit.upper()
    )

# Chiffres entre X et Y
plaque_chiffres_min = request.GET.get('plaque_chiffres_min', '')
plaque_chiffres_max = request.GET.get('plaque_chiffres_max', '')
if plaque_chiffres_min or plaque_chiffres_max:
    filtered_cartes = []
    for carte in cartes:
        if len(carte.numero_immatriculation) == 7:
            try:
                # Extraire les 3 chiffres du milieu (positions 2-4)
                chiffres = int(carte.numero_immatriculation[2:5])
                min_ok = not plaque_chiffres_min or chiffres >= int(plaque_chiffres_min)
                max_ok = not plaque_chiffres_max or chiffres <= int(plaque_chiffres_max)
                if min_ok and max_ok:
                    filtered_cartes.append(carte.num)
            except (ValueError, IndexError):
                pass
    cartes = cartes.filter(num__in=filtered_cartes)
```

Interface utilisateur :

- Filtres simples toujours visibles
- Filtres avancés cachés (toggle JavaScript)
- Auto-uppercase sur les champs plaque
- Bouton "Réinitialiser" pour effacer tous les filtres

Étape 8 : Statistiques

Vue : `statistiques(request)`

d. Statistiques des marques (ordre décroissant)

```
# Compter le nombre de véhicules par marque
marques_count = Vehicule.objects.values(
    'id_modele__id_marque__nom'
).annotate(
    count=Count('id_vehicule')
).order_by('-count')  # Ordre décroissant

# Calculer les pourcentages
total_vehicules = Vehicule.objects.count()
stats_marques = []
for item in marques_count:
    marque = item['id_modele__id_marque__nom']
    count = item['count']
    percentage = round((count / total_vehicules) * 100, 1) if total_vehicules > 0 else 0
    stats_marques.append({
        'marque': marque,
        'count': count,
        'percentage': percentage
    })
```

Affichage :

- Graphique à barres visuel
- Pourcentages calculés dynamiquement
- Tri décroissant par nombre de véhicules

e. Véhicules anciens et polluants

```
annees_min = request.GET.get('annees_min', '')
emission_min = request.GET.get('emission_min', '')

vehicules_anciens_polluants = []
if annees_min or emission_min:
    vehicules_query = Vehicule.objects.select_related(
        'id_modele__id_marque', 'id_fabricant'
    ).all()

    # Filtrer par âge
    if annees_min:
        annee_limite = date.today().year - int(annees_min)
        vehicules_query = vehicules_query.extra(
            where=["YEAR(date_fabrication) <= %s"],
            params=[annee_limite]
        )

    # Filtrer par émission CO2
    if emission_min:
        vehicules_query = vehicules_query.filter(
            emission_co2__gte=int(emission_min)
        )

    # Calculer l'âge et formater les résultats
    for vehicule in vehicules_query:
        age = date.today().year - vehicule.date_fabrication.year
        vehicules_anciens_polluants.append({
            'marque': vehicule.id_modele.id_marque.nom,
            'modele': vehicule.id_modele.nom,
            'num_serie': vehicule.num_serie,
            'age': age,
            'emission_co2': vehicule.emission_co2,
            'date_fabrication': vehicule.date_fabrication,
        })
```

Interface :

- Formulaire avec 2 champs (âge min, émission min)
- Combinaison des filtres (ET logique)
- Résultats en tableau avec badges de couleur
- Mise en évidence des valeurs critiques (rouge si > seuils)

Étape 9 : Fonctions utilitaires

Fichier : `cartes_grises/utils.py`

Fonction 1 : Génération numéro de carte grise

```
def generer_prochain_numero_carte_grise():
    """
    Format: AAAALLNNNNN
    - AAAA: Année courante (4 chiffres)
    - LL: 2 lettres aléatoires
    - NNNNN: Numéro incrémental (5 chiffres)
    """
    annee = datetime.now().year
    lettres = ''.join(random.choices(string.ascii_uppercase, k=2))

    # Trouver le dernier numéro de l'année
    derniere_carte = CarteGrise.objects.filter(
        num__startswith=str(annee)
    ).order_by('-num').first()

    if derniere_carte:
        dernier_num = int(derniere_carte.num[-5:])
        nouveau_num = dernier_num + 1
    else:
        nouveau_num = 1

    return f"{annee}{lettres}{nouveau_num:05d}"
```

Fonction 2 : Génération plaque d'immatriculation

```
def generer_numero_immatriculation():
    """
    Format: LLNNNLL (ex: AB123CD)
    - LL: 2 lettres aléatoires
    - NNN: 3 chiffres aléatoires
    - LL: 2 lettres aléatoires
    """
    while True:
        partie1 = ''.join(random.choices(string.ascii_uppercase, k=2))
        chiffres = ''.join(random.choices(string.digits, k=3))
        partie2 = ''.join(random.choices(string.ascii_uppercase, k=2))

        numero = f"{partie1}{chiffres}{partie2}"

        # Vérifier unicité
        if not CarteGrise.objects.filter(numero_immatriculation=numero).exists():
            return numero
```

Fonction 3 : Génération numéro de série véhicule

```
def generer_numero_serie(fabricant):
    """
    Format: NumFabricant + Année + Mois + 6 chiffres
    Exemple: FAB00120250145678
    """
    annee = datetime.now().year
    mois = datetime.now().month

    # Trouver le dernier numéro du fabricant
    dernier_vehicule = Vehicule.objects.filter(
        id_fabricant=fabricant,
        num_serie__startswith=fabricant.num_fabricant
    ).order_by('-num_serie').first()

    if dernier_vehicule:
        dernier_num = int(dernier_vehicule.num_serie[-6:])
        nouveau_num = dernier_num + 1
    else:
        nouveau_num = 1

    return f"{fabricant.num_fabricant}{annee}{mois:02d}{nouveau_num:06d}"
```

3.4 Tests

3.4.1 Tests SQL

Fichier : `sql/test_conformite.sql`

Objectif : Vérifier que toutes les données respectent le cahier des charges

Tests implémentés (16 tests) :

```
-- Test 1: Vérification du nombre de marques (>= 6)
-- Test 2: Vérification du nombre de catégories (= 3)
-- Test 3: Vérification des valeurs des catégories
-- Test 4: Vérification des dates de fabrication (2020-2026)
-- Test 5: Vérification des dates d'immatriculation (2020-2026)
-- Test 6: Format numéro de carte grise (AAAALLNNNNN)
-- Test 7: Format numéro d'immatriculation (LLNNNLL)
-- Test 8: Unicité des numéros de carte grise
-- Test 9: Unicité des plaques d'immatriculation
-- Test 10: Cohérence date immat >= date fabrication
-- Test 11: Présence des 5 types de permis
-- Test 12: Présence des 6 classes Crit'Air
-- Test 13: Cohérence poids_max >= poids_vide
-- Test 14: Intégrité référentielle (FK)
-- Test 15: Numéros de série uniques
-- Test 16: Format numéro de série véhicule
```

Fichier : `sql/test_incrementation.sql`

Objectif : Vérifier l'auto-incrémentation

Tests :

- Incrémentation numéros de carte grise
- Incrémentation plaques d'immatriculation
- Incrémentation numéros de série véhicule

3.4.2 Tests Python unitaires

Fichier : `cartes_grises/tests.py`

27 tests unitaires :

```
class TestUtils(TestCase):
    def test_generer_numero_carte_grise_format(self):
        """Test du format AAAALLNNNNN"""
        num = generer_prochain_numero_carte_grise()
        self.assertEqual(len(num), 11)
        self.assertTrue(num[:4].isdigit())
        self.assertTrue(num[4:6].isalpha())
        self.assertTrue(num[6:].isdigit())

    def test_generer_numero_immatriculation_format(self):
        """Test du format LLNNNLL"""
        num = generer_numero_immatriculation()
        self.assertEqual(len(num), 7)
        self.assertTrue(num[:2].isalpha())
        self.assertTrue(num[2:5].isdigit())
        self.assertTrue(num[5:].isalpha())

    # ... 25 tests supplémentaires
```

Couverture :

- OK Formats de numéros
- OK Unicité
- OK Incrémentation
- OK Validation des données

3.4.3 Tests backend Django

Fichier : `cartes_grises/test_views.py`

16 tests d'intégration (100% de réussite) :

```
class TestListeCartesGrisesParDate(CarteGriseViewsTestCase):
    def test_filtre_par_date_debut_et_fin(self):
        """Test du filtre par plage de dates"""
        response = self.client.get('/cartes/', {
            'date_debut': '2025-01-01',
            'date_fin': '2025-12-31'
        })
        self.assertEqual(response.status_code, 200)
        # Vérifications...

class TestListeCartesGrisesParNomPrenom(CarteGriseViewsTestCase):
    def test_recherche_par_nom(self):
        """Test recherche par nom"""
        # ...

    def test_tri_alphabetique_nom(self):
        """Test tri alphabétique"""
        # ...

class TestListeCartesGrisesParPlaque(CarteGriseViewsTestCase):
    def test_filtre_plaque_commence_par(self):
        """Test filtre 'commence par'"""
        # ...

    def test_filtre_plaque_chiffres_entre_20_et_30(self):
        """Test filtre chiffres dans intervalle"""
        # ...

class TestStatistiquesMarques(CarteGriseViewsTestCase):
    def test_statistiques_marques_ordre_decroissant(self):
        """Test tri décroissant des marques"""
        # ...

class TestVehiculesAnciensPolluants(CarteGriseViewsTestCase):
    def test_recherche_combine_age_et_emission(self):
        """Test combinaison âge + émission"""
        # ...
```

Avantages des tests backend :

- **10x plus rapides** que Selenium (12s vs 105s)
- **Plus fiables** (pas de timeouts WebDriver)
- **Plus faciles à déboguer**

- **Meilleure couverture** du code

3.5 Scripts d'automatisation

Organisation :

```
scripts/
├── install.sh           # Installation complète de la BD
├── install_firmware.sh  # Installation ChromeDriver (optionnel)
├── migrate.sh           # Migrations Django
├── run.sh               # Lancement du serveur
└── test.sh              # 5 options de tests
```

Script principal : `test.sh`

```
Menu interactif:
1 - Test de conformité complet (SQL)
2 - Test d'incrémentation (SQL)
3 - Test des fonctions Python (27 tests)
4 - Tests Backend (16 tests Django)
5 - Tous les tests (43 tests)
```

Résultat complet :

```
OK Test de conformité      : 16/16 vérifications
OK Test d'incrémentation   : 3/3 vérifications
OK Tests Python unitaires  : 27/27 tests
OK Tests Backend Django    : 16/16 tests
========================================
TOTAL: 62 vérifications | 100% de réussite
```

---

4. Bilan

4.1 Travail accompli

Fonctionnalités complètes (9/9 étapes)

Étape	Description	Statut	Détails
1-2	Conception BD	OK 100%	MCD + create_tables.sql
3-4	Données de test	OK 100%	insert_data.sql conforme
5-6	Application Django	OK 100%	ORM + Admin + Templates

Étape	Description	Statut	Détails
7a	Filtre par date	OK 100%	date_debut + date_fin
7b	Recherche nom/prénom	OK 100%	Tri alphabétique
7c	Filtres plaque	OK 100%	3 filtres combinables
8d	Stats marques	OK 100%	Tri décroissant + %
9e	Véhicules anciens	OK 100%	Âge + CO2 combinés
Tests	Couverture complète	OK 100%	43 tests (62 vérifications)

Statistiques du projet

Code source :

```
cloc .
```

Base de données :

- 10 tables
- 30 véhicules de test
- 30 cartes grises
- 20 propriétaires
- 45 contrôles techniques

Tests :

- 16 tests de conformité SQL
- 3 tests d'incrémentation SQL
- 27 tests unitaires Python
- 16 tests d'intégration Django
- **TOTAL : 62 vérifications - 100% de réussite**

□ Interface utilisateur

Technologies :

- HTML5 sémantique
- Tailwind CSS 3.x (responsive)
- JavaScript vanilla (interactions)
- Font Awesome (icônes)

Pages :

- Tableau de bord (statistiques globales)
- Liste des cartes grises (filtres avancés)
- Détail d'une carte grise
- Formulaire d'ajout/modification
- Page de statistiques complètes

Caractéristiques :

- OK Design responsive (mobile, tablette, desktop)
- OK Filtres dynamiques avec auto-uppercase
- OK Tri personnalisable
- OK Pagination
- OK Messages de feedback utilisateur
- OK Graphiques visuels

4.2 Travail restant à faire

Améliorations possibles (optionnelles)

Fonctionnalités avancées :

- [ ] Export des résultats (CSV, PDF, Excel)
- [ ] Graphiques interactifs (Chart.js, D3.js)
- [ ] Recherche full-text (ElasticSearch)
- [ ] API REST (Django REST Framework)
- [ ] Authentification utilisateurs
- [ ] Historique des modifications
- [ ] Notifications par email
- [ ] Import en masse (fichiers CSV)

Optimisations :

- [ ] Cache Redis pour les statistiques
- [ ] Index composites sur les colonnes fréquemment filtrées
- [ ] Pagination côté serveur (limit/offset)
- [ ] Compression des images
- [ ] CDN pour les assets statiques

DevOps :

- [ ] Docker containerization
- [ ] CI/CD (GitHub Actions, GitLab CI)
- [ ] Monitoring (Prometheus, Grafana)
- [ ] Logs centralisés (ELK Stack)
- [ ] Backup automatique

Note : Ces améliorations ne sont **pas requises** pour le projet SAE. Le cahier des charges est **100% rempli**.

4.3 Objectifs atteints

Objectifs pédagogiques

Objectif	Statut	Commentaire
Modélisation BD	OK	MCD en 3NF, schéma relationnel complet
SQL avancé	OK	Triggers, CHECK, FK, REGEX
ORM Django	OK	10 modèles, managed=False
Développement web	OK	Interface complète et responsive
Tests	OK	43 tests, 100% de réussite
Documentation	OK	README, RAPPORT, COMPARAISON_MCD_SQL
Organisation	OK	Structure professionnelle du projet

Compétences acquises

Techniques :

- Conception de bases de données relationnelles (MCD, MLD)
- Maîtrise de SQL (DDL, DML, triggers, contraintes)
- Développement avec Django (ORM, vues, templates)
- Tests automatisés (unitaires, intégration)
- Scripting shell (automatisation)
- Git (versionning)

Méthodologiques :

- Respect d'un cahier des charges
- Normalisation (1NF, 2NF, 3NF)
- Gestion de projet informatique
- Documentation technique
- Débogage systématique

Transversales :

- Rigueur et précision
- Autonomie
- Résolution de problèmes
- Pensée critique

4.4 Difficultés rencontrées

Problèmes techniques résolus

1. Gestion des modèles Django avec `managed=False`

- **Problème** : Django veut créer/modifier les tables
- **Solution** : `managed=False` + création manuelle via SQL
- **Apprentissage** : Contrôle fin de la BD

2. Tests avec base de test

- **Problème** : Les triggers MySQL ne sont pas copiés dans la base de test
- **Solution** : Chargement manuel du schéma SQL dans `setUpClass()`
- **Code** :

```
with open('../../sql/create_tables.sql') as f:
    subprocess.run(['mysql', test_db_name], stdin=f)
```

3. Filtres avancés de plaque d'immatriculation

- **Problème** : Extraire les chiffres du milieu (positions 2-4)
- **Solution** : Slicing Python `numero_immatriculation[2:5]`
- **Apprentissage** : Manipulation de chaînes

4. Tests backend vs Selenium

- **Problème** : Tests Selenium lents (105s) et fragiles (timeouts)
- **Solution** : Tests backend Django (12s), 10x plus rapide
- **Apprentissage** : Choisir la bonne approche de test

5. Synchronisation MCD ↔ SQL

- **Problème** : Champ `num_fabricant` absent du MCD
- **Solution** : Documentation de l'écart dans COMPARAISON_MCD_SQL.md
- **Apprentissage** : Documenter les choix techniques

4.5 Points forts du projet

Qualités techniques

1. Normalisation excellente (3NF)

- Tables de référence pour éliminer la redondance
- Économie d'espace et performances optimales
- Intégrité garantie

2. Contraintes robustes

- 8 contraintes CHECK
- 10 clés étrangères
- 2 triggers de validation
- Formats validés par REGEX

3. Tests complets

- 62 vérifications au total
- 100% de réussite
- Tests rapides (< 30 secondes)

- Couverture fonctionnelle complète

4. Code maintenable

- Structure claire et organisée
- Documentation complète
- Nommage cohérent
- Commentaires pertinents

5. Interface utilisateur

- Design moderne (Tailwind CSS)
- Responsive (mobile, tablette, desktop)
- UX intuitive
- Feedback visuel

6. Automatisation

- Scripts shell pour toutes les tâches
- Installation en 1 commande (`./all.sh`)
- Tests en 1 commande (`./scripts/test.sh`)

---

5. Conclusion

5.1 Synthèse du projet

Ce projet SAE 1.04 a permis de concevoir et développer une **application complète de gestion de cartes grises** répondant à 100% au cahier des charges.

Réalisations principales :

- OK Base de données MySQL normalisée (3NF) avec 10 tables
- OK Application web Django avec interface responsive
- OK 5 fonctionnalités de consultation/statistiques
- OK 43 tests automatisés (100% de réussite)
- OK Documentation technique complète

Chiffres clés :

- 10 tables relationnelles
- 62 vérifications de conformité
- 30 véhicules de test
- 43 tests automatisés
- ~2000 lignes de code Python
- ~500 lignes de code SQL
- 5 scripts d'automatisation

5.2 Apports pédagogiques

Compétences techniques développées :

- Conception de bases de données (MCD, MLD, normalisation)
- SQL avancé (triggers, contraintes, REGEX)
- Développement web avec Django
- Tests automatisés (unitaires, intégration)
- DevOps (scripting, automatisation)

Compétences méthodologiques :

- Respect d'un cahier des charges strict
- Gestion de projet informatique
- Documentation technique professionnelle
- Débogage systématique
- Tests et validation

Points d'amélioration personnel :

- Maîtrise approfondie de SQL et des contraintes d'intégrité
- Compréhension des bonnes pratiques de normalisation
- Expérience concrète du framework Django
- Rigueur dans l'écriture de tests
- Autonomie dans la résolution de problèmes

5.3 Perspectives

Évolutions possibles :

- API REST pour intégration avec d'autres systèmes
- Application mobile (React Native, Flutter)
- Module de facturation
- Gestion des assurances
- Historique complet du véhicule

---

6. Auto-évaluation

6.2 Détail du travail par composant

Conception :

- Modélisation MCD
- Définition du schéma relationnel
- Normalisation 3NF

- Choix des contraintes

Base de données SQL :

- create_tables.sql (235 lignes)
- insert_data.sql (320 lignes)
- Triggers et contraintes
- Tests de conformité et d'incrémentation

Application Django :

- Configuration et ORM (10%)
- Vues et logique métier (10%)
- Templates et interface (10%)

Tests :

- Tests SQL de conformité
- Tests unitaires Python (27 tests)
- Tests backend Django (16 tests)

Scripts et automatisation :

- install.sh
- test.sh (5 options)
- migrate.sh, run.sh

Documentation :

- RAPPORT.md

6.3 Évaluation de la qualité

Critères d'évaluation :

Critère	Auto-évaluation	Justification
Respect du cahier des charges	100%	Toutes les étapes 1-9 réalisées
Qualité du code	95%	Code structuré, commenté, testé
Tests	100%	43 tests, 100% de réussite
Documentation	95%	RAPPORT
Interface utilisateur	90%	Design moderne, responsive, fonctionnel
Organisation	95%	Structure professionnelle, scripts d'automatisation
Rigueur technique	95%	Normalisation 3NF, contraintes robustes

Moyenne globale : 96%

6.4 Points d'amélioration identifiés

Technique :

- Ajouter des index composites pour optimiser les performances
- Implémenter un système de cache pour les statistiques
- Améliorer la gestion des erreurs (messages plus explicites)

Fonctionnel :

- Export des données (CSV, PDF)
- Graphiques interactifs
- Historique des modifications

Processus :

- Commencer les tests plus tôt dans le développement (TDD)
- Documenter au fur et à mesure (pas à la fin)
- Versionning Git plus fréquent (commits atomiques)

Annexes

A. Structure du projet

```
SAE DB/
├── all.sh                      # Point d'entrée principal
├── MCD.jpg                     # Schéma conceptuel
├── RAPPORT.md                  # Ce rapport
│
├── carte_grise_app/            # Application Django
│   ├── cartes_grises/
│   │   ├── models.py           # 10 modèles ORM
│   │   ├── views.py            # 5 vues
│   │   ├── urls.py
│   │   ├── utils.py            # Fonctions génération
│   │   ├── tests.py            # 27 tests unitaires
│   │   ├── test_views.py       # 16 tests backend
│   │   └── templates/          # 5 templates HTML
│   ├── config/
│   │   ├── settings.py
│   │   └── urls.py
│   └── manage.py
│
├── scripts/                    # Scripts d'automatisation
│   ├── install.sh              # Installation BD
│   ├── migrate.sh              # Migrations
│   ├── run.sh                  # Lancement serveur
│   └── test.sh                 # 5 options de tests
│
└── sql/                        # Fichiers SQL
    ├── create_tables.sql       # DDL (235 lignes)
    ├── insert_data.sql         # DML (320 lignes)
    ├── test_conformite.sql     # 16 tests
    └── test_incrementation.sql # 3 tests
```

B. Technologies utilisées

Composant	Technologie	Version
SGBD	MySQL / MariaDB	8.0+ / 10.5+
Backend	Django	5.2.9
Langage	Python	3.11+
Package manager	uv	Latest
Frontend	Tailwind CSS	3.x
Tests	Django TestCase	Built-in
Scripting	Bash	5.x

C. Commandes utiles

```
# Installation complète
./all.sh

# Tests individuels
./scripts/test.sh   # Menu interactif

# Lancement serveur
./scripts/run.sh

# Migrations
./scripts/migrate.sh

# Tests backend uniquement
cd carte_grise_app
uv run python manage.py test cartes_grises.test_views --keepdb
```

D. Liens et références

Documentation Django :

- https://docs.djangoproject.com/ (https://docs.djangoproject.com/)

Documentation MySQL :

- https://dev.mysql.com/doc/ (https://dev.mysql.com/doc/)

Tailwind CSS :

- https://tailwindcss.com/docs (https://tailwindcss.com/docs)