

Système de Gestion d'une Institution Éducative

Pour ce projet voici ma conception d'une base de données SQLite pour gérer une université. Le système permet de suivre:

- Les départements académiques
- Les enseignants et leurs spécialités
- Les cours offerts et leurs crédits associés
- Les étudiants et leurs informations personnelles
- Les inscriptions aux cours et les notes obtenues
- Les salles disponibles et leurs capacités
- La planification des séances de cours

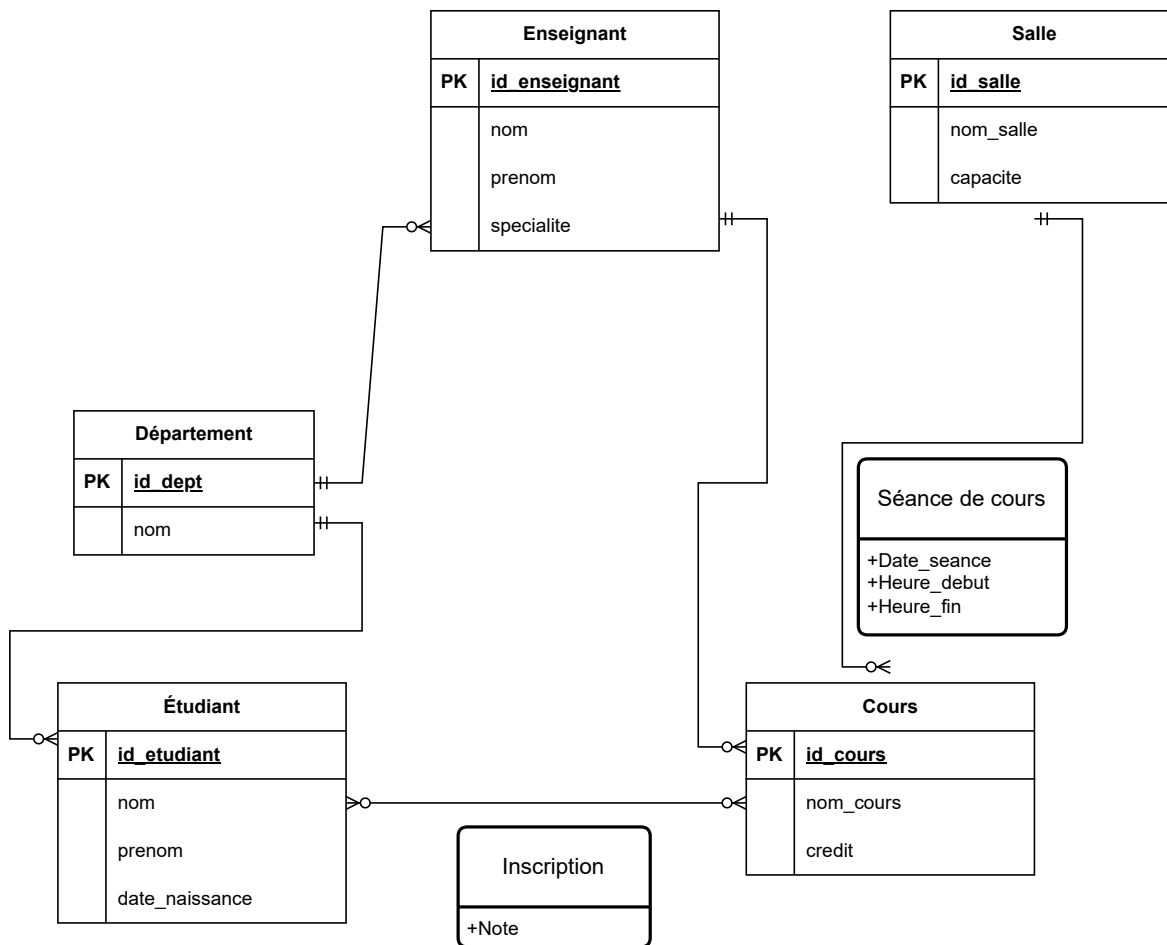
Le schéma comprend 7 tables:

- Departement
- Enseignant
- Cours
- Etudiant
- Inscription
- Salle
- Seance

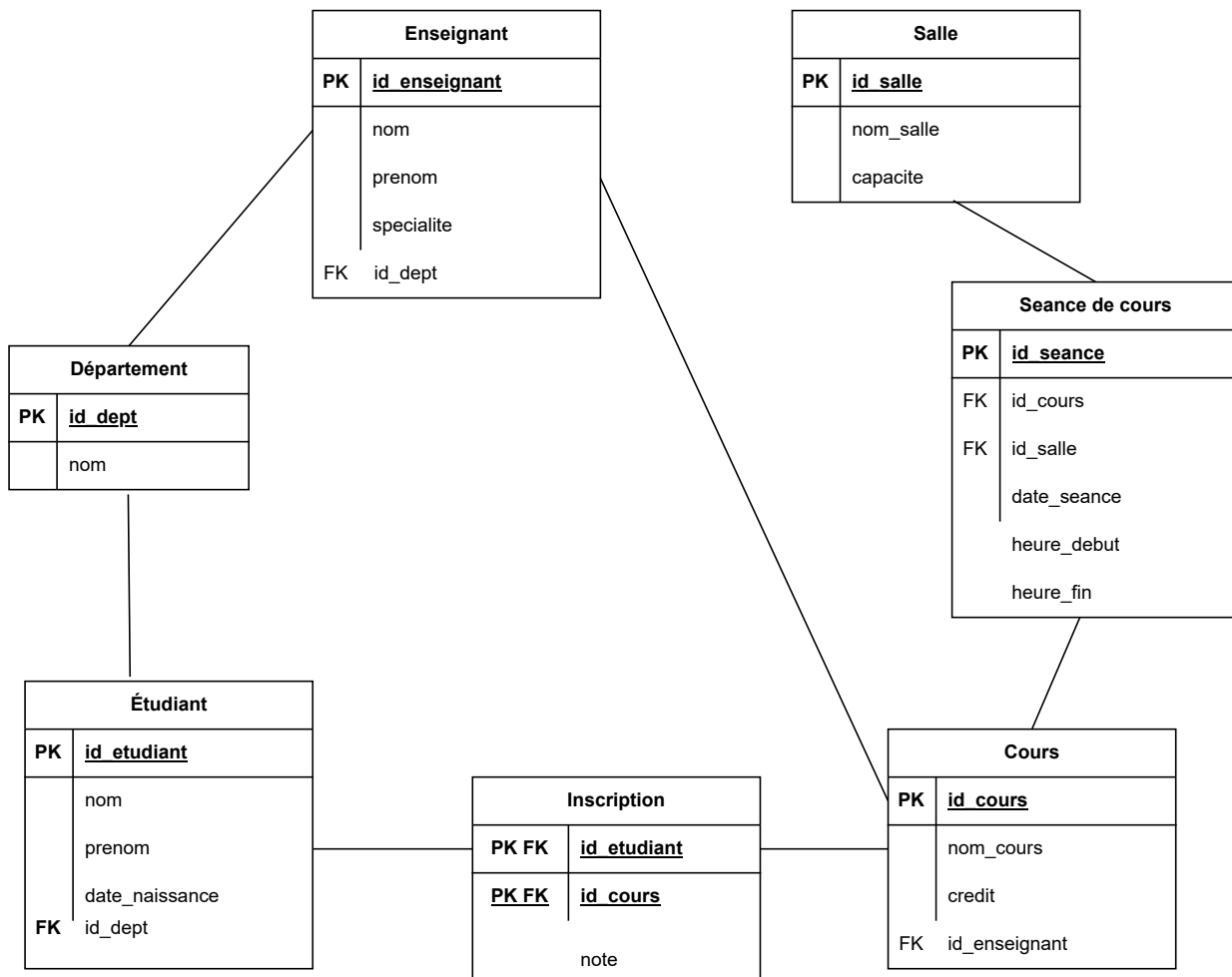
Ce système pourrait être utilisé pour la planification des cours, avec sa salle, son professeur, et ses élèves, au sein d'un département.

1. Modèles

Entité Association



Relations



2. Implémentation SQLite

2.1 Structure et peuplement

Le fichier `ecole-sqlite.sql` contient :

- 7 tables avec clés primaires et étrangères
- 1 trigger pour éviter les chevauchements de séances
- Population avec données de test réalistes

2.2 Trigger implémenté

```
CREATE TRIGGER verifier_disponibilite_salle
BEFORE INSERT ON Seance
FOR EACH ROW
BEGIN
    SELECT RAISE(ABORT, 'La salle est déjà réservée à ce moment')
    WHERE EXISTS (
        SELECT 1 FROM Seance
        WHERE id_salle = NEW.id_salle
        AND date_seance = NEW.date_seance
        AND (conditions de chevauchement horaire)
    );
END;
```

2.3 Requêtes SQL (15 requêtes testées)

Requêtes sans jointure (2) :

1. **Étudiants en informatique** : `SELECT * FROM Etudiant WHERE id_dept = 1`
2. **Salles capacité > 30** : `SELECT * FROM Salle WHERE capacite > 30`

Requêtes avec jointures (4) :

3. **Cours > 4 crédits avec enseignants** : Jointure INNER Cours-Enseignant
4. **Étudiants note > 16** : Jointure INNER Inscription-Etudiant-Cours
5. **Nombre d'étudiants par cours** : Jointure LEFT Cours-Inscription
6. **Jointure complète Salles-Séances** : UNION de LEFT JOINS

Requêtes avancées :

7. **Avec NOT IN** : Salles disponibles à un créneau donné
8. **Avec EXISTS** : Enseignants ayant au moins un cours
9. **Regroupement** : Moyenne des notes par cours
10. **Regroupement + HAVING** : Cours avec moyenne > 15
11. **Sous-requête** : Meilleur étudiant par cours
12. **Sous-requête** : Total crédits par étudiant
13. **ORDER BY** : Enseignants triés par nombre de cours
14. **Modification** : Mise à jour note d'un étudiant
15. **Suppression** : Suppression inscriptions note < 14

3. Migration vers MongoDB

3.1 Processus d'exportation

Le script `sqlite-to-mongodb.py` :

- Extrait les données SQLite vers format JSON
- Gère la conversion des types DATE/TIME
- Produit 7 fichiers JSON pour import MongoDB

3.2 Commandes d'importation

```
mongoimport --db ecole --collection Departements --file departement.json --jsonArray
mongoimport --db ecole --collection Enseignants --file enseignant.json --jsonArray
mongoimport --db ecole --collection Cours --file cours.json --jsonArray
mongoimport --db ecole --collection Etudiants --file etudiant.json --jsonArray
mongoimport --db ecole --collection Inscriptions --file inscription.json --jsonArray
mongoimport --db ecole --collection Salles --file salle.json --jsonArray
mongoimport --db ecole --collection Seances --file seance.json --jsonArray
```

3.3 Requêtes MongoDB implémentées

1. Insertion

```
db.Etudiants.insertOne({
  id_etudiant: 8,
  nom: "Dubois",
  prenom: "Alice",
  date_naissance: new Date("2002-07-19"),
  id_dept: 3
});
```

2. Modification

```
db.Enseignants.updateOne(
  { nom: "Martin", prenom: "Sophie" },
  { $set: { id_dept: 3 } }
);
```

3. Suppression

```
db.Salles.deleteMany({ capacite: { $lt: 30 } });
```

4. Recherches avec projection/sélection

```
// Étudiants nés après 2000
db.Etudiants.find(
  { date_naissance: { $gt: new Date("2000-12-31") } },
  { nom: 1, prenom: 1, date_naissance: 1 }
);

// Cours avec plus de 4 crédits
db.Cours.find(
  { credit: { $gt: 4 } },
  { nom_cours: 1, credit: 1 }
);
```

5. Agrégation - Moyenne par cours

```
db.Inscriptions.aggregate([
  {
```

```

        $lookup: {
            from: "Cours",
            localField: "id_cours",
            foreignField: "id_cours",
            as: "cours"
        }
    },
    { $unwind: "$cours" },
    {
        $group: {
            _id: "$cours.nom_cours",
            moyenne: { $avg: "$note" }
        }
    }
}
});

```

6. MapReduce - Total crédits par étudiant

```

var mapFunction = function() {
    emit(this.id_etudiant, this.credit);
};

var reduceFunction = function(key, values) {
    return Array.sum(values);
};

db.temp_inscriptions_credits.mapReduce(
    mapFunction,
    reduceFunction,
    { out: "credits_etudiants" }
);

```

4. Instructions de test

SQLite :

1. Exécuter `sqlite3 ecole.db < ecole-sqlite.sql`
2. Les 15 requêtes s'exécutent automatiquement avec résultats

MongoDB :

1. Lancer MongoDB : `mongod --dbpath="chemin/vers/db"`
2. Importer les données avec les commandes `mongoimport`
3. Exécuter : `mongosh ecole --quiet requests.js`
4. Visualiser avec MongoDB Compass

5. Fichiers livrés

- `ecole-sqlite.sql` : Base SQLite complète avec requêtes
- `sqlite-to-mongodb.py` : Script de conversion
- `*.json` : Fichiers de données pour MongoDB
- `requests.js` : Requêtes MongoDB

Note : DeprecationWarning: `Collection.mapReduce()` est déprécié !