Projet Systèmes de Bases de Données

EL HAKOUNI Yassin
05/06/2025

1. Introduction

1.1 Contexte

La restauration rapide et la livraison à domicile reposent aujourd'hui sur des systèmes d'information capables de suivre en temps réel les menus proposés, les commandes des clients et la composition détaillée de chaque plat. L'objectif de ce travail pratique est de concevoir puis de mettre en œuvre un tel système — d'abord sous SQLite, un SGBD relationnel embarqué, puis de le migrer vers MongoDB, SGBD orienté documents très utilisé pour les applications web temps réel.

1.2 Objectifs

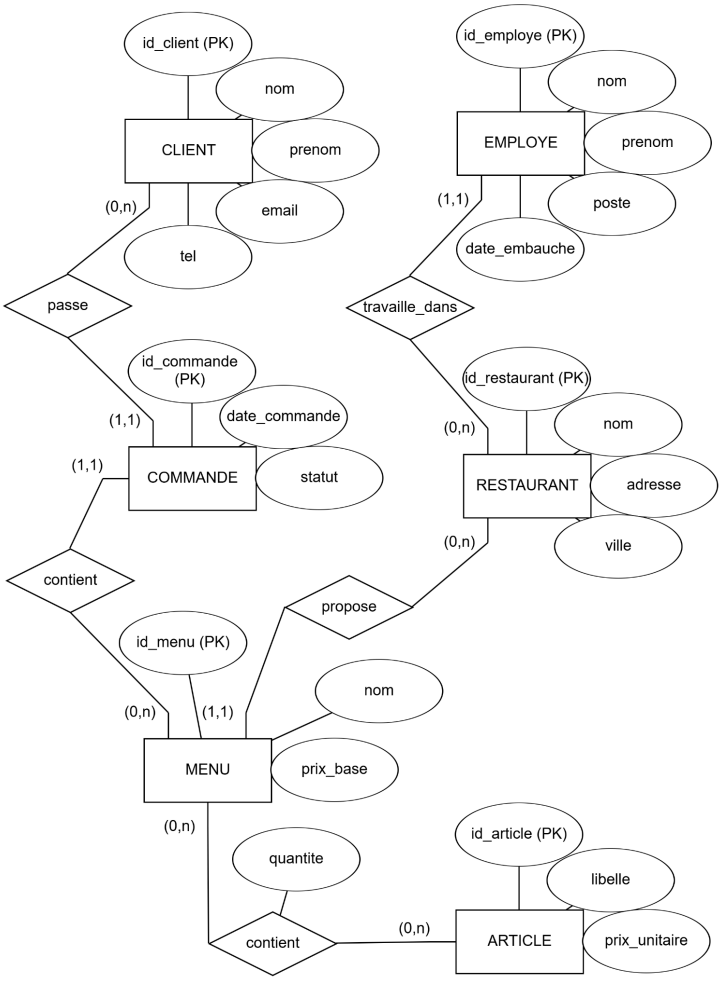
Les deux parties du projet s'enchaînent pour couvrir l'ensemble du cycle de vie d'une base :

- Modélisation relationnelle
    - passer d'un diagramme Entité-Association complet aux tables physiques ;
    - identifier les limites fonctionnelles (ex. : pas d'historique des prix, stock non géré).
- Implémentation SQLite
    - créer au moins six tables et leurs contraintes d'intégrité ;
    - écrire un déclencheur qui recalcule automatiquement le prix d'un menu ;
    - insérer un jeu d'essai cohérent et vérifier quinze requêtes couvrant :
      – filtres simples,
      – jointures diverses (inner, gauche, « full » simulée),
      – IN / EXISTS,
      – agrégats avec HAVING,
      – sous-requêtes, tri, mise à jour et suppression.

- Migration vers MongoDB
    - exporter les données SQLite au format JSON et les importer via `mongoimport` ;
    - réécrire les principales opérations (insertion, mise à jour, suppression) en style MongoDB ;
    - réaliser deux requêtes de recherche avec projection, une agrégation `pipeline` et un Map-Reduce, en montrant comment les exécuter et interpréter leurs résultats.

2. Modèle Entité-Association du projet

Le modèle couvre les besoins essentiels d'un service de prise de commandes pour restaurants : clients, restaurants, employés, menus, articles et commandes.



La relation Commande ↔ Menu est matérialisée par l'entité associative COMMANDE_MENU ; elle trace quels menus composent chaque commande.

La composition Menu ↔ Article passe par l'entité associative MENU_ARTICLE avec un attribut

quantité (≥ 1) ; un même article peut apparaître dans plusieurs menus et réciproquement.

Le prix d'un menu n'est pas saisi : un déclencheur recalcule automatiquement prix_base dès qu'on ajoute, modifie ou retire un article du menu.

Un employé travaille dans un seul restaurant ; le modèle ne conserve pas l'historique des transferts éventuels.

Si un restaurant est supprimé, tous ses menus, leurs liaisons et les lignes de commandes correspondantes disparaissent en cascade ; la suppression d'un client est bloquée s'il possède encore des commandes.

Le modèle ne gère pas :

– les stocks d'articles ni leurs fournisseurs ;

– les remises, taxes ou moyens de paiement ;

– l'historique des changements de prix ;

– plusieurs adresses ou succursales pour un même restaurant.

3. Modèle Relationnel

RESTAURANT(id_restaurant PK, nom, adresse, ville)

EMPLOYE(id_employe PK, id_restaurant FK → RESTAURANT, nom, prenom, poste, date_embauche)

CLIENT(id_client PK, nom, prenom, email UNIQUE, tel)

ARTICLE(id_article PK, libelle, prix_unitaire)

MENU(id_menu PK, id_restaurant FK → RESTAURANT, nom, prix_base -- calculé par trigger)

COMMANDE(id_commande PK, id_client FK → CLIENT, date_commande, statut)

COMMANDE_MENU(-- table associative id_commande PK FK → COMMANDE ON DELETE CASCADE, id_menu PK FK → MENU ON DELETE CASCADE)

MENU_ARTICLE(-- table associative id_menu PK FK → MENU ON DELETE CASCADE, id_article PK FK → ARTICLE, quantite CHECK (quantite ≥ 1))

Contraintes, règles métier & limitations

- Restaurant – Menu : relation 1 : N
  un restaurant peut proposer 0,n menus ; chaque menu appartient à un seul restaurant.

- Restaurant – Employé : 1 : N
  un employé travaille dans un seul restaurant.

- Client – Commande : 1 : N
  un client peut passer plusieurs commandes ; chaque commande appartient à un seul

client.

- Commande – Menu : relation 1 : N matérialisée par la table COMMANDE_MENU
  clé primaire composite *(id_commande, id_menu)* ; suppression d'une commande ⇒ suppression en cascade des lignes enfants.

- Menu – Article : relation N : N via MENU_ARTICLE
  clé primaire composite *(id_menu, id_article)* ; attribut quantite ≥ 1 grâce au `CHECK`.

- Article : `prix_unitaire` est un REAL (prix décimal).
  Aucune contrainte d'unicité sur le libellé pour autoriser plusieurs tailles/variantes.

- MENU.prix_base n'est pas saisi ; il est mis à jour par le trigger `maj_prix_menu`
  (recalcul après chaque INSERT/UPDATE/DELETE dans MENU_ARTICLE).

- INTÉGRITÉ RÉFÉRENTIELLE activée par `PRAGMA foreign_keys = ON;`.
  Toutes les FK sont en `ON DELETE CASCADE` lorsqu'elles sont sur des tables d'association, pour éviter les orphelins.

- CLIENT.email unique : empêche la création de doublons.

- Pas d'historique des menus supprimés : si un restaurant disparaît, ses menus et les liaisons correspondantes sont effacés (cascade).

4. Implémentation et tests SQLite

Fichier `restaurant.sql` (dans l'archive *sources.zip* jointe).

Exécution :

```
sqlite3 restaurant.db < restaurant.sql
```

Ce script :

- crée les 8 tables, active `PRAGMA foreign_keys = ON;`, installe le déclencheur maj_prix_menu ;
- renseigne 3 restaurants, 5 clients, 4 articles, 2 menus, 4 employés et 2 commandes.

Les 15 requêtes ci-dessous ont alors été lancées dans DB Browser for SQLite et donnent toutes un résultat.

```
/* 1. Sans jointure + WHERE  : articles à plus de 10 € */

SELECT *

FROM   ARTICLE

WHERE  prix_unitaire > 10;

/* 2. Sans jointure + WHERE  : clients Gmail */
```

```
SELECT nom, email

FROM   CLIENT

WHERE  email LIKE '%@gmail.com';

/* 3. INNER JOIN (1 jointure) + WHERE : menu  ↔ restaurant */

SELECT m.id_menu,

    m.nom AS menu,

    r.nom AS restaurant

FROM   MENU       m

JOIN   RESTAURANT  r USING(id_restaurant)

WHERE  m.prix_base IS NOT NULL;

/* 4. INNER JOIN (2 jointures) + WHERE : commande ➜ client + menu */

SELECT c.id_commande,

    cl.nom AS client,

    m.nom  AS menu

FROM   COMMANDE      c

JOIN   CLIENT        cl ON cl.id_client = c.id_client

JOIN   COMMANDE_MENU   cm USING(id_commande)

JOIN   MENU          m  USING(id_menu)

WHERE  c.statut = 'en cours';

/* 5. LEFT JOIN + WHERE : restaurants sans menu */

SELECT r.id_restaurant,

    r.nom AS restaurant,

    m.nom AS menu

FROM   RESTAURANT r

LEFT   JOIN MENU m USING(id_restaurant)

WHERE  m.id_menu IS NULL;

/* 6. “FULL JOIN” simulé (UNION de deux LEFT) + WHERE */

SELECT r.id_restaurant,

    r.nom  AS restaurant,

    m.nom  AS menu
```

```
FROM   RESTAURANT r
LEFT   JOIN MENU m USING(id_restaurant)
WHERE  m.id_menu IS NULL              -- restos sans menu
UNION
SELECT r.id_restaurant,
    r.nom  AS restaurant,
    m.nom  AS menu
FROM   MENU m
LEFT   JOIN RESTAURANT r USING(id_restaurant)
WHERE  r.id_restaurant IS NULL;       -- menus orphelins
/* 7. IN : menus de la commande 1 */
SELECT nom
FROM   MENU
WHERE  id_menu IN (SELECT id_menu
            FROM   COMMANDE_MENU
            WHERE  id_commande = 1);
/* 8. EXISTS : restaurants ayant au moins un employé */
SELECT nom
FROM   RESTAURANT r
WHERE  EXISTS (SELECT 1
          FROM   EMPLOYE e
          WHERE  e.id_restaurant = r.id_restaurant);
/* 9. GROUP BY : nombre de menus par restaurant */
SELECT id_restaurant,
    COUNT(*) AS nb_menus
FROM   MENU
GROUP  BY id_restaurant;
/* 10. GROUP BY + HAVING : restaurants avec > 3 employés */
SELECT id_restaurant,
    COUNT(*) AS nb_emp
```

```
FROM   EMPLOYE

GROUP  BY id_restaurant

HAVING COUNT(*) > 3;

/* 11. Sous-requête corrélée : articles plus chers que la moyenne */

SELECT libelle, prix_unitaire

FROM   ARTICLE

WHERE  prix_unitaire >

    (SELECT AVG(prix_unitaire) FROM ARTICLE);

/* 12. Sous-requête + NOT EXISTS : clients sans commande */

SELECT *

FROM   CLIENT c

WHERE  NOT EXISTS (SELECT 1

            FROM   COMMANDE co

            WHERE  co.id_client = c.id_client);

/* 13. ORDER BY : commandes les plus récentes d'abord */

SELECT *

FROM   COMMANDE

ORDER  BY date_commande DESC;

/* 14. Mise à jour : marquer la commande 2 comme livrée */

UPDATE COMMANDE

SET    statut = 'livrée'

WHERE  id_commande = 2;

/* 15. Suppression : retirer l'article 3 du menu 1 */

DELETE

FROM   MENU_ARTICLE

WHERE  id_menu    = 1

 AND  id_article = 3;
```

5. Principe général de la migration

5.1 Export des données

Les huit tables ont été exportées en JSON (dossier *sources/*).

5.2 Import dans MongoDB

```
cd sources/json_mongo

for f in *.json; do

  mongoimport --db resto --collection "${f%.json}" --file "$f" --jsonArray

done
```

On obtient les collections : article, client, commande, commande_menu, employe, menu, menu_article, restaurant.

5.3 Jeu de requêtes MongoDB

Catégorie	Requête
Insertion	`db.menu.insertOne({ id_menu: 3, nom: "Menu du soir", id_restaurant: 1, prix_base: null })`
Mise à jour	`db.commande.updateOne({ id_commande: 2 }, { $set: { statut: "livrée" } })`
Suppression	`db.article.deleteOne({ id_article: 4 })`
Recherche (proj.)	`db.menu.find({ prix_base:{ $gt: 20 } }, { _id:0, id_menu:1, nom:1, prix_base:1 })`
Recherche (filtre)	`db.restaurant.find({ ville:"Lyon" }, { _id:0, nom:1, adresse:1 })`
Agrégation	**mongo_pipeline.js** : chiffre d'affaires par restaurant (pipeline `$lookup` + `$group`).
Map-Reduce	**mapreduce_articles.js** : quantités totales vendues par article (résultat dans `totaux_articles`).

5.4 Test rapide

```
mongod     # si nécessaire

mongo

use resto  # puis copier-coller les requêtes ci-dessus
```

Les insert, update et delete renvoient acknowledged: true.

Le pipeline affiche le CA par id_restaurant.

Le Map-Reduce crée la collection totaux_articles; vérifier : db.totaux_articles.find().pretty().

5.5 Limitations assumées

- Identifiants numériques conservés (pas d'ObjectId).
- Tables d'association gardées telles quelles pour rester isomorphes à la partie SQL.
- Pas de transactions multi-collections : à prévoir en production.

Tous les fichiers (restaurant.sql, JSON, mongo_pipeline.js, mapreduce_articles.js) sont regroupés dans sources.zip. Tout est détaillé précisément et étape par étape.