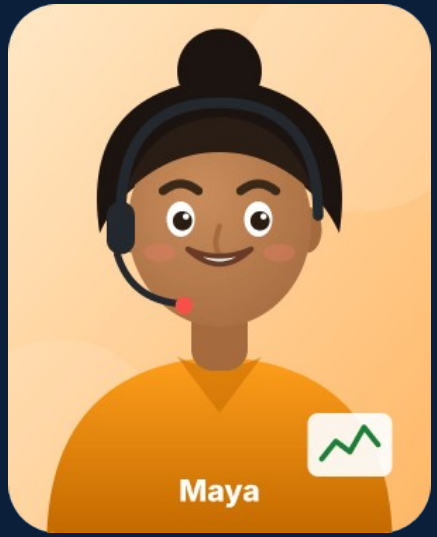
HEF-070 · MISSION 02 / 10

Modélisation et normalisation

Un bon schéma évite les erreurs avant même qu'elles ne se produisent.



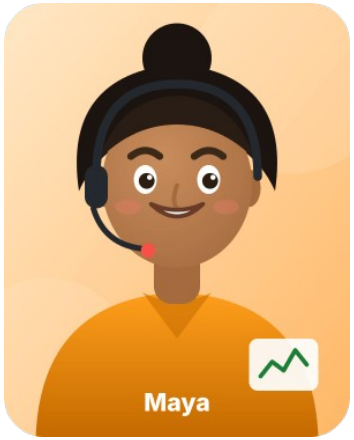


Badge : Architecte de schémas

80 min · intermédiaire

L'histoire

Kodi a créé une table unique `inscriptions_brutes` qui contient tout : nom du membre, ville, titre du cours, nom du formateur, adresse du formateur. Quand le formateur déménage, il faut modifier 400 lignes. Quand le dernier membre d'un cours est supprimé, on perd aussi le formateur et son adresse.



> *« Ta table fait trop de choses à la fois, dit Sam. Il faut la **normaliser** : découper pour que chaque fait ne soit stocké qu'une seule fois. »*

Tu apprends à repérer les anomalies et à normaliser un schéma.

Ta mission

À la fin, je serai capable de…

1. identifier les anomalies d'insertion, de modification et de suppression
2. énoncer les formes normales 1FN, 2FN et 3FN
3. décomposer une table mal conçue
4. construire un diagramme entité-association simple
5. savoir quand dénormaliser volontairement

Défi d'échauffement

Défi d'échauffement

Voici un petit tableau :

élève	classe	professeur	salle du professeur
Ada	6A	M. Durand	B12
Bob	6A	M. Durand	B12
Cléo	6B	Mme Martin	C03

Si M. Durand change de salle, combien de lignes modifies-tu ? Si Cléo quitte l'école, perds-tu des informations sur Mme Martin ? Si une nouvelle professeure arrive sans élève, où l'écris-tu ?

Le concept

Les trois anomalies

Quand une table mélange plusieurs sujets, on rencontre :

Anomalie	Exemple
De modification	changer la salle d'un professeur oblige à modifier plusieurs lignes ; on peut en oublier une → incohérence
D'insertion	impossible d'ajouter un professeur qui n'a pas encore d'élève
De suppression	supprimer le dernier élève d'une classe efface les informations sur le professeur

La normalisation

Le concept

La normalisation (suite)

La **normalisation** consiste à décomposer les tables selon des règles (les **formes normales**) pour éliminer ces anomalies.

1FN — chaque case contient une seule valeur

❌ `cours = "HEF-000, HEF-010"` (une liste dans une case) ✅ une ligne par couple (membre, cours).
2FN — tout dépend de TOUTE la clé (cas des clés composées)

Si la clé est `(membre, cours)` et que la colonne `ville_du_membre` ne dépend que de `membre`, elle n'a rien à faire dans cette table : on la déplace dans `membres`.

3FN — rien ne dépend d'une colonne non clé

Le concept

La normalisation (suite)

Si `classe → professeur` et `professeur → salle`, alors `salle` dépend de `classe` **par l'intermédiaire** du professeur. On sépare : `professeurs(id, nom, salle)` et `classes(id, professeur_id)`.

La formule à retenir

> *Chaque colonne dépend **de la clé, de toute la clé, et rien que de la clé** (so help me Codd).*

Dépendances fonctionnelles

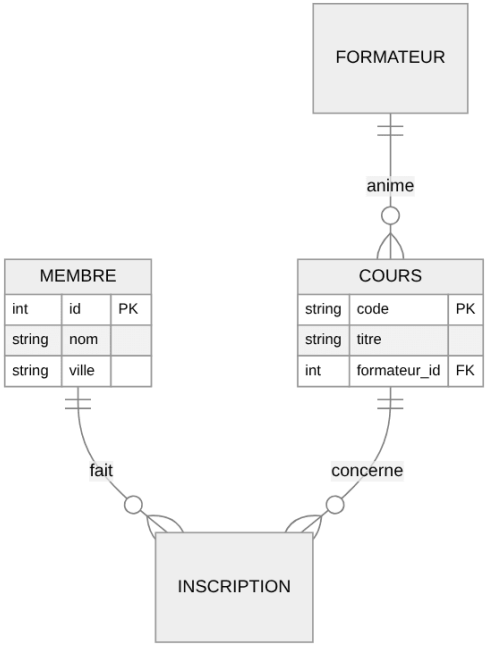
On dit que `A → B` si connaître `A` détermine `B`. Exemple : `professeur → salle`. Trouver les dépendances fonctionnelles est la clé de la normalisation.

Le concept

Diagramme entité-association

Pour concevoir, on commence par un schéma conceptuel :

Diagramme entité-association (suite)



Le concept

Diagramme entité-association (suite)

Description du schéma : un membre peut faire plusieurs inscriptions ; un cours en reçoit plusieurs ; un formateur anime plusieurs cours. Les attributs listés sont : membre (id, nom, ville), cours (code, titre, formateur).

Dénormaliser, parfois

La normalisation évite les incohérences mais multiplie les jointures. Dans certains cas (rapports, performances de lecture), on **duplique volontairement** une donnée. Ce choix est **assumé et documenté** : il demande de garder les copies synchronisées.

> 🧠 ***Astuce*** *: normalise d'abord. Dénormalise seulement quand une* ***mesure*** *montre que c'est nécessaire.*

Exemple pas à pas

Problème : normaliser la table de Kodi.

```
import sqlite3

bd = sqlite3.connect(":memory:")
bd.executescript('''
CREATE TABLE brute (
    membre TEXT, ville TEXT, cours TEXT, formateur TEXT, adresse_formateur TEXT
);
INSERT INTO brute VALUES ('Ada', 'Lyon', 'HEF-000', 'Sam', '1 rue Alpha');
INSERT INTO brute VALUES ('Ada', 'Lyon', 'HEF-010', 'Maya', '2 rue Beta');
INSERT INTO brute VALUES ('Bob', 'Lille', 'HEF-000', 'Sam', '1 rue Alpha');
INSERT INTO brute VALUES ('Cléo', 'Lyon', 'HEF-000', 'Sam', '1 rue Alpha');
''')
```

Dépendances fonctionnelles :

- `membre → ville`

Exemple pas à pas

Exemple pas à pas (suite)

- `cours → formateur`
- `formateur → adresse_formateur`

Décomposition en 3FN :

```
bd.executescript('''
CREATE TABLE membres (nom TEXT PRIMARY KEY, ville TEXT);
CREATE TABLE formateurs (nom TEXT PRIMARY KEY, adresse TEXT);
CREATE TABLE cours (code TEXT PRIMARY KEY, formateur TEXT REFERENCES formateurs(no
   m));
CREATE TABLE inscriptions (membre TEXT REFERENCES membres(nom), cours TEXT REFEREN
   CES cours(code), PRIMARY KEY (membre, cours));

INSERT INTO membres SELECT DISTINCT membre, ville FROM brute;
INSERT INTO formateurs SELECT DISTINCT formateur, adresse_formateur FROM brute;
INSERT INTO cours SELECT DISTINCT cours, formateur FROM brute;
INSERT INTO inscriptions SELECT DISTINCT membre, cours FROM brute;
```

Exemple pas à pas (suite)

```
''')
for table in ("membres", "formateurs", "cours", "inscriptions"):
    print(table, bd.execute(f"SELECT COUNT(*) FROM {table}").fetchone()[0])
# membres 3, formateurs 2, cours 2, inscriptions 4
```

Gain : l'adresse de Sam est écrite **une fois** (au lieu de 3). Pour la changer :

```
bd.execute("UPDATE formateurs SET adresse = '9 avenue Gamma' WHERE nom = 'Sam'")
```

Une seule ligne modifiée : plus d'incohérence possible. On peut aussi **reconstituer** la table brute avec des jointures, sans perte d'information.

À toi, avec aide

À toi, avec aide

Voici une table de commandes :

commande	produit	quantité	prix_unitaire	client	ville_client

La clé est `(commande, produit)`.

1. Écris les dépendances fonctionnelles.
2. Dis quelles colonnes violent la 2FN et la 3FN.
3. Propose la décomposition.

À toi, avec aide · réponse

Indices progressifs

Indice 1

`commande → client` : le client dépend d'une partie de la clé seulement.

Indice 2

`produit → prix_unitaire` : le prix ne dépend que du produit.

Indice 3

`client → ville_client` est une dépendance transitive (3FN).

À toi, avec aide · réponse

Réponse

Dépendances : `(commande, produit) → quantité` ; `produit → prix_unitaire` ; `commande → client` ; `client → ville_client`.

- Violation **2FN** : `prix_unitaire` (dépend de `produit` seul), `client` et `ville_client` (dépendent de `commande` seule).
- Violation **3FN** : `ville_client` dépend de `client`, qui n'est pas la clé.

Décomposition :

- `produits(id, nom, prix_unitaire)`
- `clients(id, nom, ville)`
- `commandes(id, client_id)`
- `lignes(commande_id, produit_id, quantite)`

À toi, en solo

Normalisation d'un tableur réel. Voici un fichier de suivi (une ligne par évaluation) :

```
etudiant;email;groupe;tuteur;email_tuteur;module;note;date_note
```

1. Liste les dépendances fonctionnelles plausibles.
2. Propose un schéma 3FN.
3. Écris le code SQLite qui crée le schéma, charge 8 lignes de données dans la table brute et les redistribue dans les tables normalisées avec des `INSERT ... SELECT`.
4. Montre qu'un changement d'e-mail du tuteur se fait en une ligne, puis qu'une requête de jointure redonne la vue d'origine.

À toi, en solo · réponse

Pistes de correction

```
import sqlite3

bd = sqlite3.connect(":memory:")
bd.executescript('''
CREATE TABLE brute (etudiant TEXT, email TEXT, groupe TEXT, tuteur TEXT, email_tut
   eur TEXT, module TEXT, note REAL, date_note TEXT);
INSERT INTO brute VALUES
 ('Ada','ada@x.fr','G1','Sam','sam@x.fr','Algo',15,'2025-01-10'),
 ('Ada','ada@x.fr','G1','Sam','sam@x.fr','Réseaux',13,'2025-01-20'),
 ('Bob','bob@x.fr','G1','Sam','sam@x.fr','Algo',11,'2025-01-10'),
 ('Bob','bob@x.fr','G1','Sam','sam@x.fr','Réseaux',9,'2025-01-20'),
 ('Cléo','cleo@x.fr','G2','Maya','maya@x.fr','Algo',17,'2025-01-10'),
 ('Cléo','cleo@x.fr','G2','Maya','maya@x.fr','Réseaux',16,'2025-01-20');

CREATE TABLE tuteurs (nom TEXT PRIMARY KEY, email TEXT NOT NULL);
CREATE TABLE groupes (nom TEXT PRIMARY KEY, tuteur TEXT NOT NULL REFERENCES tuteur
   s(nom));
CREATE TABLE etudiants (nom TEXT PRIMARY KEY, email TEXT NOT NULL, groupe TEXT NOT
    NULL REFERENCES groupes(nom));
```

Pistes de correction (suite)

```
CREATE TABLE modules (nom TEXT PRIMARY KEY);
CREATE TABLE notes (etudiant TEXT REFERENCES etudiants(nom), module TEXT REFERENCE
   S modules(nom), note REAL NOT NULL, date_note TEXT NOT NULL, PRIMARY KEY (etudi
   ant, module));

INSERT INTO tuteurs SELECT DISTINCT tuteur, email_tuteur FROM brute;
INSERT INTO groupes SELECT DISTINCT groupe, tuteur FROM brute;
INSERT INTO etudiants SELECT DISTINCT etudiant, email, groupe FROM brute;
INSERT INTO modules SELECT DISTINCT module FROM brute;
INSERT INTO notes SELECT etudiant, module, note, date_note FROM brute;
''')
bd.execute("UPDATE tuteurs SET email = 'sam@nouveau.fr' WHERE nom = 'Sam'")
vue = bd.execute('''
SELECT e.nom, e.email, g.nom, t.nom, t.email, n.module, n.note, n.date_note
FROM notes n JOIN etudiants e ON e.nom = n.etudiant JOIN groupes g ON g.nom = e.gr
   oupe JOIN tuteurs t ON t.nom = g.tuteur
ORDER BY e.nom, n.module''').fetchall()
print(len(vue), vue[0])
```

Pistes de correction (suite)

Dépendances : `etudiant → email, groupe` ; `groupe → tuteur` ; `tuteur → email_tuteur` ; `(etudiant, module) → note, date_note`.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Listes dans une case	Impossible de filtrer ou compter proprement	Une ligne par valeur (1FN)
Redondance « pratique »	Incohérences au premier changement	Une information, un endroit
Normaliser à l'excès	Dix jointures pour une page	Arrêter à la 3FN sauf besoin précis
Dénormaliser sans mesure	Complexité sans gain	Mesurer d'abord, documenter ensuite

Mini-boss

Mini-boss

Revue de schéma. On te fournit ce schéma défectueux pour une plateforme de covoiturage :

```
trajets(id, conducteur, tel_conducteur, ville_depart, ville_arrivee, passagers, pr
   ix, date)
```

où `passagers` contient les noms séparés par des virgules.

1. Liste au moins 4 problèmes (anomalies, formes normales violées).
2. Propose un schéma normalisé (avec clés).
3. Écris le code SQLite de migration : créer les nouvelles tables, remplir à partir de l'ancienne (y compris éclater la colonne `passagers`), vérifier les comptes.
4. Identifie une dénormalisation légitime (par exemple `places_restantes`) et explique comment la garder cohérente.

Mini-boss · réponse

Pistes de correction

Problèmes : liste de passagers dans une case (1FN) ; `tel_conducteur` dépend du conducteur, pas du trajet (2FN/3FN) ; les villes sont répétées en texte libre (fautes de frappe) ; impossible de stocker un passager sans trajet ou de supprimer un trajet sans perdre le conducteur. Schéma : `conducteurs(id, nom, tel)`, `villes(id, nom)`, `trajets(id, conducteur_id, depart_id, arrivee_id, prix_centimes, date)`, `passagers(id, nom)`, `reservations(trajet_id, passager_id)`. La migration éclate `passagers` avec `str.split(',')` en Python (SQLite n'a pas de fonction d'éclatement native simple) puis insère. Dénormalisation légitime : `places_restantes` dans `trajets` pour l'affichage rapide, maintenue par une transaction qui insère la réservation et décrémente le compteur, avec une requête de contrôle `places_restantes = places_total - COUNT(reservations)`.

Quiz éclair

Quiz éclair

1. La première forme normale exige :

A. une clé étrangère par table
B. des index sur toutes les colonnes
C. une seule valeur par case

2. Une anomalie de modification se produit quand :

A. une même information est stockée à plusieurs endroits
B. un index est manquant
C. la base est trop grande

3. La 3FN élimine les dépendances :

A. vers la clé primaire

Quiz éclair (suite)

B. transitives (via une colonne non clé)
C. entre deux tables

4. 🔁 *Révision mission 01* : Une clé primaire :

A. relie deux bases
B. chiffre les données
C. identifie chaque ligne de façon unique

5. 🔁 *Révision mission 01* : Pour une relation n-n, on utilise :

A. une table de liaison
B. une colonne contenant une liste
C. un fichier texte

Quiz éclair · réponse

Réponses

1. **C**.
2. **A**.
3. **B**.
4. **C**.
5. **A**.

Je réfléchis

Je réfléchis

1. Quelle information dans tes propres fichiers est écrite à plusieurs endroits ?
2. Dans quel cas accepterais-tu de dupliquer volontairement une donnée ?

À retenir

À retenir

1. Anomalies : modification, insertion, suppression.
2. 1FN : une valeur par case. 2FN : toute la clé. 3FN : rien que la clé.
3. Dépendances fonctionnelles : A → B.
4. Décomposer sans perdre d'information.
5. Dénormaliser : un choix mesuré et documenté.

MISSION ACCOMPLIE

+100 XP

Badge débloqué : Architecte de schémas

Prochaine mission : SQL de lecture : interroger les données

