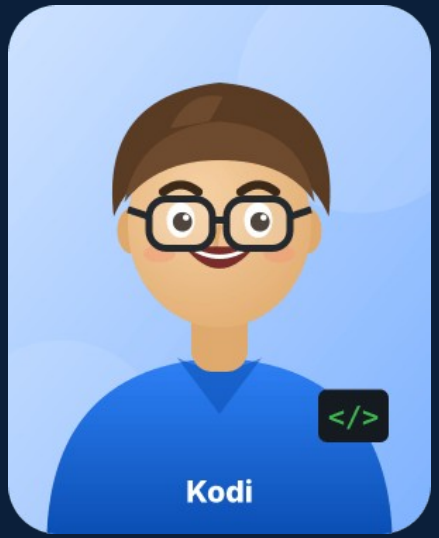
HEF-070 · MISSION 05 / 10

Contraintes et intégrité

La meilleure validation est celle que la base impose à tout le monde, même au code qui l'a oubliée.



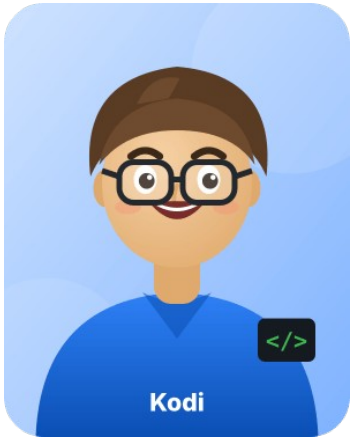


Badge : Gardien de la cohérence

75 min · intermédiaire

L'histoire

Dans la base de HashBox, quelqu'un a saisi un score de −500, un e-mail sans arobase, et deux fois le même identifiant de membre. L'application vérifiait ces règles… sauf dans le script d'import, qui a écrit directement dans la base.



> *« Les règles dans le code se contournent, dit Sam. Les règles dans la base ne se contournent pas. Elles protègent les données de tous les programmes, présents et futurs. »*

Tu apprends à déclarer des contraintes que la base fera respecter à ta place.

Ta mission

À la fin, je serai capable de…

1. utiliser NOT NULL, UNIQUE, CHECK, DEFAULT
2. déclarer des clés étrangères et choisir ON DELETE
3. distinguer erreurs de contrainte et erreurs applicatives
4. tester qu'une contrainte refuse bien les données invalides
5. expliquer pourquoi les règles métier critiques vivent aussi dans la base

Défi d'échauffement

Un formulaire papier demande : âge (entre 0 et 120), e-mail (obligatoire), numéro d'adhérent (jamais deux fois le même). Quelles erreurs de saisie peux-tu imaginer ? Qui doit les attraper : la personne qui remplit, l'accueil, ou le système de classement ? Que se passe-t-il si **un seul** de ces contrôles est oublié ?

Le concept

Les contraintes de colonne

Contrainte	Effet
`NOT NULL`	la valeur est obligatoire
`UNIQUE`	pas deux fois la même valeur
`PRIMARY KEY`	`NOT NULL` + `UNIQUE` + identité de la ligne
`DEFAULT valeur`	valeur par défaut si absente
`CHECK (condition)`	la condition doit être vraie
`REFERENCES t(c)`	clé étrangère : la valeur doit exister dans `t.c`

Le concept

Les contraintes de colonne (suite)

```
CREATE TABLE membres (
    id       INTEGER PRIMARY KEY,
    email    TEXT NOT NULL UNIQUE CHECK (email LIKE '%_@_%'),
    age      INTEGER CHECK (age BETWEEN 0 AND 120),
    score    INTEGER NOT NULL DEFAULT 0 CHECK (score >= 0),
    ville_id INTEGER REFERENCES villes(id)
);
```

Les clés étrangères et leurs actions

Que se passe-t-il si on supprime une ligne **référencée** ?

Action	Effet
`NO ACTION` / `RESTRICT`	refus de la suppression (par défaut)
`CASCADE`	supprime aussi les lignes qui la référencent

Le concept

Les clés étrangères et leurs actions (suite)

Action	Effet
`SET NULL`	met la référence à NULL
`SET DEFAULT`	met la valeur par défaut

```
REFERENCES membres(id) ON DELETE CASCADE        -- supprimer un membre supprime se
   s inscriptions
REFERENCES categories(id) ON DELETE RESTRICT    -- refuser de supprimer une catégo
   rie utilisée
```

CASCADE est puissant et dangereux : à réserver aux données qui n'ont pas de sens sans leur parent.

Contraintes sur plusieurs colonnes

Le concept

Contraintes sur plusieurs colonnes (suite)

```
UNIQUE (cours_code, numero)            -- une seule mission n°3 par cours
CHECK (date_fin >= date_debut)
```

Pourquoi dans la base ?

Dans le code	Dans la base
contournable (un autre script, une console SQL)	imposé à tous
à répéter dans chaque application	écrit une fois
messages d'erreur adaptés à l'utilisateur	protection garantie

Les deux se complètent : le code donne un bon message à l'utilisateur, la base garantit qu'aucune donnée invalide n'entre jamais.

Le concept

Attention (SQLite)

- les clés étrangères sont **désactivées par défaut** : `PRAGMA foreign_keys = ON` à chaque connexion ;
- SQLite est flexible sur les types (« typage dynamique ») : un `CHECK (typeof(age) = 'integer')` peut aider ; en PostgreSQL les types sont stricts.

Capturer les erreurs

```
import sqlite3
try:
    bd.execute("INSERT INTO membres (email) VALUES ('pas-un-email')")
except sqlite3.IntegrityError as e:
    print("refusé par la base :", e)
```

Exemple pas à pas

Problème : un schéma qui impose les règles de HashBox, et des tests qui prouvent qu'il refuse bien les données invalides.

```
import sqlite3

bd = sqlite3.connect(":memory:")
bd.execute("PRAGMA foreign_keys = ON")
bd.executescript('''
CREATE TABLE membres (
    id    INTEGER PRIMARY KEY,
    email TEXT NOT NULL UNIQUE CHECK (email LIKE '%_@_%'),
    score INTEGER NOT NULL DEFAULT 0 CHECK (score >= 0)
);
CREATE TABLE cours (code TEXT PRIMARY KEY, titre TEXT NOT NULL);
CREATE TABLE inscriptions (
    membre_id INTEGER NOT NULL REFERENCES membres(id) ON DELETE CASCADE,
    cours_code TEXT NOT NULL REFERENCES cours(code) ON DELETE RESTRICT,
    PRIMARY KEY (membre_id, cours_code)
```

Exemple pas à pas

Exemple pas à pas (suite)

```
);
INSERT INTO membres (email) VALUES ('ada@x.fr');
INSERT INTO cours VALUES ('HEF-000', 'Mentalité');
INSERT INTO inscriptions VALUES (1, 'HEF-000');
''')

tests = {
    "e-mail sans @": "INSERT INTO membres (email) VALUES ('adaxfr')",
    "e-mail en double": "INSERT INTO membres (email) VALUES ('ada@x.fr')",
    "score négatif": "INSERT INTO membres (email, score) VALUES ('b@x.fr', -5)",
    "e-mail manquant": "INSERT INTO membres (score) VALUES (3)",
    "membre inexistant": "INSERT INTO inscriptions VALUES (99, 'HEF-000')",
    "supprimer un cours utilisé": "DELETE FROM cours WHERE code = 'HEF-000'",
}
for nom, sql in tests.items():
    try:
        bd.execute(sql)
        print("ACCEPTÉ ", nom)
    except sqlite3.IntegrityError as e:
```

Exemple pas à pas (suite)

```
        print("refusé  ", nom, "→", e)

bd.execute("DELETE FROM membres WHERE id = 1")                        # CASCADE
print("inscriptions restantes :", bd.execute("SELECT COUNT(*) FROM inscriptions").
   fetchone()[0])   # 0
```

Lecture : les six écritures invalides sont refusées sans une seule ligne de code de validation. La suppression d'un membre supprime automatiquement ses inscriptions (CASCADE), alors que la suppression d'un cours **utilisé** est interdite (RESTRICT).

À toi, avec aide

Ajoute les contraintes pour une table `reservations(id, salle, debut, fin, personne)` :

1. aucune valeur manquante ;
2. `fin` strictement après `debut` ;
3. une salle ne peut pas être réservée deux fois **à la même heure de début** ;
4. teste chaque règle avec une insertion invalide.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Utilise `CHECK (fin > debut)`.

Indice 2

`UNIQUE (salle, debut)` pour la règle 3.

Indice 3

Les dates au format texte ISO (`'2025-03-01 09:00'`) se comparent correctement.

Réponse

```
import sqlite3

bd = sqlite3.connect(":memory:")
bd.executescript('''
CREATE TABLE reservations (
    id INTEGER PRIMARY KEY,
    salle TEXT NOT NULL,
    debut TEXT NOT NULL,
    fin TEXT NOT NULL,
    personne TEXT NOT NULL,
    CHECK (fin > debut),
    UNIQUE (salle, debut)
);
INSERT INTO reservations (salle, debut, fin, personne) VALUES ('A1', '2025-03-01 0
   9:00', '2025-03-01 10:00', 'Ada');
''')
for nom, sql in (("fin avant début", "INSERT INTO reservations (salle, debut, fin,
    personne) VALUES ('A1','2025-03-01 11:00','2025-03-01 10:00','Bob')"),
                 ("même salle même début", "INSERT INTO reservations (salle, debut
```

À toi, avec aide · réponse

Réponse (suite)

```
    , fin, personne) VALUES ('A1','2025-03-01 09:00','2025-03-01 09:30','Bob')"),
                 ("personne manquante", "INSERT INTO reservations (salle, debut, f
    in) VALUES ('B2','2025-03-01 09:00','2025-03-01 10:00')")):
     try:
         bd.execute(sql); print("ACCEPTÉ", nom)
     except sqlite3.IntegrityError as e:
         print("refusé", nom, "→", e)
```

Limite : ces contraintes n'empêchent pas deux réservations qui se **chevauchent** avec des débuts différents. PostgreSQL offre pour cela des contraintes d'exclusion (`EXCLUDE USING gist`) ; en SQLite on utilise un déclencheur (`TRIGGER`) ou une vérification dans une transaction.

À toi, en solo

Chevauchement de réservations. Écris un **déclencheur** (`CREATE TRIGGER ... BEFORE INSERT`) SQLite qui refuse (via `RAISE(ABORT, 'chevauchement')`) toute réservation qui chevauche une réservation existante dans la même salle (condition : `nouveau.debut < existant.fin AND nouveau.fin > existant.debut`).
Teste avec : cas qui se touche (fin = début : accepté), cas strictement à l'intérieur (refusé), cas qui englobe (refusé), autre salle (accepté).

À toi, en solo · réponse

Pistes de correction

```
import sqlite3

bd = sqlite3.connect(":memory:")
bd.executescript('''
CREATE TABLE reservations (
    id INTEGER PRIMARY KEY, salle TEXT NOT NULL, debut TEXT NOT NULL, fin TEXT NOT
    NULL, personne TEXT NOT NULL,
    CHECK (fin > debut)
);
CREATE TRIGGER pas_de_chevauchement BEFORE INSERT ON reservations
BEGIN
    SELECT RAISE(ABORT, 'chevauchement')
    WHERE EXISTS (SELECT 1 FROM reservations
                  WHERE salle = NEW.salle AND NEW.debut < fin AND NEW.fin > debut)
   ;
END;
INSERT INTO reservations (salle, debut, fin, personne) VALUES ('A1', '09:00', '10:
   00', 'Ada');
''')
```

À toi, en solo · réponse

Pistes de correction (suite)

```
cas = {
    "touche (10h-11h)": ("A1", "10:00", "11:00"),
    "à l'intérieur": ("A1", "09:15", "09:45"),
    "englobe": ("A1", "08:00", "11:00"),
    "autre salle": ("B2", "09:00", "10:00"),
}
for nom, (s, d, f) in cas.items():
    try:
        bd.execute("INSERT INTO reservations (salle, debut, fin, personne) VALUES
   (?, ?, ?, 'Bob')", (s, d, f))
        print("accepté", nom)
    except sqlite3.DatabaseError as e:
        print("refusé ", nom, "→", e)
```

Un déclencheur s'exécute à l'intérieur de la base : aucune application ne peut le contourner. Attention : sous forte concurrence, deux transactions simultanées pourraient passer le contrôle en même temps, d'où l'intérêt des contraintes d'exclusion ou des verrous (mission 9).

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Valider seulement dans l'application	Un script direct contourne tout	Contraintes dans la base
CASCADE par défaut	Suppressions en chaîne imprévues	RESTRICT sauf quand le sens est clair
Oublier `foreign_keys=ON` (SQLite)	Aucune vérification	Activer à chaque connexion
NULL dans une colonne UNIQUE	Plusieurs NULL sont autorisés	Ajouter NOT NULL si la valeur est obligatoire

Mini-boss

Mini-boss

Schéma blindé pour un système de réservation de salles et de matériel.

Règles :

1. une salle a une capacité > 0 ; son nom est unique ;
2. un utilisateur a un e-mail unique et valide ;
3. une réservation a une fin postérieure au début, pas de chevauchement sur une même salle, un nombre de participants ≤ capacité ;
4. le matériel (vidéoprojecteur…) peut être associé à une réservation, mais un même matériel ne peut pas être réservé deux fois en même temps ;
5. on ne peut pas supprimer une salle qui a des réservations futures ; supprimer un utilisateur supprime ses réservations futures mais conserve l'historique (suppression logique).

Mini-boss

Mini-boss (suite)

Écris le schéma SQLite, les déclencheurs nécessaires, et une **batterie de 15 tests** (cas valides et invalides) qui prouvent que chaque règle est appliquée.

Mini-boss · réponse

Pistes de correction

Un bon travail combine : `CHECK`, `UNIQUE`, clés étrangères avec `ON DELETE` adaptés, des déclencheurs `BEFORE INSERT/UPDATE` pour le chevauchement (salle et matériel) et la capacité (`NEW.participants <= (SELECT capacite FROM salles WHERE id = NEW.salle_id)`), un déclencheur `BEFORE DELETE ON salles` qui refuse s'il existe des réservations futures, et une colonne `supprime_le` pour la suppression logique des utilisateurs. La batterie de tests utilise un dictionnaire `{nom: (sql, params, doit_etre_accepte)}` et vérifie le résultat attendu pour chaque règle, y compris les cas limites (fin = début suivant, capacité exacte).

Quiz éclair

Quiz éclair

1. Une contrainte CHECK :

A. crée un index
B. relie deux tables
C. impose une condition sur les valeurs

2. ON DELETE CASCADE signifie :

A. supprimer aussi les lignes qui référencent la ligne supprimée
B. refuser la suppression
C. mettre la référence à NULL

3. Pourquoi mettre les règles dans la base ?

A. elles sont plus rapides à écrire

Quiz éclair

Quiz éclair (suite)

B. elles s'appliquent à tous les programmes
C. elles remplacent les tests

4. 🔁 *Révision mission 04* : Un UPDATE sans WHERE :

A. ne fait rien
B. provoque une erreur
C. modifie toutes les lignes de la table

5. 🔁 *Révision mission 04* : Un upsert est :

A. insérer ou mettre à jour si la clé existe
B. supprimer puis insérer
C. un type d'index

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quelle règle de ta vie professionnelle devrait être « impossible à contourner » plutôt que simplement rappelée ?
2. Pourquoi un message d'erreur clair est-il important pour l'utilisateur, en plus de la contrainte ?

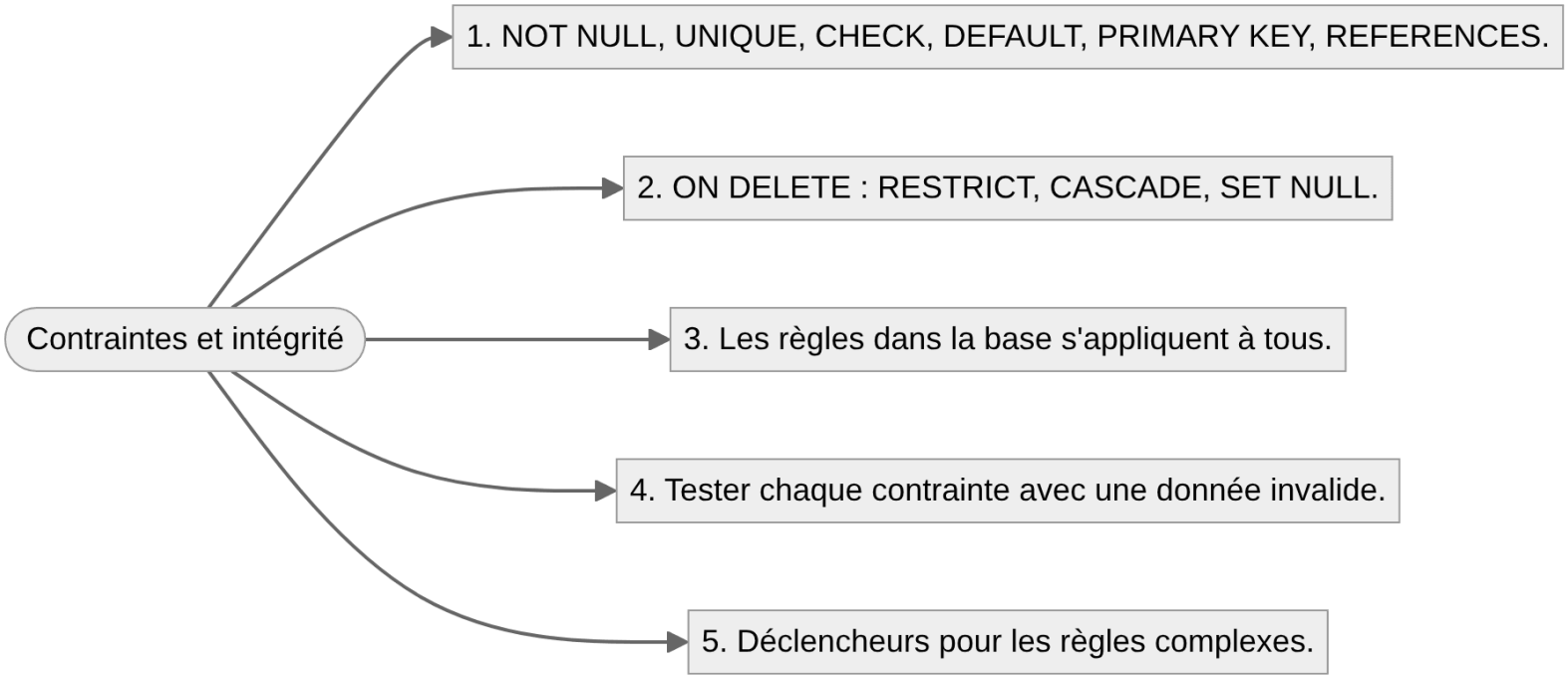
À retenir

À retenir

1. NOT NULL, UNIQUE, CHECK, DEFAULT, PRIMARY KEY, REFERENCES.
2. ON DELETE : RESTRICT, CASCADE, SET NULL.
3. Les règles dans la base s'appliquent à tous.
4. Tester chaque contrainte avec une donnée invalide.
5. Déclencheurs pour les règles complexes.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Gardien de la cohérence

Prochaine mission : Jointures et agrégations

