Fiche à étudier — Relier deux entités (1-N)

Tout ce qui s'apprend par cœur pour relier deux entités : le vocabulaire, les notations, la démarche du texte à la base, la syntaxe SQL, les pièges et une série de questions pour te tester.

4TTR 5TTR 6TTR Synthèse

Cette fiche rassemble ce qu'il faut savoir par cœur pour relier deux entités : découper les données, poser une association 1-N, la traduire en clé étrangère et interroger les deux tables ensemble. Elle ne contient pas d'exercice et peut s'imprimer seule. Les exemples portent sur une école : des élèves, des classes, et chaque élève appartient à une classe.

Le vocabulaire

Terme	En anglais	Définition
Redondance	GB ***redundancy***	Un même fait écrit à plusieurs endroits d'une base.
Anomalie	GB ***anomaly***	Problème qui apparaît quand on ajoute, modifie ou supprime des données da
Association	GB ***relationship***	Lien entre deux entités. Son nom est un verbe conjugué à la 3e personne : `ap`
Schéma entité-association	GB ***entity-relationship diagram*** (ERD)	Dessin qu'on réalise pour analyser un domaine : entités dans des rectangles
MCD (modèle conceptuel de données)	GB ***conceptual data model***	Description des entités, de leurs attributs et des associations qui les relient, s
Merise	—	Méthode française de conception de bases de données, qui fixe les règles du
Cardinalité	GB ***cardinality***	Couple (minimum, maximum) écrit à côté d'une entité : à combien d'occurren
Association 1-N	GB ***one-to-many relationship***	Association dont le maximum vaut 1 d'un côté et n de l'autre.
Clé étrangère	GB ***foreign key***	Colonne qui contient la clé primaire d'une ligne d'une autre table. Elle fonctio
MLD (modèle logique de données)	GB ***logical data model***	Traduction du MCD en liste de tables : chaque entité devient une table, chaqu
`REFERENCES`	GB ***foreign key constraint***	Mot-clé SQL qui déclare qu'une colonne est une clé étrangère vers une autre
PRAGMA `foreign_keys`	GB ***pragma***	Réglage propre à SQLite qui active la vérification des clés étrangères. Désact
Intégrité référentielle	GB ***referential integrity***	Contrainte qui garantit que chaque clé étrangère désigne une ligne qui existe
Ligne orpheline	GB ***orphan row***	Ligne dont la clé étrangère désigne une ligne qui n'existe pas, ou plus.
Produit cartésien	GB ***cartesian product***	Toutes les combinaisons possibles entre les lignes de deux tables. Nombre
Jointure	GB ***join***	Assemblage des lignes de deux tables qui correspondent l'une à l'autre, selo
Alias de table	GB ***table alias***	Surnom donné à une table dans une requête (`FROM eleve e`), utilisé pour pr
`LEFT JOIN`	GB ***left join***	Jointure qui garde toutes les lignes de la table de gauche, même sans corres

Rappel des mots des autres niveaux : une **entité** est une chose dont on veut garder la trace ; une **occurrence** en est un exemplaire concret ; une **contrainte** est une règle que la base vérifie à chaque écriture (`NOT NULL` , `UNIQUE` , clé étrangère).

Les conventions de notation

Point	Convention	Exemple
Cardinalité	Merise min-max, écrite du côté de l'entité qu'elle décrit	`ELEVE (0`
Les quatre cardinalités	Entre parenthèses, `n` minuscule	`(0,1)` `(`
Lecture	« un » + l'entité du côté où les chiffres sont écrits	« Un élèv
Association	Un verbe conjugué à la 3e personne	`appartie`
Entité (MCD)	Singulier, MAJUSCULES	`ELEVE`
Table (base)	Singulier, minuscules, sans accent	`eleve`
Clé primaire	`id`	`classe.i`
Clé étrangère	Table désignée + `_id`	`classe_i`

Point	Convention	Exemple
Clé étrangère dans le MLD	`#` + colonne + flèche vers la colonne désignée	`#classe_`
Placement	La clé étrangère va dans la table de l'entité qui porte le maximum 1 — celui qui n'en a qu'un garde l'adresse de l'autre.	`classe_i`

Cardinalité	Se lit
`(0,1)`	au plus un, éventuellement aucun
`(1,1)`	exactement un
`(0,n)`	autant qu'on veut, éventuellement aucun
`(1,n)`	au moins un, autant qu'on veut

MLD de l'école (clé primaire soulignée) :

- classe (id, nom, annee, local)
- eleve (id, nom, prenom, date_naissance, commune, email, absences, #classe_id → classe.id)

Du texte à la base, étape par étape

1. **Repérer les entités et les associations.** Chercher les phrases sujet – verbe – complément dont le sujet et le complément sont deux entités. « Chaque élève appartient à une classe » donne l'association `ELEVE –appartient– CLASSE`. Sans verbe, c'est souvent un attribut.
2. **Poser les cardinalités.** De chaque côté, se demander le minimum et le maximum, et écrire la phrase qui commence par « Un … ». Le minimum est une décision de gestion. Résultat : `ELEVE (0,1) –appartient– (0,n) CLASSE`.
3. **Écrire le MLD.** Une table par entité. Pour chaque association 1-N, ajouter une clé étrangère dans la table de l'entité qui porte le maximum 1. Vérifier : autant de clés étrangères que d'associations 1-N.
4. **Activer le pragma, avant tout le reste.** DB Browser : onglet *Éditer les pragmas*, case *Foreign Keys*, *Enregistrer*. En SQL : `PRAGMA foreign_keys = ON;`
5. **Créer la clé étrangère.** `CREATE TABLE` pour la nouvelle table, `ALTER TABLE … ADD COLUMN … REFERENCES …` pour la colonne, puis `UPDATE` pour remplir les liens.
6. **Faire les deux tests.** Rattacher une ligne à une ligne inexistante, puis supprimer une ligne encore désignée. Les deux doivent afficher `FOREIGN KEY constraint failed` et ne rien modifier.

La syntaxe

Déclarer une clé étrangère dans une nouvelle table ou dans une table existante :

```sql
CREATE TABLE eleve (
    id        INTEGER PRIMARY KEY,
    nom       TEXT NOT NULL,
    classe_id INTEGER REFERENCES classe(id)
);

ALTER TABLE eleve ADD COLUMN classe_id INTEGER REFERENCES classe(id);
```

Activer, contrôler et vérifier les clés étrangères :

```sql
PRAGMA foreign_keys = ON;      -- activer (pour la session en cours)
PRAGMA foreign_keys;           -- 1 = activé, 0 = désactivé
PRAGMA foreign_key_check;      -- liste les lignes dont la clé étrangère est invalide
```

Joindre deux tables, avec des alias :

```sql
SELECT e.prenom, e.nom, c.nom AS classe
FROM eleve e
JOIN classe c ON c.id = e.classe_id;
```

Garder aussi les lignes sans correspondance, ou les isoler :

```sql
SELECT e.prenom, e.nom, c.nom AS classe
FROM eleve e
LEFT JOIN classe c ON c.id = e.classe_id;

SELECT e.prenom, e.nom
FROM eleve e
LEFT JOIN classe c ON c.id = e.classe_id
WHERE c.id IS NULL;
```

Compter par groupe, groupes vides compris :

```sql
SELECT c.nom AS classe, COUNT(e.id) AS nb_eleves
FROM classe c
LEFT JOIN eleve e ON e.classe_id = c.id
GROUP BY c.nom;
```

Depuis Python :

```python
conn = sqlite3.connect("mon-ecole-classes.db")
conn.execute("PRAGMA foreign_keys = ON")
conn.row_factory = sqlite3.Row
```

Mot-clé	Traduction
`REFERENCES`	fait référence à
`ALTER TABLE … ADD COLUMN`	modifier la table … ajouter une colonne
`JOIN … ON`	joindre … sur
`LEFT JOIN`	jointure à gauche
`AS`	comme
`IS NULL`	est vide
`FOREIGN KEY constraint failed`	la contrainte de clé étrangère a échoué
`ambiguous column name`	nom de colonne ambigu

Les pièges à connaître

Lire une cardinalité à l'envers

Le `(0,1)` écrit à côté de ELEVE parle d'**un élève** et compte ses classes. Il ne veut pas dire « zéro ou un élève ». Parade : toujours formuler la phrase « Un élève … ».

Placer la clé étrangère du mauvais côté

La clé étrangère va du côté du maximum 1, jamais du côté du n. Une colonne `eleve_id` dans `classe` obligerait à écrire plusieurs élèves dans une seule case, ou à recopier la classe : retour à la redondance.

Oublier le pragma

Sans `PRAGMA foreign_keys = ON`, SQLite accepte n'importe quelle valeur dans une clé étrangère et laisse supprimer des lignes encore désignées. Les lignes orphelines créées pendant ce temps restent dans la base quand on réactive le pragma. Il faut l'activer dans chaque logiciel et chaque programme, avant toute écriture.

Le produit cartésien

`FROM eleve, classe` sans condition donne toutes les combinaisons : 12 × 3 = 36 lignes, sans message d'erreur. Symptôme : un résultat beaucoup trop gros, où chaque ligne se répète. Une condition `ON` fausse (`ON c.id = e.id`) donne aussi des lignes fausses sans erreur.

Le `LEFT JOIN` annulé par un `WHERE`

Une condition du `WHERE` sur la table de droite (`WHERE c.nom <> '4TTR'`) élimine les lignes où cette table vaut `NULL`, c'est-à-dire celles que le `LEFT JOIN` devait garder. Parade : ajouter `OR c.id IS NULL` .

✅ Teste-toi

1. Qu'est-ce que la redondance ? Donne un exemple.
2. Cite les trois anomalies provoquées par la redondance, avec un exemple pour chacune.
3. Pourquoi une faute de frappe comme `4TRR` ne provoque-t-elle aucun message d'erreur dans un tableau unique ?
4. Dans la phrase « Un client passe des commandes », quelles sont les entités et quel est le nom de l'association ?
5. Que signifient les deux chiffres d'une cardinalité ? Cite les quatre cardinalités possibles.
6. Lis à voix haute les deux cardinalités de `ELEVE (0,1) —appartient— (0,n) CLASSE` .
7. Pourquoi dit-on que le minimum est une décision de gestion ?
8. Qu'est-ce qu'une association 1-N ?

9. Énonce la règle de placement de la clé étrangère. Où va la clé étrangère de `VEHICULE (1,1) —appartient— (0,n) CLIENT` ?
10. Que signifie `#classe_id → classe.id` dans un MLD ?
11. Pourquoi faut-il activer `PRAGMA foreign_keys` dans chaque programme ?
12. Quels sont les deux tests à faire après avoir posé une clé étrangère, et quel message doivent-ils afficher ?
13. Qu'est-ce qu'une ligne orpheline ?
14. Combien de lignes renvoie `SELECT * FROM eleve, classe;` avec 12 élèves et 3 classes ? Comment s'appelle ce résultat ?
15. Tu veux la liste de tous les élèves avec leur classe, y compris ceux qui n'en ont pas. `JOIN` ou `LEFT JOIN` ? Pourquoi ?

▶ Réponses