Identifier ses données : la clé primaire

Deux élèves peuvent porter le même nom. Comment la base sait-elle de quelle ligne tu parles ? Identifiant naturel ou artificiel, clé primaire, et les contraintes qui empêchent les doublons et les cases vides.

4TTR 5TTR 6TTR Découverte

Une table peut contenir des milliers de lignes. Pour corriger l'une d'elles, la relier à autre chose ou simplement en parler, il faut pouvoir la **désigner sans hésitation**. Cet article montre comment on donne à chaque ligne une identité, et comment la base la protège.

> 💡 Télécharge `mon-ecole-depart.db` dans la sidebar et ouvre-la dans DB Browser for SQLite. Elle contient une table `eleve` de douze élèves, avec une colonne `id` qui numérote les lignes.

🎯 Objectifs

À la fin de cet article, tu seras capable de :

1. Expliquer pourquoi chaque table a besoin d'un identifiant.
2. Distinguer un identifiant naturel d'un identifiant artificiel.
3. Juger si une information fait un bon identifiant, à l'aide de quatre critères.
4. Reconnaître une clé primaire et prévoir comment SQLite la remplit.
5. Poser les contraintes `NOT NULL` et `UNIQUE` avec DB Browser et vérifier qu'elles fonctionnent.

Deux lignes identiques

Deux nouveaux élèves arrivent à l'école. Par hasard, ils s'appellent tous les deux Lucas Adam et sont nés le même jour.

```
nom  | prenom | date_naissance | commune
-----+--------+----------------+---------
Adam | Lucas  | 2010-03-12     | Jodoigne
Adam | Lucas  | 2010-03-12     | Jodoigne
```

Essaie de répondre à ces questions :

- Lequel des deux a obtenu 14 en mathématiques ?
- Lequel déménage à Wavre le mois prochain ?
- Comment corriger l'adresse e-mail de l'**un** des deux sans toucher à celle de l'autre ?

C'est impossible. Rien dans ces lignes ne permet de les distinguer. Le problème n'est pas qu'elles se ressemblent : c'est qu'**aucune information ne désigne une ligne et une seule**.

> 📖 NOUVELLE NOTION : L'IDENTIFIANT
>
> Un **identifiant** est une information qui désigne une ligne et une seule. Deux lignes ne peuvent jamais avoir le même identifiant. Grâce à lui, on peut retrouver, corriger ou relier une ligne précise.
>
> En anglais : GB ***identifier***.

Deux façons d'identifier

L'identifiant qui existe déjà

Certaines choses possèdent déjà un numéro unique dans la vie réelle.

Ce qu'on identifie	Son identifiant dans la vie réelle
un citoyen belge	le numéro de registre national
un livre publié	l'ISBN
un produit en magasin	le code-barres
une voiture	la plaque d'immatriculation
un pays	son code à deux lettres (`BE` , `FR`)

> 📖 **NOUVELLE NOTION : L'IDENTIFIANT NATUREL**
>
> Un **identifiant naturel** est une information qui existe en dehors de la base et qui est déjà unique : un ISBN, une plaque d'immatriculation. Il a du sens pour tout le monde.
>
> En anglais : GB ***natural key***.

L'identifiant qu'on invente

L'autre solution consiste à inventer un numéro qui n'existe que dans la base : 1 pour le premier élève encodé, 2 pour le suivant, et ainsi de suite. C'est le rôle de la colonne `id`.

> 📖 **NOUVELLE NOTION : L'IDENTIFIANT ARTIFICIEL**
>
> Un **identifiant artificiel** est un numéro créé uniquement pour la base de données. Il ne veut rien dire en dehors d'elle, et c'est justement sa force : comme il ne décrit rien, il n'a jamais de raison de changer.
>
> En anglais : GB ***surrogate key***.

Ce qui fait un bon identifiant

Un bon identifiant respecte quatre critères. S'il en rate un seul, il finira par poser problème.

Critère	La question à se poser
Unique	Deux éléments peuvent-ils avoir la même valeur ?
Obligatoire	Peut-il manquer, ne serait-ce qu'une fois ?
Stable	Peut-il changer au cours du temps ?
Non signifiant	Contient-il une information qui pourrait devenir fausse ?

Appliquons ces critères aux candidats pour identifier un élève :

Candidat	Unique	Obligatoire	Stable	Verdict
`nom`	❌ plusieurs Dubois	✅	⚠️ un mariage, une adoption	inutilisable
`nom` + `prenom`	❌ deux Lucas Adam	✅	⚠️	inutilisable
`email`	✅	❌ 3 élèves n'en ont pas	⚠️ une adresse change	non
numéro national	✅	✅	✅	bon… mais donnée sensible
`id`	✅	✅	✅	**oui**

> ⚠️ **LE NUMÉRO NATIONAL EST UN BON IDENTIFIANT SUR LE PAPIER, ET UN MAUVAIS CHOIX EN PRATIQUE.** C'est une donnée personnelle protégée par le RGPD, la loi européenne sur la protection des données. On ne la collecte que si on en a vraiment besoin, jamais « au cas où ». Un `id` inventé fait le même travail sans aucun risque.

> 🧠 **LA RÈGLE DU PARCOURS.**
>
> Chaque table a pour identifiant un **numéro artificiel** `id` . Si un identifiant naturel existe, comme l'adresse e-mail, on le garde dans une colonne ordinaire et on interdit les doublons dans cette colonne.

La clé primaire

Ouvre l'onglet **Structure de la base** et déplie la table `eleve`. La colonne `id` porte une petite clé : c'est la **clé primaire** de la table.

> 📖 **NOUVELLE NOTION : LA CLÉ PRIMAIRE**
>
> La **clé primaire** est la colonne choisie comme identifiant officiel d'une table. La base garantit que sa valeur est **unique** et **jamais vide**. Une table n'a qu'une seule clé primaire.
>
> En anglais : GB ***primary key***, souvent abrégé PK.

Dans le SQL de la table, elle apparaît sur la dernière ligne :

```
CREATE TABLE "eleve" (
    "id"            INTEGER,
    "nom"           TEXT,
    ...
    PRIMARY KEY("id")
);
```

`PRIMARY KEY` se traduit littéralement par « clé principale ».

Une règle que la base fait respecter

Essaie de tricher. Dans **Parcourir les données**, double-clique sur l'`id` d'Emma Bastin (le `2`) et remplace-le par `1`, qui appartient déjà à Lucas Adam.

DB Browser refuse la modification et affiche un message qui contient :

```
UNIQUE constraint failed: eleve.id
```

Traduit mot à mot : « contrainte d'unicité non respectée : eleve.id ». La base vient de refuser une donnée qui aurait cassé une règle.

📖 **NOUVELLE NOTION : LA CONTRAINTE**

Une **contrainte** est une règle attachée à une table, que la base vérifie à chaque ajout ou modification. Si une donnée ne respecte pas la règle, la base la refuse et affiche un message d'erreur.

En anglais : GB ***constraint***.

Le numéro qui se remplit tout seul

Quand tu insères une ligne sans donner d'`id`, SQLite choisit lui-même le numéro : **le plus grand `id` présent dans la table, plus un**. C'est pour ça que la colonne se remplit toute seule dans DB Browser.

Conséquence : si tu supprimes une ligne au milieu de la table, son numéro laisse un **trou** qui ne sera jamais rebouché.

```
avant    : 1, 2, 3, 4, 5
on supprime la ligne 3
après    : 1, 2, 4, 5
on ajoute une ligne : elle reçoit 6
```

⚠️ **LES TROUS DANS LA NUMÉROTATION SONT NORMAUX.** Un identifiant sert à **désigner** une ligne, pas à compter les lignes. Ne cherche jamais à « renuméroter proprement » : tout ce qui faisait référence à l'ancien numéro désignerait soudain une autre ligne.

💡 **ET LA CASE AI ?** DB Browser propose une case **AI**, pour `AUTOINCREMENT`, et tu la verras dans beaucoup d'exemples sur internet. En SQLite, elle est inutile : une colonne `INTEGER` en clé primaire se numérote déjà toute seule. Laisse-la décochée.

Interdire les cases vides : NOT NULL

Pour l'instant, rien n'empêche d'enregistrer un élève sans nom. Une ligne sans nom n'a pourtant aucun sens.

📖 **NOUVELLE NOTION : LA CONTRAINTE NOT NULL**

La contrainte **NOT NULL** interdit qu'une colonne reste sans valeur. Toute ligne doit obligatoirement y contenir quelque chose.

En anglais : *not null*, « pas nul », c'est-à-dire « jamais vide ».

Pose-la sur `nom` et `prenom` :

1. Onglet **Structure de la base** : clic droit sur la table `eleve`, puis **Modifier la table**.
2. Coche la case **NN** sur la ligne de `nom`, puis sur celle de `prenom`.
3. Clique sur **OK**, puis sur **Écrire les modifications** (Ctrl+S).

Vérifie que la contrainte fonctionne. Dans **Parcourir les données**, fais un clic droit sur le prénom de Noah Charlier et choisis **Définir comme NULL**. DB Browser refuse et affiche :

```
NOT NULL constraint failed: eleve.prenom
```

⚠️ **UNE CASE VIDÉE N'EST PAS UNE CASE NULL.** Si tu effaces le texte d'une case au clavier, DB Browser y range un texte vide `''`, qui compte comme une valeur. `NOT NULL` le laisse passer. Seule l'absence totale de valeur est refusée.

Interdire les doublons : UNIQUE

L'adresse e-mail est un identifiant naturel : deux élèves ne partagent jamais la même. On ne l'utilise pas comme clé primaire, mais on veut quand même empêcher les doublons.

> 📖 **NOUVELLE NOTION : LA CONTRAINTE UNIQUE**
>
> La contrainte **UNIQUE** interdit que deux lignes aient la même valeur dans une colonne. Contrairement à la clé primaire, une table peut avoir plusieurs colonnes `UNIQUE`, et ces colonnes peuvent rester vides.
>
> En anglais : *unique*, « unique ».

Pose-la sur `email` : **Modifier la table**, coche **U** sur la ligne de `email`, **OK**, puis Ctrl+S.

Vérifie : copie l'adresse de Lucas Adam (`lucas.adam@ecole.be`) dans la case e-mail d'Emma Bastin. DB Browser refuse :

```
UNIQUE constraint failed: eleve.email
```

Maintenant, observe les élèves sans adresse e-mail : Noah Charlier, Chloé Fontaine et Camille Jacques ont tous les trois `NULL`. La contrainte a pourtant été acceptée.

> 🧠 **UNIQUE ACCEPTE PLUSIEURS NULL.** Pour la base, deux valeurs inconnues ne sont pas égales : on ne peut pas affirmer que deux choses qu'on ignore sont identiques. Si tu veux « pas de doublon **et** toujours rempli », il faut cocher **NN** et **U**.

	Clé primaire	UNIQUE
Combien par table ?	une seule	autant qu'on veut
Case vide autorisée ?	non	oui, même plusieurs fois
Rôle	l'identifiant officiel de la ligne	une protection contre les doublons

Regarde enfin le SQL de la table dans **Structure de la base**. Tes clics y ont ajouté les contraintes :

```sql
CREATE TABLE "eleve" (
    "id"             INTEGER,
    "nom"            TEXT NOT NULL,
    "prenom"         TEXT NOT NULL,
    "date_naissance" TEXT,
    "commune"        TEXT,
    "email"          TEXT UNIQUE,
    "absences"       INTEGER,
    PRIMARY KEY("id")
);
```

Ta base correspond maintenant à `mon-ecole-eleves.db`, disponible dans la sidebar.

> ⚠️ **ON NE POSE PAS UNE CONTRAINTE SUR DES DONNÉES DÉJÀ FAUSSES.** Si deux élèves avaient eu la même adresse e-mail, DB Browser aurait refusé de poser `UNIQUE`. C'est une bonne nouvelle : la base t'aurait signalé un problème. Il faut alors corriger les données avant de contraindre.

🧪 Exercices

Exercice 1 — Bon ou mauvais identifiant ? ★☆☆☆☆

Pour chaque proposition, dis si elle ferait un bon identifiant. Si ce n'est pas le cas, nomme **le critère qu'elle rate**.

Table	Identifiant proposé
`eleve`	l'adresse du domicile
`livre`	l'ISBN
`voiture`	la plaque d'immatriculation
`commune`	le code postal
`professeur`	les initiales (`LL` , `SD`)
`article`	le nom du produit
`classe`	le nom de la classe (`4TTR`)

> 💡 Pour le code postal : plusieurs communes partagent parfois le même, et certaines villes en ont plusieurs. Qu'en conclus-tu ?

Exercice 2 — Provoquer les refus ★★☆☆☆

Travaille sur une copie de ta base. Pour chaque cas, fais la manipulation dans **Parcourir les données** et recopie le message exact.

1. Donner à un élève l'`id` d'un autre élève.
2. Donner à un élève l'adresse e-mail d'un autre élève.
3. Définir comme NULL le nom d'un élève.
4. Effacer au clavier le nom d'un élève. Que se passe-t-il, et pourquoi ?
5. Définir comme NULL l'adresse e-mail de Lucas Adam. Pourquoi est-ce accepté ?

Termine par **Annuler les modifications**.

Exercice 3 — Les trous dans la numérotation ★★☆☆☆

Crée une base d'essai avec une table `joueur (id, pseudo)`, où `id` est une clé primaire `INTEGER`.

1. Insère cinq joueurs et note leurs `id`.
2. Supprime le joueur `3` (bouton **Supprimer l'enregistrement**), puis insère un nouveau joueur. Quel `id` reçoit-il ?
3. Supprime maintenant le dernier joueur, puis insère un nouveau joueur. Quel `id` reçoit-il cette fois ? Explique la différence avec la question 2.
4. Ton chef te demande de « renuméroter les joueurs de 1 à 5 ». Donne deux raisons de refuser.

Exercice 4 — Les manuels scolaires ★★☆☆☆

L'école veut gérer ses manuels : ISBN, titre, éditeur, année de parution et nombre d'exemplaires.

1. L'ISBN est-il un bon identifiant naturel ? Vérifie les quatre critères.
2. En appliquant la règle du parcours, écris le MLD de la table `manuel` et souligne la clé primaire.
3. Quel type choisis-tu pour l'ISBN ? Attention : un ISBN peut se terminer par la lettre `X`.
4. Quelles contraintes poses-tu, et sur quelles colonnes ?

Exercice 5 — Une table sans identifiant ★★★☆☆

Un collègue a créé une table `contact (nom, prenom, email)` sans clé primaire. Elle contient déjà 300 lignes, dont quelques doublons.

1. Explique à ton collègue, en trois phrases simples, ce qu'il ne pourra pas faire avec cette table.
2. Pourquoi ne peut-il pas simplement cocher **U** sur `email` ?
3. Dans quel ordre faut-il travailler pour arriver à une table correcte ? Quelle question dois-tu lui poser avant de supprimer quoi que ce soit ?

✍️ À retenir

- Un **identifiant** désigne une ligne et une seule ; sans lui, on ne peut ni corriger ni relier une ligne.
- Un identifiant **naturel** existe dans la vie réelle ; un identifiant **artificiel** est un numéro inventé pour la base.
- Un bon identifiant est **unique, obligatoire, stable et non signifiant**.
- La **clé primaire** est l'identifiant officiel de la table : unique, jamais vide, une seule par table. Dans ce parcours, c'est toujours `id`.
- SQLite numérote tout seul une clé primaire `INTEGER` ; les trous dans la numérotation sont normaux.
- Une **contrainte** est une règle vérifiée par la base : `NOT NULL` interdit l'absence de valeur, `UNIQUE` interdit les doublons mais accepte plusieurs `NULL`.

Suite

Ta table est prête et protégée. Dans l'article Lister le contenu d'une table, tu poses ta première question à la base en SQL.