

COURS COMPLET : CONCEPTION DE BASE DE DONNÉES SQL

Objectif : Comprendre comment concevoir une base de données relationnelle de A à Z, appliqué au projet FITNESS CLASH.

Prérequis : Aucun (tout est expliqué depuis zéro)

Durée estimée : 2-3 heures de lecture + exercices

TABLE DES MATIÈRES

1. Les Fondamentaux
 2. Concepts Essentiels
 3. Méthodologie de Conception
 4. Schéma BDD FITNESS CLASH
 5. Types de Données SQL
 6. Exercices Pratiques
 7. Vocabulaire Technique
-

PARTIE 1 : LES FONDAMENTAUX

C'est quoi une Base de Données ?

Définition simple :

Une **base de données** est un système organisé qui permet de **stocker**, **gérer** et **récupérer** des informations de manière structurée et efficace.

Sans base de données : Tu stockes tout dans des fichiers texte en vrac. Bonne chance pour retrouver l'information rapidement !

Avec une base de données : Tu as un système intelligent qui organise tout et te permet de retrouver n'importe quelle info en quelques millisecondes.

Analogie : La Bibliothèque

Imagine une **grande bibliothèque municipale** :

Élément Bibliothèque	Équivalent Base de Données
Étagères thématiques	Tables (Users, Workouts, Exercises)
Livres	Enregistrements (lignes/rows)
Caractéristiques d'un livre (titre, auteur, année, ISBN)	Colonnes/Attributs
ISBN (numéro unique)	Clé primaire (Primary Key)
Système de classement (Dewey)	Index et Relations
Fiches de prêt (qui a emprunté quoi)	Relations entre tables
Bibliothécaire	Système de Gestion de BDD (PostgreSQL)

Exemple concret :

Sans BDD = Tu jettes tous les livres en vrac dans une pièce

"Harry Potter, JK Rowling, 1997, fantasy"
 "1984, George Orwell, 1949, dystopie"
 "Le Seigneur des Anneaux, Tolkien, 1954, fantasy"
 ...

Avec BDD = Tout est organisé dans des tables

Table: Books

id	title	author	year	genre
1	Harry Potter	JK Rowling	1997	Fantasy
2	1984	George Orwell	1949	Dystopia
3	Le Seigneur...	Tolkien	1954	Fantasy

Maintenant tu peux facilement :

- Trouver tous les livres de fantasy : `SELECT * FROM Books WHERE genre = 'Fantasy'`
- Trouver tous les livres publiés après 1990 : `SELECT * FROM Books WHERE year > 1990`
- Compter combien de livres tu as : `SELECT COUNT(*) FROM Books`

SQL vs NoSQL : Comprendre la différence

Il existe **deux grandes familles** de bases de données :

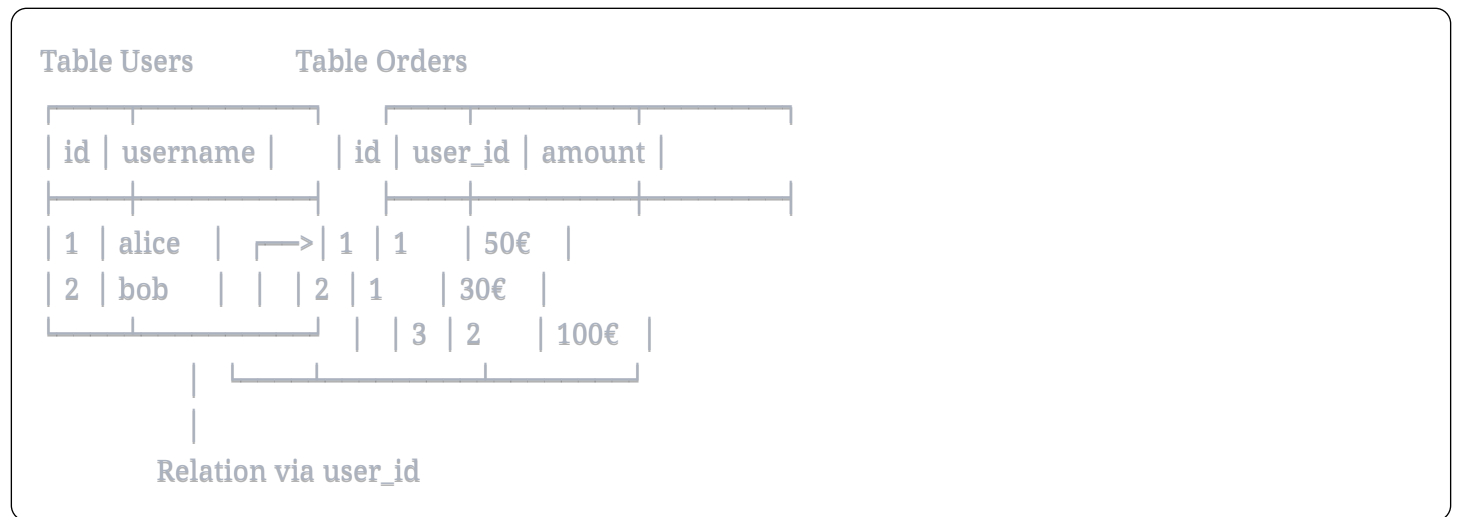
SQL (Bases de Données Relationnelles)

Principe : Les données sont organisées en **tables** avec des **colonnes fixes** et des **relations** entre elles.

Analogie : Un **fichier Excel** bien structuré avec plusieurs feuilles reliées entre elles.

Exemples : PostgreSQL, MySQL, SQLite, Oracle

Structure :



Avantages :

- ✓ Structure claire et prévisible
- ✓ Relations fortes entre données
- ✓ Intégrité garantie (pas de données incohérentes)
- ✓ Langage standard (SQL) utilisé partout
- ✓ Transactions sécurisées (ACID)

Inconvénients :

- ⚠ Moins flexible (colonnes fixes)
- ⚠ Scalabilité horizontale plus complexe

📦 NoSQL (Bases de Données Non-Relationnelles)

Principe : Les données sont stockées sous forme de **documents JSON** flexibles, sans structure fixe.

Analogie : Des **post-its** que tu colles sur un mur. Chaque post-it peut contenir n'importe quelle info.

Exemples : MongoDB, Firebase, CouchDB




Structure :

json




```
// Document User 1
{
  "id": 1,
  "username": "alice",
  "email": "alice@mail.com",
  "preferences": {
    "theme": "dark",
    "language": "fr"
  }
}

// Document User 2 (structure différente, c'est OK!)
{
  "id": 2,
  "username": "bob",
  "age": 25
}
```

Avantages :

-  Très flexible (structure adaptable)
-  Scalabilité horizontale facile
-  Rapide pour certaines opérations

Inconvénients :

-  Pas de relations fortes
-  Risque de données incohérentes
-  Chaque BDD a son propre langage

 **Pour FITNESS CLASH : On choisit SQL (PostgreSQL)**

Pourquoi ce choix ?

1. Relations claires :

- Un utilisateur a plusieurs séances
- Une séance contient plusieurs exercices
- Ces relations sont naturelles en SQL

2. Intégrité des données :

- Si on supprime un utilisateur, toutes ses séances sont supprimées automatiquement
- Impossible d'avoir une séance sans utilisateur associé

3. Requêtes complexes faciles :

- "Donne-moi tous les utilisateurs qui ont fait au moins 3 séances cette semaine"
- "Calcule le classement de la compétition"
- SQL excelle dans ce genre de requêtes

4. SQLAlchemy :

- ORM Python qui facilite l'utilisation de SQL
- Tu codes en Python, il génère le SQL pour toi

5. Gratuit et performant :

- PostgreSQL est open-source
- Utilisé par des millions d'applications

PARTIE 2 : CONCEPTS ESSENTIELS

Vocabulaire Fondamental

1 Table (ou Entité)

Définition :

Une **table** est un ensemble de données du même type, organisées en lignes et colonnes.

Analogie :

Une table = un **type de fiche** dans un classeur administratif.

Exemple :

Table Users = tous les utilisateurs de FITNESS CLASH

Table: Users

id	username	email	created_at
1	alice	alice@mail.com	2025-01-15
2	bob	bob@mail.com	2025-02-20
3	charlie	charlie@mail.com	2025-03-10

Règles de nommage :

- ☒ Utilise le pluriel : `Users`, `Workouts`, `Exercises`
- ☒ Pas d'espaces : utilise underscore `workout_exercises` ou CamelCase `WorkoutExercises`
- ☒ Noms clairs et descriptifs

2 Enregistrement (Row/Ligne)

Définition :

Un **enregistrement** est une instance spécifique dans une table. C'est une ligne du tableau.

Analogie :

Un enregistrement = une **fiche remplie** dans un classeur.

Exemple :

Dans la table `Users`, l'enregistrement avec `id=1` représente l'utilisateur "alice".

id	username	email	created_at
1	alice	alice@mail.com	2025-01-15

← Ceci est UN enregistrement

En SQL :

- Ajouter un enregistrement : `INSERT INTO Users ...`
- Lire des enregistrements : `SELECT * FROM Users`
- Modifier un enregistrement : `UPDATE Users SET ...`
- Supprimer un enregistrement : `DELETE FROM Users WHERE id = 1`

3 Attribut (Colonne/Champ)

Définition :

Un **attribut** est une caractéristique d'une entité. C'est une colonne du tableau.

Analogie :

Les attributs = les **cases à remplir** sur un formulaire (Nom, Prénom, Date de naissance).

Exemple :

Dans la table **Users** :

- **id** est un attribut
- **username** est un attribut
- **email** est un attribut
- **created_at** est un attribut

↓ Attributs (colonnes)

id	username	email	created_at
1	alice	alice@mail.com	2025-01-15

Chaque attribut a :

- Un **nom** : **username**
- Un **type de données** : **VARCHAR(50)** (texte de max 50 caractères)
- Des **contraintes** optionnelles : **UNIQUE**, **NOT NULL**, etc.

4 Clé Primaire (Primary Key - PK)

Définition :




Une **clé primaire** est un attribut (ou ensemble d'attributs) qui identifie **de manière unique** chaque enregistrement d'une table.

Analogie :

La clé primaire = ton **numéro de sécurité sociale**.

- Unique dans tout le pays
- Ne change jamais
- Permet de t'identifier sans ambiguïté

Règles STRICTES :

1.  **Unique** : Pas de doublon possible
2.  **Non nul** : Toujours présent (jamais vide)
3.  **Immuable** : Ne change jamais une fois créé

Convention :

Presque toujours appelée `id` et de type `INTEGER AUTO-INCREMENT`.

Exemple :

Table: Users

id	username	email	
1	alice	alice@mail.com	← id=1 identifie uniquement alice
2	bob	bob@mail.com	← id=2 identifie uniquement bob

↑
PRIMARY KEY

En SQL :

```
sql

CREATE TABLE Users (
  id SERIAL PRIMARY KEY, -- SERIAL = auto-incrémentation
  username VARCHAR(50),
  email VARCHAR(100)
);
```

Pourquoi c'est important ?

Sans clé primaire :

username	email	
alice	alice@mail.com	
alice	alice@mail.com	← Doublon ! C'est la même personne ?

Avec clé primaire :


```
sql

CREATE TABLE Workouts (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES Users(id),
  date DATE NOT NULL
);
```

Protection de l'intégrité :

❌ **Impossible** de créer un workout pour un utilisateur qui n'existe pas :

```
sql

INSERT INTO Workouts (user_id, date) VALUES (999, '2025-10-01');
-- ❌ ERREUR : user_id=999 n'existe pas dans Users
```

✅ **Possible** si l'utilisateur existe :

```
sql

INSERT INTO Workouts (user_id, date) VALUES (1, '2025-10-01');
-- ✅ OK : alice (id=1) existe
```

Action CASCADE :

Si tu supprimes un utilisateur, que se passe-t-il avec ses workouts ?

Option 1 : **ON DELETE CASCADE** (supprimer aussi les workouts)

```
sql

user_id INTEGER REFERENCES Users(id) ON DELETE CASCADE
```

Option 2 : **ON DELETE RESTRICT** (empêcher la suppression si workouts existent)

```
sql

user_id INTEGER REFERENCES Users(id) ON DELETE RESTRICT
```

6 Relations entre Tables

Il existe **3 types de relations** possibles entre deux tables.

A. Relation 1-à-N (One-to-Many) 1 2 3 4

Définition :

Une entité A peut être liée à **plusieurs** entités B, mais chaque entité B n'est liée qu'à **une seule** entité A.

Notation : $A (1) \longleftrightarrow (N) B$

Analogie :

Un **parent** peut avoir plusieurs **enfants**, mais chaque enfant n'a qu'un seul parent biologique (dans cet exemple simplifié).



Exemple FITNESS CLASH :

Un **utilisateur** peut avoir **plusieurs séances**, mais chaque **séance** appartient à **un seul utilisateur**.



En SQL :

```
sql

-- Table "Mère" (côté 1)
CREATE TABLE Users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50)
);

-- Table "Fille" (côté N)
CREATE TABLE Workouts (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES Users(id), ← Foreign Key
  date DATE
);
```

Règle : La clé étrangère se place toujours du côté N (Many).

B. Relation N-à-N (Many-to-Many) ✂

Définition :

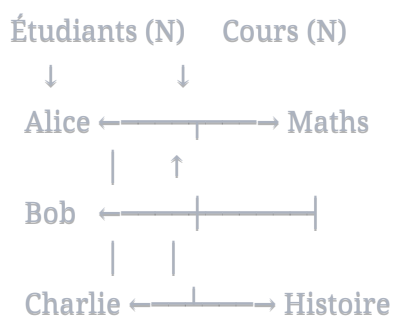
Plusieurs entités A peuvent être liées à plusieurs entités B, et inversement.

Notation : $A(N) \longleftrightarrow (N) B$

Analogie :

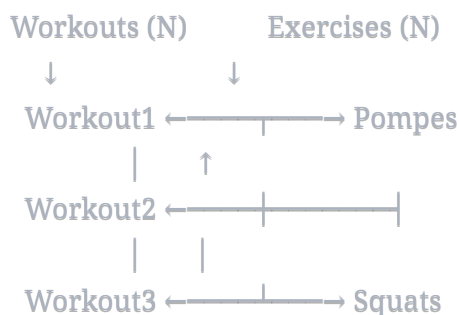
Des **étudiants** et des **cours**.

- Un étudiant suit plusieurs cours
- Un cours a plusieurs étudiants



Exemple FITNESS CLASH :

Une **séance** contient **plusieurs exercices**, et un **exercice** peut apparaître dans **plusieurs séances**.



PROBLÈME : Comment représenter ça en SQL ?

✂ Impossible directement :

```
sql
-- ✂ Ça ne marche pas !
CREATE TABLE Workouts (
  id SERIAL PRIMARY KEY,
  exercises ??? -- Comment stocker plusieurs exercices ici ?
);
```

✓ **SOLUTION** : Créer une **table de liaison** (junction table).

En SQL :

```
sql

-- Table "Mère" 1
CREATE TABLE Workouts (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES Users(id),
  date DATE
);

-- Table "Mère" 2
CREATE TABLE Exercises (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100),
  category VARCHAR(50)
);

-- Table de LIAISON (junction table)
CREATE TABLE WorkoutExercises (
  id SERIAL PRIMARY KEY,
  workout_id INTEGER REFERENCES Workouts(id), ← FK vers Workouts
  exercise_id INTEGER REFERENCES Exercises(id), ← FK vers Exercises
  reps INTEGER,
  duration INTEGER
);
```

Schéma relationnel :

Workouts (1) ↔ (N) WorkoutExercises (N) ↔ (1) Exercises

Exemple de données :

Workouts:

id	user_id	date
1	1	2025-10-01
2	1	2025-10-03

Exercises:

id	name	category
1	Pompes	upper
2	Squats	lower
3	Planche	core

WorkoutExercises (table de liaison):

id	workout_id	exercise_id	reps	duration
1	1	1	20	NULL
2	1	2	15	NULL
3	2	1	25	NULL
4	2	3	NULL	60

Interprétation :

- Workout 1 contient : Pompes (20 reps) + Squats (15 reps)
- Workout 2 contient : Pompes (25 reps) + Planche (60 secondes)
- L'exercice "Pompes" apparaît dans 2 workouts différents

C. Relation 1-à-1 (One-to-One)

Définition :

Une entité A est liée à **une seule** entité B, et inversement.

Notation : $A(1) \leftrightarrow (1) B$

Analogie :

Ton **passport** et **toi**.

- Un passeport = une personne
- Une personne = un passeport

Exemple (rare) :

Un **utilisateur** a un **profil détaillé**.

```
sql

CREATE TABLE Users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50)
);

CREATE TABLE UserProfiles (
  id SERIAL PRIMARY KEY,
  user_id INTEGER UNIQUE REFERENCES Users(id), ← UNIQUE = relation 1-1
  bio TEXT,
  avatar_url VARCHAR(255)
);
```

Note : Relation 1-1 rare en pratique. On préfère souvent tout mettre dans la même table.

Pour FITNESS CLASH : Pas nécessaire dans votre MVP.

PARTIE 3 : MÉTHODOLOGIE DE CONCEPTION

Les 5 Étapes pour Concevoir une BDD

Étape 1 : Identifier les Entités

Question : *Quels sont les "objets" principaux de mon application ?*

Méthode :

1. Relis tes **User Stories**
2. Souligne tous les **noms** importants
3. Garde seulement ceux qui doivent être stockés

Exemple pour FITNESS CLASH :

User Story :

"En tant que **sportif**, je veux lancer une **séance** contenant des **exercices** et valider ma **séance complétée** pour suivre mes **progrès**. Je veux aussi participer à une **compétition** hebdomadaire pour obtenir un **score**."

Entités identifiées :

- ☒ **Utilisateur** (sportif)
- ☒ **Séance** (workout)
- ☒ **Exercice** (mouvement au poids du corps)
- ☒ **Compétition** (mini-compét' hebdo)
- ☒ **Score** (résultat de compétition)
- ☒ **Animation** (stickman pour chaque exercice)

Résultat : 6 entités principales

Étape 2 : Définir les Attributs

Question : *Quelles informations dois-je stocker pour chaque entité ?*

Règles :

1. ☒ Toujours avoir un `id` (clé primaire)
2. ☒ Ajouter `created_at` et `updated_at` pour traçabilité
3. ☒ Ne stocker que l'essentiel (pas de sur-ingénierie)
4. ☒ Éviter les données calculables (ex: `age` → calculé depuis `birthdate`)

Méthode :

- Pose-toi la question : "De quoi ai-je besoin pour cette entité ?"
- Pense aux User Stories et aux fonctionnalités

Exemple pour `Users` :

Besoins :

- Identifier l'utilisateur → `id`
- Se connecter → `username`, `email`, `password_hash`
- Traçabilité → `created_at`, `updated_at`

```
sql

CREATE TABLE Users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Explication des contraintes :

- **SERIAL** : auto-incrémentation (1, 2, 3, ...)
 - **PRIMARY KEY** : clé primaire (unique et non nul)
 - **VARCHAR(50)** : texte de max 50 caractères
 - **UNIQUE** : pas de doublon (deux users ne peuvent avoir le même email)
 - **NOT NULL** : obligatoire (ne peut pas être vide)
 - **DEFAULT CURRENT_TIMESTAMP** : valeur par défaut = date/heure actuelle
-

Étape 3 : Identifier les Relations

Question : *Comment les entités sont-elles liées entre elles ?*

Méthode : Pour chaque paire d'entités, demande-toi :

- *Combien de A peuvent être liés à un B ?*
- *Combien de B peuvent être liés à un A ?*

Tableau des questions :

Question	Réponse	Type de relation
Combien de workouts peut avoir un user ?	Plusieurs (N)	1-N
Combien de users peut avoir un workout ?	Un seul (1)	1-N

→ **Relation 1-N** entre Users et Workouts

Exemples FITNESS CLASH :

Entité A	Entité B	Type	Explication
Users	Workouts	1-N	1 user a plusieurs séances
Workouts	Exercices	N-N	1 séance a plusieurs exercices, 1 exercice dans plusieurs séances
Exercices	Animations	1-1	1 exercice a 1 animation stickman