COURS

Bases de données relationnelles — SQL

Tle

Programme de terminale générale, enseignement de spécialité — BO spécial n°8 du 25 juillet 2019

> Interroger deux tables de mille lignes chacune en oubliant la condition qui les relie ne produit pas une erreur : cela produit un million de lignes. La requête est valide, elle s'exécute, et son résultat n'a aucun sens. C'est la faute la plus fréquente du chapitre, et la plus silencieuse.

1 Le modèle relationnel

DÉFINITION

Une **clé primaire** identifie de façon **unique** chaque ligne d'une table : deux lignes ne peuvent pas la partager, et elle ne peut pas être vide. Une **clé étrangère** est une colonne qui **référence** la clé primaire d'une autre table : c'est elle qui établit le lien entre deux tables. La contrainte d'**intégrité référentielle** garantit qu'une clé étrangère pointe toujours vers une ligne qui **existe**.

L'intérêt de cette organisation est d'éviter la **redondance** : une information n'est stockée qu'à **un seul endroit**, et les autres tables y renvoient. Sans cela, modifier une donnée obligerait à la corriger partout où elle figure, et la moindre omission produirait deux versions contradictoires dans la même base.

2 Interroger une table

PROPRIÉTÉ

Une requête **SQL** se lit dans un ordre précis. **SELECT** indique les colonnes à afficher, *FROM* la table, *WHERE* la condition de **filtrage des lignes**. **GROUP BY** regroupe les lignes partageant une valeur, ce qui permet d'appliquer une fonction d'agrégat — compte, somme, moyenne. *HAVING* filtre ensuite ces **groupes**, là où *WHERE* filtre les **lignes**. *ORDER BY* trie le résultat.

$$WHEREfiltreleslignes\cdot HAVINGfiltrelesgroupes$$

La distinction entre *WHERE* et *HAVING* découle de l'**ordre d'exécution** : les lignes sont d'abord filtrées, puis regroupées, puis les groupes filtrés. Une condition portant sur un agrégat — « les clients ayant passé plus de trois commandes » — ne peut donc **pas** figurer dans un *WHERE* : au moment où il s'applique, les groupes n'existent pas encore.

3 Joindre deux tables

RÈGLE

Une **JOIN** rapproche deux tables selon une **condition** qui relie la **clé étrangère** de l'une à la **clé primaire** de l'autre. Sans cette condition, la requête n'échoue pas : elle associe **chaque ligne de la première table à toutes les lignes de la seconde**, produisant un nombre de lignes égal au **produit** des deux tailles. Le résultat est volumineux, dénué de sens, et parfaitement valide du point de vue du moteur.

$$sansconditiondejointure, n \times mlignes$$

LE PIÈGE QUI COÛTE LE PLUS DE POINTS

Interroger deux tables ensemble en oubliant la condition qui relie la clé étrangère de l'une à la clé primaire de l'autre. — Le moteur n'a aucun moyen de deviner quelles colonnes doivent se correspondre : il exécute ce qu'on lui demande, c'est-à-dire toutes les combinaisons possibles. Sur de petites tables de test, le résultat contient quelques dizaines de lignes et l'anomalie passe inaperçue ; sur les données réelles, elle produit des millions de lignes et un résultat faux. Le symptôme apparaît donc **après** la mise en production.

Le réflexe : Compter les tables jointes : il faut **une condition de moins** que de tables.

4 L'injection SQL

PROPRIÉTÉ

Une **injection SQL** consiste à faire interpréter comme du **code** ce qui était censé être une **donnée**. Elle survient dès qu'une requête est construite en **concaténant** une saisie d'utilisateur : la saisie peut alors contenir des caractères qui referment la chaîne attendue et ajoutent une condition, un opérateur ou une instruction entière. La faille ne vient pas du langage : elle vient du **mélange** entre la structure de la requête et les valeurs qu'on y place.

La parade est structurelle, non cosmétique : les **requêtes préparées** déclarent d'abord la **structure** de la requête, puis y passent les valeurs **séparément**. Le moteur sait alors que ces valeurs sont des données et ne les interprète jamais comme du code, quel que soit

leur contenu. Filtrer les caractères « dangereux » est une parade **partielle** — elle suppose qu'on a pensé à tous les cas, ce qui n'est jamais démontrable.

CONSTRUIRE UNE REQUÊTE SUR PLUSIEURS TABLES

1. Identifier les **tables** nécessaires et, pour chacune, ce qu'on en attend.
2. Repérer les **clé étrangère**s qui les relient : ce sont elles qui donnent les conditions de **JOIN**.
3. Écrire la jointure avec sa condition, et vérifier qu'il y a **une condition de moins que de tables**.
   Trois tables jointes demandent deux conditions de rapprochement.
4. Ajouter le filtrage : *WHERE* pour les **lignes**, *HAVING* pour les **groupes** après **GROUP BY**.
5. Vérifier le **nombre de lignes** obtenu : un résultat anormalement volumineux signale une condition manquante.

À VÉRIFIER

- Chaque **JOIN** a-t-elle sa **condition** de rapprochement ?
- Ai-je **une condition de moins** que de tables jointes ?
- Mon filtre porte-t-il sur des **lignes** (*WHERE*) ou des **groupes** (*HAVING*) ?
- Ai-je vérifié le **nombre de lignes** du résultat ?
- Mes requêtes construites avec une saisie utilisateur sont-elles **préparées** ?
- Ai-je évité de **concaténer** une saisie dans une requête ?

À RETENIR

- **Clé primaire** : identifie de façon **unique**. **Clé étrangère** : référence une autre table.
- L'**intégrité référentielle** garantit que la référence pointe vers une ligne existante.
- Éviter la **redondance** : une information à **un seul endroit**.
- **SELECT** les colonnes, *FROM* la table, *WHERE* la condition sur les **lignes**.
- **GROUP BY** regroupe ; *HAVING* filtre les **groupes**, *WHERE* filtre les lignes.
- *WHERE* s'applique **avant** le regroupement : d'où l'impossibilité d'y mettre un agrégat.
- Une **JOIN** exige une **condition** reliant clé étrangère et clé primaire.
- Sans condition : **n × m** lignes, résultat valide et dénué de sens.
- **Une condition de moins que de tables** jointes.
- **Injection SQL** : faire interpréter une **donnée** comme du **code**.
- Parade : **requêtes préparées**, jamais la concaténation.