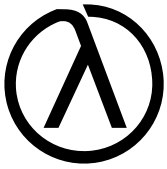

Les clés dans les bases de données relationnelles

1. Les bases théoriques

Les bases de données relationnelles s'appuient sur l'**algèbre relationnel**.

On y manipule des **relations**[1] (des tables ou des vues), dont chacune doit notamment être dotée d'une **clé primaire**.

La clé primaire d'une relation doit être composée d'un ensemble de colonnes de cette relation (pouvant ou non être limité à une seule colonne) dont l'ensemble des valeurs sera forcément **unique** pour toute ligne de la relation.

Les clés primaires sont automatiquement **indexées** par les systèmes de gestion de bases de données, ce qui permet de considérablement accélérer les recherches quand on veux accéder à des données en connaissant les valeurs de leurs clés.

Enfin, ces clés primaires peuvent avoir vocation à être reprises sous forme de **clés étrangères** dans d'autres tables associées.

2. Clés naturelles et clés de substitution

On appelle « clé naturelle » une clé constituée de colonnes naturellement présentes dans la relation, par opposition à des colonnes qui lui auraient été rajoutées pour des raisons purement techniques.

Des colonnes peuvent effectivement être ajoutées à une relation, précisément pour servir de clé. On parle alors de « clé de substitution » (*suroggate key* en anglais).

Quoi qu'il en soit, dans une base de données correctement constituée, toutes les relations (tables) devraient avoir chacune une ou plusieurs **clés naturelles candidates**, parmi lesquelles on pourrait choisir une clé primaire naturelle. Si on ne parvient pas à en identifier au moins une par relation, c'est qu'il y a un défaut dans l'analyse du modèle de données et que la copie est à revoir.

Une fois que l'on s'est assuré qu'il y a bien au moins une clé naturelle candidate par relation :

1. On peut déclarer une contrainte d'unicité sur chacune de ces clés candidates.

---

1. Une petite précision pour certain·es lecteur·ices qui pourraient confondre les termes « relation » et « association ». Certain·es appellent parfois « relation » le fait qu'une table soit reliée à une autre (donc « mise en relation », pourrait on dire) par le biais d'une clé étrangère ou d'une table de jointure. Mais dans le vocabulaire consacré, issu le l'algèbre relationnel, chaque table, prise indépendamment des autres, constitue à elle seule une relation, en ce qu'elle met en relation les colonnes qui la composent. Pour parler des liens entre tables, on pourra plutôt utiliser les termes « association » ou « jointure ».

On peut également vouloir déclarer des index sur les colonnes correspondant à ces clés candidates (un index sur l'ensemble des colonnes consituant chaque clé candidate).

2. On peut se poser la question de préférer ou non une clé de substitution à une clé naturelle.

Cette question trouve généralement vite une réponse : à moins de se trouver dans un environnement technique très contraint en quantité de mémoire disponible, dans lequel l'économie de chaque colonne serait bonne à prendre, on a tout lieu de préférer une clé de substitution.

La principale raison à ceci est qu'une clé de substitution est systématiquement composée d'une seule colonne, alors qu'une clé naturelle sera souvent composée de plusieurs colonnes. Or, la clé primaire d'une relation peut avoir vocation à être reprise sous forme de clés étrangères dans d'autres relations.

À ce titre, il est plus simple de ne reprendre qu'une colonne plutôt que plusieurs. Et cela rendra les jointures moins fastidieuses quand il s'agira d'écrire des requêtes pour interroger la base de données.

3. Entier ou UUID ?

Une fois qu'on a déterminé qu'on allait doter chaque table de notre base de données d'une clé primaire de substitution, on pourra se trouver face à un choix : de quel type de données devra être la colonne constituant cette clé ? La plupart des systèmes de gestion de bases de données modernes proposent :

- Un type **nombre entier auto-incrémenté** ou, à défaut, un mécanisme de **séquence** pour générer des nombres entiers croissants.
- Un type **UUID**.

Les UUID sont intéressants en ce que se sont des identifiants universels, dont chacun sera unique non seulement dans la base de données, mais en fait également dans tous l'Univers. En comparaison, les nombres entiers auto-incrémentés ne sont uniques que pour une table donnée dans une base donnée.

Seulement, les UUID ne sont pas **monotones** : il ne sont pas soit croissants, soit décroissants, quand on les génère les uns à la suite des autres. Cette absence de monotonie présente un inconvénient majeur vis à vis des algorithmes utilisés pour l'indexation des données : elles les rend beaucoup moins performants que quand ils opèrent sur des données monotones.

À l'inverse, des nombres entiers auto-incrémentés ou issus de séquences sont, par construction, forcément croissants (donc monotones) dans le temps. Cela les rends idéaux pour l'indexation des données.

Pour cette raison, on préférera donc utiliser des nombres entiers auto-incrémentés ou issus de séquences pour constituer nos clés primaires.

Toutefois, rien n'empêche d'également doter la table d'une colonne de type UUID qui sera, comme les clés naturelles candidates, dotée d'une contrainte d'unicité et éventuellement d'un index.

Cette colonne pourra s'avérer utile, notamment, dans le cas où il sera demandé d'effectuer des copies de données d'une base à une autre (par exemple d'un environnement de production vers un environnement de développement, pour y reproduire des dysfonctionnements constatés). Les mêmes données auront des clés primaires (incrémentées à partir de compteurs distincts) différentes d'une base à l'autre, mais toujours les mêmes UUID.

En outre, ces UUID pourront éventuellement servir d'identifiants publics aux données.

Il existe un débat sur le fait qu'il est acceptable ou non, voire de bonne pratique ou non, de faire apparaître un identifiant technique plutôt que naturel dans l'URI d'une ressource servie par un service web ou même, plus discrètement, sous forme de propriété `id` dans des échanges en JSON (par exemple).

Si on opte pour ce qui est une forme de publication de ces identifiants techniques, ce ne devrait jamais être les entiers incrémentés qui constituent les clés primaires des données en base. Ces entiers et le fait qu'ils soient incrémentés relèvent de détails d'implémentation qui devraient demeurer privés.

Des UUID sont de meilleurs (ou moins mauvais[2]) choix en la matière.

4. Mes préconisations résumées

Voici donc comment je procède pour des bases de données relationnelles sur lesquelles j'ai la main. Pour chaque table :

1. Identification des clés naturelles candidates.

   S'il n'y en a pas, le modèle de données devra retourner à la case analyse, car il y a quelque chose qui cloche.

2. Pour chacune des clés naturelles candidates identifiées, mise en place d'une contrainte d'unicité et d'un index.

3. Ajout d'une colonne de type entier auto-incrémenté / numéro de séquence.

   Déclaration de la clé primaire de la relation comme étant constituée de cette seule colonne. À ce titre, c'est cette colonne qui sera éventuellement référencée comme clé étrangère dans d'autres relations de la base de données.

   Cette colonne peut éventuellement être nommée `id`. Les colonnes y faisant références sous forme de clés étrangères dans d'autres tables seront alors nommées de la forme `xxx_id` où `xxx` décrit le rôle de la table référencée dans l'association qu'elle a avec la table dans laquelle figure la clé étrangère.

---

2. L'auteur de ces mots est plutôt d'avis de bannir les identifiants techniques des routes des API web, pour y faire plutôt figurer des identifiants naturels ayant du sens métier. L'URL des ressources adressées n'en sont ainsi que plus intelligibles. Pour ce qui est de propriétés id dans des structures JSON, il est plus partagé.

4. Ajout d’une colonne de type UUID.

   Cette colonne servira d’identifiant unique au delà de la base de données. Il permettra d’identifier les lignes et leurs correspondances lors d’opérations multibases.

   Cette colonne peut éventuellement être nommée `public_id`.