# Les Index

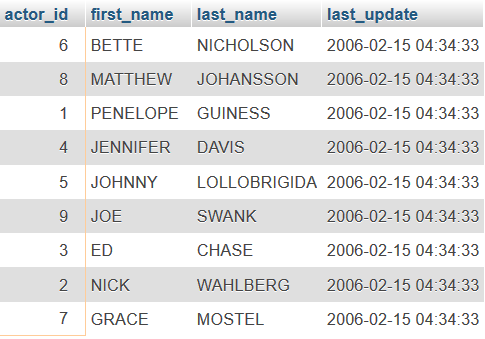
## Qu'est-ce que c'est un index et à quoi ça sert ?

Dans le domaine des BD, un index est un outil pour accélérer les requêtes. En général on les utilise pour accélérer les requêtes de sélection, mais aussi pour accélérer tout type de requête contenant plusieurs tableaux.

L'index lui-même est une **structure de données** (arbre binaire BTREE) mais pour comprendre son fonctionnement plus facilement on le représente avec une colonne de données.

Si on crée un index sur une colonne une structure contenant les données triées de cette colonne sera créée. On peut faire ceci en choisissant l’option INDEX dans phpMyAdmin (structure de table) ou en tapant un requête SQL de création (dans phpMyAdmin ou dans la console).

Considérons ce tableau :



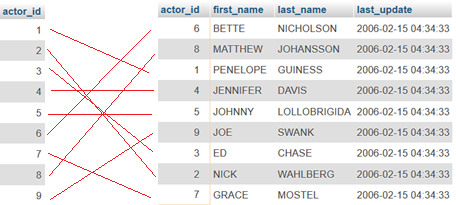
Si on **crée un index** **sur une certaine colonne** (ex : sur l’id de **l’actor**), **MySQL créera une structure contenant les données de cette colonne triées et un ensemble de "pointers".** **Chaque pointer pointe vers une donnée.**

**IMPORTANT**: pour cette explication on partira d’un cas où **le champ indexé n’a pas des valeurs déjà ordonnées**. C’est fictif, car quand on crée un autonumber en MySQL il sera consécutif et sans trous (ex : 1,2,3,...5 et pas 1,3,2 ou 1,4,5) sauf si on efface de lignes.

Pour garder l’explication simple, on utilise une colonne clé primaire (actor\_id) qui, de manière fictive, n’est pas trié à la base.

On peut appliquer la même explication pour de champs qui ne sont pas naturellement triés tels que un prix, une année ou nom (on peut crée des index pour de champs varchar aussi).

Ex: créer un index sur la colonne **actor\_id** provoquera la création d'une structure équivalente à celle-ci :



Ce système a deux **avantages** :

Si **on veut sélectionner toutes les lignes dont l'id est plus grand ou plus petit qu'une certaine valeur** (ex : les acteurs qui portent l'id < 3), on ne doit plus parcourir le tableau en cherchant toutes les lignes qui puissent correspondre (id<3). Au lieu de cela, nous prendrons directement les 2 premières lignes en utilisant l’index.

Si **on veut sélectionner une ligne dont on connaît l'id, on peut utiliser un algorithme de recherche de données sur l'index tel que la recherche dichotomique**. Cette méthode de recherche compare la valeur de l'id recherché avec l'id de la ligne au milieu du tableau. Si elles ne sont pas égales, la moitié des lignes dans laquelle la valeur ne se trouve pas sera supprimée et la recherche continuera dans l’autre moitié. On répète l’opération jusqu'à ce que la valeur soit trouvée.

Pour pouvoir comprendre cette idée voyons un exemple.

**Exemple :** on a 100 acteurs et on cherche celui dont l'id est le 28. On divise le total de lignes (100) en deux groupes.

On aura deux ensembles de lignes : les lignes ayant l’id de 1 à 50 et les lignes ayant l’id de 51 à 100.

Puis on regarde si l'id recherché (28) est plus grand ou plus petit que l’id ‘moitie’ (ici 50).

L’id est plus petit que l’id ‘moitie’, alors on sait qu’on doit chercher dans la moitie inférieure du tableau (ids de 1 à 50) . La moitie supérieure sera ignoré pour toujours.

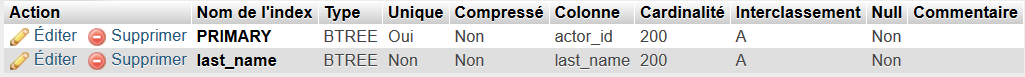
On répète toute la procédure sur la moitie inférieur (entre 1 et 50). On aura à nouveau deux tableaux (1-25 et 26-50). On compare et, cette fois, on prend la moitié supérieure (26-50). On jette à la poubelle la moitie inférieure.

On répète la procédure jusqu’au moment où on trouve la ligne 28. Cette procédure est beaucoup plus rapide qu’aller ligne par ligne en vérifiant si la ligne à l’id 28.

## Création de plusieurs indexes sur un même tableau

On pourrait se demander quelle est la raison de créer un index sur l'id d'un tableau... il aurait suffit tout simplement de trier le tableau par id! La raison pour laquelle nous ne le ferons pas est que **nous pouvons créer plusieurs index sur un même tableau**.

**Exemple** : on crée un index sur **l'id** et un autre sur le **last**\_**name**



MySQL créera une structure équivalente à celle-ci :

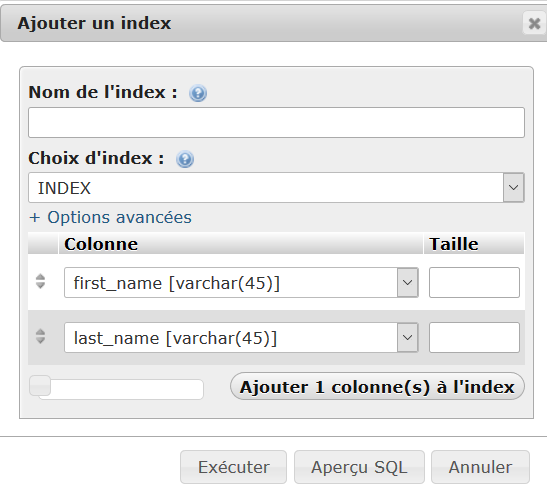


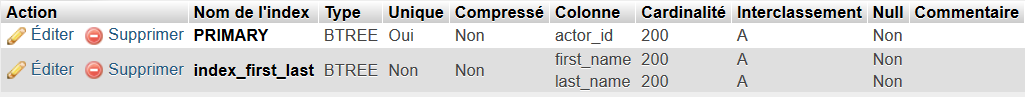
## Création des index qui affectent plusieurs colonnes.

On peut créer un index basé sur plusieurs colonnes. On utilise des index multi-colonne quand on fait beaucoup de recherches qui concernent ces colonnes.

**Exemple :** créons un index sur deux champs (first\_name et last\_name)







Dans ce cas, la structure de données sera équivalente à celle d’un tableau contenant deux colonnes (first\_name, last\_name). Les données seront tries d'abord par la première colonne, puis par la deuxième, etc…

## Désavantages d'utiliser des index

Les index ont deux **désavantages** :

* Sa création prend de la place en mémoire (comme l’index d’un livre demande de rajouter quelques pages en plus!)
* Les requêtes d’insertion et élimination (et dans certains cas celles de modification) seront ralenties car l’index doit être mis à jour à chaque modification de la table.

## Contraintes sur les indexes

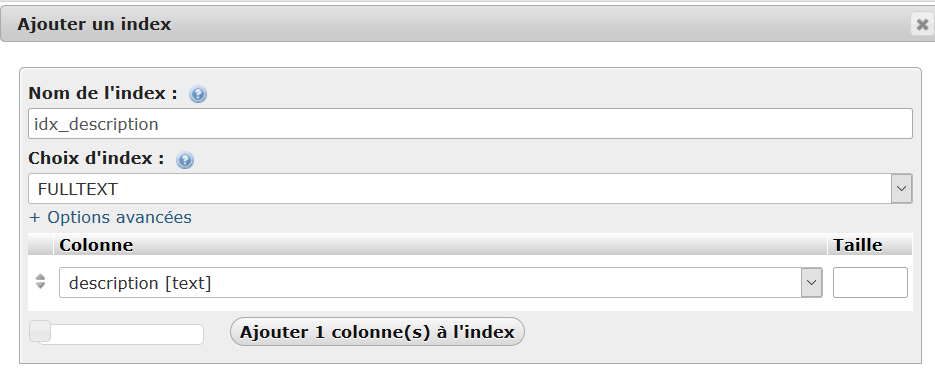
Lors de la création d'un index on peut établir plusieurs contraintes :

* **UNIQUE** : la valeur de la colonne ne pourra pas se répéter. C'est les cas des clés primaires ou de certains champs tels que "Login" !
* **NOT NULL** : une valeur devra être donné à cette colonne lors de l’insertion d'une ligne dans le tableau
* **PRIMARY :** la colonne sera clé primaire
* **FULLTEXT** : expliqué à continuation, nous permet d'optimiser les recherches à l’intérieur d'un champ. On l'utilise uniquement avec de colonnes CHAR, VARCHAR et TEXT

## Les index Fulltext

Un index FULLTEXT nous permet d'optimiser les recherches à l’intérieur d'un champ. On l'utilise uniquement avec de colonnes CHAR, VARCHAR et TEXT

**Exemple :** On veut accélérer les recherches d'une partie de la description d'un film. On crée un index sur la colonne **description** du type FULLTEXT



MySQL créera une structure de données en divisant le contenu de la colonne (ici **description**). La division se réalisera par mots. Un mot peut contenir de lettres et de chiffres, ainsi que des apostrophes. Un espace créera deux mots séparés, un tiré ("-") aussi

Pour réaliser une requête on utilisera les mots clés **MATCH** et **AGAINST**

SELECT \* FROM film WHERE **MATCH** (description) **AGAINST** ('Monkey')

**Exemple :** recherche qui trouve le mot 'Monkey' à l’intérieur de la description. Par défaut, le moteur cherche le **mot exact**. Si on avait cherché 'Mon' on n'aurait obtenu aucun résultat.

Il faut savoir que les recherches de ce type sont réservées aux colonnes qui possèdent un index **FULLTEXT**

Pour la création de l'index, **MySQL suit ces trois réglés** :

1. Les mots très fréquents dans la colonne (trouvés dans au moins la moitié des lignes) ne sont pas pris en compte.
2. Les mots très courts (moins de 4 lettres) ne sont pas pris en compte
3. Les mots très fréquents en anglais ne sont pas pris en compte