T. D. n° 5
Index en MySQL

Résumé

Ce document est le TD n° 5 du module **Base de données**. Il reprend rapidement des éléments du cours et propose une mise en pratique interactive des index.

1 Index

source :`Meme.xyz`

Reprenez votre base de donnée élevage. Cette base contient table *Animal* doit être composée *a minima* de six colonnes : *id*, *espece*, *sexe*, *date_ naissance*, *nom* et *commentaires*.

Les index sont des structures permettant de lister de façon ordonnée vos lignes dans une table. Par exemple si l'index est sur la colonne date de naissance, lorsque vous exécuté une recherche recherche sur cette colonne, les éléments seront affichés selon la date la plus récente à la plus ancienne. Les index permettent d'accélérer le temps de calcule associé à des requêtes qui utilisent des colonnes indexées comme critères de recherche. Si vous avez l'intention de faire beaucoup de recherches sur une colonne particulière, il vaut mieux l'indexer. Ils vous permettent également de fixer des contraintes sur les valeurs contenues par les colonnes. Cependant, ils prennent de la place en mémoire et contraignent l'utilisateur lorsqu'il souhaite modifier une table. Il existent trois type d'index particulier : `UNIQUE`, `FULLTEXT`, et `SPACIAL` (ce dernier ne sera pas vu ici cependant.

1.1 Index défini avec la création de la table

Pour définir un index on utilise `INDEX` ou `KEY` (la différence entre ces deux commandes sera vue plus tard, considérons uniquement `INDEX` pour l'instant).

```
CREATE TABLE nom_table (
    colonne TYPE_COLONNE,  --  index simple sur colonne
    INDEX nom_index (colonne)
);
```

Par défaut, MySQL défini un nom automatiquement pour votre index. Cependant définir le nom d'un index peut être utile si vous devez potentiellement le supprimer par la suite. Si vous souhaitez choisir vous même le nom à donner à votre index utilisez syntaxe suivante :

```
CREATE TABLE nom_table (
    colonne TYPE_COLONNE,
    INDEX nom_index (colonne)
);
```

Pour lister les index d'une table, vous pouvez utiliser la commande suivante :

```
SHOW INDEX FROM nom_table
```

À vous !

a) Créez la table *toto* avec une colonne *id* de type `TINYINT` qui sera également un index.

b) Quel est le nom qu'a automatiquement choisi MySQL pour cet index ?

c) Créez la table *toto2* avec une colonne *id* de type `SMALLINT` qui sera également un index. Le nom de cet index sera *monIndex*

d) Vérifiez que l'index *monIndex* a correctement été crée.

e) Supprimez la table *toto* et *toto2*

1.2 Définir un index après la création de la table

Pour définir un index sur une table déjà définie, deux commandes sont possibles : `ALTER TABLE` et `CREATE INDEX` Personnellement, je recommande l'utilisation d'`ALTER TABLE`, qui évite d'apprendre une commande supplémentaire. De plus, `CREATE INDEX` est plus limité qu'`ALTER TABLE` (suppression de l'index impossible par exemple).

```
ALTER TABLE nom_table
    ADD INDEX nom_index (colonne);

CREATE INDEX nom_index
ON nom_table (colonne);
```

Si vous choisissez de définir un index sur une ou plusieurs colonnes, il peut être parfois nécessaire de redéfinir le type de la colonne. Pour cela, la commande `PROCEDURE ANALYSE()` s'avère très utile.

Cette commande analyse pour vous vos tables et vous propose le type idéal pour vos données... si vous en avez bien sûr, ça ne peut pas vous aider lors d'une création de table. En revanche, elle permet « d'auditer » vos enregistrements actuels, d'en tirer quelques statistiques et propose le type le plus adapté :

```
SELECT nom_col FROM nom_table PROCEDURE ANALYSE(10,256)
```

`PROCEDURE ANALYSE` comprend deux paramètres optionnel

- max_elements (par défaut 256) qui est le nombre maximum de valeurs distinctes que `ANALYSE()` remarque par colonne. Ceci est utilisé par `ANALYSE()` pour vérifier si le type de données optimal devrait être de type `ENUM` ; s'il y a plus que des valeurs distinctes max_elements, alors `ENUM` n'est pas un type suggéré.
- max_memory (par défaut 8192) qui est la quantité maximale de mémoire qu'`ANALYSE()` devrait allouer par colonne tout en essayant de trouver toutes les valeurs distinctes.

À vous !

a) Ajoutez un index *ind_nom* sur la colonne *nom* votre table **Test_tuto**.

b) Proposez la commande similaire avec l'opérande `CREATE INDEX`

c) Peut-on créer 2 index sur une même colonne ?

d) Peut-on créer 2 index avec le même nom ?

e) Que propose `PROCEDURE ANALYSE()` comme typage des colonnes *id*, *espece*, *sexe*, *nom*, *date_naissance* et *commentaires*.

1.3 Index multiples

Vous pouvez également spécifier plusieurs colonnes pour définir un index (par exemple nom, prénom, date et lieu de naissance). On peut se servir de cet index même si on ne fait pas une recherche sur toutes les colonnes indexées. C'est ce qu'on appelle l'index par la gauche.
Reprenons par exemple un index défini sur nom, prénom, date et lieu de naissance. L'index triera par ordre les noms, puis les prénoms, puis la date de naissance et enfin le lieu. Si vous voulez faire une recherche sur les noms uniquement, l'ordre sera exactement le même que si on avait créé un index uniquement sur les noms.
De même si vous faite une recherche sur nom et prénom, l'ordre ne bouge toujours pas comparé à un index sur ces deux colonnes. En revanche, si on veut faire une recherche sur date de naissance uniquement ou prénom uniquement, ça ne marchera pas.
En effet, les dates de naissances sont ordonnés par les nom et les prénoms qui les précèdent dans les colonnes de gauches, avant d'être trié eux-même par date croissante. C'est pour cette raison que l'on dit « par la gauche », cela marche tant que l'on effectue la recherche sur toutes les colonnes qui sont à gauches. Dans le cas contraire, il faut créer un autre index.
Pour définir un index sur plusieurs colonnes, la syntaxe est la suivante :

```
CREATE TABLE nom_table (
    colonne_1 TYPE_COLONNE,
    colonne_2 TYPE_COLONNE,
    ...
    colonne_n TYPE_COLONNE,
    INDEX nom_index (colonne1,..,colonne_n)
);
```

À vous !

a) Créez la table *Animal_2* avec un index sur la date de naissance et un autre sur les 10 première lettre du nom.

2 Index `UNIQUE`

L'index `UNIQUE` vous permet d'imposer à votre table qu'une colonne (ou une combinaison de colonne) ne comprenne que des valeurs différentes pour chaque ligne. Par exemple un index `UNIQUE` sur la colonne *pseudo* imposera que tous les objets dans la table aient un pseudo différent. Le fait d'indexer cette colonne permet également d'avoir une réponse rapide lorsque vous interroger votre base via le pseudo d'un utilisateur.
Il est possible de spécifier votre colonne indexe unique via `UNIQUE` lors de la création de la table via les syntaxes suivante :

```
CREATE TABLE nom_table (
    colonne TYPE UNIQUE , -- index unique sur colonne1
);

CREATE TABLE nom_table (
    colonne TYPE,
    UNIQUE INDEX nom_index (colonne)
);
```

Personnellement je préfère la deuxième syntaxe.
Notez que `UNIQUE` peut être renseigné directement dans la description de la colonne, pas `INDEX`.

À vous !

a) Créez la table *toto* avec une colonne *id* de type entier et une colonne *name* de type caractère (20 max)

b) Après la création de votre table, créez un index unique sur la colonne *id*

c) Ajoutez un élément dont l'*id* est nul.

d) Que pouvez-vous conclure sur les contraintes qu'impose l'index unique.

e) Dans la table Animal créer un index unique sur les colonnes espèce et nom (un seul index, sur les deux colonnes).

3 Index FULLTEXT

La recherche plein texte (FULLTEXT), ou FTS, est une technique utilisée par les moteurs de recherche pour trouver des résultats dans une base de données. Vous pouvez l'utiliser pour optimiser les résultats de recherche sur des sites Web tels que les boutiques, les moteurs de recherche, les journaux, et plus encore.
Plus précisément, FTS récupère les documents qui ne correspondent pas parfaitement aux critères de recherche. Les documents sont des entités de base de données contenant des données textuelles. Cela signifie que lorsqu'un utilisateur recherche "chats et chiens", par exemple, une application soutenue par FTS est capable de retourner des résultats qui contiennent les mots séparément (seulement "chats" ou "chiens"), contiennent les mots dans un ordre différent ("chiens et chats"), ou contiennent des variantes des mots ("chat" ou "chien"). Cela donne aux applications l'avantage de deviner ce que l'utilisateur veut dire et d'obtenir des résultats plus pertinents plus rapidement.
Techniquement parlant, les systèmes de gestion de bases de données (SGBD) comme MySQL permettent généralement des recherches partielles de texte à l'aide des clauses LIKE. Toutefois, ces demandes ont tendance à être moins performantes dans le cas de grands ensembles de données. Ils se limitent également à faire correspondre exactement l'entrée de l'utilisateur, ce qui signifie qu'une requête peut ne produire aucun résultat même s'il y a des documents contenant des informations pertinentes. En utilisant FTS, vous pouvez construire un moteur de recherche texte plus puissant sans introduire de dépendances supplémentaires sur des outils plus avancés.
Pour mettre en pratique l'utilisation des index FULLTEXT exécuter les instructions SQL suivantes, qui servent à créer la table que nous utiliserons pour illustrer cette section. Nous sortons ici du contexte de l'élevage d'animaux.

```
CREATE TABLE Livre (
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    auteur VARCHAR(50),
    titre VARCHAR(200)
) ENGINE = MyISAM;

INSERT INTO Livre (auteur, titre)
VALUES ('Daniel Pennac', 'Au bonheur des ogres'),
('Daniel Pennac', 'La Fée Carabine'),
('Daniel Pennac', 'Comme un roman'),
('Daniel Pennac', 'La Petite marchande de prose'),
```

```
('Jacqueline Harpman', 'Le Bonheur est dans le crime'),

('Jacqueline Harpman', 'La Dormition des amants'),

('Jacqueline Harpman', 'La Plage d''Ostende'),

('Jacqueline Harpman', 'Histoire de Jenny'),

('Terry Pratchett', 'Les Petits Dieux'),

('Terry Pratchett', 'Le Cinquième éléphant'),

('Terry Pratchett', 'La Vérité'),

('Terry Pratchett', 'Le Dernier héros'),

('Terry Goodkind', 'Le Temple des vents'),

('Jules Verne', 'De la Terre à la Lune'),

('Jules Verne', 'Voyage au centre de la Terre'),

('Henri-Pierre Roché', 'Jules et Jim');


CREATE FULLTEXT INDEX ind_full_titre

ON Livre (titre);


CREATE FULLTEXT INDEX ind_full_aut

ON Livre (auteur);


CREATE FULLTEXT INDEX ind_full_titre_aut

ON Livre (titre, auteur);
```

Il existe trois types de recherche `FULLTEXT` : la recherche naturelle, la recherche avec booléen, et enfin la recherche avec extension de requête.

3.1 Recherche naturelle

Lorsque l'on fait une recherche naturelle, il suffit qu'un seul mot de la chaîne de caractères recherchée se retrouve dans une ligne pour que celle-ci apparaisse dans les résultats. Attention, cependant, au fait que le mot exact doit se retrouver dans la valeur des colonnes de l'index `FULLTEXT` examiné.
Voici la syntaxe utilisée pour faire une recherche `FULLTEXT` :

```
SELECT *                                -- Vous mettez évidemment
    les colonnes que vous voulez.
```

```
FROM nom_table

WHERE MATCH(colonne1[, colonne2, ...])   -- La (ou les) colonne(s)
   dans laquelle (ou lesquelles) on veut faire la recherche (index
   FULLTEXT correspondant nécessaire).

AGAINST ('chaine recherchée');            -- La chaine de caractè
   res recherchée, entre guillemets bien sûr.
```

Si l'on veut préciser que l'on fait une recherche naturelle, on peut ajouter `IN NATURAL LANGUAGE MODE`. Ce n'est cependant pas obligatoire, puisque la recherche naturelle est le mode de recherche par défaut.

```
SELECT *

FROM nom_table

WHERE MATCH(colonne1[, colonne2, ...])

AGAINST ('cha ne recherchée' IN NATURAL LANGUAGE MODE);
```

À vous !

a) recherchez "Terry" dans la colonne auteur de la table Livre .

b) recherchez d'abord "Petite", puis "Petit" dans la colonne titre.

c) recherchez "Henri" dans la colonne auteur.

d) recherchez "Jules", puis "Jules Verne" dans les colonnes titre et auteur.

e) l'ordre des colonnes dans `MATCH` a-t-il de l'importance ?

f) quelle est la différence entre `MATCH` et `MATCH AGAINST`

3.2 La pertinence

La pertinence est une valeur supérieure ou égale à 0 qui qualifie le résultat d'une recherche `FULLTEXT` sur une ligne. Si la ligne ne correspond pas du tout à la recherche, sa pertinence sera de 0. Si par contre elle correspond à la recherche, sa pertinence sera supérieure à 0. Ensuite, plus la ligne correspond bien à la recherche (nombre de mots trouvés, par exemple), plus la pertinence sera élevée. Vous pouvez voir la pertinence attribuée à une ligne en mettant l'expression `MATCH... AGAINST` dans le `SELECT`.

```
SELECT *, MATCH(col1, col2) AGAINST ('chaine de charactères')

FROM Table;
```

À vous !

a) Affichez la pertinence de la recherche 'Jules Verne Lune' sur les colonnes *titre* et *auteur*.

b) Comparez avec le résultat de la recherche 'Jules Verne Lune' dans les colonnes titre et auteur.

3.3 Recherche avec booléens

La recherche avec booléens possède les caractéristiques suivantes :

— elle ne tient pas compte de la règle des 50 % qui veut qu'un mot présent dans 50% des lignes au moins soit ignoré ;
— elle peut se faire sur une ou des colonnes sur lesquelles aucun index `FULLTEXT` n'est défini (ce sera cependant beaucoup plus lent que si un index est présent) ;
— les résultats ne seront pas triés par pertinence par défaut.

Pour faire une recherche avec booléens, il suffit d'ajouter `IN BOOLEAN MODE` après la chaîne recherchée.

```
SELECT *

FROM nom_table

WHERE MATCH(colonne)

AGAINST('cha ne recherchée' IN BOOLEAN MODE);  -- IN BOOLEAN MODE
   à l'intérieur des parenthèses !
```

La recherche avec booléens permet d'être à la fois plus précis et plus approximatif dans ses recherches.

— Plus précis, car on peut exiger que certains mots se trouvent dans la ligne ou soient absents de la ligne pour la sélectionner. On peut même exiger la présence de groupes de mots, plutôt que de rechercher chaque mot séparément.
— Plus approximatif, car on peut utiliser un astérisque * en fin de mot, pour préciser que le mot peut finir de n'importe quelle manière.

Pour exiger la présence ou l'absence de certains mots, on utilise les caractères + et -. Un mot précédé par + devra être présent dans la ligne et inversement, précédé par - il ne pourra pas être présent.

```
SELECT *
FROM nom_table
WHERE MATCH(colonne)
AGAINST ('+mot1 -mot2' IN BOOLEAN MODE);
```

À vous !

a) Recherche sur le titre, qui doit contenir "bonheur", mais ne peut pas contenir "ogres".

b) recherche sur titre, qui doit contenir tout le groupe de mots "Terre à la Lune"

c) recherche sur titre, qui doit contenir tout le groupe de mots "Lune à la Terre"

d) recherche sur titre, qui doit contenir tout le groupe de mots "Terre la Lune"

e) Comparez les trois requêtes précédentes.

f) recherche sur titre, sur tous les mots commençant par "petit".

g) recherche sur titre et auteur, de tous les mots commençant par "d".

h) recherche sur titre, qui doit contenir un mot commençant par "petit", mais ne peut pas contenir le mot "prose".

3.4 Recherche avec extension de requête

Le dernier type de recherche est un peu particulier. En effet, la recherche avec extension de requête se déroule en deux étapes.

— Une simple recherche naturelle est effectuée.

— Les résultats de cette recherche sont utilisés pour faire une seconde recherche naturelle.

La syntaxe à utiliser est

```
SELECT *

FROM nom_table

WHERE MATCH(colonne1, colonne2)

AGAINST ('mot' WITH QUERY EXPANSION);
```

À vous !

a) Réaliser une recherche naturelle effectuée avec la chaîne "Daniel" sur les colonnes auteur et titre.

b) Refaire la requête précédente en utilisant l'extension de requête `WITH QUERY EXPANSION`.

c) Comparez le résultats.