

Bases de données, ENS Cachan & Ulm

TP n° 4 – Plans d'exécution

Pierre Senellart (pierre@senellart.com)

28 mars 2008

Le but de ce TP est de comprendre la manière dont MySQL exécute les requêtes, et de comment les optimiser en créant des index.

I Prérequis

Ce TP est court ; avant de l'aborder, assurez-vous d'avoir terminé :

- les exercices 1, 2 et 4 du TP n° 3 ;
- les deux premières questions du complément au TP n° 3.

2 Explications de plans d'exécution

Afin d'optimiser une requête, de savoir quels index créer ou de comprendre pourquoi une requête simple prend beaucoup de temps, on peut utiliser l'instruction MySQL EXPLAIN devant un ordre SELECT. Le SELECT n'est alors pas exécuté, mais un plan d'exécution en est donné.

On rappelle que des index (UNIQUE ou non) peuvent être créés sur chacune des colonnes d'une table. Un index UNIQUE est également automatiquement associé à une clef primaire est automatiquement. La contrepartie d'un index est une augmentation de la taille occupée par la table et du coût des mises à jour.

1. Afficher le plan d'exécution de la requête 1 de l'exercice 2 du TP n° 3. On rappelle que, dans l'outil de ligne de commande `mysql`, un point-virgule peut être remplacé par un `\G` pour demander un affichage alternatif.

Chaque ligne du plan d'exécution correspond à un parcours d'une table, dans l'ordre où les accès sont effectués. `table` indique le nom de la table dont il s'agit, `type` le type d'accès (voir ci-dessous), `key` l'index utilisé, le cas échéant, `rows` une estimation du nombre de lignes de la table à parcourir, et `Extra` des informations complémentaires (voir ci-dessous). Une estimation grossière du temps d'exécution d'une requête est ainsi donnée par le produit de la valeur `rows` de chaque étape.

Les principaux types d'accès sont les suivants :

ALL : parcours linéaire de la table ;

const, eq_ref : utilisation d'un index UNIQUE pour accéder à une unique ligne (**const** est indiqué quand la valeur indexée est une constante à la compilation de la requête) ;

ref, ref_or_null : utilisation d'un index pour accéder à l'ensemble des lignes correspondant à une valeur indexée donnée ;

range : utilisation d'un index pour accéder à l'ensemble des lignes dont la valeur indexée est dans un intervalle donné ;

index : parcours linéaire de l'index.

Les principales informations complémentaires sont :

Using where : une sélection supplémentaire est effectuée sur les lignes parcourues ;

Using filesort : un tri additionnel de la table est nécessaire ;

Using temporary : le résultat doit être mis dans une table temporaire, par exemple pour faire des regroupements ;

Using index : il n'est pas nécessaire d'accéder à la table, la réponse à la requête peut être déterminée par l'index lui-même ;

Select tables optimized away : il n'est même pas nécessaire d'accéder à une entrée d'index, la réponse à la requête peut être déterminée par les méta-informations de l'index (taille, valeurs minimales et maximales), etc.

2. Que peut-on déduire de la structure physique des index, s'il y a des accès de type **range** ?
3. En créant ou en supprimant des index sur la table `Unicode`, imaginer des requêtes dont le plan d'exécution comporte des accès de tous les types cités plus haut.
4. Idem avec les informations complémentaires.

5. Comparer le plan d'exécution d'une requête affichant tous les noms de caractères Unicode ASCII (c'est-à-dire, de code Unicode inférieur ou égal à 127) et d'une requête affichant tous les noms de caractères non ASCII. Que peut-on en déduire ?
6. Dans quels cas une requête avec ORDER BY pourra-t-elle être exécutée sans tri additionnel ?
7. Considérer la requête suivante :

```
SELECT u1.name, u2.name, u3.name
FROM Unicode u1, Unicode u2, Unicode u3
WHERE u2.lowercase=u1.codepoint AND u3.lowercase=u1.codepoint AND u2.codepoint<>u3.codepoint;
```

Sans faire l'expérience, répondre aux questions suivantes :

- (a) Quelle est sa sémantique ?
- (b) Quel est son plan d'exécution ?
- (c) Y a-t-il moyen de l'optimiser ?

Vérifier.