SQL优化

优化基本原则

> 原则一：尽量避免全表扫描。 原则二：通过索引优化

> 1，**最左前**缀**匹配原**则：mysql会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配。所以要尽量把"="条件放在前面，把这些条件放在最后。复合索引也是同理。 2，尽量选择 **区分度高且是**业务**常用的列作**为**索引**。在构建复合索引时，也应注意顺序。 3，当取出的数据超过 **全表数据的20%**时，不会使用索引。【待求证】 4，**避免在 like** 查询**中将 %放在开**头：1）不使用索引：like '%L%'；2）使用索引：like 'L% 5，尽量将or 转换为 union all：这是何原理？可能和mysql实现有关，union all会并发？可能不是。1）不使用索引：select * from user where name='a' or age='20'；2）使用索引：select * from user where name='a' union all select * from user where age='20'。or可能导致不会使用索引，导致全表扫描。对于or+没有索引的age这种情况，假设它走了userId的索引，但是走到age查询条件时，它还得全表扫描，也就是需要三步过程：全表扫描+索引扫描+合并 如果它一开始就走全表扫描，直接一遍扫描就完事。mysql是有优化器的，处于效率与成本考虑，遇到or条件，索引可能失效，看起来也合情合理。 6，**字段加函数不会使用索引，字段加运算符不会使用索引，索引列不能参与**计**算** 7，使用组合索引时，必须要包括第一个列,**索引(A, B)相当于**创**建了索引(A)和索引(A, B)** 8，尽量避免使用is null或is not null 9，不等于（!=，<>）不会使用索引 10，尽量使用表连接（join）代替子查询 select * from t1 where a in (select b from t2) 11，性能方面，表连接 > (not) exists > (not) in 12，避免使用HAVING子句, **HAVING 只会在**检**索出所有**记录**之后才**对结**果集**进**行**过滤. 这个处理需要排序,总计等操作. 如果能通过WHERE子句限制记录的数目,那就能减少这方面的开销 13，**类型要一致** 14，尽量减少 select *，避免全表扫描 15，在适当的时候，使用覆盖索引：通常在使用索引检索数据之后，需要访问磁盘上数据表文件读取所需要的列，这种操作成为 **回表**。若索引中包含查询的所有列，则不需要回表操作，直接从索引文件中读取数据即可，这种索引成为 **覆盖索引** 16，=和in可以乱序，比如a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql的查询优化器会帮你优化成索引可以识别的形式 17，尽量的扩展索引，不要新建索引。比如表中已经有a的索引，现在要加(a,b)的索引，那么只需要修改原来的索引即可 18，明知只有一条查询结果，那请使用 "LIMIT 1"，避免全表扫描 19，索引数量，一般不要超过5个 20，尽可能使用varchar/nvarchar 代替 char/nchar。首先变长字段存储空间小，可以节省存储空间

优化工具与方法

> 1，explain 2，开启慢查询日志 3，show processlist； 4，日志分析工具 MySQLdumpslow 5，第三方工具：美团技术团队的 SQLAdvisor，给出索引优化建议的工具

优化案例