1. 索引优化
2. 索引的介绍

索引从字面上就可以看出类似一个指向数据库中数据的指针，没有索引的时候只能通过搜索数据库中的数据来查找；通过索引，只需要查找索引中有无关联的数据就可以返回查找结果，大大提高了查找的性能。

1. 添加索引

在创建数据表的时候，可以通过KEY关键字添加索引：

CREATE TABLE `tb\_table` (

`id` int(11) NOT NULL AUTO\_INCREMENT COMMENT '主键',

`name` varchar(20) DEFAULT NULL COMMENT '姓名',

`number` int(11) DEFAULT NULL COMMENT '编号',

PRIMARY KEY (`id`),

KEY `number` (`number`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

添加number项的索引后，当我们通过number的值来搜索数据库时，就会使用到索引。

1. 索引的类型
2. 主键索引PRIMARY KEY

特殊的唯一索引，不允许有空值，一般是在建表的时候同时创建主键索引。

PRIMARY KEY (“id”)

1. 唯一索引UNIQUE

唯一索引列的值必须唯一，但允许有空值，可以在建表的时候指定，也可以修改表结构。

UNIQUE KEY “num” (“number”) USING BTREE

1. 普通索引INDEX

基本的索引，没有限制，可在建表时指定，也可以修改表结构、

KEY “num” (“number”) USING BTREE

1. 组合索引INDEX

索引分单列索引和组合索引。单列索引即一个索引仅包含单个列，一个表可以有多个单列索引；组合索引，即一个索引中包含多个列，组合索引遵循着左前缀匹配的原则，只有使用到左边的列，索引才会被使用，如果单独使用右边的列而不使用左边的列，那么索引是无法使用的。如下所示，只有查询了number列，name列才能够使用。

KEY “num” (“number”, “name”) USING BTREE

1. 全文索引FULLTEXT

全文索引（全文检索）是目前搜索引擎使用的一种技术，能够利用分词技术分析出文本中关键字的频率和重要性，筛选出想要的搜索结果。

1. 索引的存储结构
2. BTree索引

MYSQL普遍使用B+ Tree作为索引方案，即BTree。BTree索引以B+树作为存储结构，能够加快数据的查询速度，并且适合进行范围查找。通常用于全值匹配的查询，即一个值要完整地匹配上；同样在联合索引时遵循最左前缀匹配原则，还能够匹配列的前缀进行模糊查询，最重要的是可以进行范围查询，对一个范围的值进行查找，而不只是搜索一个相等的值。同时，它还是一个只访问索引的查询。

1. 哈希索引

哈希索引在MYSQL中使用的并不多，目前主要是Memory存储引擎使用，是默认的索引类型。所谓的哈希索引，就是通过一定的hash算法，将需要索引的键值进行Hash运算，然后将得到的Hash值存入一个Hash表中，每次需要检索时，对于检索条件进行同样的Hash运算，并与Hash表比较得出结果。

Hash索引只能进行单值相等查询，不能进行范围查询；

无法对数据进行排序，因为Hash索引中存储的是Hash运算的结果值，与原有的值排序无关；

不能利用部分索引键查询，对于联合索引，Hash索引是将联合索引中的所有列作为键值计算Hash值的，因此无法使用其中的部分索引键查询；

Hash索引无法避免表扫描，因为Hash运算可能会对不同的键值产生相同的Hash值，需要通过全表扫描的方式来搜素出所有Hash值相等的键值，进行排除；

Hash索引遇到大量Hash值相等的情况性能不一定就比BTree好。

1. Full-text全文索引

全文索引，在MYSQL中仅有MyISAM和InnoDB存储引擎支持。对于文本对象，如果使用普通的索引，只能匹配前几个字符或者通过LIKE %word%来匹配中间的字符，这样非常麻烦且低效。全文索引会为文本生成一份单词的清单，在索引时根据这个单词的清单来索引。

对于较大的数据集，将数据添加到一个没有全文索引的表，然后再添加全文索引的速度比将数据直接添加到一个已有全文索引的表更快；

生成全文索引会消耗大量时间和空间；

5.6版本前全文索引只能使用MyISAM存储引擎，之后添加了InnoDB存储引擎；

5.7版本后使用ngram插件开始支持中文；

检索的字符串长度至少4个字节，如果太短无法得到预期的结果。

1. 索引的使用
2. 索引的缺点

索引会增加写操作的成本，如空间成本；

索引过多，则生成的查询计划也更多，查询优化器的查询时间也会更多；

只有当创建索引的优势大于消耗时，才是最优的选择。

1. 使用索引的场景

主键自动建立唯一索引；

常作为查询条件的列要建立索引，如WHERE和ORDER BY语句中的列；

作为排序的列要建立索引；

查询中与其他表关联的字段，外键关系需要建立索引；

高并发条件下需要建立组合索引；

1. 不使用索引的场景

列中有大量重复的值不单独建立索引，因为没有意义；

表中记录过少不要建立索引，浪费资源；

不会用于查询的类不要建立索引；

1. 存储优化

通过show engines 命令可以查询出MYSQL支持的存储引擎。这里主要讨论Innobdn, MyISAM 存储引擎。

1. InnoDB存储引擎
2. 特点
3. InnoDB存储引擎提供了具有提交，回滚和崩溃恢复能力的事务安全，也正是因为事务，写操作的处理效率更差且会占用更多空间保存数据和索引；
4. 提供了对数据库事务ACID（原子性，一致性，隔离性，持久性）的支持，实现了SQL标准的四种隔离级别；
5. 内存中建立缓冲池，用于缓冲数据和索引；
6. 使用InnoDB引擎的表不会保存表的具体行数，所以需要扫描整个表才能够计算行数；
7. 使用行锁，粒度更小，写操作不会锁定整个表，高并发时，效率更高；
8. 清空数据量大的表时，非常缓慢，因为InnoDB必须处理表的每一行，根据InnoDB的事务设计规则，首先需要将“删除动作”写入“事务日志”，然后写入实际的表，所以清空大表时，直接drop table然后重建。。
9. 使用场景

频繁的update/insert操作，处理高并发的写请求；

只有InnoDB支持事务，可通过日志+事务回滚恢复；

支持外键约束，列属性AUTO\_INCREMENT支持；

1. MyISAM存储引擎
2. 特点
3. 不支持事务，不支持外键；
4. 每个MyISAM在存储时分成三个文件，扩展名分别为

frm ：存储表定义，结构

MYD(MYData): 存储数据

MYI（MYIndex）: 存储索引

1. 不同MyISAM表的索引文件和数据文件可以放到不同的路径中；
2. MyISAM类型的表提供修复的工具；
3. 5.6以前，只有MyISAM支持全文索引；
4. 使用场景

插入不频繁，查询频繁的场景，小型应用考虑使用MyISAM，效率更高。

1. MyISAM和InnoDB的区别
2. MyISAM是非事务安全的，InnoDB是事务安全的；
3. MyISAM的锁粒度是表级别的，InnoDB是行级别的；
4. MyISAM不支持外键，InnoDB支持；
5. MyISAM简单，效率上优于InnoDB，小型应用选择MyISAM;
6. InnoDB表比MyISAM表更加安全；
7. 存储优化

在存储数据时，可以通过以下的设置来优化性能。

1. 禁用索引

对于使用索引的表，插入记录时，Mysql会对插入的数据建立索引，如果插入大量的数据，同步建立索引会降低插入速度，我们可以在批量插入数据之前禁用索引，在数据插入完成后再开启索引，这样建立索引的操作就只需要一次即可。

禁用索引的语句：

ALTER TABLE table\_name DISABLE KEYS

开启索引的语句：

ALTER TABLE table\_name ENABLE KEYS

InnoDB引擎需要这样操作，而MyISAM引擎是直接对空表批量插入数据后再建立索引，因此不需要这样做。

1. 禁用唯一性检查

唯一性校验会降低插入数据的速度，同样的可禁用在完成插入后开启：

禁用唯一性检查的语句：

SET UNIQUE\_CHECKS = 0

开启唯一性检查的语句：

SET UNIQUE\_CHECKS = 1

1. 禁用外键检查

禁用：

SET foreign\_key\_checks = 0

开启：

SET foreign\_key\_checks = 1

1. 禁止自动提交

禁止事务的自动提交操作，在数据完成之后统一进行事务的提交。

禁用：

SET autocommit =0

开启：

SET autocommit =1