关于 mysql 相关知识

数据表引擎

1. innodb 引擎

默认事务型引擎，最重要最广泛的存储引擎，性能非常优秀

数据存储在共享表空间，可通过配置分开

对主键查询的性能高于其他类型的存储引擎

内部做了很多优化，从磁盘读取数据时自动在内存构建 hash 索引，插入数据时自动构建插入缓冲区

通过一些机制和工具支持真正的热备份

支持崩溃后的安全恢复

支持行级锁

支持外键

1. MyISAM 引擎

5.1 版本前是默认引擎

拥有全文索引、压缩、空间函数

不支持事务和行级锁，不支持奔溃后安全恢复

表存储在两个文件，MYD 和 MYI

设计简单，某些场景下性能很好

1. 其他引擎：

Archive、Blackhole、CSV、Memory

MySQL 锁机制

当多个查询同一时刻进行数据修改时，会产生并发控制的问题

1. 共享锁（读锁）
2. 排他锁（写锁）

锁粒度

1. 表锁

系统性能开销最小，会锁定整张表，myisam 使用表锁

1. 行锁

最大程度的支持并发处理，但是也带来了最大的锁开销，innodb 实现行级锁

char 与 varchar

1. char

char 是定长的，根据定义的字符串长度分配足量空间 **char** 会根据需要采用空格进行填充以方便比较 **char** 适合存储很短的字符串，或者所有值都接近同一个长度 **char** 长度超过设定的长度，会被截断

1. varchar

varchar 用于存储可变长字符串，比定长类型更加节省空间

varchar 使用 1 或 2 个额外字节记录字符串的长度，列长度小于 255 字节用 1 个字节表示，否则用 2 个

varchar 长度超过设定的长度，会被截断

1. 比较

对于经常变更的数据，**char** 比 varchar 更好，**char** 不容易产生碎片

对于非常短的列，**char** 比 varchar 在存储空间上更有效率

只分配真正需要的空间，更长的列会消耗更多的内存

索引

1. 大大减少服务器需要扫描的数据量
2. 帮助服务器避免排序和临时表
3. 将随机 I/O 变顺序 I/O

4. 大大提高查询速度，降低写的速度，占用磁盘空间

索引类型

1. 普通索引
2. 主键索引
3. 唯一索引
4. 组合索引
5. 外键索引
6. 全文索引

索引创建原则

1. 最适合索引的列是出现在 where 子句的列，或连接子句中的列，而不是出现在 select 的关键字后的列
2. 索引列的基数越大，索引效果越好
3. 对字符串进行索引，应指定一个前缀长度，可以节省大量的索引空间
4. 根据情况创建复合索引，复合索引可以提高查询效率
5. 避免创建过多索引，索引会额外占用磁盘空间，减低写操作效率
6. 主键尽可能选择较短的数据类型，可以有效减少索引的磁盘占用，提高效率

索引的注意事项

1. 复合索引遵循前缀原则
2. like 查询，%不能在前，可以使用全文索引
3. column is null 可以使用索引
4. 如果 MySQL 估计使用索引比全表扫描更慢，会放弃使用索引

mysql 优化

查询速度慢的原因

1. 打开慢查询日志，通过 pt-query-dugest 分析
2. show profile，通过 set profiling=1;开启，服务器上执行的所有语句消耗时间都会记录到临时表。show profile for query QUERY_ID 查询指定查询
3. show status，查询一些计数器，猜出哪些代价高或消耗时间多
4. show processlist，查询线程状态进行分析
5. explain，分析单个 SQL 语句查询

优化查询过程中的数据访问

1. 访问数据太多导致性能下降
2. 确定应用程序是否检索大量超过需要的数据，可能是太多列或者行
3. 确定 mysql 是否分析大量不必要的数据行
4. 查询不需要的记录，使用 limit 限制
5. 夺标关联返回全部列指定 A.id,A.name
6. 总数取出全部列，select * 会让优化器无法完成所有覆盖扫码的优化
7. 重复查询相同的数据，可以缓存数据
8. 改变数据库和表的结构，修改数据表范式
9. 重写 SQL 语句，让优化器可以更优的执行

优化长难得查询语句

1. MySQL 内部每秒能扫描内存中上百万行数据，相比之下，响应数据给客户端就要慢得多
2. 使用尽可能少的查询是好的，但是有时将一个大的查询分解为多个小的查询是很有必要的

3. 分解关联查询，将一个关联查询分解为多个 sql 来执行，让缓存效率更高，执行单个查询可以减少锁的竞争，在应用层做关联可以更容易对数据库进行拆分，查询效率会有大幅提升，较少冗余记录的查询

优化特定类型的查询语句

1. 优化 count()查询，count(*)会忽略所有列，直接统计所有列数，因此不要用 count(列名)
2. 优化关联查询，确定 ON 或者 USING 子句的列上有索引；确保 GROUP BY 和 ORDER BY 中只有一个表的列，这样 MySQL 才有可能使用索引
3. 优化子查询 建议使用关联查询替代
4. 优化 GROUP BY 和 DISTINCT，建立索引进行优化
5. 优化 LIMIT 分页，可以通过记录上次查询的最大 ID，如果根据 id 排序时，下次查询根据该 ID 来查询（如：ID > maxID）
6. 优化 UNION 查询，UNION ALL 性能比 UNION 高

MySQL 提升（高可扩展和高可用）

分区表

工作原理

对用户而言，分区表是一个独立的逻辑表，但是底层 MySQL 将其分成了多个物理子表，对于用户来说是透明的，每一个分区表都会使用一个独立的表文件。

创建表的时候使用 partition by 子句定义每个分区存放的数据，执行查询时，优化器会根据分区定义过滤那些没有我们需要数据的分区，这样查询只需要查询所需数据在的分区即可

分区的主要目的是将数据按照一个较粗的粒度分在不同的表中，这样可以将相关的数据存放在一起，而且如果想一次性删除整个分区的数据也很方便

适用场景

1. 表非常大，无法全部存在内容，或者只有表的最后有热点数据，其他都是历史数据
2. 分区表的数据更易维护，可以对独立的分区进行独立操作
3. 分区表的数据可以分布在不同机器上，从而高效使用资源
4. 可以使用分区表来避免某些特殊瓶颈
5. 可以备份和恢复独立分区

限制

1. 一个表最多只能有 1024 个分区
2. 5.1 版本中，分区表表达式必须是整数，5.5 可以使用列分区
3. 分区字段中如果有主见和唯一索引列，那么主键和唯一列都必须包含进来
4. 分区表中无法使用外键约束
5. 需要对现有表的结构进行改变
6. 所有分区都必须使用相同的存储引擎
7. 分区函数中可以使用的函数和表达式会有一些限制
8. 某些存储引擎不支持分区
9. 对于 MyISAM 的分区表，不能使用 load index into cache
10. 对于 MyISAM 表，使用分区表时需要打开更多的文件描述符

分库分表

工作原理：

通过一些 HASH 算法或者工具实现将一张数据表垂直或者水平物理切分

适用场景

1. 单表记录条数达到百万到千万级别时
2. 解决表锁的问题

分别方式

1. 水平切分：表很大，分割后可以减低在查询时需要读的数据和索引的页数，同时也减低了索引的层数，提高查询速度

使用场景:1. 表中数据本身就有独立性，例如表中分别记录各个地区的数据或者不同时期的数据，特别是有些数据常用，有些不常用 2. 需要把数据存放在多个介质

缺点：1. 给应用增加复杂度，通常查询时需要多个表名，查询所有数据都需要 UNION 操作 2. 在许多数据库应用中，这种复杂性会超过他带来的优点，查询时会增加读一个索引层的磁盘次数

1. 垂直分表：把主键和一些列放在一个表，然后把主键和另外的列放在另一张表中

使用场景：1. 如果一个表中某些列常用，而另外一些列不常用 2. 可以使数据行变小，一个数据页能存储更多数据，查询时减少 I/O 次数

缺点：1. 管理冗余列，查询所有数据需要 JOIN 操作 2. 有些分表的策略基于应用层的逻辑算法，一旦逻辑算法改变，整个分表逻辑都会改变，扩展性较差 3. 对于应用层来说，逻辑算法无疑增加开发成本

主从复制

工作原理

1. 在主库上把数据更改记录到二进制日志
2. 从库将主库的日志复制到自己的中继日志
3. 从库读取中继日志中的事件，将其重放到从库数据中

解决问题

1. 数据分布：随意停止或开始复制，并在不同地理位置分布数据备份
2. 负载均衡：减低单个服务器压力
3. 高可用和故障切换：帮助应用程序避免单点失败
4. 升级测试：可以使用更高版本的 MySQL 作为从库

MySQL 安全

安全操作

1. 使用预处理语句防 SQL 这几日
2. 写入数据库的数据要进行特殊字符转移

3. 查询错误信息不要返回给用户，将错误记录到日志

安全设置

1. 定期做数据备份
2. 不给查询用户 root 权限，合理分配权限
3. 关闭远程访问数据库权限
4. 修改 root 口令，不用默认口令，使用较复杂的口令
5. 删除多余的用户
6. 改变 root 用户的名称
7. 限制一般用户浏览其他库
8. 限制用户对数据文件的访问权限