聚簇索引（主键索引，唯一性索引）：

该索引如果是主键自增，存储时索引叶大小一致，对应的存储的叶子节点也是连续的，每页逻辑大小为16k；这样插入就会变的高效，读取I/O就会是连续的；性能会很高

倘若不是自增或者说不是连续的；这样在插入的时候就会把原本存储满的叶子节点的空间进行分页，导致页的分裂；从而导致查询时不是连续I/O操作，会出现多次；因此可能会使用命令optimize table进行表存储的重新排列，存储做调整

因此二级索引可能比想象的更大，因为在二级索引的叶子节点上包含了引用行的主键列值，二级索引访问需要两次索引查找，而不是一次（二级索引保存的不是指向物理行的位置，而是行的主键值）

总结：主键的值是顺序的，所以innodb把每一条记录都存储在一条记录的后面，当达到页的最大填充因子（15/16），下一条记录就会记录在下一页中；一旦数据按照这种方式加载，主键页就会近似于被顺序的记录填满；这就是最好的结果

非递增的主键插入数据会导致页分裂，因为新的记录很可能插入到之前记录的中间，导致需要强制移动之前的记录，如果新记录要插入的位置所在的页被写满且已经被刷新到磁盘上，会导致重新从磁盘读取该页，innodb引擎表中非递增主键的一些缺点：

1. 可能导致大量的随机I/O
2. 频繁的页分裂操作
3. 不规则的填充导致产生数据碎片

问题：为什么varchar 在存储索引时会是最长的？

因为varchar字段的存储是变长的，因此索引页表比较大，从而导致页表会很多毕竟一个页表不能超过4k;因此会增加树的高度，导致同样大小的数据，用固定长度的检索出的数据会更快。

Btree索引，按照顺序存储数据，所以mysql可以用来做order by 与 group by操作，因为数据是有序的，所以btree也就会将相关的列值存放在一起，因为索引中存储了实际的值，所以某些查询就只使用索引就能够完成全部查询结果。

1. 索引大大减少了服务器需要扫描的数量（特别是全表扫描）
2. 索引可以帮助服务器避免排序和临时表（当然排序的列是在索引列中，不然最终还是会建立临时表以及重排序结果集）using filesort ,temprory
3. 索引可以尽量将随机I/O变为顺序I/O

评价一个索引是否适合某个查询的三要素，索引将相关的记录放到一起，索引中的数据顺序和查询中的排序一致，索引中的列包含了查询中需要的全部列。(三星索引)

hash索引倘若出现hash冲突时，会遍历链表中所有行，这样遍历代价非常之高，毕竟冲突会很常见无法避免，所以会少用。

连接类型（type）

1. All 全表扫描
2. Index 按照索引顺序进行全索引扫描，例如取出组合索引的所有列不加条件，或者加比较大范围的条件（较大很大的时候就直接全变扫描，不然既然扫描数又要扫描全表划不来）
3. Range 按索引范围查找，当范围较大时不按照索引来
4. Ref 按照索引准确找出某一行
5. eq\_ref 按照外键进行精确匹配某一行
6. Const/system 主键索引，唯一索性

建立高效索引：

1. 使用组合索引，避免建立多个单独的索引，因为要判断用哪个单独索引消耗性能
2. 避免使用独立的列，如果使用独立的列就不会用索引 where customer\_code = 4006 当customer\_code 为char或者varchar类型时,并建立索引;当用整数去取值，索引会失效。但是这种情况除外 customer\_id = ‘1’
3. 选择使用前缀索引；字符串比较长，选择前面几个字符来作为索引；这里会有一个比值；到底选取多长作为 count(distinct left(address,n))/count(address) 当n取值得到的值趋近于95以上就阔以
4. 选择合适的索引顺序，命中率越高的字段，越靠左，向左原则
5. 如果一个索引包含我们需要查询的所有列就叫做覆盖索引，也叫覆盖扫描
6. 排除冗余索引，重复索引，较少开销以及判断

不适合创建索引的条件：

1. 频繁跟新的字段不适合建立索引
2. 参与计算的列不适合
3. 数据重复唯一性太差的字段不适合：性别
4. 表数据比较少
5. Where条件中用不到的字段

行碎片 ，行间碎片（比较常见） ， 剩余的空间碎片 检查与修复损坏的表（check table /alter table customer ENGINE=Innodb ）

show variables like ‘query%’

1. query\_cache\_limit 缓存存储的限制，最大不超过这么多就进入到缓存
2. query\_cache\_min\_res\_unit 设置最小开辟缓存空间
3. query\_cache\_size 设置整体缓存大小
4. query\_cache\_type off /demad /on sql中加入SQL\_CACHE 表示进入缓存， SQL\_NO\_CACHED 表示不需要进入缓存

flush query cache 整理缓存块的碎片

reset query cache 清空缓存

show status like ‘Qcache%’

1. Qcache\_free\_memory 空闲的大小
2. Qcache\_hits 命中缓存个数
3. Qcache\_insert 插入缓存次数 2与3的比值至少3:1 才是有效缓存
4. Qcache\_lowmem\_prunes 因为内存不足而被丢弃的次数

语句优化提示器：

HIGH\_PRIORITY 与 LOW\_PRIORITY 多个语句同时访问某个表时，那个语句优先级更高

DELAYED 立即响应 并非所有引擎都支持，会导致last\_insert\_id无效

STRAIGHT\_JOIN 关联顺序 join 按照我们指定的方式来进行关联

SQL\_SMALL\_RESULT 和SQL\_BIG\_RESULT 对结果集排序 内存 文件

SQL\_BUFFER\_RESULT 缓存策略

SQL\_CACHE 与 SQL\_NO\_CACHED

FOR UPDATE 锁表

USE INDEX , IGNORE INDEX , FORCE INDEC

Select actor\_id , last\_name from actor use index(idx\_last) where actor\_id=1 and last\_name =’sdf’

USE INDEX 与 FORCE INDEX会强制使用指定的索引

优化count()

优化关联查询: on 子句上要有索引，并考虑关联顺序，关联的两个列字段类型 长度 必须相同

优化group by ： 通常关联分组，通常采用被扫描的表的列作为分组 a left join b group by a.id 不要用b.id 否则会采用临时表并排序；因为是以a表为准

limit优化: 延迟关联：select actor\_id , last\_name from actor join (select actor\_id from actor limit 100 ,10) as tem using (actor\_id)

Union 与 union all ：当两个结果集相同，有限选用union all 不然会排序，建立临时表

服务器优化：

mysqld --verbose --help|grep -A 1 ‘Default options’ 查看mysql配置文件；后面会覆盖前面的配置文件

max\_connections 允许客户端并发的连接数151 一般设置500-2000

max\_connect\_errors 如果客户端尝试连接的错误数量超过某一个值，则服务器不接受新的连接

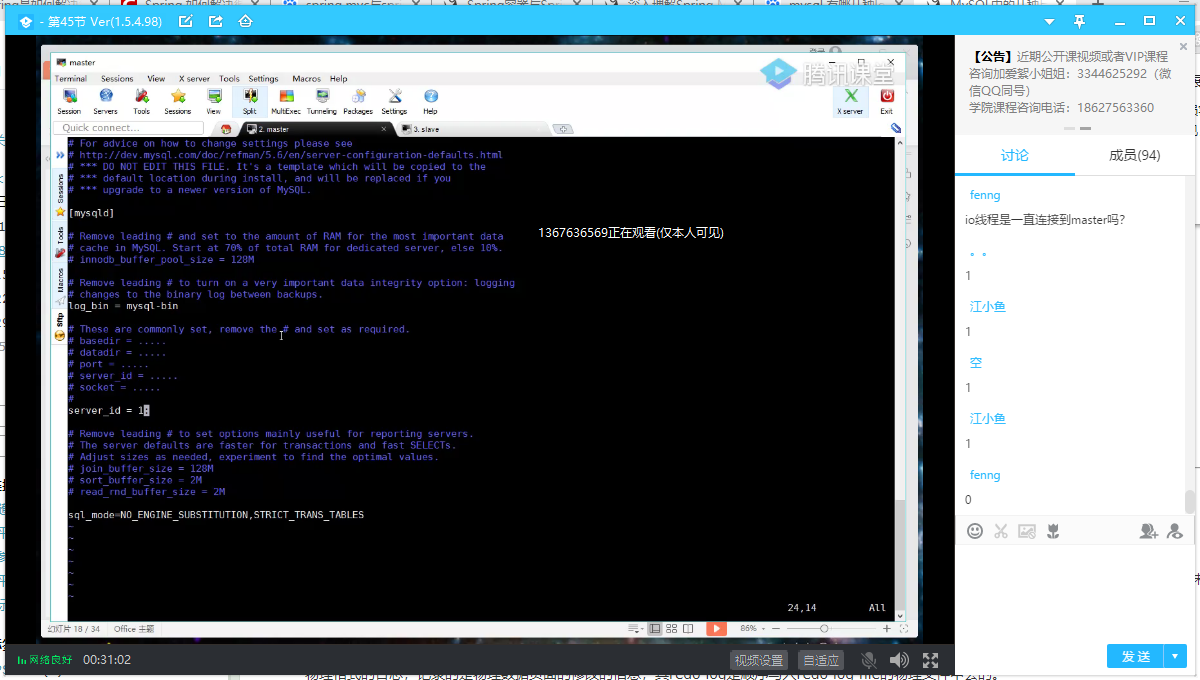
back\_log 服务器连接请求队列所能处理的最大连接数（default 80）

open\_file\_limit 操作系统允许mysql服务打开的文件数量

table\_open\_cache 所有线程能打开表的数量

thread\_cache\_size 服务缓存重用的线程数（coreSize）

Binlog日志深度研究解析：



Log 的三种方式

