MySQL精选问答500题

1、我创建了一个没有select权限的用户，执行select * from T where k=1，报错“select command denied”，并没有报错“unknown column”，是不是可以说明是在打开表之后才判断读取的列不存在?

答：这个是一个安全方面的考虑。你想想一个用户如果没有查看这个表的权限，你是会告诉他字段不对还是没权限？如果告诉他字段不对，其实给的信息太多了，因为没权限的意思还包含了：没权限知道字段是否存在。

2、wait_timeout 是客户端 非交互式的连接时间，如果程序连接mysql SERVER，是交互连接，关联的时间参数为interactive_timeout, 这两个时间参数需要尽量一致吗,一般设置多少合适?

query_cache_size 参数虽然不用了,我想确认下,关闭情况是 query_cache_size=0 要匹配参数query_cache_type=off吗？ 还是直接query_cache_size=0 即可?

答：第一个问题：是的，这两个尽量设置成相同。值的话取决于业务。如果你面对的是成熟的开发（比如公司内部团队），可以设置小些，分钟级别就行。

第二个问题：这两个都可以，不过用query_cache_type会好些（代码判断路径更短）。

3、文里遇到临时内存会跟随长连接的存在而存在，直到连接被销毁。想问的是，这部分临时内存是由于什么产生？为什么不提前释放呢？mysql有用到内存池吗?

答：排序、变量这些会占用内存，如果要复用，以前释放反而影响性能，MySQL没法自己决定，所以5.7之后提供了方法让客户端有命令来做；MySQL的内存池一般说的是查询缓存和引擎里面的（比如InnoDB 的buffe pool), 跟线程内缓存不是一个概念。

4、文中join的例子，内表和外表 都有高区分度条件的情况下，先过滤出两表符合条件的记录，再对这些记录做join，可不可以？感觉这样性能更高。

答：MySQL确实有你提到的这种算法，叫做index_merge. 不过我们这个case里面不会，由于实现的原因，MySQL至今没有支持跨表index_merge。

5、末尾问题有个疑问。是在分析器里面检查的列是否存在。且文章中说到执行器会检查有没有权限。那为什么一个没有查询权限的用户进行查询语句，提示的是”没有权限”而不是”列不存在”。按照这个逻辑权限应该是在执行器做的，那这个”权限不存在”是哪个模块检查的?

答：查询的时候权限不存在是在执行前单独多的一次判断，是为了安全考虑。就是如果没有权限，你连“字段不存在”这个信息都不应该暴露给客户端。

6、如果你用的是 MySQL 5.7 或更新版本，可以在每次执行比较大的操作, 请问比较大的操作的判断维度是什么呢? 即怎么判断是比较大的操作?

答：其实就是类似于很多数据排序，或者一个操作复杂的存储过程。这个是可以在测试环境稳定复现的。 如果语句都差不多，或者比较随机，或者可以用另外一种方法，比如一个连接执行了N个语句以后做reset这个动作。

7、既然在链接阶段已经通过权限表获取了这个该连接所具有的权限，那么在执行阶段再检查一次的意义何在?

答：执行器阶段会碰到需要再判断权限的情况，这时候读内存中事先存好的权限，而这个权限是在连接器阶段算出来存进去的。

8、请教个业务操作表的场景，如果对一个业务数据表的操作既有查询（分页查询批处理），又有更新，此场景下都是操作主库好呢? 还是说查询走从库，更新操作走从库? 您有没有好的建议呢，如果遇到了此类业务场景您会怎么分析解决此困扰呢?

答：必须做语句分类，延迟敏感的走主库，不敏感的走从库。

9、上面说如果开启缓存查询在不命中的情况下开始执行分析器，分析语句，那么我发送语句的key是什么时候生成的还是直接拿语句作为key，这时候我过我语句的列中包含了错误的列名在这个阶段是不检查的吗?

答：就是语句做key, 这时候还没到分析器阶段，确实还没检查。 不过不影响逻辑正确性，你分析的时候要带上失效逻辑哈。

10、我们有什么办法实时获取目前已查询到多少行的数据了？比如我用php语言怎么调用rows_examined实时获取查询的进度?

答：查询进度是没有的，如果是查询结束后，看查到多少行，就是MySQL_num_rows. 不过这个跟rows_examined不一样哦。

11、权限为什么不在句法分析之前呢，此时已经完成了词法分析，知道了查询的表和列了，完全可以分析。

答：执行过程可能会有触发器这些，还是得到这一步才判断。 但是为了安全，有一个precheck会在优化器之前判断一下权限。

12、关于在连接器方面，在连接器中怎样给新的连接归类为长连接还是短连接的，判断的标准是什么?

答：不用判断，连接建立成功后就是连接，断开就是断开。只是我们建议减少创建连接到次数。

13、文中说明权限判断是在执行器中进行的，那么假如查询的语句命中了查询缓存，按流程就直接返回了，那这种情况权限判断是怎么处理的呢?

答：从query_cache查到结果后，结果里面有表的信息，要做权限验证的，权限不够还是报错处理。

14、c mysql接口连接设置好mysql_ping自动重连之后。查询时候会自动调用吗？还是要自己主动定时检测。

答：虽然会自动了，但是如果是程序里有用到事务，还是需要处理的，否则事务被拆成两个（前面一半自动回滚）业务还不知道，也不是好事哦。

15、在mysql中的多字段模糊查询同一个值！怎么写的执行效率高?

答：MySQL 支持全文检索(fulltext index)的，比顺序遍历快，不过相比于ES这样的系统，MySQL 的全文检索能力还是有待加强。

16、“MySQL 在执行过程中临时使用的内存是管理在连接对象里面的”

请问“管理在连接对象里面”这个怎么理解呢？是还有管理在其它什么地方的吗？是因为管理在对象里面才会被 OOM 的吗?

答：每个连接对应一个THD对象，这些内容都在这个对象里。

17、关于不建议用查询缓存的，虽然查询缓存命中率很低，但是一旦命中不是就能减少查询时间吗。不使用查询缓存是能节省去查询缓存中匹配查询的流程，还有什么其它好处吗？从你说的内容中我还不是很明白不用查询缓存带来了什么好处?

答：不是，不使用查询缓存就不需要维护，就是查到结果不用写到查询缓存去，这个才是最大的收益哦。

18、查询缓存的时候需要执行select语句，这个时候不需要经过分析器对语法词法进行分析吗?
答：查询缓存不是”select语句”，是查询内部的一个数据结构。

19、java用jdbc或者mybatis连接数据库，对于定时任务（每天执行一次），连接器会在8小时自动断开链接？那每天执行的时候是重新获取连接?
答：你要看下这个客户端的代码实现。看看他是不是自己会重新获得连接，或者是不是会维持心跳。

20、Write-Ahead Logging–先写日志再写磁盘，空闲的时候写入磁盘。那么如何判别空闲呢?假如我增加一条数据 ，而这条数据只写入了日志 ，还没来得及写进磁盘，而我还要读取刚写入的这条数据怎么办呢?
答：读内存。至于空闲，系统有判断规则的，比如QPS这类。

21、问题1：

在缓存开启状态下， 要判别sql是select还是其他类型，保证只有select才去查询缓存，那么这个判断操作是何时进行的呢？ 我猜测在查询缓存之前， 但是分析器也要对sql关键字进行分析，再分析一遍岂不是重复？ 因为我没有看源码，所以这一块不明白， 如果我来设计的话， 可能是把查询缓存这个模块在分析器阶段来执行， 由分析器进行关键字解析，判断是何种sql语句，如果是select， 再去查询缓存， 如果不是继续往下进行好像更合理，看到大家都说先查询缓存，没有命中再去分析器， 感觉不太通。

问题2：

在mysql中通过“explain extended和show warnings”命令来查看的sql， 是不是优化器重写之后的sql？ 以前我看到一本书中提到，也可能是一篇博客，mysql的单表的union查询其实会优化成or查询， 但是show warnings看到的还是union查询语句，所以有此疑问。

问题3：

update等sql清空缓存是在sql执行完返回结果给客户端之前还是之后， 还是说同时进行的，会不会存在清空缓存失败， 但是告诉客户端update success， 缓存的地方一直不是很清楚，所以正好 will be removed in a future releas， 真是一个大快人心。

答：1. 实际上肯定是要判断出是select语句， 但是之后的解析都没做哈。 要理解成“包含在内”也可以。

2. Show wanings不会把重写后的显示出来，只是格式化后。

3. 清空缓存失败这个不太可能，这个内部结构的操作没什么依赖。

22、数据库所在的docker配了80G内存。但因数据库配置文件里的参数配置不好，通过计算，达到了1024G，而，现在已经到达容器内存100%了。之后我把参数改到合理值，但内存还没降，应该是保持会话的没生效，新会话生效了。现在数据库不能重启，也不能把会话都kill掉。要怎么处理才能使内存降下来的?

答：会话不能kill? 这样程序应该升级，正常的程序要能够处理异常连接连接断开哦。

23、如果用户名密码认证通过，连接器会到权限表里面查出你拥有的权限和执行阶段去查询有没有select的权限这两个有什么区别?

答：连接器阶段是去系统表读数据，结果放在变量；执行器使用这个变量。

24、文中写到，执行器会调用引擎提供的借口查询数据。是不是可以认为，最终遍历数据是引擎来完成当的。

答：“遍历”这个动作是执行器让做的，引擎给了两个接口：取第一个、取下一个。

25、文章提到：

1. 建立连接成功之后，在连接器阶段会先到查询缓存查看是否存在该 SQL 的缓存。

2. 针对一条 SQL 语句，在分析器的词法分析阶段，从关键字 select 中识别出是一条查询语句。

这里有一个疑问，在分析器阶段才识别出 SQL 是查询语句，而在连接器阶段，如何决定是否需要到查询缓存看看呢?

答：嗯，确实是“识别出这个是一个查询缓存”这个动作是在连接器之后，查询缓存之前的，但是在查询缓存之前也只做了这一步小小的解析哈。

26、MySQL的连接池就是把用完的链接放回池子里，给别的业务线程复用，变短链接为长链接。

MySQL的连接池是在InnoDB 的buffe pool里面的吗？和MySQL企业版的thread_pool的实现原理类似吗?

答：1. 不是，连接池是server层的。

2. Thread_pool是线程池，给不同的连接复用线程；连接池是复用连接。

27、MySQL 拿到一个查询请求后，会先到查询缓存看看，之前是不是执行过这条语句。之前执行过的语句及其结果可能会以 key-value 对的形式，被直接缓存在内存中。key 是查询的语句，value 是查询的结果。如果你的查询能够直接在这个缓存中找到 key，那么这个 value 就会被直接返回给客户端。

老师请问关于上面的这段，如果执行语句相同，仅仅是条件不同，会走查询缓存么?

如：select * from user where id=6

select * from user where id=7

这两句.

答：不会，查询缓存要求语句一摸一样才行，别说你这个参数变了，就算两个语句差异只是多一个空格都不行哦。

28、时间区间查询，between and跟<=>=有性能差异吗?

答：没有，不止时间区间，所有类型这两个都是等价操作。

29、执行器去操作引擎获取数据的时候，因为引擎查询数据也需要时间，那执行器是不是有个等待的过程？您说的rows_examined是记录执行器每调用引擎获取数据刚时累加，那是不是可以这样理解在一个查询过程中，执行器会间隔一段时间去引擎获取数据？如果是这种的话，我又有一个疑问了，执行器怎么知道引擎已经查询完了，最终返回数据的呢?

答：不是。执行器是同步调用引擎接口，不是异步调用。执行过程中累计的。

30、问题1：和其他同学有类似的疑问：关于没有权限操作表的时候的报错：

如果分析器已经知道了“unknown column”之后，为什么还会报错：“select command denied”。你之前的回答是：由于安全考虑，用户没有权限操作这个表，那么也不需要告诉用户这么多，直接告诉用户没有权限。

【重点疑问】如果是按照这样的话，且对表的权限检查都在执行器的话，那么每个sql在执行的时候至少都会执行到执行器吗?

因为同时看到文中在执行器章节：于是就进入了执行器阶段，开始执行语句。开始执行的时候，要先判断一下你对这个表 T 有没有执行查询的权限，如果没有，就会返回没有权限的错误，如下所示。

SELECT command denied to user 'b'@'localhost' for table 'T'。

问题2：两位测试结果不一样，A的测试结果是Complete optimizer trace；B从他的描述来看，测试结果应该是分析器。没有本项目上的MySQL权限，等明天自己本地版下好后，测试下结果。

答：1. 不是的，由于工程实现的原因，会有前置检查。

2. 这个也类似，其实如果看调用路径，执行器阶段都在mysql_parse函数里面调用的。 做这些阶段的拆分我觉得主要是方便我们从概念上理解，工程实现的时候确实不是1234这么按部就班的。

31、执行 mysql_reset_connection 来重新初始化连接资源，我想请问下，如果在主从DB服务器执行这个操作会影响主从同步吗?

答：不会，这个操作没有写binlog,只是更新连接的线程内部数据。

32、“在工程实现上，如果命中查询缓存，会在查询缓存放回结果的时候，做权限验证。查询也会在优化器之前调用 precheck 验证权限。”不是很理解这块。

1.命中查询缓存之前在连接阶段不是已经做过权限验证了吗,为什么查询缓存放回结果的时候要再做权限验证。

2.优化器之前调用 precheck 验证权限和执行阶段的权限验证有何区别。

答：1. 连接阶段只是“获得权限信息”， 真正开始查询动作，才判断“有没有操作这个表的权限”。

2. 不同的阶段， 执行器阶段是判断一些关联操作，比如更新一行的时候，由于触发器会再更新别的表，这种情况。

33、文中执行器模块给的例子sql是select * from ，引擎层返回了一整行数据，请问老师如果我只是select id from ，引擎层是只返回id一列，还是一整行。

答：只取id的话，引擎就只返回id给执行器。

34、怎样查看是长连接还是短连接，定期断开长连接怎样配置?另外最近做性能测试时发现大并发用户同时连接MySQL时会有部分用户连接超时(设置的最大连接数1000，实际使用的最大连接不会超过200)，关于MySQL连接管理这块该怎样优化?

答：那你这个就是典型的短连接了，建议改成长链接方案。

35、为什么分析器完了还要走查询缓存？不是之前查询缓存没命中吗？查询缓存是在分析器阶段之前的。

答：只是如果有更新语句，要去失效查询缓存。

36、MySQL 在执行过程中临时使用的内存是管理在连接对象里面的。这些资源会在连接断开的时候才释放。这里的内存是什么？不是BP，对吧。

答：是的，不是BP。 BP是在InnoDB里面维护的，只有InnoDB才；执行过程中的临时内存，是在连接对象里。

37、第一次调用的是“取满足条件的第一行”这个接口，之后循环取“满足条件的下一行”这个接口，这些接口都是引擎中已经定义好的。这里面的接口怎么理解，我们在数据库中说的接口是什么。

答：这里的接口是指“引擎定义好的函数调用方法”，只是这个定义不是乱定义的，MySQL是一个插件式的结构，要求每个引擎以固定的方式提供调用方法。

38、skip-name-resolve这个参数是不是都需要配置优化呢，之前由于这个参数没有配置导致sql查询特别慢，好多接口也因为这个响应时间特别长，请问老师域名解析是在连接器阶段吗？还有这个参数是否为必须优化参数呢?

答：是在连接器阶段，建议全部设置为 skip-name-resolve。

39、mysql connection 内存占用过多，有没有办法在shell中看到具体的内存占用呢?

mysql_reset_connection 能不能再shell中执行?

答：官方版本还没有支持直接看内存消耗的，不能再shell执行的，是一个c的api。

40、请问老师，短连接和长连接是怎么设置的？是连接数据库的时候有什么参数吗?

答：不是，这是“行为”，比如连接完，执行一个查询，就断开，这是短连接；执行一个查询，不断开，下次查询还用这个连接，持续使用，就是长链接。

41、短链接。 假设 代码执行2次select语句。每次都会重新建立连接吗?

答：这个是程序行为，看你程序怎么写的哈，也不一定就是2个。

比如以前用php写web页面，一个页面逻辑执行过程中用同同一个连接，这个页面用完就关掉连接。再刷新页面的时候再创建一个新连接，这个我也认为是短连接。

42、“输完命令之后，你就需要在交互对话里面输入密码。虽然密码也可以...”

如果是生产上面的数据库，为了查询，不在交互模式里直接输入密码，那么按照正确流程是如何进行连接的？场景是：在生产环境访问数据库。

发散一下，如果为了要养成良好习惯，我们在测试或者开发环境也不直接输入密码，那在测试或者开发环境怎样访问数据库是正确的?

答：1. 最好是在交互模式里面输入密码。

2. 对，测试或开发环境，最好也是交互模式里面输入密码。

43、逻辑架构图中,查询缓存有连接器和分析器两条线，是否说明有时候直接从连接器去查询缓存查看是否命中，有时候会先经过分析器之后再去找查询缓存？如果是的话，为什么会有两种情况呢？缓存的key是一条sql语句吗?

答：分析器那条线是“更新”查询缓存。

44、MySQL 在执行过程中临时使用的内存是管理在连接对象里面的。 这句话啥意思?

答：就是说在线程断开的时候，内存肯定就会回收，不跟别的线程混用。

45、“数据库里面，长连接是指连接成功后，如果客户端持续有请求，则一直使用同一个连接。短连接则是指每次执行完很少的几次查询就断开连接，下次查询再重新建立一个。”想问下长连接和短连接是不是人为的主观上来判断的，还是通过某些量化标准来定义的?

答：确实有主观成分，衡量标准就是“一次连接持续期间，执行了多少个sql语句”。

46、我可以认为redo log 记录的是这个行在这个页更新之后的状态，binlog 记录的是sql吗?

Redo log不是记录数据页“更新之后的状态”，而是记录这个页 “做了什么改动”。

答：Binlog有两种模式，statement 格式的话是记sql语句， row格式会记录行的内容，记两条，更新前和更新后都有。

47、我想问下如果提交事务的时候正好重启那么redo log和binlog会怎么处理？此时redo log处于prepare阶段，如果不接受这条log，但是binlog已经接受，还是说binlog会去检查redo log的状态，状态为prepare的不会恢复?

答：Binlog如果已经接受，那么redolog是prepare, binlog已经完整了对吧，这时候崩溃恢复过程会认可这个事务，提交掉。 （你可以分析下这种情况下，是否符合我们要达到的“用binlog恢

复的库跟原库逻辑相同”这个要求）。

48、有两个地方不太理解，一是，binlog也可以被引擎操作，那么binlog除了由执行器记录逻辑信息，是否引擎也会参与进binlog的记录？另外就是小结中提到的将log文件在磁盘中持久化，这是一个怎样的操作？因为一想到外部磁盘总感觉和内存相比会很慢，每次更新都持久化磁盘会不会影响效率?

答：记录binlog这个动作是执行器做的，引擎会提供信息；写盘点代价比内存高的。但是这个是没法避免的。除非业务可以接受主机掉电丢数据。对于不能接受的业务，这个是绕不开的成本。 所以幸好我们已经过了机械硬盘时代。

49、比如因为删表等操作需要回滚，但不希望数据库执行binlog中全部的语句 比如说删除语句不执行的话 可以通过自己手动筛选数据库执行的binlog类型吗？还是必须都要执行呢?

答：可以在线程里面单独设置 是否使用binlog, 也可以单独设置binlog格式，前提是要有super权限。

50、也就是说状态恢复的过程会去同时检查redo log和binlog？不然怎么能确定一个prepare的redo log已经写好了binlog，因为不检查的话不能确定到底是写好了binlog之后奔溃的还是写之前奔溃的，也就不能确定这个prepare log的合法性。如果检查的话那么不用两阶段提交好像也没什么问题，无论先写哪个日志都可以。

答：很好的思考，你考虑一下这个场景，redolog没写，binlog写好了，怎么算。 注意由于binlog写完了，在备份恢复回来以后是会重放这个事务的。

51、在两阶段提交那部分，每当一个动作完成后，会有事件通知吗？比如在binlog写入成功后，进行commit redo log，这个commit动作在成功或者失败后，会有回调或者事件通知吗？我感觉如果有通知机制的话，能更好的保持两个log的一致性，老师怎么认为呢?

答：这个就涉及“组提交”概念。是的，有通知，commit完成然后才继续后面的流程（比如返回ack给客户端）。

52、redo log是为了快速响应SQL充当了粉板，这里有两个疑问：

1. redo log本身也是文件，记录文件的过程其实也是写磁盘，那和文中提到的离线写磁盘操作有何区别?

2.响应一次SQL我理解是要同时操作两个日志文件？也就是写磁盘两次?

答：你的理解是对的。

1. 写redo log是顺序写，不用去“找位置”，而更新数据需要找位置。

2. 其实是3次（redolog两次 binlog 1次）。不过在并发更新的时候会合并写。

53、请问用redolog恢复时还写binlog吗？反之呢?

答：崩溃恢复过程不写binlog了，用binlog恢复实例（或搭建备库）的时候，是会写redolog的。

54、频繁的更新写入，导致redolog过大，适合短周期的备份?

答：Redolog固定大小，不会“导致过大”的。

55、binglog 一般建议设置啥模式，假如row模式binglog 怎么回复?

答：建议设置row模式，row模式日志一样的可以用于重放的。

56、既然write pos和checkout都是往后推移并循环的，而且当write pos赶上checkout的时候要停下来，将checkout往后推进，那么是不是意味着write pos的位置始终在checkout后面，最多在一起，而这和老师画的图有些出入，不知道我的理解是不是有些错误。

答：因为是“循环”的，图中这个状态下，write_pos 往前写，写到3号文件末尾，就回到0号继续写，这样你再理解看看“追”的状态。

说明一下，文中“write pos和checkpoint之间的是’粉板’上还空着的部分，可以用来记录新的操作。” 这句话，说的“空着的部分”，就是write pos 到3号文件末尾，再加上0号文件开头到checkpoint 的部分。

57、innodb结构为内存池和后台线程 ， 其中内存池里存在redo日志缓冲 和 缓冲池，其中缓冲池以页为单位缓冲磁盘数据 。

问题:

1.之前老师说更新语句时执行器调用innodb存储引擎先从内存读数据，指的就是从innodb缓存池中读取数据把，没有的话就去读取磁盘找， 找到后再同步到缓冲池。

2. 在 server层中， 连接器先 查询缓存 ， 这个缓存指的也是 innodb 的缓冲池么。mysql 通过公有的缓冲组件调用 innodb 的缓冲池来获取值。 获取不到在走 分析器 --> 优化器 --> 执行器的流程 。

3. 从缓冲中取的时候 ，为什么没有 分析器 --> 优化器 --> 执行器 的流程呢?

答：1. 是的。

2. 不是，server 层的查询缓存就是（query_cache）Server层跟innodb的buffer pool是没有交集的，引擎没有暴露这个接口，只有操作数据的接口。
3. 跟2类似，不是一个东西哦。

58、redo 是引擎提供的，binlog 是server 自带的，文中提到前者用在crash的恢复，后者用于库的恢复，两者是否在某种程度上是重复的？如果在都是追加写的情况下，是否两种日志都能用于 crash 与 库 的恢复呢?

答：Crash-safe是崩溃恢复，就是原地满血复活；binlog恢复是制造一个影分身（副本）出来。

59、是否在redo没写时重启数据库，会使事务丢失？这个交易也失败。怎样情况重启会出现数据库数据不一致呢？oracle都存在重启后数据不一致的情况。数据不一致后怎么恢复?

答：Redolog没写，这个不算丢数据的。交易失败是预期的，也应该这样处理。就靠业务重试。

60、老师，我现在有点搞不懂 mysql 缓存机制和 innodb缓存机制的关系 ，以及 redo日志也有缓存，如果在缓存中还没同步到 redo 日志里的时候，mysql 发生的 crash 时，是否更新记录就真的没了?

答：会没了，但是没关系，因为redolog都没写完，表示你事务还没提交，中间出了错，本来就应该没的。

61、老师 如果 binlog 或者 redo log 在写入磁盘的过程中故障了怎么保障数据一致?

答：整个事务回滚，数据也没有，日志也没有，也是一致的。

62、问题一：写redo日志也是写io（我理解也是外部存储）。同样耗费性能。怎么能做到优化呢?

问题二：数据库只有redo commit 之后才会真正提交到数据库吗?

答：1. Redolog是顺序写，并且可以组提交，还有别的一些优化，收益最大是是这两个因素；

2. 是这样，正常执行是要commit 才算完，但是崩溃恢复过程的话，可以接受“redolog prepare 并且binlog完整”的情况。

63、两阶段提交那里不太懂，我理解是不是只要binlog保存成功，数据就能恢复，redolog 是不是就为了加快innodb引擎的写速度的?

答：Binlog写完就算，但是并不是用binlog恢复，恢复这个工作还是innodb做的。

64、redo log的机制看起来和ring buffer一样的；如果在重启后，需要通过检查binlog来确认redo log中处于prepare的事务是否需要commit，那是否不需要二阶段提交，直接以binlog的为准，如果binlog中不存在的，就认为是需要回滚的。这个地方，是不是我漏了什么，拉不通。

答：文章中有提到“binlog没有被用来做崩溃恢复”，历史上的原因是，这个是一开始就这么设计的，所以不能只依赖binlog。操作上的原因是，binlog是可以关的，你如果有权限，可以set sql_log_bin=0关掉本线程的binlog日志。 所以只依赖binlog来恢复就靠不住。

65、明白double write是为了保证数据页面的完整性，我上午问的是db 异常关闭起来需要做恢复的时候，因为MySQL 的redo 是物理逻辑日志，记录的是页面逻辑的改动，所以要保证数据页面的完整性，这个就是靠double write 去保证的，是这样吗？而Oracle因为redo是物理日志，所以不需要double write技术的保证?

答：对，这样说就精确 。不过现在有些硬盘支持16K的原子写保证，就可以关掉double write buffer，MariaDB 就提供了这个参数允许关掉。

66、多写场景需要吧 指标应该是RTO。有个问题既然innodb事务性这么强，为什么在恢复临时库不用redo恢复呢？直接对页进行恢复 数据一致性强，恢复时长也短，不用多写一份binlog，减少写放大。这样做主从直接redo同步岂不是更好吗(就是redo量有点大)？

答：Redo主要就是因为循环使用会覆盖；不过你说的这个逻辑，业界已经有团队通过改造InnoDB行为在做了，只是不是官方做的，所以正文中没提。

67、binlog为什么说是逻辑日志呢？它里面有内容也会存储成物理文件，怎么说是逻辑而不是物理?

答：这样理解哈。 逻辑日志可以给别的数据库，别的引擎使用，已经大家都讲得通这个“逻辑”；

物理日志就只有“我”自己能用，别人没有共享我的“物理格式”。

68、MYSQL第二讲中提到binlog和redo log, 我感觉binlog很多余，按理是不是只要redo log就够了?

您讲的时候说redo log是InnoDB的要求，因为以plugin的形式加入到MySQL中，此时binlog作为Server层的日志已然存在，所以便有了两者共存的现状。但我觉得这并不能解释我们在只用InonoDB引擎的时候还保留Binlog这种设计的原因。

答：binlog还不能去掉。一个原因是，redolog只有InnoDB有，别的引擎没有。另一个原因是，redolog是循环写的，不持久保存，binlog的“归档”这个功能，redolog是不具备的。

69、Bin log 用于记录了完整的逻辑记录，所有的逻辑记录在 bin log 里都能找到，所以在备份恢复时，是以 bin log 为基础，通过其记录的完整逻辑操作，备份出一个和原库完整的数据。

在两阶段提交时，若 redo log 写入成功，bin log 写入失败，则后续通过 bin log 恢复时，恢复的数据将会缺失一部分。(如 redo log 执行了 update t set status = 1，此时原库的数据 status 已更新为 1，而 bin log 写入失败，没有记录这一操作，后续备份恢复时，其 status = 0，导致数据不一致）。若先写入 bin log，当 bin log 写入成功，而 redo log 写入失败时，原库中的 status 仍然是 0 ，但是当通过 bin log 恢复时，其记录的操作是 set status = 1，也会导致数据不一致。其核心就是， redo log 记录的，即使异常重启，都会刷新到磁盘，而 bin log 记录的， 则主要用于备份。

我可以这样理解吗？还有就是如何保证 redo log 和 bin log 操作的一致性啊?

答：几乎全对，除了这个“两阶段提交时，若redo log写入成功，但binlog写入失败…”这句话。实际上，因为是两阶段提交，这时候redolog只是完成了prepare, 而binlog又失败，那么事务本身会回滚，所以这个库里面status的值是0。如果通过binlog 恢复出一个库，status值也是0。这样不算丢失，这样是合理的结果，两阶段就是保证一致性用的。你不用担心日志写错，那样就是bug了。

70、问题一：redolog记录更新日志，当更新完一个表之后再进行查询，此时会将这个表的redolog刷到物理表中再进行查询吗？如果不是的话是如何保证数据最新?

问题二：数据恢复是以binlog的记录为准吗？如果要考虑redolog，那redolog如何帮助恢复?

答：1. 不会。就直接从内存读走，内存是对的就行。

2. 用binlog是用在备份恢复到过程中，这个过程就是用全量备份+binlog,你可以认为没redolog的帮助。

71、binlog感觉像oracle的归档日志，那么也有设置binlog日期的参数，这个参数是？默认值又是多少?

答：expire_logs_days。默认是0 表示不自动删除；非零的话，单位是天。

72、如果认可这个事务可以从crash恢复，起码要求服务器告知客户端事务成功了吧，那您说redo log prepare + bin log ok，就可以恢复了，说明bin log写入成功就算事务成功了？那最后一步确实没什么用呀。

答：有用，如果redolog 已经commit了，崩溃恢复就不需要去找binlog。

73、请教下redo log 和 binlog 也是需要写文件的，也会消耗IO，这个不会影响数据库的性能吗？ 从内容看redo log首先写道缓存中，如果这个时候断电了，缓存也清空了，尚未写道磁盘，这个时候不是redo log 不完整的如何保证恢复的完整性呢?

答：要写IO，是会影响，但是为了数据可靠性，还是要做的。Redolog是直接写盘的，更新的内存是数据的内存。如果没有写完就重启，就回滚，不算的丢数据。

74、pos和checkpoint的位置一開始是怎麼確定的呢？checkpoint什麼時候移動呢？應該怎麼移動？移動多少?

答：一开始就是从头了，中间如果有重启，也从重启前的位置继续。 移动就是向右，到末尾循环；移动多少 取决于事务要写下多少的日志。

75、怎么知道binlog是完整的?

答：一个事务的完整binlog是有固定格式，也就是说有固定结尾的。

76、两个控制日志写磁盘的性能参数 如果都设置1 会不会对io影响很大 对大并发也要设成1吗?

答：一般这两个都建议设置成1（有时候称双1），除非你的系统IO撑不住了要临时应急。

77、两阶段提交是针对自动提交事务的语句来说的吗？如果是手动提交事务呢?

执行更新语句，只更新当前session缓存。当手动commit的时候，才是 记录redo log 处于prepare状态，执行器执行，记录binlog，然后 redlog commit。是这样吗?

如果用户手动commit，但是第三步redlog变成 commit的时候挂掉，这时候用户收到的执行结果是什么?---这时候恢复的时候是认可这次commit的。

答：不是，这个跟手动还是自动无关。你说的commit是一个命令，“提交这个事务”，包含了整个提交过程（也就是说两次redolog和一次binlog操作，都是在你的这个commit **命令**中做的）。

78、关于提到的'数据页'这个词我没有太理解，是一种存储方式么?

答：MySQL的记录是以“页”为单位存取的，默认大小16K。也就是说，你要访问磁盘中一个记录，不会只读这个记录，而会把它所在的16K数据一起读入内存。

79、update后先写入内存，然后写redo log，那么写入的这条数据在内存中的驻留时间是怎么处理的？是redo log写入磁盘后从内存中删除吗?

答：不会删除，除非内存不够用。不过在数据从内存删除之前，系统会保证它会被更新到磁盘数据上。

80、我想模式redo_log成功，bin_log失败；或者反过来。该如何模拟呢?

答：这个稍微有点麻烦，最直接方法是GDB，停在两个函数中间，然后coredump出来。

81、如果更新写入redolog后处于prepare状态，此时系统突然断电，那么binlog没有写入记录，重启后数据库的实际数据根据redolog更新了，但与binlog的语句记录不符，那么binlog怎么办?

答：不会，这时候事务会回滚，binlog也没有，所以是一致的。

82、在一个大事物或长事物里，一边执行sql一边写redo log吗？未提交的事物也会写到redo log file吗?

答：会先写一块内存，叫做log_buffer里面，在提交的时候再一次性写到磁盘。

83、在binlog写入磁盘后，commit提交前发生crash，由于commit没有成功，那返回给客户端的消息是事物失败，但是在系统恢复的时候却会重新提交事物，使之成功，这不是在欺骗客户端嘛？问题很大啊。

答：不是，并没有返回“事务失败”呀，而是“执行异常”。执行异常本来就可能成功也可能失败的。你想一下这个场景：执行全部完成了，在回复客户端的时候网络断了，这怎么算。

84、redo log是prepare状态，写binlog时crash；以及binlog写成功，但是redolog更新为commit状态时crash；重启后的回复机制又是怎么做的呢?

答：第一个回滚，第二个事务有效。

85、定时备份也是通过执行当前时间到上一次备份时间之间的binlog来完成的吗?

答：这个其实也是一种做法，不会大家用的更多的还是直接找个备库再全量备一次。

86、有没有这样的命令dump日志得到查询历史SQL语句?

答：更新在binlog里面有记录，用mysqlbinlog工具可以，仅限更新语句。

87、有一段不太懂：具体来说，当有一条记录需要更新的时候，InnoDB 引擎就会先把记录写到 redo log（粉板）里面，并更新内存，这个时候更新就算完成了。

这里所说的，并更新内存，值得是更新内存里面的数据吗？ 即id为2的数据，

另一个问题，按照前面所讲的分区，那内存里面最大存储的数据量是否为4个G？（即，老板一直没时间写入账本，小黑板记录的最大数据量）。

答：1. 对，更新内存

2. 粉板记录的最多的更新记录是4G。 我们说的内存是放数据的内存，就是buffer_pool的大小。

88、问题一：我的数据表设计 id user ud,先做个多线程拿号登陆的功能，未避免拿到重复的号码，我百度到使用UPDATE $db SET ud=1, id=LAST_INSERT_ID(id) WHERE ud=0 LIMIT 1；SELECT user FROM $db WHERE ROW_COUNT()>0 and id=LAST_INSERT_ID()；这两句来取号，但是效率很低，数据一大，mysql还容易崩溃。请问有更好的方法吗?

问题二：我的第二个项目数据表设计 id user info。数据利润 1，user1，1001;1002;1003。我想查询包含1001的数据，用like的话，数据少的话还可以接受，当百万数据的时候，就会很卡很慢。想请教老师有其他的方案吗?

答：1. 你这个做法不好，应该先insert,拿到的这个last_insert_id就行

2. 如果有这种需求，最好不要把这些拆成单行。 或者创建另外一个表单独维护包含“1001”对应的集合。

89、redo日志在prepare阶段之后就已经刷盘啦？我按我的理解，innodb_flush_log_at_trx_commit 应该是控制在commit之后才会写磁盘啊？如果是没有问题，如果不是，binlog刷盘后是无法保证redolog刷盘的。

答：这个prepare已经是在你执行“COMMIT”语句里面了。

90、innodb_flush_log_at_trx_commit这个参数设置成 1，是不是等于binlog的归档了呢?

答：不是，这个但是是控制redolog的。 binlog存起来就有“归档”的功能。

91、问题一：文中‘’update 语句执行流程‘’示意图，是先修改数据，在记日志。不应该是先写日志（生成重做日志），在修改数据（缓存里）吗?

问题二：redo log两阶段提交具体是怎么实现呢？我理解就是直接记录哪个页面做了什么修改到redolog（硬盘日志文件）就结束了，已经落盘了，prepare，和commit怎么理解呢?

答：1. 这两个其实是交叉发生的（对于一个有多次更新日志的事务来说），所以可以忽略先后。关键是他们都在“把日志写入redolog物理文件之前。

2. 有了binlog, 需要两阶段。

92、为什么binlog没有crash-safe的能力呢？不是写磁盘了吗？可以在重启后把宕机前的binlog中记录下来的异常事务sql语句执行一遍?

答：一开始没这么设计，实际上现在的binlog也不够，奔溃恢复需要日志和引擎内存状态配合的。

93、mysql的redo log有没有log buffer？对于高并发事务或者批量事务 如何优化redo文件的写入瓶颈?

答：有buffer的，事务整个完成才学到redolog文件，redolog是顺序写。

94、checkpoint是指redo buffer落盘的位置，还是指日志更新写到表中的位置（crash recovery的起始点），我在理解oracle的checkpoint是后者的。。就不知道他们俩是否都一样?

答：Write pos是redo buffer接下来要写的位置， crash recovery 的起点就是checkpoint. (Crash recovery起点。

95、如果第三步用户不是想commit而是想rollback,但不巧在第三步的时候崩溃了，回复后根据redo log和binlog一致来自动commit?

答：如果用户执行rollback语句，整个事务就放弃了，不会进入“prepare”状态。

96、更新停留在redo-log bin-log上时，查询是如何把新数据也包括进来？是在查询启动的时候把redo-log的数据刷新到页数据里面么？同一个数据库里，查询和更新同样多时，岂不是会让redo-log的“粉笔板”效果失效?

答：不会，最新的结果会保存在内存里面，就是“掌柜的记忆”，查询直接从内存返回就可以了。

97、情形是这样的，我们因为磁盘IO异常导致数据库异常重启了，并发现某个表数据损坏了（无数据备份）且mysql无法开启，一开始的做法innodb_force_recovery设置到1-6直到数据库可以启动，然后进行mysqldump备份，然而并不能备份，报错是mysqldump: Error 2013: Lost connection to MySQL server during query when dumping table `XX表`at row: 31961089。于是就跳过这个表进行备份，并修复了其他数据库，但是这个表的数据丢失最终请了外援修复好了，但是具体没告诉我们，不知道和我们这个有没联系？有没提示让我学学习下这方面？是可以用ibd文件修复吗？

答：你说的这个case我分析了一下。

1. 这个现象应该是数据页（page）的内容错了。InnoDB在读到数据页的时候会判断checksum，碰到有不对的页，就会自己crash掉。

2. 修改 force_recovery启动以后，mysqldump会显示Lost connection，表示出错的表是在主键索引上（如果是叶子节点，dump的时候是不会访问到的）。

3. 这种场景的坏表修复，有一种方法，是把错误的page先忽略掉。每个page有checksum，可以扫文件找到checksum不对的page，清空掉内容，初始化成空的。（另一种做法是修改MySQL的代码让MySQL忽略这个错误，而不是crash）。

4. 总之第3步的目的就是跳过那些已经错了的数据页，把表里面别的数据取出来。

5. 按照前说的，可以判断出这个错误的页是在主键上面，所以会丢几行的。

6. 找回这几行的方法，有几个地方： 这个表其他索引里面有部分字段；undolog；binlog；业务对账。 但是这些都不能保证能够100%拿回全部数据。

一般尽量用binlog，业务对账已经是属于数据库本身没辙了。

我碰到过这种情况，没有binlog的情况下，做到第5步，业务就满意了。

最后总结一下 ： 一定要备份，包括数据和binlog。

98、问题一：当mysql重启，对于innodb的启动是如何实现的，听文章说redo log会很大，比如一个G，需要全都加载一边才能算启动完成吗？

问题二：二段提交，如果在commit之前crash了，innodb是如何界定这一条数据全是丢弃还是不丢弃？因为有两种情况，一种是bin log写成功了，但是commit没成功，另一个是bin log压根就没写成功。

答：1. 从checkpoint开始到writepos结束，不需要全部4G。

2. Binlog写成功事务算成功，就提交事务；binlog没写成功就回滚。

99、文章提到如果redo log被写满了，需要停下来将数据写入磁盘。这个时候，整个数据库是处于阻塞状态吗？当有大量数据涌入的时候，会存在丢失数据的风险吗？

答：更新都会被堵住，但是不算数据丢失，因为事务根本就提交不了。

事务/数据丢失的定义应该是本应该存在的结果没了。但是这时候并没有告知客户端“成功”，也就不算“丢失”。

100、问题一：同样都是调用引擎接口， 为什么写入新行操作算执行器的， 提交事务操作就算引擎的了？具体描述如下：

update 语句时的内部流程时，2.执行器拿到引擎给的行数据，把这个值加上 1，比如原来是 N，现在就是 N+1，得到新的一行数据，再调用引擎接口写入这行新数据。

这里调用引擎接口时， 算执行器的操作，5.执行器调用引擎的提交事务接口，引擎把刚刚写入的 redo log 改成提交（commit）状态，更新完成。

这里调用引擎接口， 提交事务为什么就算引擎的操作了？ 跟上面是不是有冲突，还是因为什么原因呢?

问题二：我对redolog的存在原因的理解： redlog存在时为了数据库的内存操作的数据安全。 虽然写日志有io成本， 但是比每条sql都直接操作硬盘的成本依然还是低，所以综合来说，保证内存操作，然后日志硬盘io，最划算。

问题三：比较好奇redolog设计固定大小4g的历史原因。

答：1、其实真正做数据更新和事务维护的都是引擎。但是引擎接口是执行器来调用的，调用关系。这里只需要明确，“加1”这个操作是在server层的执行器做的就行啦。在这个流程里，逻辑计算在server, 引擎层做读写。

2、理解正确。

3、不是固定为4G哦，可以配置的，只是在现在普遍的大内存场景下，要支持稳定的写入性能，可以设置4G。

101、客户端发起commit的时候，收到commit成功是在把事务写到innodb_log_buffer_size就返回成功，还是要刷到binlog后才返回成功啊?

答：都要。而且不止写到log_buffer,还要写到redolog文件。

102、write pos 是当前记录的位置，checkpoint 是当前要擦除的位置。

请问老师，这个checkpoint 是不是就像图片一样，当写到3的时候，0已经被擦除了，并且checkpoint已经在1上面了。就是，checkpoint 是预先擦除一个文件的位置留给 write pos写?

答：对，但是并不保证能够多出一个文件，也有可能write pos追上来就接近chenckpoint,这两位置在同一个文件。

103、关于崩溃恢复 数据库目标既然是保持数据一致可能还原也可能执行 对于数据库的客户端这时候是什么状态呢 对于这些没有收到成功答复的更新操作，客户端需要一直重试吗?

答：代码里面要有确认的逻辑。

104、既然write pos和checkout都是往后推移并循环的，而且当write pos赶上checkout的时候要停下来，将checkout往后推进，是不是意味着，后续的更新请求都会等待，等checkout往后推到一定空间后，再继续执行。如果上述成立的话，在大量的更新或插入操作的时候，write pos可能会很快的追上checkout，从而再次产生等待的过程，这个是不是一个在性能上产生瓶颈。

答：是的，所以1. 要设置大一点的innodb redolog的大小 2. 用好点的磁盘。

105、有一个疑问 redo log记录 在某个数据页做了什么修改，这里数据页要怎么理解，数据查询的结果数据?

答：不是，存放数据的地方，你可以理解为按块儿存，一块儿16K。

106、文中有两处内容让我迷惑，感觉描述不一致，如下：

“具体来说，当有一条记录需要更新的时候，InnoDB 引擎就会先把记录写到 redo log（粉板）里面，并更新内存”

“3.引擎将这行新数据更新到内存中，同时将这个更新操作记录到 redo log 里面”

该过程是先把新数据更新到内存中再写到redo log里吗?

答：如果是MySQL 代码里的精确顺序，是先更新内存的。这里主要是要说明WAL机制，这两个的顺序你可以认为“同时”，会更方便理解。

107、一天一备份, 每份bin log都比较小, 数据恢复比较快, 但是每天数据库备份占用的硬盘空间会比较大, 相比每周备份几乎多了一个数量级。

答：对的，存储成本和RTO的权衡。

108、请问数据库备份是怎么做？可以通过主从数据库，将从库的复制时间设置为1天后复制的方法实现备份么?

答：这个叫延迟从库，你可以在这个库上备份。不过一般建议在一个普通从库上备份。

109、我采用mysqxtrabackup来全量和增量备份数据库,没有备份binlog，这样可以的吧?

答：看你的系统可靠性要求哈。两次增量备份之间如果机器挂了怎么解决的，要有个预案。

110、如果先写binlog 再写redolog ，出现crash时我以之前全备份加binlog进行恢复，不需要redolog了,出现不一致也没有问题啊?

答：嗯，你这么做是可以的，但是正常重启innodb 要做崩溃恢复的，它一看，没redolog, 这个事务是没有的，但是binlog里面已经有了，用你的方法事务就存在，设计上要避免使用方式的问题出现不一致的结果。

111、我这个小白不懂，字段由0变1时，在“先写redo后写bin”下，写完redo就crush，您说用binlog恢复会导致结果是0和原库不一致。但那也还没commit啊，原库不还是0，用binlog恢复出来数据库也还是0，那不是反而保证了和原库一致吗?

答：Innodb里面，redolog写完就算成功了，崩溃恢复以后这个行是1。

112、如果将innodb_flush_log_at_trx_commit参数设置为1，那么每次redo log都要被刷新到磁盘，这样效率也会比较低吧。感觉就不太符合您讲的“掌柜写粉板提高效率”例子了。

答：日志顺序写盘，比直接写数据（随机写盘）还是快的。

113、如果是redo log是直接写磁盘，为什么不把更新的数据直接写磁盘？是因为慢太多?

答：是的，顺序写比随机写快。

114、如果在非常极端的情况下，redo log被写满，而redo log涉及的事务均未提交，此时又有新的事务进来时，就要擦除redo log，这就意味着被修改的的脏页此时要被迫被flush到磁盘了，因为用来保证事务持久性的redo log就要消失了。但如若真的执行了这样的操作，数据就在被commit之前被持久化到磁盘中了。当真的遇到这样的恶劣情况时，mysql会如何处理呢，会直接报错吗？还是有什么应对的方法和策略呢?

答：这些数据在内存中是无效其他事务读不到的（读到了也放弃），同样的，即使写进磁盘，也没关系，再次读到内存以后，还是原来的逻辑。

115、请问如果写完binlog后crash，redolog还没commit，这时候会发生什么?
答：重启后崩溃恢复时提交事务。

116、上面id为某人的关于二阶段提交，也就是14/15描述貌似有点问题，preper阶段是刷redo到磁盘，commit阶段才刷binlog到磁盘，根据淘宝同学提的建议，5.7.之后某一个版本后改到了只需要确保刷binlog之前刷redo(根据lsn判断)，不知道我描述的对不对，请老师确认下!
答：是的，但是两阶段的逻辑没变。

117、如果两种log都写完成了，但是redolog没有写磁盘，物理机挂了，redolog在内存中就丢失了吧，再启动跟磁盘中的binlog就不一致，后期恢复数据的时候会有问题吧?
答：是的，这就trx_commit没设置成1的后果。

118、只有redo-log，没有undo-log，innodb应该没有办法做到crash safe，楼主可不可以把这块也稍微讲一下呢？目前有点一知半解。
答：Undo log跟数据一样，也是WAL机制写入，需要依赖redolog 恢复出来。

119、两阶段提交为了保证binlog和redolog的一致性，1 prepare阶段 2 写binlog 3 commit阶段，假如在3之前崩溃，恢复时，虽没有commit，但prepare和binlog完整，所以重启后会自动commit，以此来保证有binlog一致性。
既然能这样做，那为什么不把commit阶段去掉，恢复时，只要redolog和binlog完整就认为redolog有效，否则回滚呢？ 是因为效率还是其他的原因?
答：如果redolog是完整的，（包括了prepare和commit），就直接认为成功，不去判断binlog。

120、对于“redo log 用于保证 crash-safe 能力”这个描述，如果在写redo log的时候发生异常(比如断电)，那重启后该条更新是不是失败了？但是在这种情况下redo log、binlog和磁盘上的数据文件里都没有该条更新的记录，从数据一致性角度来看，整个MySQL系统是“安全”的。更新失败的事务可以通知客户端重新提交。不知道我这样理解对不对?

答：这时候崩溃恢复肯定是要回滚了，但是确没办法再“通知客户端”了，因为之前连接就已经断开了。只能靠客户端发起新连接去查数据确认。

121、两阶段提交

1、prepare 2、写入binlog 3、commit

如果到2后，系统奔溃，binlog没写入，事务回滚，

如果到3后，系统奔溃，binlog写入，事务自动提交，

那有个疑问在这过程中，事务回滚和提交mysql是怎么判断的?

答：只好靠恢复后，应用程序自己来查数据验证。

122、装库的时候，redo 应该设置成几个文件，以及每个文件的大小怎么设置的，应该从哪些方面去考虑呢?

答：主要看你预设多大的TPS，现在SSD机器+32G 以上内存，建议设置4个1G。

123、两阶段提交的第一阶段，将变更的记录写入redo日志，这个阶段是写到了log buffer还是log file? binlog也是一样。在第二阶段提交，binlog和redo是这个时候进行刷盘，还是目前已经完成刷盘，二阶段提交确保两者已经刷盘的情况下才会真正意义上完成commit,否则进行rollback。

答：1. 到了提交阶段，就是把logbuffer写入redolog duke。

2. 第二阶段，其实就是binlog和redolog最终提交的阶段。

124、文章写到redolog是物理日志，记录在某个数据页做了什么修改，binlog是逻辑日志，是原始逻辑。我是否可以这样理解，redolog是增加新的内容，binlog是对表中已有数据做改动。回到老师的问题，一天一备份与一周一在什么指标上更有优势，我的理解是在优化指标上更有优势，备份时间越长，之后查询建立索引上就需要花更多的时间，所以对优化指标有影响。

答：Redolog和binlog都是记录数据的改动，前者是针对数据页，后者是记录修改逻辑（这一行要怎么改）。

125、两阶段提交那个地方用了反证法，个人认为并不一定用了两阶段提交就一定可靠，写入redolog-->写binlog-->提交事务处于commit状态，这个是3步，可能在写完binlog后直接服务器断电，碰到过从库的binlog比主库还超前的状况（可能是binlog已通过网络传给从库，但是主库断电数据没落盘）。

答：这就是为什么MySQL 在崩溃恢复的时候，会在主库提交这个事务。

126、1 prepare阶段 2 写binlog 3 commit

.....

当在3之前崩溃，重启恢复：虽没有commit，但满足prepare和binlog完整，所以重启后会自动commit。备份：有binlog.一致。

疑问：虽没有commit，但满足prepare和binlog完整，所以重启后会自动commit。

--这里的binlog完整，通过什么方式或机制来判断?

答：每个事务的binlog日志都有标准的结尾的

127、问题一：

老师您回复：WAL机制里面有“已经提交但是还没落盘”的数据，binlog恢复不了这些。

这是因为一开始 binlog 就是设计为归档日志，因此恢复不了“已经提交但是还没落盘”的数据么？从图“update 语句执行流程”来看，提交前已经写 binlog 日志了，即使没落盘，这部分数据仍然是未丢失的。

问题二：

文章的例子：某天下午两点发现中午十二点有一次误删表，需要找回数据

最近一次全量备份文件 + 重放到误删表之前那一时刻，这样十二点到两点的数据是不是丢失了，恢复不了了?

答：1. 但是binlog并不知道他们没有落盘，这个是引擎内部机制。不会去重新执行。

2. 对。 一般是这样，恢复出一个临时库是误删前的。然后把里面的数据按需取出来放到线上的库里。

128、update最后一步，既修改对应redo日志的状态为commit，这个时候，系统crash，会出现什么情况?

答：重启后，崩溃恢复过程中提交该事务。

129、开启了半同步复制after_sync以后，假设一个事务在已经把binlog sync到磁盘了,在传输binlog到从库上时,主库挂了。如果这时发生了主从切换,从库是没有这个事务的.但是挂掉的主库重新启起来,由于binlog已经fsync到磁盘,虽然引擎层未commit,但是会根据binlog来恢复这个事务.这个时候就是主从不一致了,那么和after_commit那就没区别呀。请问这个事务是恢复还是回滚呢？如果是回滚,内部又是怎么判断的呢?

答：如果你设置来双MM，启动以后binlog还会传到新主库的。但是如果你设置的是单向的，那就会不一致了。

130、最近生产环境遇到一个问题，show full processlist中抓到一个update语句，根据主键更新的，时间是30s，state为updating，但是slow log中没有记录，我知道slow log中只会记录执行时间>long_query_time的查询，不包括lock_time。我的问题是updating过程中做了哪些事情，是有加锁的动作吗?
答：那看来就是被行锁锁住了。

131、据之前的理解，应该是commit完成后才会写入binlog到binlog cache。看您说的update过程是先落盘binlog再commit。假设这样的话，在主从复制中，在主库事务未提交的情况下，binlog就已经被同步到从库了。
答：不是哦，我们说的都是“提交过程”哦。在事务“执行计期间”，binlog 没有写盘，如你所说，这时候在binlog cache，没往备库发的。

132、innodb_flush_log_at_trx_commit 和 sync_binlog 都设置成1，那么都是每个事务结束直接写磁盘，那是不是就浪费了WAL的作用，这样每一次的更新操作都需要写进磁盘，然后磁盘也要找到对应的那条记录，然后再更新，整个过程 IO 成本、查找成本都很高。
答：这个是为了保证数据安全必须的成本，但是WAL还是有用的。

133、执行一条Update 语句后，马上又执行一条 select * from table limit 10。如果刚刚update的记录，还没持久化到磁盘中，而偏偏这个时候的查询条件，又包含了刚刚update的记录。
那么这个时候，是从日志中获取刚刚update的最新结果，还是说，先把日志中的记录先写磁盘，再返回最新结果?
答：都不是，直接读内存。

134、执行最后commit，innodb的commit具体是做了哪些操作呀？移动checkpoint指针吗?
答：不是，Redolog上打上commit 标签，写盘。

135、在一个表上有更新的时候，跟这个表有关的查询缓存会失效，所以这条语句就会把表 T 上所有缓存结果都清空。如果做了主从，把所有的读操作都指到从库上进行，主库update表，

从库同步更新表数据，这个时候从库会不会有清空查询缓存的操作?

答：会，数据同步到从库其实也是在从库做更新，相同逻辑。

136、如果把redolog的innodb_flush_log_at_trx_commit设置为1，不就相当于每一次都要提交到磁盘，就用不到他的记录到小黑板上的功能（先都写到日志，找一个合适的时间记录到磁盘），不是把他的优势给弄丢了吗?

答：没丢，顺序写比随机写的成本低。

137、我之前是做运维的，通过binlog恢复误操作的数据，但是实际上，我们会后知后觉，误删除一段时间了，才发现误删除，此时，我把之前误删除的binlog导入，再把误删除之后binlog导入，会出现问题，比如主键冲突，而且binlog导数据，不同模式下时间也有不同，但是一般都是row模式，时间还是很久，有没什么方式，时间短且数据一致性强的方式?

答：其实恢复数据只能恢复到误删之前到一刻，误删之后的，不能只靠binlog来做，因为业务逻辑可能因为误删操作的行为，插入了逻辑错误的语句，所以之后的，跟业务一起，从业务快速补数据的。只靠binlog补出来的往往不完整。

138、我发现ibdata0文件无限膨胀并且不分裂初ibdata1文件，具体会是什么原因导致，有没什么有效的方式解决?

答：就是回滚段导致的。

Ibdata不会帮你自动生成ibdata1,会变大是因为有事务没提交，导致别的事务更新的回滚段不能回收，最终把文件撑大。查一下数据库里的长事务，kill掉。不过这样只能避免继续膨胀，现有的空间不能回收了，实在要回收只能靠把数据逻辑导入另外一个干净的库。

139、如果有一个备份库的时间点，最接近误删除的时间点，但这个备份库的时间比误删除的晚，能不能用binlog倒过来回溯?

答：binlog都保留了、并且是row格式、并且误删操作是“delete”才行。

140、您说MySQL 使用WAL，先写日志再写磁盘。请教一个问题，执行一条Update 语句后，马上又执行一条 select * from table limit 10。

如果刚刚update的记录，还没持久化到磁盘中，而偏偏这个时候的查询条件，又包含了刚刚update的记录。那么这个时候，是从日志中获取刚刚update的最新结果，还是说，先把日志中的记录先写磁盘，再返回最新结果？“

答：都不是，直接从内存读走，内存里可是有修改的最新的结果哦。

141、写入新行、新行更新到内存、写入redolog写入binlog。这4个操作应该是原子性的吧?

这边的写内存，写redolog，写binlog是怎么保证原子性的呢?

答：只是逻辑上原子，实现上不会加这么大一个锁哈。

142、写内存的时候查询到的数据是不是最新的数据？如果写入内存成功，redolog，binlog失败了，会不会造成读到这个非法数据呢?

答：不会，事务没提交呢，MVCC会让别的线程看不到这个内存的修改的。

143、目前更新数据的过程是在确保数据能更新成功的情况下就写binlog了，然后事务才提交commit，如果通过binlog来同步数据，当接收到binlog后还需要获取其它数据进行计算，此时立刻去读数据库的会发现有时候读不到刚更新的这条数据，这个有没有什么办法避免?

答：这个是读写分离常见的问题...只能做好区分，对延迟敏感的请求，需要到主库上去查。

144、接前一个问题

原问题：更新数据的过程是在确保数据能成功更新的情况下就写binlog，然后事务才提交commit，通过binlog来同步数据，当接收到binlog后还需要获取其它数据进行计算，此时立刻去读数据库会发现读不到刚更新的数据。

您的回复：做好区分，对延迟敏感的请求，需要去主库上查询。

感谢老师的回复，之前表达的不完善。这里就是接收到主库发出的binlog后，立刻去主库查询数据，有时查不到此条binlog对应的更新内容，当sleep 0.5s(或更久)再去获取便能取到正确的数据，初步怀疑是binlog发送时主库的事务尚未commit导致的，不知道有没有什么好的解决方法。

答：确实有可能，binlog确实先发了，你还有这种场景啊，

比较少见到这种方案哈，我现在能想起来的就是轮询了。

当然还有可以去查解析出binlog里面的线程id和时间戳，然后去查select * from information_schema.innodb_trx where trx_mysql_thread_id=$tid and trx_started < $timestamp;

必须返回空才行，本质上还是轮询就是了，不过，都拿到binlog了，看看能不能考虑设置成row格式，直接解析更更好些。

145、关于先写redo log再写binlog的分析有些不理解。假设redo log写完后binlog还没写完，这时数据库出现意外宕机，修改数据的物理记录虽然记录到redo log，但是事务还没提交，我理解未提交意味着数据没有最终落盘，您说的c=1是因为innodb的崩溃恢复机制能把未提交的事务恢复到提交状态么?

答：如果redo没落盘，那会回滚的。

146、关于两阶段提交有个问题想请教一下。写redo log处于prepare阶段后，如果写binlog失败是会回滚redo log的记录吗?

答：不用回滚日志，只需要打上“rollback”标签。

147、【update 语句执行流程】这张图的update更新流程是不是有点问题啊，既然是WAL了，为什么是将这一行的c值加1，写入新行，然后才是写入redo log呢?

答：WAL是指在“数据真正写入对应的磁盘数据页“ 之前写日志。

148、任何事务如果在回滚的时候系统异常重启了或者回滚操作失败了， 那是不是就导致数据状态可能不一致？ 这样情况会发生吗？ 如果发生了怎么避免?

答：没问题的。回滚中系统异常重启，重启后会继续回滚，是通过扫描redo log状态来发现的。

149、WAL可以保证之前提交的记录不会丢失。所以是不是只能是checkpoint过的记录不会丢失呢？那没有checkpoint过的数据但是已经write到redo log里的，在数据库发生问题了，应该都丢失了吧。如果发生这样的问题，如何恢复呢?

答：不会呀，崩溃恢复的时候，会从checkpoint开始往后主动去刷数据的，没有checkpoint过的并不会丢失，只是崩溃恢复的时候花点时间。

150、问题一：如果更新语句涉及到索引，索引是什么时候被更新的，是否跟数据一样，先更新内存，然后再持久化到磁盘?

问题二：如果事务失败，回滚的时候，是否也会写redo log和binlog。回滚涉及到索引的更新话，索引如何回滚的呢?

答：1. 索引也是数据的一部分哈，一样的策略。

2. 会写redolog，打上回滚标签，binlog就直接丢弃啦。回滚都是把没用的内存版本删掉。

151、下面这段话是不是矛盾了：

既然redo log 本身就配置了一组4个文件,每个文件的大小是 1GB，那么这块“粉板”总共就可以记录 4GB 的操作,这个时候不是持久化到磁盘吗?

为什么还要设置innodb_flush_log_at_trx_commit 这个参数设置成 1 的时候，表示每次事务的redo log 都直接持久化到磁盘。

答：innodb_flush_log_at_trx_commit =1 的时候，才是确保redo log里的数据是持久化到磁盘的；如果是innodb_flush_log_at_trx_commit=2， 其实只是“写到redo log文件，但是没有持久化(fsync)”。

152、文中的redo log其实是redo+undo log吧，因为记录更新前和更新后的值都记录了的。

答：这里主要还是讲“数据更新逻辑”哈， undo log是有的。

153、redo log处在commit状态就是已经在check point了吗?

答：没有，checkpoint是后台线程追上来的。

154、您说“redo log 可以是4组文件，每个文件大小1G”，从大并发update性能方面考虑，这个文件大小及几组一般怎样配置比较合适?

答：我觉得如果是1T以上的磁盘&不小于64G的内存，都可以按照4*1G来配置redo log。

155、不太理解为何要用redo.log, 前面举的那个记账的例子我可以理解, 但是binlog只是记录变化, 不是应该就是不停的写下去吗, 就不像是账本需要找到人, 然后对应的计入这个人的账目..... 所以我觉得binlog就是那个粉板了...因为我是初学者, 所以可能很多基础知识有缺失, 老师直接指出搜索哪些相关知识可以帮助我理解这个问题也可以, 我可以自己去看。

答：redo log和binlog其实都是粉板的概念，只是redo log是专门给innodb用的，而且在MySQL的实现里面，是用redo log来达到“崩溃恢复”这个作用的，binlog一直的定位是“归档日志”。

156、文章中写到：write pos 和 checkpoint 之前是“粉板”上还空着的部分，我的疑问是checkpoint到write pos 这块内容的大小是如何定的，是不是一直是固定的大小?

答：不是固定的大小哦。整个文件系统空间是固定大小的（根据系统配置），write pos是“当前写入的位置”；checkpoint是“已经将更新持久化到数据的位置”。所以如果系统TPS短时间内增大，这两个距离就会增大；反之如果系统空闲的时候，这两个距离就会减小。

157、日志写刷盘跟修改库的数据这两者什么关系。也就是粉板的数据什么时候算总账，在没算账之前如何能查到最新的值?

答：内存里是正确的值，直接从内存读走就行；日志和数据的关系，是日志记录了数据从状态1到状态2的过程。这个过程持久化了，状态1也是已经持久化到磁盘的，那么状态2就可以不着急一定要写到磁盘。这个是WAL机制的逻辑。

158、innodb_flush_log_at_trx_commit 设置 1，并且sync_binlog也设置为1，就表示redo log和bin log都是同步刷盘的，而库的数据先写内存，后写磁盘。库的数据保存在内存是安全，因为log中有全量数据可以恢复，请问库的数据何时刷盘，是依赖操作系统的pdflush还是mysql主动调用刷盘的系统调用?

答：“请问库的数据何时刷盘”----两种情况，redo log在推进checkpoint的时候，和脏页被淘汰出内存的时候。

159、两阶段提交是原子性的吧?

答：不是，如果保证原子性，就不用两阶段啦。

160、两阶段提交的重点是不是应该是redo log bin log 的刷盘，而非写日志?

答：都是重点，不过如果说耗时，确实是刷盘才是主要的耗时的点。

161、更新语句应该是在分析器以后才能确定表名的吧。那它怎么在这之前去清空该表的缓存的呢?

答：在之前是查缓存，执行语句以后才更新（失效）缓存。

162、如果不分为两个阶段，而把写redo和binlog放在一个事务里面，如果写redo失败，则失败回滚，binlog写失败，则同样回滚，只有两个一同成功才算真正成功，这样也可以保证数据一致性，为什么一定需要两阶段呢?

答：redolog是innodb里面的，如果按照你说的这种方式，redolog提交后，在引擎内就得算是事务完成了（因为没有第二阶段让事务“真正完成”）这样binlog失败的时候，想让引擎回滚就回滚不了了。

163、关于redo 和binlog写入完成但是不commit的情况，此时我断开连接，这个事务应该是失败的。按照上面的说法，恢复的时候binlog这一条记录存在了，所以这条记录应该存在。

这个观点有问题还是mysql的bug。“此时我断开连接，这个事务应该是失败的。”

----这里有问题，此时你主动断开连接，这个事务就是“异常”，而不是“失败”。

答：异常的事务，状态可能是提交，也可能是为提交，所以记录存在，是合理状态的一种。

164、如果提交事物失败怎么处理，修改redo log状态失败，怎么回滚binlog?

答：binlog这时候还没写到磁盘上的，直接删掉binlog cache李忠的内容就可以。

165、ddl语句也会写入redo log吗？而且redo log是物理日志，同样也要I/O开销吧？另外实在看不懂redo log里面到底存的是什么，不是更新后的值吗?

答：1. ddl语句会写redo log的。

2. 也要io开销，不过因为是顺序写，开销小。

3. 不是更新后的值，而是“数据页的内容是怎么变化的”。

166、看到留言里面说既然恢复的时候可以通过事务 id 去检查 binlog 的完整性，就不需要两阶段提交了。个人认为这可能是一种优化，如果每次都去检查 binlog 就太低效率了，而加上两阶段提交的步骤，进行恢复的时候只有当 redo log 没有提交的时候需要去检查 binlog，而一般在commit阶段失败的概率都很小，所以需要检查binlog完整性的情况也比较少。不知道这样理解正确否?

答：虽然两阶段不是因为这个原因设计的，不过有了两阶段，确保了，只要redo log完整就可以直接认为事务有效，确实减少了崩溃恢复的时间。

167、redo log等系统不忙更新数据的时候，也是需要先从磁盘读出数据到内存，然后用redolog里面的记录更新内存，最后再一起写入磁盘吗?

答：对，这个就是“推进checkpoint”的过程，当然也有可能redo log要更新的那个数据页刚好在内存，就跳过了你说的第一步。

168、在redo log commit的时候，想我一下，是追加写呢，还是找到prepare的地方进行修改呢?

答：prepare写日志的最后一个512字节，会在commit的时候被改掉。

169、redo log为什么要设计成环形的？环形缓冲区的设计又有什么好处?

答：我觉得最重要的好处，是可以避免新建、删除文件带来的抖动。

170、是不是磁盘中的数据也有事务id，这样如果写完redolog，内存中的数据还没刷到磁盘，系统奔溃重启后可以用redolog来检查当前数据是不是最新的用来恢复。

答：你的理解是对的， trx_id是数据的一个字段，会持久化到磁盘。

171、参考书《数据库系统实现》中，将redo log的记录格式描述为<t,表，字段，值>，即幂等。这里是否和您的操作记录说法有冲突?

答：有表空间id，勉强可以跟“表”对应起来，不过没有 “字段”的。

172、缓存失效是在刷新内存之后呢，还是在commit成功之后呢?

答：你可以用事务验证一下。

先构造一个querycache然后启动一个事务

begin;

update ....

select ... //看看这里还能不能用上查询缓存

Commit

173、innodb_flush_log_at_trx_commit 和sync_binlog 两个参数不同的值对mysql 两阶段提交的影响。

答：关了binlog没有两阶段提交，这两个参数只是控制写潘策略，两阶段的每个阶段执行步骤都还在的。

174、如果在写入redo log之前就异常重启了，数据还是会丢失的。

答：那样不算丢失，因为并没有告诉用户“成功”。

175、之前好像说redo log是prepare状态，当写完bin log后，才会调用commit接口，让redo log变成commit，那么如果bin log写完后，调用commit的时候出异常了，是不是bin log和redo log的记录不一致了?

答：不会，redo 如果没有commit，这个事务就回滚。

176、这个程序是什么时候获取到返回值的呢 比如我update一条正常来说程序是返回1 是在commit之后获取到的吗？如果是commit之前返回给程序 然后crash了 会给用户带来错觉吧?

答：顺序上是这一样的：

数据库里面写日志，此时算事务提交；

返回响应包给客户端；

客户端收到响应包

客户端解析响应包里面的affacted_rows，得到1.

所以没有你说的这种情况哈。

177、问题一：redo log写入内存的数据，是跟各个事务绑定的。假设 ID为2的数据c=1。

执行更新语句 update T set c=c+1 where ID=2; 在更新语句执行过程中，写入了redo log，但未两阶段提交。此时，是不是只有当前事务才能读取到c=2?

问题二：其他留言下“即使写进磁盘，也没关系，再次读到内存以后，还是原来的逻辑”，老师能否扩展下这个解答。是不是指，即便写到磁盘，磁盘的数据也有事务版本信息来保障?

答：1. 是的，这个事务还没有提交，没有提交的事务的更新，对其他事务是不可见的；

2. 是的，一个数据行的trx_id是一个数据的一部分，写磁盘的时候也是带着的。

178、看上去数据库正常关闭会清空redolog?

答：不会。就算事务放弃了，redolog也留着，redolog循环写的，下次循环回来就覆盖掉了。

179、既然write pos 会赶上checkout的时候对吧，那我的问题是当粉版满了，暂停后做一部分归档 ，那么是不应该有阈值，比如达到90%的时候，启动一个线程，来做归档，这样就不用暂停了吧?

答：一直有一个线程在归档的。

180、两种交叉写的方式依然会出问题啊，假如redolog在没有commit的时候，系统宕机以后，回复的数据也还是不一样的啊?

答：redo如果没有commit，那系统宕机恢复后，事务是会回滚的。其实这样是没问题的。因为还没有commit之前，你用别的线程去查，也查不到数据；崩溃恢复后，也查不到数据。这样就是一致的。

181、更新redo log 和binlog log 成功后 。InnoDB 在空闲的时候才把日志文件写入到数据文件。如果查询的时候不使用缓存，日志文件还没有来得及同步到数据文件，那么怎么保证查询到的数据文件与日志文件一致的呢?

答：这时候查询的时候，直接从内存返回就可以了。内存里是正确的数据，磁盘上数据文件其实还是旧数据，不过因为请求查不到，所以没关系。

182、怎样让数据库恢复到半个月内任意一秒的状态？这个是不是还需要redo log啊？为啥我感觉仅仅binlog不够。
答：binlog记录了所有操作，操作记录里的时间也是精确到秒的。

183、#比如突然停业几天，恢复生意后依然可以通过账本和粉板上的数据明确赊账账目。#
对于这一句话,不太清楚.如果崩溃之后,对于没有提交的事务来说,粉板上的记录难道不是不存在了吗?
答：没有提交的意思是，客人买一半（比如刚点菜完还没上菜），突然失火都跑了，这样是不应该记录到粉板上的。

184、今天面试遇到了一个问题，就是高并发的插入下，有哪些情况下会导致数据库插入变慢。对于这篇提到的redo log和binlog有关系吗?
答：这两个会影响，不过在高并发的问题里，锁的影响可能会大些。

185、python里面，autocommit默认是0的，这种情况下，一条select语句如果不写begin，是否需要，commit?
答：不commit如果连接保持着，就可能变成长事务哦。

186、作为开发负责人，为开发人员制定规范，避免Long running transcation，
作为DBA，定时检查老师提到的information_schema.innodb_trx表，将超过阈值的transaction终止掉，这一点看到其他同学说能够定时执行+邮件通知，不知道该怎么具体实现?
答：Kill query + thread_id 可以要求一个语句终止执行。

187、长事务回滚也会写日志吗？是写redolog？之前看理解是需要写日志，当日志空间不够，数据库就会挂起不提供服务。
答：回滚就不写到redolog了， 嗯，空间满了的话，binlog写不下去，事务就执行不下去。

188、事务的回滚日志难道直接写到磁盘吗？我猜想应该是先存在内存然后在某一个时间写到磁盘，那么问题又来了？这个时间是什么时候呢？如果猜想正确那是不是意味着长事务还会导致短

时间内内存占用较大?

答：回滚日志和数据的写入策略一样（WAL），但是不一定不写盘，比如checkpoint 推进过程就可能把undolog的内容从内存写到日志文件，所以不会因为undolog导致占用内存很大的情况。

189、怎么理解事务是基于连接的这句话？如果我在多线程中共用一个连接，并执行事务会有什么后果?

答：多线程不能同时用一个连接，你说的共同，是不是指连接池复用？这时候对一个连接是“先后使用”，不是“同时使用”。

190、回滚语句就是传说中的undo吧?

答：不是。undo叫做 回滚段，回滚语句就是”rollback”这个命令。

191、在更新语句的时候 会两阶段提交形式写入 redo log 和 bin log . 那记录旧版本的 undo log 是这么时候写入的呢?

答：更早之前了，写redolog和binlog是提交过程。在此之前更新数据的时候就已经写了undolog。

192、在事务开启前是不是可以把需要查询的数据查询出来，做好逻辑运算，只有写数据库阶段才开启事务，是不是可以避免长事务了?

答：要注意的是，这时候查询没在事务里面，所以等开始事务进行更新的时候，可能数据已经变了。如果业务逻辑可以接受这么做，那这样是很好的方法。也是我比较建议的业务优化方向。

193、这个read-view里存储什么范围的数据呢？rr模式下，事务begin就创建一个空的read-view，然后语句执行后，把符合条件的记录存在这个视图中，commit后再释放这个视图，这么理解对吗?

答：不是begin,而是begin之后的第一个语句执行开始的时候。

194、mysql上千万表数据，如何快速删除自增，也就是取消自增?

答：如果只是去掉自增属性，你可以直接忽略它，每次插入数据的时候自己指定值就可以了。

195、如果一个mtr更改多个数据页，会对整个b+树上锁？还是只锁住页所在的分支呢?

答：如果是整个B+树加锁就是表锁啦 InnoDB是支持行锁的哦。

196、关于长事务，我的理解是这样，因为会存在大段的回滚日志，而回滚日志可能其他未提交的事务也有关联，所以当这个长事务提交之后，这些回滚日志也会被保留，那当其他的事务提交后，这个回滚日志不会被删除吗?

答：会被复用，但是在ibdata中占用的空间恢复不回去了。

197、为什么回滚段删除了，硬盘空间大小没变大呢?

答：因为它占了空间就没法删回去了。

198、如果有主从配置 是不是就不需要备份了?

答：要，错误执行的drop table 也会同步给备库的。

199、当隔离级别是“读未提交“的时候，当读数据读到了一个未提交事务的值，比如文章中的V1=2，但事务B回滚了，实际上V1的值仍然是1，可事务A已经查到结果是2，这样结果不就不一致了？ 该如何解决?

答：解决方法就是不用这个隔离级别，读未提交也称为“脏读”，就是你说的这个场景得名的。

200、我们保证断电数据不丢失，可以在事务每次commit时候直接写磁盘落地，当然这性能太差，我们采取写日志的形式，日志是顺序写的，性能会高些，那这样的话我觉得可以直接使用binlog做断电后的恢复，每次写入binlog后再做commit，不写binlog当他没提交，这样理论上和直接写数据感觉是一样的，为什么还是要是用redo log呢?

答：“当他没提交”表示要回滚对吧，可是没有redolog, 回滚日志也丢了，回滚不了了。

201、您在文中提到关于“串行化”有”后访问的事务必须等前一个事务执行完成，才能继续执行。”，我想问的是两个事务A和B它们有可能同时启动吗？如果可以同时启动，是不是说先进行操作(读或写)的事务就是前面提到的“前一个事务”，后进行操作的事务就是“后访问的事务”？还是说两个事务不会同时启动，先启动的就是“前一个事务”，后启动的就是“后访问的事务”?

答：启动事务一定有“先后”的，每个事务要加入系统统一管理，加入的时候加锁，不会精确地“同时”。

202、如何理解“即使长事务最终被提交了，回滚段被清理，文件也不会变小”？为什么清理了文件大小还不变呢?

答：因为默认回滚段是在一个大文件里，和数据字段放一起。空间不够文件膨胀，空间逻辑删除，文件不能随便删除。

203、开始以为自己懂了，然后想起参数 autocommit自动提交 后就又迷糊了，事物隔离级别和自动提交这俩是什么关系呢？这俩参数会打架吗?

答：自动提交只是把事务提交掉，提交结束事务。事务隔离级别是事务运行期间能看到的什么数据。这俩不会“打架”呀。

204、您提到：set autocommit=0，这个命令会将这个线程的自动提交关掉……（中间省略），如果是长连接，就导致了意外的长事务。

我的问题： 这个命令是针对这个线程有效，还是针对这个连接有效？如果这个线程A结束，而连接没有关闭，另外一个线程B继续使用该连接，这个时候这个事务有没有自动提交呢？谢谢！

答：连接有效。不过你这个问题要情况。如果你说的是MySQL 线程池，MySQL 在复用的时候会重新初始化，那线程和连接其实一个意思；如果是说中间件(proxy)的连接复用，那就要看这个中间件怎么做的。

205、提到的一天一备，指的是自己手动备份一份新的吗，还是可以设置时间点，系统自动备份?

如果可以回到一个月之前的数据，按一天一备来算，是否要保存30份binlog文件?

答：这个需要外部系统，MySQL 自己做不了的哈。

是的，30天的日志，不一定是30个哈，一天可能多个，也可能好几天一个。

206、现在有另外一个事务正在将 4 改成 5，这个事务跟 read-view A、B、C 对应的事务是不会冲突的。我想问下，假如现在回滚到事物1最开始中的值1，这是另一个事物将值改为5，那么现在这条记录中最终的值是1还是5呢?

答：这俩应该是行锁了，不存在“同时修改”的情况。

207、在业务系统中，往往会存在数据库的并发操作，比如对同一个行数据的update操作。我认为只要数据库的隔离级别在 串行化 之上（可重复读或者读提交），那么要防止这种并发，往往

是在业务方进行 一锁，二判，三更新的这种操作。如果是串行化的隔离级别，那就不需要这么麻烦的操作了。

答：确实是可以这么做，就是并发行不够好。对于不冲突的，额外加锁也是有额外消耗，需要看业务冲突率。冲突率高就用你说的这种方法，低低就用类似CAS的思想。

208、view A 拿到的值，直接就是1吧？不需要从当前值回滚得到吧？

答：1 不是物理存在的，是算出来的。

209、问题一：undo log是在什么时候写入的呢，前面讲语句更新的流程中没有提到undo log，是它的写入结果对更新操作没有影响吗？

问题二：“当系统中没有比某一回滚日志更早的read-view时，回滚日志会被删除”。这个操作是定时的执行的吗？是不是如果某个数据库在一定时间内完全处于空闲状态，undo log就被删除完了？

答：1. 更新过程一边更新一边生成undo log。

2. 是后台，有定时的逻辑。如果空闲很久，确实有可能删除完了。

210、在可重复读的隔离级别下，两个事务都对同一行进行读出，事务一把该行某字段的值更改并提交了。事务二还在用该字段旧值做业务逻辑，这样不会有问题吗？

答：得看业务逻辑，有的业务就需要这个特点… 如果有影响的，就要有对应的策略处理。改隔离级别、加锁读啥的。

211、问题：set autocommit=1，自动提交模式下，是不是每个sql都会自动提交，最后的commit语句是不是就没用了；如果把commit改成rollback，是不是也执行不了回滚操作了？

答：自动提交的意思是“在没有begin或start transaction 的时候”，自动提交。

212、引用文章原文：

-“在MySQL 5.5及以前的版本，回滚日志是跟数据字典一起放在ibdata文件里的，即使长事务最终提交，回滚段被清理，文件也不会变小。”这里不是说“回滚段被清理了”，但为什么文件不会变小？

答：这个“清理”的意思是“逻辑上这些文件位置可以复用”，但是并没有删除文件，也没有把文件变小。

213、如果每一个select都是一个完整的事务，那为什么innodb_trx没有数据呢，我是在本机上玩的，按理说我在本地数据库玩过很多，innodb_trx不应该没数据呀。

答：innodb_trx 显示的是“当前正在执行的事务”如果语句执行完成了，你在innodb_trx是看不到的哈。

214、文中示例提出“同时你会发现，即使现在有另外一个事务正在将 4 改成 5，这个事务跟 read-view A、B、C 对应的事务是不会冲突的。”，如果这个时候将数据改为4的事务1回滚了，另一个将4改为5的事务2如何处理?

答：不会出现这种情况。”将数据改为4的事务1回滚了“ 表示这个事务还没提交，那么就有行锁，“另一个将4改为5的事务2” 这个就会锁进入等待，不会让他执行的。

215、老数据库可以设置事务超时时间吗，一般会不会设置?

答：没有直接的“事务时间”，有“语句执行时间”和“等待时间”，线上比较重要的业务，一般都要设置。

216、串行化隔离级别的锁是什么粒度的？合理的做法应该是只在读写记录时加锁，但从结果来看，是对整个事务加锁。如果事务A和事务B不按相同顺序都访问了多个row，会不会出现死锁?

答：1. 串行化隔离级别的锁是事务级别，既然是串行化，就是保证一个事务结束前，其他事务不能访问它锁住的资源；

2. 会

217、如何修改 sync_binlog 这个参数？我用的是 mysql 8.0，在 my.ini 中找到了 innodb_flush_log_at_trx_commit 参数（默认值为1），但是没有 sync_binlog 参数，是用其他参数代替了吗?

答：没有啊。不是每个参数都会默认写在my.ini里的，没写的时候就用默认值，你可以在innodb_flush_log_at_trx_commit 的下一行加上sync_binlog=1就是设置上了。

218、既然读提交每次都要新建一个视图，而可重复读只建立一次，那么为什么读提交的效率要比可重复读高呢？？

答：建立视图没什么成本的，就是拷贝一个事务数组；所以性能的差异不是体现在这里；

一般我们说可重复的效率相对的低（其实也还好，不会低多少），主要还是因为可重复读的锁范围可能更大（有gap lock），锁时间更长（事务结束才释放），影响并发度。

219、什么是长事务?

答：执行完语句但是一直不断开连接。

220、“读提交”隔离级别，视图在每个SQL语句开始执行时创建。这句话结合到文中示例，按照我的理解，事务A的“查询得到值V1”会创建新视图，但是事务B的“将1改成2”已经执行了，那么事务A得到的V1应为2才对呀。我觉着是否改成“读提交隔离级别，视图在有其他事务提交后，自身事务中有SQL语句开始执行时会创建”，会更准确些?

答：不是的，跟事务是否提交无关，这个事务自己会创建事务视图。

221、根据您的经验，什么场景下用RC，什么场景下用RR，能否举些场景出来?

答：业务场景里面需要明确的“可重复读”的能力，比如我见过的一些金融类的业务里面需要。

大多数时候可以优先考虑 row 格式 + RC。

222、备份是Mysql自动为我们做的吗？还是需要做什么操作?

答：不是，需要外部的备份系统。

223、事务的acid特性中，具有隔离性，那为什么脏读中，一个事务可以读到另一个事务更新但未提交的数据呢？还有就是为什么会有事务隔离级别呢?

答：acid里面说的隔离性是指RC或者RR哈，事务隔离还是很必要的吧，脏读的话，很多业务逻辑都会出问题。

224、假如默认的mysql数据库已经有隔离级别了，为什么还要开启事务呢？没有开启事务，实际开发中会怎么样?

答：多个语句要作为一个原子逻辑的时候。

225、一条单独insert语句构成一个事务吗?

答：如果没有显式的启动事务，一条单独insert语句构成一个事务的。

226、不知道还能不能被看到，对于作者的两个事务的例子，如在事务A提交前有一个根据查询到的v2值更新另外一个表的值，如果采用rc,那么这个v2就有可能变化了，导致业务执行结果
答：不一致。可以通过在事务A中增加getid for update来解决吧？但是我们在写代码时这种情况大多是没有这么去实现的，是会有问题的吧?
如果业务逻辑有这种依赖，就是需要用for update去做的；没这么写就会出现bug。

227、为什么说set autocommit =0的时候，会产生长事务？虽然不能自动提交，但是每次执行几个sql就commit不就可以了吗，如果它会产生长事务，那么set autocommit = 1配合显示启动事务也会长事务，一直不commit，执行100个sql再commit，这样不也是会产生长事务么?
答：这样是没问题的，但是很容易忘记，我经常碰到的情况是，大家以为select就不是事务，在autocommit=0的模式下，执行一个select，然后连接就那么放着。

228、意味着如果你只执行一个 select 语句，这个事务就启动了，而且并不会自动提交。这个事务持续存在直到你主动执行 commit。 请问一下为什么一个Select语句会自动开启事务呢？Oracle从来不会这样的。除非DML 语句，以及Select for update。我在5.7执行一个Select后再事务表并没该语的事务。
答：autocommit=0就会。

229、如果事务执行了一半后失败了，会自动回滚，那回滚就一定成功吗，如果回滚也失败了怎么办?
答：回滚失败一种常见情况是回滚一般crash了， 重启以后继续回滚。

230、如果不begin，那么select语句会开启一个事务吗?
答：如果select的是一个innodb表，会的。

231、在commit一个trx后开始一个begin，万一这个thread挂掉了怎么办?
答：前面提交的事务就提交了，新的事务没有走到提交阶段，就回滚。

232、关于可重复读隔离级别下，一致性视图的创建时间有点疑惑，文中说的在事务启动时创建，但我实验的结果是在第一次查询语句执行时创建的。
答：这个是一致的哈，事务启动就是在“第一次执行查询语句”的时候哦。

233、底层数据结构在页内是逻辑有序的吗?

答：物理上无序的，并且采用双向指针，页内逻辑有序，这个很精确。物理上，顺序插入的数据页连续的概率大（但也不是精确的），非顺序写入的索引基本可以认为物理无序的。

234、“N叉树”的N值在MySQL中是可以被人工调整的么？曾经面试被问到过这问题，当时就懵逼了...

答：面试中题面越简单的问题越暗藏凶险，可见一斑…

可以按照调整key的大小的思路来说；如果你能指出来5.6以后可以通过page大小来间接控制应该能加分吧，面试回答不能太精减，计算方法、前缀索引什么的一起上。

235、如果磁盘中的主键索引已经存储了这个表的全部数据的话，那常说的没走索引是遍历整个B+树还是其他地方还有整个表的数据呢?

答：就是遍历这个主键索引的意思。

236、索引本身在mysql里又是用什么数据结构管理的，内存里会有副本吗?

答：内存是一大片buffer，内存的数据内容跟磁盘一样的。

237、没有主键的表，有一个普通索引。怎么回表?

答：没有主键的表，innodb会给默认创建一个Rowid做主键。

238、1、什么情况下创建索引才有意义？有哪些限制？比如字段长度

2、如何查看索引占用多少空间?

3、查看索引数的结构，比如多少个层，多少节点?

4、如何查看索引的利用率。比如我创建了一个索引，是否可以有记录这个索引被调用了多少次?

答：1. 有这个索引带来的查询收益，大于维护索引的代价，就该建 对于可能变成大表的表，实际上如果不建索引会导致全表扫描，这个索引就是必须的。

2. 可以估算出来的，根据表的行数和索引的定义。

3. 跟 2 一 样 。 如 果 要 精 确 的 ， 就 要 解 数 据 文 件 ， 这 个 工 具 可 以 看 看 https://github.com/jeremycole/innodb_diagrams

4. performance_schema.table_io_waits_summary_by_index_usage能看到一些信息。

239、目前我们在初级阶段设置主键和索引一般就是id，也有使用到例如手机号或者身份证号，银行卡号等作为辅助索引，因为数据量小，几乎都没感觉到有什么区别，通过老师对哈希表，N叉树的点播，课下再去补充了相关的知识点，对一张表的执行逻辑和更快节约时间有了进一步认识，对于优化数据库有用，只不过重建这一块还是没懂，我们操作数据库行为时通常没写重建这个动作啊。

答：因为我们说不论是删除数据，还是增删过程都有可能导致数据页的空间利用率降低，所以有重建需求哦。

240、 关于数据库主键的建立文章中提到一般是自增较好,我的理解是主要是索引采用了树的结构,主要是索引列的有序性,如果用uuid或其它string类型做主键,是否无法进行索引优化呢,或者说索引会失效呢,分布式系统中是否还是应该采用类似于snowflake来保证主键唯一的同时又具备索引优化的条件,而不应该使用uuid呢,之前的一个系统中,在存储订单的场景中,使用了订单表order和订单明细表order_detail,order_detail表通过关联order表的主键order_id,order表和order_detail表均为uuid生成,这样当数据量很大的时候(目前已经order_detail表有200多万条记录,order表大于10万记录,且不断递增),是否无法进行相关索引优化提高查询效率呢,主要报表统计时查询很慢,订单明细表是否应该通过订单号order_number(长整型数字)来关联从而索引列建在订单号上呢?小白求教,感谢

答：不是的，即使是随机顺序插入的索引也是，查询也是能用上，也是能提升查询性能的。 只是插入数据的时候维护代价比自增主键大。

241、假如说我的查询都是根据身份证号查询的话,如果身份证号是主键索引,不就不用回表了嘛,这样性能会更高吧?

答：是，但是身份证长19位，如果你还有别的索引，比如按照名字查、按照城市查这种，就要考虑主键值在其他索引的占用空间。还有，随机插入速度慢些。

242、问题一：如果插入的数据是在主键树叶子结点的中间，后面的所有页如果都是满的状态，是不是会造成后面的每一页都会去进行页分裂操作，直到最后一个页申请新页移过去最后一个值?

问题二：之前看到过说是插入数据如果是在某个数据满了页的首尾，为了减少数据移动和页分裂，会先去前后两个页看看是否满了，如果没满会先将数据放到前后两个页上，不知道是不是有这种情况?

答：1. 不会不会，只会分裂它要写入的那个页面。每个页面之间是用指针串的，改指针就好了，不需要“后面的全部挪动。

2. 对，减为了增加空间利用率。

243、在 InnoDB 中，表都是根据主键顺序以索引的形式存放的，这种存储方式的表称为索引组织表。这句话怎么理解?

索引表内部是B+的结构，那所有索引表在innodb中是怎么的结构，一条语句如何快速定位到某一个索引表的？我的猜测是不是所有的索引表也是存在于一棵B+树上呢？通过一些索引表自身的标志?

答：比如一个innodb 有一个主键索引和2个普通索引。这样就三棵树对吧。然后InnoDB只要记录每棵树的根节点的位置就行了，有根节点，B+树就能拉出一整棵树。

244、有一张用户表，user_id是唯一标识，user_id是随机的字符串，user_id长度也不等，表大概10来个字段。只需要以user_id为索引进行查询。如果以user_id建立主键索引，考虑数据量达到千万级别，这种情况下插入性能会比自增主键大概会差多少。数据量大了以后，本身以字符串做主键是不是也会成为瓶颈。

答：User_id业务有要求必须是唯一索引的话，你这种情节就直接用来当主键吧。

245、查询出来的结果集默认是有序的吗，还是按使用索引的正序?

答：按照索引遍历的顺序，这样不用再排序。

246、哈希表这种索引模型，为什么数组后面会拉出一个链表而不是数组呢？是不是就是因为链表的插入比数组的时间复杂度要低?

答：其实更重要的原因是不定长...一开始不知道要开多大的数组，每个值后面的item个数也不同。

247、文中提到KV场景适合用业务字段直接做主键，因为没有其他索引，不会浪费空间，但岂不是分裂现象仍然严重？因为不能保证业余字段顺序插入啊。

答：嗯嗯，鱼和熊掌不可兼得。不过我们是假设业务必须把这个字段设置为唯一索引。不把它设置成主键，设置成唯一索引也一样要面对这个问题。

248、非聚集索引上为啥叶子节点的value为什么不是地址，这样可以直接定位到整条数据，而不用再次对整棵树进行查询。

答：这个叫作“堆组织表”，MyISAM就是这样的，各有利弊。你想一下如果修改了数据的位置的情况，InnoDB这种模式是不是就方便些。

249、为什么二级索引的value是主键索引呢？不这么做，一旦数据页的分解或合并会更改所有的B+树，特别对于二级索引要对很多不连续的数据页修改，会降低并发性能。这么做，数据页的操作只会影响主键的B+树，但这会影响二级索引的查询效率。还有个问题，如果用户显示两二级索引建在主键上，那会不会有两份数据?

接着问上次的问题啊，如果主从复制传redo到从机，在回放时，一个mtr更改多个数据页，除了锁住整个B+树外，还有没有其他能解决B+树分解的原子性呢?

答：前面的分析很好。不能直接动手，要交给IO线程处理并发冲突。所以物理复制是一个新技术，要解决很多问题。

250、自增主键是不是只对数据插入操作会简化索引的维护，如果是对数据的删除操作，因为删除操作会涉及页合并，所以对索引的维护是不是就跟普通主键索引一样了?

答：合并不是频繁操作，所以对删除的性能影响是可以忽略的。如果是随机删除数据确实差不多。

251、KV场景，我们是假设业务必须把这个字段设置为唯一索引。不把它设置成主键，设置成唯一索引也一样要面对这个问题。

我很疑惑：业务字段设成唯一索引，不设成主键索引。岂不是唯一索引经常分裂，并且还需要回表查询，双重缺点。不如把业务字段设置成主键了吧？还是说讨论的是innodb之外的引擎，不需要非得有个主键?

答：还是专指Innodb, 所以说，如果没有别的索引，就特别合适把这个唯一索引当主键索引用，没额外成本。如果有别的索引就要权衡大小的问题。

252、数据库做集群，用自增索性，需要给每台服务器设置不同步长，感觉很麻烦。但用自己生成的唯一id，像老师说的可能存储空间和效率没自增好，请问这是时候怎么考虑?

答：自己生成唯一ID就是实现麻烦一点，如果是bigint 类型自增的，存储空间和效率其实也不错的。

253、查了一下，max_execution_time 是影响查询语句的超时时间：

The execution timeout for SELECT statements, in milliseconds. If the value is 0, timeouts are not

enabled.

max_execution_time applies as follows:

• The global max_execution_time value provides the default for the session value for new connections. The session value applies to SELECT executions executed within the session that

include no MAX_EXECUTION_TIME(N) optimizer hint or for which N is 0.

• max_execution_time applies to read-only SELECT statements. Statements that are not read

only are those that invoke a stored function that modifies data as a side effect.

这样对长事务还有影响吗？进而思考了一下，只读事务内会维护回滚段吗？jdbc客户端驱动说可以用Connection.setReadOnly(true) 来提高性能，是不是同此有关呢，能有哪些优化？

答：单语句的时间是算在整个事务中的，所以长查询也会导致长事务。只读事务没更新，不会写回滚段，不过由于有一致性读，还是有可能要访问回滚段的。

254、这里讲的索引B+树的叶子节点是一行数据，那对于每一行数据，内部是怎么存储的呢，有没有用到什么数据结构？如果仅仅是顺序写磁盘，那修改表结构的时候是不是每一个叶子节点对应的磁盘都要改，这样不会很低效吗？

答：叶子节点之间是指针链接的关系，改一个不用改别的哦。

255、如果主键不是递增的，插入数据，会导致页分裂。那么说，索引在存储引擎中，肯定会有一部分缓存在内存中，这里插入数据，更新索引的时候，实际上是现在内存中更改索引（页满了进行页分裂）。那么索引的物理存储会有什么变化吗？或者说索引的物理存储格式是怎样的？

答：Innodb 里面内存和数据页是一一对应的，内存分裂了，等到写回磁盘，看上去就是“磁盘上那棵树的叶子节点，也分裂了”。

256、唯一索引是和主键一样作为一级索引吗？

答：不是。这里说的唯一索引还是非主键索引的一种。

257、二叉树树高20就有可能要访问20个数据块，是不是意思每层对应一个数据块存储？树高多少就对应几个数据块？n叉树也是这样吗？

答：是，不过N叉树的树更“矮”。

258、在 InnoDB 中，表都是根据主键顺序以索引的形式存放的，老师，这句该怎么理解？还有，除了索引组织表，还有别的种类么？

答：就是所有的数据都在索引里，有，堆组织表，比如myisam 。

259、现在一般自增索引都设置为bigint，这点老师这么看？

答：特别合理，因为现在很多业务插入数据很凶残，容易超过int 上限，实际上是建议设置bigint unsigned。

260、非主键索引的叶子节点内容是主键的值。在 InnoDB 里，非主键索引也被称为二级索引（secondary index）。我看图上一个非主键索引只有一个叶子节点，实际上应该会有多个叶子节点吧，内容是主键的值吧？

答：有多个叶子节点的。每个叶子节点是一个数据页，每个数据页16K，里面是放多个值的。

261、回表只是普通索引才会有的吗？主键和数据放在同一个树中，根据主键查询的时候，就可以直接获得数据了。那select ＊from table where id＝xx和select id from table where id＝xx的效率是一样的吗？（id是主键）

答：这两个语句是都不用回表了，在“查找行”这个逻辑上是一样的，但是select *要读和拷贝更多列到server,还要发送更多列给客户端，所以还是select id更快的。

262、文中说非主键索引会存储主键的值，而文中举例的非主键索引值刚好没有重复，所以想请问下，如果记录表中 R1~R5 的 (ID,k) 值分别为（100,1)、(200,1)、(300,1)、(500,1) 和 (600,1)，那么非主键索引k=1的节点，应该记录100，200，300，500，600的值，是这样理解么？

答：不是，非主键索引上有5个值，分别是（1，100），（1，200）... （1，600）。

263、二叉搜索树的特点是：每个节点的左儿子小于父节点，父节点又小于右儿子，那上面那个例子要查ID_card_n2的话可不可以理解为先查userB节点，依次向下找，找不到再来找userC节点？

答：不是呀，跟父节点比，小的走左，大的走右，不需要左右都扫。

264、对于索引树，它是如何决策哪一个做为父节点，哪一个是叶子结点呢?

比如文中的：100，200，300，500，600，700，mysql是根据什么策略来构建这个b+树的，又是在什么时候决定增加树的高度呢?

答：核心目标就是“减少树高”，最底层满了就要往下扩展。

265、“以 InnoDB 的一个整数字段索引为例，这个 N 差不多是 1200。”这个怎么理解呢?

答：一个page的大小是固定的（默认16k），索引大小固定的情况下，一个page可以放的item数是固定的，如果是int型，是1200个左右。

266、在插入数据的时候，主键类型为字符串，ID为uuid的形式，插入时会导致分裂吗?

答：会，特别不建议uuid做主键。。

267、身份证和id为主键选择的问题:老师写了这样一句话’显然，主键长度越小，普通索引的叶子节点就越小，普通索引占用的空间也就越小。’但普通索引的引占用空间不光是叶子节点决定吧，应该还有中间的索引节点，这里如果使用id当主键，身份证当普通索引相当于普通索引的索引节点空间增大。其实从空间的角度来说应该都差不多吧。

答：是这样的，这个主要是考虑到，一个表可能有多个非主键索引，这样每个非主键索引的叶子节点就包含了主键id的值。

268、uuid作为主键索引的，它在插入数据时是追加插入？还是需要挪动数据?

答：uuid不是持续递增的，中间还是不免会插入到“中间”。

269、自增主键的插入数据模式，每次插入一条新记录，都是追加操作，都不涉及到挪动其他记录，也不会触发叶子节点的分裂。这表述不对吧，插入无数个数据，数据页早就满了啊，应该分裂啊?

答：不用“分裂”，已经有了12345， 要插入6的时候，如果叶子满了，就申请一个空的页，然后单独放入6，这个过程不叫做“分裂”。

270、请问这个B Tree B+Tree有什么区别?

答：B+Tree是数据只存在叶子节点，中间节点都只有索引字段。

271、为什么二叉树会访问更多的数据块?

答：因为二叉树更高，比如树高如果为20，就可能落在不同的20个数据块上。

272、全表扫描，是扫描整张表;按索引查找，不也是扫描整个数组吗？难道不是类似于扫描整张表?

答：如果是“按索引查找”，那就不是扫描，而是用B+树查找，这个是可以快速定位记录的。

273、譬如身份证做主键，可以保证数据唯一性。如果一个表，是自增主键，那么插入数据不会造成页分裂，但是如果有一天删除一条记录，是不是会出现页合并的现象，因为主键不连续了

答：删除一条不会的，主键没有要求“连续”存放哈，中间有个把空洞没事的，空洞太大了才会触发合并。

274、为什么一定有一个回表过程。因为保存了主键数据量小吗?

答：需要回表，是因为使用的索引上的信息不够，需要到主键索引上去取数据。

275、既然非主键查询还需要回表，那就没有存在的意义了，为什么要有非主键的查询呢?

答：索引可以快速定位记录，这个好处还是需要的。

276、请问老师是否有讲外键的利与弊?

答：外键可以用来做约束，但是这种约束关系是在数据库里面做的（类似于存储过程，其实是一种逻辑）。这种情况下，等于你的数据库里面也有业务逻辑，这个就要看项目管理上做得怎么样。

如果能够把这些关系也作为代码的一部分，其实是可以的。之前很多人会觉得说加了存储过程、触发器、外键这些以后，代码逻辑混乱，一个原因也是因为没有把数据库里的逻辑像代码一样管理好。

277、看到n叉树的n是由数据块大小决定的。既然，一般情况下是1000多，那么也就是说一个数据块只能存放在大概1000多的数据块?

还在想另外一个问题？如何保证整个节点刚好在一个数据快上面，但是刚好分布在不同的数据块上面，那岂不是还是要查询两次?

答：“那么也就是说一个数据块只能存放在大概1000多的数据块”准确说是 一个数据页 放1000多个记录。一个数据页大小是16k， 如果有的数据超过16k，就会放在多个不同的数据页上，查这样的记录是要访问多个数据页的。

278、1. 有些资料提到，在不影响排序结果的情况下，在取出主键后，回表之前，会在对所有获取到的主键排序，请问是否存在这种情况?

2. 索引下推那个例子，感觉5.6之前的机制很匪夷所思：感觉判断'张%'之后再“看age的值”是顺理成章的事。难道联合索引的底层实现结构在这期间发生了变化?

答：1. 有的， Multi-Range Read (MRR) 由于不论是否使用这个策略，SQL语句写法不变，就没有在正文中提。

2. 不是，是接口能力发生了变化，以前只能传“搜索关键字”。如果你用过5.1 甚至5.0， 在从现在的观点看，你会发现很多“匪夷所思”。还有：并行复制官方5.6才引入、MDL 5.5 才有、Innodb 自增主键持久化、多源复制、online DDL。只能说，持续进化，幸甚至哉。

279、建立了(name, age)的联合索引之后，例子中的检索如下：

mysql> select * from tuser where name like '张 %' and age=10 and ismale=1;

如果改为：

mysql> select * from tuser where age=10 and name like '张 %' and ismale=1;

这样优化器是否能够作出优化调整，还是直接走全表搜索?

答：既不会走全表扫描，也不会调整优化 。还是跟原来一样的执行计划。SQL语句的条件是会自动调整的。

280、大家都知道建立索引可以提高查询效率，但是索引太多会导致索引占用的存储空间比原始数据都要大，请问这两个要怎么去平衡?

答：现在的磁盘其实空间问题大多是时候不是主要问题了。所以一般会建议优先考虑查询性能。

281、原文：在引擎内部覆盖索引在索引 k 上其实读了三个记录，R3~R5（对应的索引 k 上的记录项）.

我的问题是在 k 索引树上搜索 k=6时，发现条件不满足，不就结束了吗，难道还会回到主键索引去读一遍 R5 的值吗? 所以这里的读了“三个记录”，是不是有问题? 应该是两个吧!

答：覆盖索引本来就不会回表哈，第三次读出来，判错就结束的意思。

282、如果一张表有ab两个单独的索引字段，在一条查询语句中where a=”” and b=””，优化器阶段会选择哪个索引，还是两个索引都走?

答：有三种可能，

1. 只选a, 然后用b过滤
2. 只选b ，然后用a过滤
3. 选a和b,然后两个单独跑出来的结果取交集

优化器选择的策略是数据分布

283、为什么第一张图的300/700，3/7 是在上面一层，另外为什么要跳过了400/4？其实看上一节举例时就有这个疑问。

答：上面一层是用做搜索功能（查询的时候确定往下找哪个儿子节点的）。跳过400是为了说“如果要插入一行（400，4）”

284、结合之前那个where in问题，用union all会造成整体sql语句很庞大的，这个对效率没影响吗？union all是最优的吗，有没有其他办法，目前搞工作中遇到了这个问题?怎么解决呢?

答：备注 in后面有很多，至少上万的了，union all的话，相当于上万条select union all。不需要union all，就用in 好了。上万看上去不太美，拆成多个in ()吧。

285、为什么要有联合索引这个东西呢？看样子是为了配合使用“索引下推”，减少回表?

答：索引下推是后来才有的。 就是为了快速定位记录。（name,age)对于where name=“a” and age=10就特别快。

286、视图能做索引吗，或者视图查询效率该怎么优化?

答：视图不能建索引，视图本身是SQL语句，关键就是优化视图定义就是。

287、mysql为什么要查一次索引回一次表？而不是查完索引后把结果存储起来，一次回表查询？这样应该更容易用到局部性原理。

答：全存起来变成需要临时表。还有，现在还不支持“传入一批ID给主键，’同时回表’ ”这样的能力。

289、思考题：老师的回答是ca不必要，cb需要保留，我已经听明白了，但是既然有了cb这个索引，那么c这个索引也没有必要了吧（最左前缀原则）？为什么老师没说把c也可以去掉?

答：只有cb的话， where c= N order by a还是要排序的。

290、图4 ID4 ID5 只要取回数据吧，是不是不用再取回数据判断，索引里有age。

答：还要判断ismale，前面还是一个select * , 所以还是要回表的。

291、linux中tokudb引擎有没有类似alter table T engine=InnoDB这样的操作来释放空间呢?

答：对于server层来说，这个动作是“重建表”，但是重建表会不会收缩空间，就得看引擎怎么做了，tokudb应该会有效的。

292、为什么还要去k索引树搜索k＝6，发现不满足条件，循环结束呢？关于这个语句:select * from T where k between 3 and 5 。

答：因为5还满足，这个循环得继续找，找到第一个不满足的呀。

293、在覆盖索引这一小段，为什么name的字段比age大，就推荐(name,age)和age这两个索引呢？联合索引的两个字段反过来，然后再单独创建name的索引不可以吗？感觉空间上占用的差不多啊。

答：单独一个name字段 和 单独一个age 字段，还是后者小一点。

294、根据最左前缀原则以及索引结构实现原理应该只需要保留ab主键以及cb索引就能满足问题中貌似的查询语句。

答：但是要避免排序，最好有个cab。

295、这个语句（从文稿中copy出来的） select * from tuser where name like ’张 %’ and age=10 and ismale=1; ，“张”字后面有一个空格，去掉之后，执行计划就变了 ，发现查询列只要带上不在索引列中的列，执行计划 的 type 就变成了 ALL，rows 也变成了行数总和，即使去掉“and age=10 and ismale=1”？ 难道是否全表扫描，不是光看type？还有执行计划的每一列，究竟是怎么看的呀?

答：可能是数据量太少，你模拟一万行进去看看，应该还是能用上索引的。

296、老师讲的索引都是基于有主键的表。如果一个表里边只有唯一索引和普通索引那么唯一索引，联合唯一索引，普通索引都是怎么执行的呢？效率上又会如何呢?

答：这时候Innodb 引擎会自己创建一个内置主键。

297、既然是between 3 and 5，它该查询4吧，怎么会查询6呢，在k索引树中。

答：没有4呀，5的下一个是6，得判断到“6不满足”才结束。

298、在不影响排序结果的情况下，在取出主键后，回表之前，会在对所有获取到的主键排序，请问是否存在这种情况？不是取出一个主键回表一次吗，什么情况下一次取出多个主键再回表？

答：”存在，在MySQL 里面叫做“MRR优化”这个是自动的，代码版本支持就会这么做”。

299、看到某位同学提了个问题。关于的 truncate 与 delete.也就是说 truncate 会删除所有（数据、索引等一系列相关）。delete 虽然删除了数据，但是实际索引依然存在。 这就是他们之间的区别。实际上是 truncate 就行是么？能解释一下 “”自适应哈希索引“”么。

答：Delete其实索引也删除的。就是如果要全表情况，就truncate好了。自适应哈希主要是偏理论的，我得想个应用场景…”。

300、delete也删除索引， truncate 也删除索引。但是实际他们是删除了 索引内部的内容，而非索引是吧？查询了一些资料是这么说的，想跟您确认一下。

答：额…都是删除数据，表结构、索引定义是没删的。

301、select * from T where k between 3 and 5 为啥要找k=6呢?

答：因为要找到6，才知道“后面没有符合条件的了”。

302、面试官问：说下怎么让mysql的myisam引擎支持事务，网上搜了下，也没有结果！

答：我怀疑他是想说用lock table 来实现，但是这样只能实现串行化隔离级别，其它隔离都实现不了。但是因为mysiam不支持崩溃恢复，所以即使用lock table硬实现，也是问题多多：ACID里面， 原子性和持久性做不到；隔离性只能实现基本用不上的串行化；一致性在正常运行的时候依赖于串行化，在异常崩溃的时候也不能保证。这样实现的事务不要也罢。你这么答复面试官，应该能加到分吧。

303、请教下主键索引的存储顺序

```
CREATE TABLE `demo`.`test_pk_vs_non_unique_index`(
  `id` INT NOT NULL,
  `val` INT,
```

```
  PRIMARY KEY (`id`),
  INDEX `idx_val` (`val`)
) ENGINE=INNODB;

INSERT INTO test_pk_vs_non_unique_index VALUES(3, 33);
INSERT INTO test_pk_vs_non_unique_index VALUES(1, 11);
INSERT INTO test_pk_vs_non_unique_index VALUES(5, 55);
INSERT INTO test_pk_vs_non_unique_index VALUES(2, 22);

SELECT * FROM test_pk_vs_non_unique_index;
    id      val
------  --------
     1        11
     2        22
     3        33
     5        55

INSERT INTO test_pk_vs_non_unique_index VALUES(4, NULL);
INSERT INTO test_pk_vs_non_unique_index VALUES(6, NULL);

SELECT * FROM test_pk_vs_non_unique_index;
    id      val
------  --------
     4    (NULL)
     6    (NULL)
     1        11
     2        22
     3        33
     5        55
```

问题：为什么行的返回顺序不是1，2，3，4，5，6?

因为明明ID列是主键，主键索引就是聚集索引，就应该是顺序排列。

为什么另一列上的索引上有空值就影响了主键的存储顺序？”

答：你每个语句explain下，看看怎么使用索引的，对照再分析看看。

304、 关于“根据身份证查询地址”这个有点疑问，当有索引（身份证号，姓名）这个索引，确实按照最左前缀原则可以走索引，但是查询地址是要回表的吧，如果加索引（身份证号，地址）就不回回表了，但是会有维护索引的成本，那该如何舍取呢?

答：如果第二个查询不多，我会选择就只有 （身份证号，姓名）这个索引，然后查地址的语句，可以用上这个索引，回表查。

305、在高性能mysql这本书中提到索引并不总是最好的工具，其中提到对于非常小的表，大部分情况下全表扫描更高效，对于中到大型的表，索引就非常有效？我的问题是怎么定义非常小的表呢？怎么定义中到大型的表呢?

答：其实就是看扫描行数。我觉得小于100行叫做非常小的表（一般是放些配置参数啥的）。

306、我看到你回复留言说N树的N是由页大小，和索引大小决定的。请问页是指什么呢？索引大小，是指索引包含的字段数，还是指索引包含字段数的字节呢?

答：页就是数据页（默认16K，可调），索引定义里所有字段字节数的和。

307、请问第12篇专栏为什么表数据删掉一半，表文件大小不变 在哪里可以看到呢

答：表t的数据存在文件t.ibd里面。

308、下面两条语句有什么区别，为什么都提倡使用2:

1.select * from T where k in(1,2,3,4,5)

2.select * from T where k between 1 and 5”

答：第一个要树搜素5次，第二个搜索一次。

309、【文章链接】

原文章链接地址如下

http://hedengcheng.com/?p=577

但是这个地址有时候打不开有的时候打开很慢，我找到了一个转载了这篇文章的博客，地址如下

https://www.jianshu.com/p/7a4b4f821c8d

【例子】

```
dba@db_test 05:08:05>show create table trade_detail\G
*************************** 1. row ***************************
Table: trade_detail
Create Table: CREATE TABLE `trade_detail` (
`id` int(11) NOT NULL,
`tradeid` varchar(32) DEFAULT NULL,
`trade_step` int(11) DEFAULT NULL,
`step_info` varchar(32) DEFAULT NULL,
PRIMARY KEY (`id`),
KEY `idx_test` (`tradeid`,`trade_step`,`step_info`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8

dba@db_test 05:08:14>select * from trade_detail;
+----+----------+------------+--------------+
| id | tradeid | trade_step | step_info |
+----+----------+------------+--------------+
| 1 | aaaaaaaa | 1 | cus |
| 2 | aaaaaaaa | 2 | update |
| 3 | aaaaaaaa | 3 | commit |
| 4 | aaaaaaab | 1 | add |
| 5 | aaaaaaab | 2 | update |
| 6 | aaaaaaab | 3 | update again |
| 7 | aaaaaaab | 4 | commit |
| 8 | aaaaaaac | 1 | bbc |
| 9 | aaaaaaac | 2 | update |
| 10 | aaaaaaac | 3 | update again |
| 11 | aaaaaaac | 4 | commit |
+----+----------+------------+--------------+

SQL1
dba@db_test 05:08:26>desc select * from trade_detail where tradeid>='aaaaaaac' and
trade_step>2\G
```

*************************** 1. row ***************************

id: 1

select_type: SIMPLE

table: trade_detail

partitions: NULL

type: range

possible_keys: idx_test

key: idx_test

key_len: 104

ref: NULL

rows: 2

filtered: 33.33

Extra: Using where; Using index

SQL2

dba@db_test 05:08:47>desc select * from trade_detail where tradeid>'aaaaaaac' and trade_step>2\G

*************************** 1. row ***************************

id: 1

select_type: SIMPLE

table: trade_detail

partitions: NULL

type: range

possible_keys: idx_test

key: idx_test

key_len: 99

ref: NULL

rows: 1

filtered: 33.33

Extra: Using where; Using index

【疑问】

mysql在确定First Key的时候为什么对于>=可以继续向下匹配，而对>却停止匹配了，这个原因是什么？能否从B+树遍历的原理进行解释一下?

留言部分我咨询了这个问题，您说举一个例子，所以我就举了一个例子，这个表是引用的您其它文章里的表和数据。非常感谢答复。”

答：他这个应该只是对于First Key，比如条件是比如条件是 trade_id > ’aaaaaaaa’ and trade_step=3 and step_info=’update’

这时候优化器第一次查询的是 拿到id=1的这一行，但是因为是大于，所以不需要继续判断 trade_step=3 and step_info=’update’这两个条件，直接往后找 （这个first key逻辑就结束了），

接下来再找就找到 aaaaaaab才算第一个开始有效的，

我觉得你认为这时候命中的是 id=3 和 id=4 的这两行在idx_test的这两个key的间隙好了

然后继续往后扫，扫到找到的值，超过范围为止”。

310、文中说了利用最左前缀原则可以利用 身份证号+姓名 这个索引支持“用身份证号查询地址”这个需求，而这个需要回表一次，利用身份证号字段索引也需要回表一次，这两有什么区别么?

答：在这个查询上没有区别。主要是如果有身份证+姓名的索引（为了使用覆盖索引），那么就没有必要加身份证号单独的索引”。

311、对于最左前缀的理解，比如建立了一个联合索引(a,b,c),有以下查询条件：

1.WHERE a=xxx AND b=xxx

2.WHERE a=xxx AND b=xxx AND c=xxx

3.WHERE a=xxx AND c=xxx

4.WHERE a=xxx

5.WHERE b=xxx AND c=xxx

6.WHERE b=xxx

7.WHERE c=xxx

对于1，2，3，4情况会走索引；1，2，4情况不会进行回表操作（用到索引下沉），效率较高；

3会用到索引a，但索引c失效，会进行回表操作，效率会低一些；

5，6，7情况都不走索引，会进行全表扫描，效率低，这样理解对吗？”

答：取决于你前面是select 什么哦，如果是select *的话，124还是要回表，3只能用索引的a部分是对的，567 也对。

312、”C、a索引是不必要的，因为c索引是次级索引，指向主键索引，已经包含了a，可以使用a排序，满足第一种查询，C、b索引是必要的，在a字段未知的情况下不能使用b字段排序，我觉得除了主键索引外，只需要c、b索引就可以了。

答：再有一个(c,b)的联合索引，可以避免where c=N order by b这个语句的排序。

313、文中提到的索引下推案例：（name,age）

查询 ： like ’张%’ and age=10，会判断索引中的age值。但是我看「高性能MySQL」书中提到，对于多列索引，如果查询中有某个列的范围查询，则其右边所有列都无法使用索引优化查找。这与文章内容有些出入，是MySQL版本的问题吗？这是MySQL有索引下推之前才有的缺陷？”

答：这两个没有矛盾哈。“如果查询中有某个列的范围查询，则其右边所有列都无法使用索引优化查找”， 这里说的使用索引优化查找，应该是指“不能用索引快速定位”。索引下推能力其实age也不是用来“快速定位”的，只是用在索引遍历中“快速过滤”的”。

314、关于覆盖索引这块，我有点疑问，如果select刚好要查询ID，我觉得是可以覆盖索引的，但是如果我要提取整个行数据，该怎么要呢，先查出ID list，然后select where id in ？

答：过程确实就是这样的，先拿到id，然后id去主键索引查整行的数据，这时候就用不上覆盖索引啦。

315、问题一：身份证号和name做联合索引，用身份证查name会快是什么原理?

问题二：查询的时候是会在索引数里直接读出name，这是下推索引才会这么做吗?

问题三：这时候查询是走这个联合索引还是身份证的主键索引，为什么?

答：1. 覆盖索引。

2. 这个不是索引下推哈，就是覆盖索引的概念（之所以要纠结概念，是因为覆盖索引5.5就有了，而索引下推5.6才有）。

3. 走这个联合索引，因为快。

316、联合索引怎么就不回表了，按照前面的例子只有查询字段是主键索引id的时候才不用回表呀！这个就不好理解了呀！

答：如果联合索引是由a,b组成的，那么 select b from t where a=N 就不用回表”。

317、对于索引的选择还是有点疑惑。业务有个表大约十列数据。其中业务上需要唯一索引 X (A, B, C)。但是90%查询语句需要列 (A, B, C, D, E)， 为了提高性能最好的选择是加一个普通索引来覆盖列 Y (A, B, C, D, E)，但是这样问题就是索引X和Y的内容重复了。在数据库层面上有没有好的解决办法，在不需要唯一索引X的办法下来确保(A, B, C)列的唯一性，还是说X索引和Y索引都是必须的?

答：就建一个X (A, B, C) 好了，X已经是唯一索引了，不需要 Y (A, B, C, D, E)了。

318、mysql> select * from tuser where name like ’张 %’ and age=10 and ismale=1;

为什么这个语句只走了name的索引，不走age，是只有name like ’张三’ and age＝10 才会走age吗?

答：对的 ，前缀匹配哈。不过不是“只走了name的索引，不走age”，这是同一个索引，比较准确的描述是：使用了这个索引的前两个字节做快速定位，然后由于有index condition pushdown优化，接下来遍历索引过程中可以用age来过滤掉不满足条件的记录。

319、b+树中的索引节点应该都是由指针和索引组成。但现在要将磁盘索引节点加载到内存中，这些指针地址是怎么映射的（磁盘和内存指针的映射）？

答：首先磁盘的数据和数据页是一样的，所以磁盘只能记录“我的第一个叶子节点在page_n”在内存里面也是，然后当把page_n读到内存以后，内存里面记录的是“page_n”的内存地址在哪里。

320、”对于覆盖索引的疑问，当 select ”非主键“ from T where k = **；的时候是不是就不算覆盖索引了，只有查询需求是主键的时候才算覆盖索引。如果是这样的话，还是有回表的操作，那当查询“非主键”时的过程是不是如下：

1. 通过索引 k = 123 找到 ID主键 1**；
2. 通过ID主键 1** 找到对应的数据行 row；
3. 对比row中的“非主键”字段，相等就返回给Server层。

答：不一定，也可能是一个联合索引的一部分，比如有一个联合索引 (k, a),然后执行 select a from t where k= ... 也是可以用上覆盖索引的。

321、问题一：之前一般认为range查询比如”a > 5 and b = ’123’”在联合索引（a,b）中b是不起作用的，在ICP下是不是意味着b就可以起到作用了，我们还是应该尽量将查询中用到的字段放入联合索引中。

问题二：针对1的问题，“a > 5 and a < 10 and b=’123’”在ICP作用下的执行过程是什么样子的？”

答：1. 是的

2. 流程是这样的：

a) 把 a>5 and b=’123’传入引擎，

b)引擎找到第一个a>5的行（这里是快速定位），如果发现b<>’123’,找下一个，直到满足b=’123’,

c)把找到的行返回给server层， server层根据a是否小于10决定要不要取下一个。

322、alter table T engine=InnoDB 重建索引是否会产生行锁?

答：5.6版本以后支持onlineddl， 行锁不会，有mdl读锁。

323、网上很多资料提到联合索引 (a, b)，在查询语句where a = xxx and b = xxx 和 where b = xxx and a =xxx 都是能走到索引的，跟您在文中提到的，（age ,name） 不太一样。

答：能走到索引，但是不能使用多name+age全字段快速定位。

324、公司有订单表，有些核心字段，比如订单号.时间(整型，时间戳，范围查找).订单状态（整型，6个值，可能in，可能=）.客户标识（整型，几百个值）.付款方式（整型，5个值），设备号（字符串，有权限需要in）,这6个字段后台都会用到查询筛选，而且不选的情况下条件就不传，按照联合索引最左原则，那么可能要建几十个索引，这是不可能的，这个表做了按月分表，数据量一张表大约1000万,不建立索引的话，后台选的条件没有建索引就会非常慢，强制最多只能查连续两个月的数据（union all），请问老师有什么好的解决方案么?

答：嗯，这种情况是挺麻烦的。

得按照查询的模式，选最常见的来创建组合索引。比如如果时间+客户标识用得最多，就创建者两个的联合索引。对于比较少用的条件，单独给这个字段建索引，然后查id出来跟别的字段的查询结果，在客户端取交集，也是一种思路。

总之，这种需求是最头大的。

325、联合索引c,b，where条件里为c,order by b,这样的语句，联合索引把c,b都用到嘛?

答：会，前提是where c= ，不能用大于或者小于。

326、CREATE TABLE c_test(

cid INT,

cnum1 INT,

cnum2 INT

);

ALTER TABLE c_test ADD INDEX index1(cnum1,cnum2);

DELIMITER ;;

CREATE PROCEDURE idata()

BEGIN

DECLARE i INT;

SET i=1;

WHILE(i<100000)DO

INSERT INTO c_test VALUES(i,i,i);

SET i=i+1;

END WHILE;

END;;

DELIMITER ;

CALL idata();

SELECT COUNT(*) FROM c_test;

EXPLAIN SELECT * FROM c_test WHERE cnum1>999 AND cnum2=1000;

老师这是我的实现过程，我用explain看并没有使用到索引啊。

id select_type table partitions type possible_keys key key_len ref rows filtered Extra

1 SIMPLE c_test \N ALL index1 \N \N \N 100089 5.00 Using where”

答：表示这时候优化器认为全表扫描比较好。

327、文中提到考虑空间时，name字段是比age字段大的，所以建议用(name，age)联合索引和age单个索引。不太明白这里说的大，指的是?

答：字段大小（单个name占用的空间比age大）。

328、mysql里的B+树这种索引结构是怎么持久化到硬盘的？数据库的运行过程中如果想把某一个B+树节点对应的内存页存到硬盘中去，具体怎么做呢？在硬盘中怎么维护这种B+树这种结构呢?

答：内存数据页和磁盘数据页是一一对应的，持久化的时候就直接覆盖写进去，内存里记了数据页在文件中的偏移量；数据页从磁盘读入内存的时候，记录了内存地址；这样实现互相找到。

329、5.6版本前只搜索了第一个字段就回表，那岂不联合索引都没有啥用了?

答：是说，用不上连续前缀索引的情况下才这样。对于索引(a,b,c)如果是a=1 and b=1 and c=1还是可以都用的，但是如果是a=1 and b<>1 and c=1就只能用第一个。

330、我发现我司的数据库（Oracle）几百张表的主键ID都是32位的UUID。看完本章，是不是说明我们的设计有问题；相比自增（Oracle用的是序列）ID，在同样多的数据下，UUID索引的文件要大大的增加。因为现在线上的数据比较庞大（表的数据动辄千万级别的），表与表之间的关联关系都是通过UUID来关联的，一张表的修改就会牵动全身，请问有没有好的优化方案?

答：Oracle的原理我不够熟悉，不能乱说哈。

如果是在MySQL 里，既然都用UUID关联了，要做改动就要很小心。可以考虑保留UUID,并在这个字段上创建一个索引。同时创建一个字增主键ID。

331、假如有一张表，有多个状态的字段，比如 T(id，..... delete_flag, order_status, handle_flag), 因为这些字段值域的区分度不大，比如都是0~3，所以按照之前的对索引的理解，针对这几个字段建索引效果应该不是很好，但是如果数据量大的话，按上面的条件查询等于全表扫描，会比较慢，实际工作中的场景，请问有什么好的优化建议?

答：这个其实是要看查询怎么写。

比如语句就是 select count(

*) from T where delete_status= ... 那也只能给delete_status创建上索引， 不过如果delete_status的区分度很低，这个语句跟全索引扫描的效果也差不多。还是要从业务设计上优化。

332、order by是对结果集排序的，与ca和cb索引关系不大。

答：还是有关系的，索引合理的话不需要排序了。

333、覆盖索引必须要覆盖所有的查询条件中的列吗，我经过试验发现，覆盖索引不能只覆盖要查询的列，同时必须将WHERE后面的查询条件的列都覆盖，能解释一下这个原因吗?

答：你的发现是对的，因为覆盖索引的目的就是“不回表”，所以只有索引包含了where条件部分和select返回部分的所有字段，才能实现这个目的哦。

334、组合索引顺序选择的时候，为什么字段大的排在前面会更省空间呢?

答：(a,b) + (b)

如果是(大，小），就需要一个额外的(小）索引；

如果是（小，大），就需要一个额外的（大）索引。

335、问mysql为什么没有采用最右原则呢?

答：只能选一个嘛，而且最左前缀才能支持自然的range查询。

336、没分析出来二级索引的叶子节点的数据结构，不知道是不是（key,value）形式？并且这个相同的key来说，value还是有序的?

答：可以理解为还是key，value结构（其实是树形式组织的），不以value的顺序来组织，而是key的顺序，key是没有“相同”的，你把主键值考虑进去就能发现了。

337、我不明白为什么执行select * from T where k between 3 and 5是为什么还要执行第5步 在 k 索引树取下一个值 k=6，不满足条件，循环结束　因为我理解的是找到3 然后找到5 就能把这里对应的数据找出，不应该再去往后请求了。

答：不是唯一索引的话，有可能5后面还有一个5。

338、覆盖索引这种情况，是不是查询的必须是主键，且是声明了主键的表。

答：只要查询的条件和返回字段中的，所有字段都能被一个非主键索引覆盖，就能用覆盖索引。

339、我刚才面试的时候 面试官说mysql索引除了叶子结点要进行磁盘io之外 其他的结点都存放在内存 你觉得这样的说法对吗 这样内存不会不够放 我的观点是每个结点会进行一次io

答：额 两种都不太准确。应该说非叶子节点也可能放在磁盘上，取决于内存大小。

只是大多数数据量下，非叶子节点不多，留在buffer pool的概率高。

340、一次查询为啥不能只回表一次呢？比如范围查询通过二级索引定位到全部符合条件的主键在回表一次。

答：这样还需要地方把”满足条件的所有主键值“存起来，就要用到临时表了，性能反而更差。

341、建立索引会增加insert和update时的消耗，但是能加快查询，如果做读写分离，主库不建立索引，从库建立索引，带来的风险是主从同步的时候延迟会更加大，不知道这种想法靠不靠谱呢?

答：有这么用的哈。主从延迟其实也不会增大很多，后面章节会介绍到 change buffer的优化，只是说如果从库重做的时候，系统要记得把索引再给加上。

342、专栏中说“那么，如果既有联合查询，又有基于 a、b 各自的查询呢？查询条件里面只有 b 的语句，是无法使用 (a,b) 这个联合索引的，这时候你不得不维护另外一个索引，也就是说你需要同时维护 (a,b)、(b) 这两个索引。”我是有疑惑地。

建表及索引语句如下：

create table abc(id int primary key,a int,b int,c int)

ALTER TABLE abc ADD INDEX idx_a_b_c (a,b,c)

这是我的执行计划：explain select * from abc where c=1;

按理说where c=1 不满足最左匹配原则，可是我的执行计划里面走了索引，很困惑。

1 SIMPLE abc index idx_a_b_c 15 1 100 Using where; Using index

答：using index表示，使用了覆盖索引，也就是说确实使用了这个索引，但是执行的时候，是在这个索引里“全索引遍历”，这个索引没有起到加速的作用。

343、对session c的写锁会阻塞session d的读锁有个疑问，因为之前session a没有释放读锁，那么session c应该一直取不到写锁，这个时候session d是不是应该有机会取到读锁先于session c执行?

答：没有，session C 在等待的时候就开始阻塞后来的请求了，不是拿到（拿到以后反而很快执行就放过了）。

344、mysql 5.6不是支持online ddl了吗？也就是对表操作增加字段等功能，实际上不会阻塞读写?

答：Online DDL的过程是这样的：

1. 拿MDL写锁

2. 降级成MDL读锁

3. 真正做DDL

4. 升级成MDL写锁

5. 释放MDL锁

1、2、4、5如果没有锁冲突，执行时间非常短。第3步占用了DDL绝大部分时间，这期间这个表可以正常读写数据，是因此称为“online ”

我们文中的例子，是在第一步就堵住了。

345、在对一个表做增删改查操作的时候，加MDL读锁；当要对表做结构变更操作的时候，加MDL写锁。老师，加了MDL读锁之后，还能增删改吗？不大明白。

答：可以，因为增删改执行都只要求MDL读锁，读锁之间不互斥的。

346、添加二级索引会加什么锁?

答：属于DDL，变更表结构，过程做要加MDL写锁。

347、update table set xx;这种更新全表的锁是mdl吗?感觉应该是第一种全表锁。

答：也不是，还是行锁，只是这个表的所有行。

348、MDL只有在事务中才会被加上么 ， 如果没有开启事务直接进行增删改 MDL不会默认加上么。 比如我没有加事务， 去执行一个更新语句， 首先会先去找这个数据，然后在进行更新 。 整个过程被拆成两步进行，这个期间没有锁的么?

答：是没有“没加事务”的概念的，你不显式地启动事务，MySQL也会按照“单语句事务”处理的。

349、比如某台服务器空间硬盘为1T，在上面部署一台单机的mysql。随着时间的增长，mysql空间不够了。然后再给这台机器挂一个1T的磁盘。请问mysql怎么使用这个1T的磁盘？我对服务器这块比较小白，是不是需要将这1T的硬盘变成服务器的系统盘才能使用?

答：一般比较简单的做法是把binlog 路径配置到新磁盘。

350、在从库执行readonly 会不会影响主从复制，也就是说会不会导致从节点无法写入master提供的数据了?

答：不会，readonly对super权限用户无效，执行binlog的线程是super权限。

351、问题一：上面的那个因为mdl锁把整个库搞挂的例子里，如果用pt工具来操作，会出现同样的情况吗?

问题二：那个例子里显示select语句前加了begin，是不是select的时候不加begin，就不会出现同样的情况呢?

问题三：online ddl 的copy方式和inplace方式，也都是需要 拿MDL写锁、降成读锁、做DDL、升成写锁、释放MDL锁吗?

答：1. Pt的过程也是有操作表结构的，所以会类似。

2. 对，没有begin的话，这样select执行完成以后，MDL就自动释放了哦。

3. 是，是否online都是第三步的区别，另外四步还是有的。

352、文中讲到了小表加字段需要注意的地方，那么大表加字段呢？如果大表的访问量和新增记录很频繁，是否无法做到灵活加字段呢?

答：大表一般反而大家会注意，大表在线更新现在用比较多的是gh– ost。

353、已经有cb索引了，为什么还要保留c呢？前缀索引不是可以复用吗?

答：有c的话，c=N order by a就需要排序。

354、老师如果用的是where A 再 group by c 那我主键是不是可以用a, c还是单独c?

答：如果你有一个索引是（a,c), 这个查询语句可以避免排序的。

355、为什么我在谷歌还有其他网页上搜索死锁的处理方法只有kill进程呢？您提到的commit也能解决死锁适用于什么场景呢?

答：不是commiit，是rollback. 当然，kill就是一种主动放弃的做法，是实现rollback的一种操作。

356、第一关于全局锁，在做备份的时候，为什么要让让全局只读的状态，我们用老师说的Mysqldump的single transaction方法不是更好吗？第二是在进行Lock tables t1 read ,t2 write，为什么其他线程不能写t1,读t2,A在执行Unlock tables 之前，为什么可以读t2?

答：1. 如果库里面有不支持事务的表。

2. 前半句因为锁就是这么定义的；后半句，write比read权限高。

357、Online DDL的过程是这样的：

1. 拿MDL写锁

2. 降级成MDL读锁

3. 真正做DDL

4. 升级成MDL写锁

5. 释放MDL锁

1、2、4、5如果没有锁冲突，执行时间非常短。第3步占用了DDL绝大部分时间，这期间这个表可以正常读写数据，是因此称为“online ”

问题：

如果第四步升级为MDL写锁的时候，这个表的MDL锁有其他事务占着，那么这个事务会阻塞，等到可以拿到MDL写锁是吧”

答：对。而且如果不幸一直拿不到，最后锁超时了，就只好回滚这个DD了操作。

358、按您的说法：select会加MDL读锁，而增删改查也会加MDL读锁，读锁互不排斥，那么未在事务中全表扫描过程中，可以执行dml操作。

以前我一直以为

1.select不会加锁，所以dml操作可以执行；

2.ddl操作被阻塞的原因是，有对该表的DML或DDL操作未执行完；没成想事务中select语句也能对DDL操作造成堵塞。

还有个疑问：如果select没在事务中执行，那么在select执行过程中，应该也是不可以执行DDL操作的吧。

答：语句执行过程中肯定是加了MDL读锁的，所以会被锁。

359、在mdl锁这一块有一点不清楚。

a获取mdl读锁，这时候c要获取mdl写锁会被阻塞，c被阻塞是因为要等a释放，才能获取写锁，为啥c在被阻塞的时候，其他的查询语句也会被阻塞，不是说读锁不互斥吗？还是说我理解错了，应该理解为c获取写锁之后，在没释放写锁期间，其他的读锁会被阻塞。

答：你没理解错，5.6的实现确实是“一个DDL，由于被其它线程的MDL读锁堵住，自己无法执行，但是会堵住后续MDL读锁的申请”。

360、您说的这个读锁之间不互斥，因此你可以有表锁多个线程同时对一张表增删改查。可是下一章又说表锁同时只能有一个做更新操作，是否矛盾?

答：没有矛盾，Innodb做数据的增删改查的时候，不需要对表加表锁。

361、原文：举个例子, 如果在某个线程 A 中执行 lock tables t1 read, t2 write; 这个语句，则其他线程写 t1、读写 t2 的语句都会被阻塞。同时，线程 A 在执行 unlock tables 之前，也只能执行读 t1、读写 t2 的操作。连写 t1 都不允许，自然也不能访问其他表。
问题是最后一句，线程 A 解锁前，不允许写 t1，自然也不能访问其他表。为什么不允许线程 A 访问其他表，读其他表的操作也不允许么？这样设计的目的是为了规避啥问题吖？
答：这个不可考据了。不过我觉得有一个好处，就是可以让当前线程尽快执行unlock tables，它自己操作不了别的表，会尽快去释放锁，否则一直锁着，影响系统并发度。

362、DML锁是不是只针对事务啊，文中说锁是在commit后才会释放，如果autocommit=1设置了，select之前不显示的开启一个事务，也没有相应的commit是不是就不存在DML锁了？
答：DML一般指的是增删改。不过若果select ... lock in share mode; select ... for update也会有行锁。

363、您说索引 ca 和索引 c 的数据是一模一样的。是否不太准确？索引 ca 是先按 c 排序，再按 a 排序。而索引 c 只是按 c 排序。因为有 order by a，所以删除索引 c 更为合适。是这样么，而索引 c 只是按 c 排序。
答：不是的，索引c是按照 cab排序的，跟索引ca一样。

364、MDL作用是防止DDL和DML并发的冲突，个人感觉应该写清楚，一开始理解为select和update之间的并发。
答：嗯，特意写了是MDL“读锁”。

365、关于MDL写锁，有个更好的处理方法不知道是否稳妥？先基于原表创建一个临时表，完成字段更新修改；然后对原表加MDL写锁，加锁成功后把临时表更名为原表，这样加锁时间很短。
答：最早的ddl内部实现就是这么做的。不过加锁时间也不短。“先基于原表创建一个临时表，完成字段更新修改；”这个过程，要包含“把原表的数据导入临时表”吧？导入的过程还是对原表加MDL锁的。

366、既然文中说，session C被阻塞之后，后面需要MDL读锁的session也都会被阻塞。那么只要前面的A，B session的长事务执行完了，C就能执行掉（我想应该是按先到先得的顺序获取锁的吧），为什么后面又说“即使把长事务kill掉也未必管用，因为新的请求马上又来了”，为什么新的请求不是排在session C后面呢？这样按理不会拖跨才对啊？就算是一些已经拿到MDL读锁的session在事务中的发送新sql，也应该为了避免死锁不需要再排队了吧。

答：“kill 也未必管用”是说，如果没有超时机制，我们只能先kill，再执行ddl，这样才保证不会堵住后面的请求。但是先kill完，可能ddl还没执行，又来新的查询，导致前面的kill白做了。是这个意思。

367、索引 cb 的组织是先按 c 排序，在按 b 排序，同时记录主键—c--|—b--|—主键部分a--主键部分为什么是a 而不是ab呢?

答：因为我们定义为（c,b)，其实就是告诉MySQL，要按照c，c有序的规则来组织索引。如果改成cab，那么cb就不一定保证有序了。

368、索引问题答案解释这个是不是再详细一点，还是搞不清楚为什么c索引和ca索引一样。

答：InnoDB会把主键字段放到索引定义字段后面，当然同时也会去重。所以，当主键是(a,b)的时候，定义为c的索引，实际上是（c,a,b)；定义为(c,a)的索引，实际上是(c,a,b)。你看着加是相同的。

Ps：定义为(c,b）的索引，实际上是（c,b,a)。

369、对于大表的ddl，是不是会一直阻塞着增删改查呢，对于大表的ddl，一般如何解决mdl锁阻塞的问题?

答：5.6以后支持online DDL，不会堵塞增删改查了。mdl锁是没办法的，不过mdl锁对正常使用没影响，因为增删改查申请对是mdl读锁，而ddl执行对大部分时间，占着的也是mdl读锁。

370、“读锁之间不互斥，因此你可以有多个线程同时对一张表增删改查。”这里是否写错了，应该是“你可以有多个线程同时对一张表执行查”，而不是“增删改查”，增删该不是写吗?

答：这里说的是“MDL读锁”，对数据表做增删改查的时候，加的都是“MDL读锁”。

371、使用 ”mysqldump –single-transaction ” 做备份，会不会存在备份期间有些数据被修改，而备份线程又无法读到最新数据，导致备份出来的数据跟真实数据之间存在差距呢?

答：对于备份行为来说，“备份出来的数据跟真实数据之间存在差距”是没关系的。

备份表示的是“在某个时刻，这个系统的状态”， 只要备份出来的数据，跟“这个时刻”是一致的就可以了，不需要跟“备份完成后”的时间点一致。

372、是在执行器执行时 生成快照 进行锁表对吧?

答：要看是执行什么语句哈。基本都是在执行器锁的，但是不同的语句阶段不同，比如lock table ，就没有生成快照，直接锁表了；如果是myisam的更新语句，是在引擎内第一次使用表的时候加。

373、mdl锁那里读锁之间不互斥，因此你可以有多个线程同时对一张表增删改查;要是对同一行数据进行处理呢,会互斥吗?

答：对同一行数据进行处更新，会互斥，但是互斥的不是mdl锁，而是行锁。

374、例题中的订票系统，影院的余额表可不可以用流水的方式来记录，每天闲时汇总一次，这样就没有update只有insert和select sum了。

答：嗯，如果是没有边界条件，比如一直加钱，这种可以的。但是如果有“退款”的逻辑，就不行了。只记日志可能会给扣成负数。

375、为减少死锁检测，用中间件或者改源码，让更新同一条记录的的操作进入引擎层前排队，也就是把并发转为串行，这样结果返回时间是不是就慢了？我的意思是假如同时很多人买一样东西，排在前面的人余额扣的快立马可以看到，后面的得等待一段时间才能刷新出来?

答：相比与死锁检测吃CPU，这还是快的。一般一秒能处理5、6千次update,出现热点更行可能就剩100不到。

376、 1行级锁只有update的时候会用到吗？insert的时候会用到？2 死锁检测时间复杂度是O(n)是怎么计算的 ，100万级 3 死锁检测发现有死锁都是先回滚再在客户端重试，数据库有对死锁检测之后的处理吗?

答：1. Insert 也有的，比如你要插入两个相同的id值的记录。

2. 每个线程在判断有没有死锁的过程中，都要扫一下所有其他人。

3. 回滚了一个事务，剩下的就继续工作了。

377、设置一致性试图命令之前和之后区别，做保存点前后有啥不一样?

答：一致性视图之后对DDL暂时没影响，保存点设置之后也是。只是“回滚到保存点”这个操作会释放MDL锁。

378、文章中提到行锁是在InnoDB 事务中需要的时候才加上，那不加事务的DML语句(比如update)会不会加上行锁啊?

答：没有DML就不需要加行锁哈。

379、主动回滚死锁链条中的某一个事务，让其他事务得以继续执行，这个回滚是回滚后面的这个事务吗?

答：系统会选，同等权重（比如相同类型的事务操作），会优先回滚最后的这个，成本最低。

380、如果有我有多个node，其中一个node 对数据加了行锁，结果挂掉了，还是只能等待InnoDB自动50S释放锁吗?

答：如果客户端断开，对应的那个事务会回滚，锁就会放弃了。

381、针对其他同学提到的，对无索引的字段做更新，会锁定全表。试过对

A事务执行：begin; update t set t.num=t.num+1 where t.code='10001'; # t.code无索引

B事务执行：begin; insert into t values (null, '10002', 100); # 第一列为id列

确实，B事务在等待状态。而分析INNODB_LOCKS表，发现：

lock_id | lock_type

B | RECORD

A | RECORD

为何 A 锁住了全表了，lock_type 不是TABLE 而是 RECORD 呢?

答：这里“锁全表”，表示的就是“锁了全表的所有行”。

382、第三讲说过，在可重复读隔离级别下，事务启动时会创建一个逻辑视图，整个事务存在期都会用这个视图。那为什么在事务启动时如果没有加WITH CONSISTENT SNAPSHOT，假设时刻2 DDL语句到达，导出数据的时候不会报table definition has changed。为什么DDL在show create table之前到达不会报这个错误，show create table是做了什么事情呢?

答：Show create 表示这个语句“看过了”表结构。表结构数据和表的数据不同，不支持一致性读。

383、本节课讲的不支持行锁的引擎，只能使用表锁，而表锁同一张表在同一时刻只能有一个更新。但是上节课讲的表级锁中的MDL锁，dml语句会产生MDL读锁，而MDL读锁不是互斥的，也就是说一张表可以同时有多个dml语句操作。感觉这两种说法有点矛盾！

答：不矛盾，MDL锁和表锁是两个不同的结构。

比如：你要在myisam 表上更新一行，那么会加MDL读锁和表的写锁；然后同时另外一个线程要更新这个表上另外一行，也要加MDL读锁和表写锁。第二个线程的*MDL读锁是能成功加上*的，但是被表写锁堵住了。从语句现象上看，就是第二个线程要等第一个线程执行完成。

384、在同一时间不是应该只有一个线程拿到锁吗?锁释放之后由其他线程争夺锁权限,这样的话为什么会出现事务A等待B释放锁,事务B又会等待事务A释放锁?

答：锁是针对“行”的，他俩现在锁的不是同一行哦。

385、如果reset以后，是不是就失去了长连接的意义了呢？相当于再次进行连接。

答：不会，reset_connection只是复位状态，恢复到连接和权限验证之后的状态，没有重连。

386、select * from t where id>=1 and id<=4. id是主键，看老师之前的文章谅解，这条语句是在主键索引里分别查询id等于1，2,3,4这几条数据，也就是在主键索引b+树中查找4次。b+树叶子节点是有序且用链表的方式链接起来，为什么不在叶子节点直接查找。上次老师讲的是非主键索引，是这样的当时，不知道主键索引是否是我说的这样查找

答：如果是你的语句这样写，是不用找四次的，找到第一个（id=1)然后向右遍历就行。之前说的不行是说，在回表过程中，回表只能一个ID一个ID回来查。

387、update t set k=k+1 where id=1，这样一个语句是先拿读锁，读到k，然后再拿写锁去将数据改成k+1么？还是说一看到update这样的关键字就直接拿读锁和写锁？如果是第一种情况，那2条同样的update语句，可能出现同时读到k的数据，然后其中一条更新后另外一条再更新，这样数据就有问题了？

答：没问题呀，后加写锁的就等待。

388、如果是并发插入，会有死锁问题吗，他锁的哪些行呢？

答：可能会死锁，一个表有多个索引的话。

389、有以下情况 帮忙分析下会锁表不

update a，b set a.name = b.name where a.uid=b.uid and b.group=1；

update c，b set c.age=b.age where c.uid=b.uid and b.group = 1；

如果两个语句同时执行期间 是不是有个执行不了 要等b解锁。还是说没有更新b的字段b不会锁，两个可并行执行?

答：这个你得同时贴表结构。还有，会不会锁，不是验证一下就可以吗，两个都用begin + 语句，两阶段锁协议会帮助你。

390、针对高频热点账户，问题一：可否根据账户id排序，来解决死锁问题，当然有可能热点账户排在前面，导致持有锁时间变长，或者加钱账户可更新，减钱账户余额不足的情况。

问题二：可否将事务拆解，将加减钱的数据库事务拆解成减钱账户update，流水记录insert一个数据库事务，然后异步补偿加钱账户比如影院账户，达到最终一。

答：1. 这个场景里其实没有死锁的。不过大家都按照同样的顺序加锁，确实是避免死锁的方法

2. 这个在没有边界条件的时候是可以的。文中说到，其实影院也可能扣款，比如退票。那扣款是可能扣成0的。要做边界条件判断的时候，日志补偿的模式就满足不了需求了。

391、你说时刻2，时刻3，时刻4是代码执行的时刻，那能请问是Q2代码，Q3代码，Q4代码执行的时刻麽?

答：不是，就是代码的注释里面写着的时刻2、时刻3…这些。

392、每个新来的被堵住的线程，都要判断会不会由于自己的加入导致了死锁，这是一个时间复杂度是O(n)的操作。假设有1000个并发线程要同时更新同一行，那么死锁检测操作就是100万这个量级的。老师，为什么1000个并发线程，死锁检测是100万量级?

答：你先考虑，假设现在已经有999个线程在同一行等锁，新来一个请求也要访问这个行，他要判断有没有死锁要判断多少次？然后这个结果乘以1000。

393、关于死锁检测innodb_deadlock_detect我想请教一下，是每条事务执行前都会进行检测吗？如果是这样，即使简单的更新单个表的语句，当每秒的并发量达到上千的话，岂不是也会消耗大量资源用于死锁检测吗?

答：如果他要加锁访问的行上有锁，他才要检测。

这里面我担心你有两个误解，说明下：

1. 一致性读不会加锁，就不需要做死锁检测；

2. 并不是每次死锁检测都都要扫所有事务。比如某个时刻，事务等待状态是这样的：

B在等A，

D在等C，

现在来了一个E，发现E需要等D，那么E就判断跟D、C是否会形成死锁，这个检测不用管B和A

394、平时工作中经常对数据库操作，对有些重要的业务库进行增删改的操作时，为了减少错误，偶尔会在命令行用begin/commit来启动一个事务，结果查询无误了才提交。这种情况是不是也会生成MDL或者行锁？是不是不应该使用这种方法?

答：这样在MDL问题不算大，这个操作会阻塞DDL，但是你们应该不会出现“你在操作的时候，刚好有另外一个同学在改表结构”的操作吧？正常大家都会知会一下避免的。这种情况，怕的是手工操作期间，从begin到commmit之间时间太长，如果是业务更新高峰期，就不好。

395、我理解同一个行记录 在每个事务里面 只出现一次update，是不回出现死锁的吧?

还有一个问题，一个事务里面修改同一条记录两次，不是也会死锁了?

答：多个事务更新同一行，只是锁等待，如果还有更新别的行，可能造成循环依赖，导致死锁。一个事务自己跟自己不会死锁。

396、之前遇到一个死锁的问题：

【描述】一个表，在2个字段上都加了索引，字段：roomId，extnumId；然后有这样的更新语句： update set .... where extnumid=? and roomId in(1,2,3); 在并发的场景下发生了死锁。后来查阅了资料，得到的知识点如下： 1. InnoDB 的 行锁建立在索引的基础上，锁的是索引。 2. 上面的语句先锁非主键锁，再锁主键锁去更新。 不知道这2点结论对不对？另外，我上面那条update 是按照什么顺序锁索引的？ 后来我的解决方式是：改成按主键更新解决的问题.

答：1. 前面结论对的.

2. 你是不是有重叠条件。就是不同的语句里，有出现相同的room id?行锁挂在索引上，如果使用有索引的字段更新；同时又有另一个事务更新同一行没有索引行会发生什么？行锁挂在索引上是什么操作?

所以你举的例子，在扣一个总账户数额一条记录的时候 应该是不会出现死锁的，就是锁等待，只是队列很长。

397、innodb行锁是通过锁索引来实现，如果id字段加索引，update 更新id = 1 和 id = 2记录，在更新id = 2时会进行阻塞吗?

答：同一行才阻塞

398、如果开启事务，然后进行死锁检测，如果发现有其它线程因为这个线程的加入，导致其它线程的死锁，这个流程能帮着分析一下么?

答：好问题。理论上说，之前没死锁，现在A加进来，出现了死锁，那么死锁的环里面肯定包含A，因此只要从A出发去扫就好了。

399、对于 回滚到 SAVEPOINT sp，在这里的作用是释放 t1 的MDL锁，这一句有疑问。

那么如果DDL语句在Q4之前到达了，会正常执行，但是这里已经执行了snapshot，然后后面又会回滚到之前的SAVEPOINT sp。中间有个DDL的，这样也回滚了了？？我不是很明白回滚到SAVEPOINT sp 这个点。

答：没有回滚DDL哦，只是释放了mdl锁

400、“如果在 Q4 语句执行之前到达，现象：没有影响，备份拿到的是 DDL 后的表结构。”

其他的都能理解，就是这点很困惑：Q2已经建立了快照，为什么还能备份快照后，对表结构修改后的表呢？不应该备份快照时的表结构吗？不然备份不就乱了吗?

答：Q2是启动了一致性视图，但一致性视图不包含表结构哦

401、大事务会影响主从延时吗？例如一个大事务里面执行更新语句，记得日志系统那章说，两段提交，redo log的prepare和commit之间已经写binlog了。这样未commit之前已经有binlog了，那事务未完成就可以发生主从复制了吧。还是说redo log的commit跟事务的提交是两码事?

答：大事务一般都会导致主从延迟，就在binlog写完之后才开始发。

402、使用 in 条件批量更新如何加锁?

答：In这个，如果你说的是后面是常数字列表，那就是多个等值查询那么处理； 如果带子查询，就复杂些，还需要根据隔离级别来判断的。

403、row 格式下，delete 一个无主键 innodb 大表，主库很快完成，可从库每删除一行都需要全表扫，导致延迟很大，除了加主键外，还有其它的方法吗?

答：有没有普通索引，MariaDB 可以根据普通索引优化的。不过还是要形成习惯，表里带主键哈。

404、上讲降到dml时会产生读MDL锁（表锁），也就是update会持有读MDL。读和读不互斥。但是对于行锁来说。两个update同时更新一条数据是互斥的。这个是因为多种锁同时存在时，以粒度最小的锁为准的原因么?

答：不是“以粒度最小为准”，而是如果有多种锁，必须得“全部不互斥”才能并行，只要有一个互斥，就得等。

405、老师看评论里说，行锁是锁在索引上的，那么如果存在两条sql，通过不同的索引去锁定同一行记录，innodb会怎么处理呢？我实验的结果是会阻塞，但是不太明白怎么阻塞的。

答：如果是update语句，一定会在主键索引上也加上行锁，这两个在主键索引上形成等待关系。

406、减少锁冲突的思想是分治法吗？类似分库分表，把压力分摊，同时增加容灾能力.

答：要看是什么场景。如果是文中提到的所有线程都在更新同一行，分表也没用过了，就不是分治；如果是那种分散的更新压力，可以用分治。

407、表U的钱包记录表是会负数的 消费就是负数 充值就是整数 所以求和之后是余额 这样和在用户表单独加一个余额字段有什么劣势吗?

答：但是很多业务其实是不能接受负数的，所以一般都是要实时判断，当然如果没有“边界条件限制”的，是可以的。

408、本课举的例子，预售一年电影票导致CPU占用率100%，这怎么可能是死锁导致的呢？这种场景会出现互等锁的情景吗?

答：不会出现死锁，就是普通的单向锁等待。但是我们知道不会死锁，InnoDB可不知道，所以它得做死锁检测才能得到“没有死锁”这个结论，我们说吃CPU的就是死锁检测这个逻辑。

409、面试官问了行锁的原理，

我：行锁的实现原理?

面试官：对

我：不知道。。。

回来又看了几遍事务隔离的这几篇文章，发现确实不知道，丁老师能简单讲讲么?

答：主要是不知道面试官想知道多深的原理。我觉得08篇的内容可以讲讲的，每次更新产生一个新的数据版本，数据版本上加了锁。 如果来了另外一个更新，就会在这个新版本上被锁住。然后看看面试官怎么追问。

410、直接用update miaosha_goods set stock_count = stock_count – 1 where goods_id = #{goodsId} and stock_count >0的方式不就解决超卖了吗，为什么有人说并不能完全解决，与今天老师讲的行锁相违背。
答：主要是并发的时候有性能问题

411、有些资料看到的：InnoDB行锁是通过给索引上的索引项加锁来实现的，只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁。老师，这个是正确的吗?
答：不对。你可以验证下，select * from t limit 1 lock in share mode, 看怎么锁的。
mysql不会先分析本次事务所需要的锁，一次性获取吗？这样会避免死锁。这样并发性能不好。

412、老师在文中说：所以，在执行事务 B 的 Q1 语句的时候，一看自己的版本号是 101，最新数据的版本号也是 101，可以用，所以 Q1 得到的 k 的值是 3。
1. 这里不参考up_limit_id了吗?
2. 如果参考，事务B的up_limit_id是在执行update语句前重新计算的，还是在执行Q1语句前重新计算的?
答：1. 判断可见性两个规则：一个是up_limit_id ,另一个是“自己修改的”；这里用到第二个规则
2. 这时候事务Bup_limit_id还是99。

413、可重复读情况下，事务c的102早于事务b的101，如果事务c再get k，那不是就取得101的值了?
答：咱们例子里面，事务C是直接提交的，再执行一个GET 就是另外一个事务了。如果你说的是用begin 来启动一个多语句事务，那么事务c在更新后查询，还是看到row trx_id是102的。
【注意：如果它还没提交，101根本生成不出来，因为事务B被行锁挡着呢】

414、文中说:“读提交是在每条语执行前计算up_limit_id”，如果不计算会怎样？就变成了读未提交吗?
答：为啥不提交，bug 吗 ？ 无关不计算也不判断，就是读未提交；如果不计算，但是判断，那就是用事务的up_limit_id, 就是可重复读。

415、在重复读级别下，b事务中，执行update t set k=k+1 where k=1这句时，where条件中得到的k=1 ，k+1以后，不应该是2，么，为什么是3，您的文章里说，如果是2，c事务就失效了，所以 执行where语句和执行k=k+1不是在同一个逻辑一致性下执行的？那这样可重复读的语义是不是没有保证?

答：Update语句是“当前读”的逻辑，所以它拿到的k 是2哦。

416、事务启动是begin就记录up_limit_id还是begin之后的第一条select语句开始记录up_limit_id?我测试中，客户端A 执行begin后但没执行select，客户端B开始事务修改提交后，客户端A再次select会读到这个修改值。而另一种begin后直接select,此后客户端B开始事务修改后，客户端Aselect就是第一次Select的值。

答：Begin之后的第一个语句算启动事务。

417、假设我的T表很大，第一种方法只需要全表扫描一次，第二种需要全表扫描20次吧?

答：有Limit ，都是总共锁10000行。

418、在可重复读隔离级别下，事务在启动的时候就“拍了个快照”。

老师，这句话有点疑问，在可重复读隔离级别下，事务视图应该是在事务第一个查询执行时建立的吧，在下面这个例子中

demo1

A1: start transaction;

A2: select * from t;

B1: start transaction;

B2: insert into t values (4, wangwu);

B3: commit;

A3: select * from t;

demo2

A1: start transaction;

B1: start transaction;

B2: insert into t values (4, wangwu);

B3: commit;

A2: select * from t;

demo1的A3和A2读到的数据是一样的即读不到B事务新插入的数据，但是demo2里的A2却可以读到B事务新插入的数据，能解答一下么?

答：我跟你是一致的，如果是 begin; select ... 这种用法，事务是在select语句才正式启动。

419、对于可重复读，查询只承认在事务启动前就已经提交完成的数据；这个应该是第一次查询时，不是事务启动时吧?

答：Begin不算事务启动哦，所以咱们俩是一致的。

420、是不是只有commit了之后的数据才会被加到有历史版本和当前版本的记录中呢 没有commit 是不是就是自己知道 存在内存的某个角落呢?

答：没commit也写，也生成版本，但是是未提交的，还加着锁。

421、对于读提交，查询只承认在语句启动前就已经提交完成的数据；而当前读，总是读取已经提交完成的最新版本。老师请问下读提交和当前读有什么区别吗，我理解这两个都是读到已提交的最新版本? 如果是的话，那为什么读提交这种隔离级别不直接基于当前读做呢，每次去算一下up_limit_id不麻烦吗?

答：不一样，读提交隔离级别下，首先是不加锁的。还有，你考虑下，一个语句开始执行的之后，执行期间别的事务修改了数据的情况。

422、对于文中的例子假设transaction id为98的事务在事务A执行select（Q2）之前更新了字段，那么事务A发现这个字段的row trx_id是98，比自己的up_limit_id要小，那此时事务A不就获取到了transaction id为98的事务更新后的值了吗?

换句话说对于文中”之后的更新，产生的新的数据版本的 row trx_id 都会大于 up_limit_id”这句话不太理解， up_limit_id是已经提交事务id的最大值，那也可能存在一个没有提交的id小于up_limit_id的事务对数据进行更新? 还是说transaction id比up_limit_id小的事务都是保证已经提交的?

答：Innodb 要保证这个规则：事务启动以前所有还没提交的事务，它都不可见。但是只存一个已经提交事务的最大值是不够的。 因为存在一个问题，那些比最大值小的事务，之后也可能更新（就是你说的98这个事务）。所以事务启动的时候还要保存“现在正在执行的所有事物ID列表”，如果一个row trx_id在这列表中，也要不可见。

423、最大事务ID是保存在系统里面的一个全局变量吗?是excution_gtid?如果并发写很大，这个产生事务id的生成器会不会成为瓶颈?

答：是全局变量，事务启动后拷贝为这个事务自己的内部变量的。不是gtid 哈，trx_id是innodb的机制，gtid 是server 层的。

424、一个历史row版本可以删除的依据是什么，您文中提到的是没有事物再使用这个版本了，感觉这句话还是很空洞，能不能具体说一下?

答：如果所有事务的一致性读的结果，都不需要再用到它的时候。比如我们例子中，Q2执行前， row trx_id=101 102的两个版本，就不能删除，因为事务A还需要依赖他们。

425、如果commit之前就生成了历史版本，当前版本的记录，那么如果之前的一个update没有commit，没有释放锁，那后来新的一个事务select会不会因为那一行还有锁而提取全表数据等待呢？一致性读的时候读不是不加锁的吗？那这种情况怎么办呢?

答：一致性读的时候，只有未提交的版本都直接认为不可见（包括可重复读隔离级别和读提交隔离级别）。

426、 另起一个事务在当前事务更新前修改， 如果新起的事物没有提交 ，那么当前的事物没有办法执行更新语句 ，会一直堵塞。如果新起的事物提交后 ，当前事务再更新，那么当前事务更新的数据行是最新的 ，再去查询也应该是当前事务更新的值，所以为什么会看不到新的值还是没有理解。

答：当前事务再更新，成功了的话，就是就是看到自己更新的，自己更新生成的版本自己是要可见的（transaction_id=row trx_id)。

427、执行事务B的Q1语句时候，一看自己版本好是101，最新数据版本号不是102？是在事物C基础上读到新的值。上面描述是不是错了?

答：执行Q1的时候，事务B已经更新过了，(1,3)已经生成出来了。

428、原文”如果落在黄色部分，那就包括两种情况

a. 若 row trx_id 在数组中，表示这个版本是由还没提交的事务生成的，不可见；

b. 若 row trx_id 不在数组中，表示这个版本是已经提交了的事务生成的，可见。”中，b段落，为什么row trx_id不在数组中，不是未来提交事务，而不是已提交事务？这个row trx_id如果是

已提交事务，怎么又在黄色区域，不在绿色已提交区域?

答：在黄色区域是因为，这些事务是在“低水位事务之后”开启的，但是又在“当前事务启动前”提交的。

429、当开启事务时，需要保存活跃事务的数组（A），然后获取高水位（B）。我的疑问就是，在这两个动作之间（A和B之间）会不会产生新的事务？如果产生了新的事务，那么这个新的事务相对于当前事务就是可见的，不管有没有提交。

答：好问题，代码实现上，获取视图数组和高水位是在事务系统的锁保护下做的，可以认为是原子操作，期间不能创建事务。

430、“InnoDB 里面每个事务有一个唯一的事务 ID，叫作 transaction id。它是在事务开始的时候向 InnoDB 的事务系统申请的，是按申请顺序严格递增的。”

老师您好，请问是begin/start transaction语句就申请到了trx_id，还是执行了第一个操作数据表的语句才有了trx_id呢?

答：都是“事务启动”的时候申请，所以是第一个操作表的语句。也可以是执行start transaction with consistent snapshot这个语句的时候。

431、想到一个业务上的问题：减库存的场景

当前库存：num=200，假如多线程并发：

AB同时开启事务，A先请求到行锁，

A：

start transaction;

select num from t where num>0;先查询当前库存值（num>0）

update t set num=num−200; 库存减量

B：

start transaction;

select num from t where num>0;先查询当前库存值（num>0）

update t set num=num−200; 库存减量

----结果---

A：查询到num=200,做了库存减量成了0

B：事务启动后，查询到也是200，等 A 释放了行锁，B进行update，直接变成 −200

但是 B 查询时，时有库存的，因此才减库存，结果变成负的。

老师，对于这种场景，怎么避免减成负值？给 select 加读锁或者写锁吗 ？ 这种select 加锁，对业务影响大吗?

答：这是个好问题，也是并发业务常见的问题。一开始Select 加锁虽然可以，但是会比较严重地影响并发数。

比较简单的做法是update语句的where 部分加一个条件： where nun >=200 .

然后在程序里判断这个update 语句的affected_rows, 如果等于1 那就是符合预期；

如果等于0，那表示库存不够减了，业务要处理一下去，比如提示“库存不足”。

432、在数据可见性规则那一部分中的第三种可能的b情况: ”若 row trx_id 不在数组中，表示这个版本是已经提交了的事务生成的，可见。”对于这部分内容我开始不是很理解, 后来反复思考了一下, 可见性规则这部分是不是在说明这种情况：因为数据的row trx_id是依次递增的, 但是事务由于创建和提交的时间不可预期所以transactionId可能是跳跃的, 所以假如有事务A, 比A的transactionId大的数据的row trx_id对于事务A一定不可见, 但是比A的transactionId小的数据的row trx_id也可能在A的事务数组中, 所以要判断一次。不知道这么理解对不对?

答：是的，最后这个“可能”说得很好，可能在，也可能不在，就用“是否在数组中”来判断。

433、数据库show processlist存在大量opening tabale的情况，此时数据库接近瘫痪，这个跟阻塞有关么?

答：阻塞一般就提示为阻塞了，也可能根本就是并发太大。

434、在事务A中update语句set部分c读取到事务B更新后的值，而where 部分id读取的c是事务A一致性视图中的值，由于两者不一致导致更新记录为零的对吗?

答：不是，整个update语句走的当前读，因此where读的就是事务B修改后的值啦。

435、我在mysql实践中遇到个问题，就是同一个后台服务的方法A中执行了插入一条数据成功后，然后方法B中去查询发现没查到这条数据，其中方法B是一个异步方法（另一线程），看日志发现查询是在插入后几十微妙，这是不是出现幻读了，有什么办法解决吗？隔离级别是Repeatedable Read，数据库没有读写分离，查询条件也是唯一索引的。

答：这个不是幻读呀，这个是写入了查不到。你看的“查询在插入之后几十微秒发起”，这个日志是客户端日志，还是数据库日志？数据库除非特别定制，否则没有微秒单位，如果是业务发送端的话，有可能查询是后发先到哦，毕竟才差几十微秒。

436、老师，有三个事务，事务A先启动tranid是100，事务B再启动tranid是101，都对id=1的行操作，B先操作然后提交，A再操作然后提交，此时C开启事务执行查询id=1的行，那它应该拿的是tranid为101的数据，这样不就读不到最新的了吗

答：不是啊，此时C拿到的是A操作后的结果哦。

437、我现在理解是：当A对一行进行操作时，开启事务，创建transid，然后B再对这一行操作时，必须要等到A提交释放锁后啊才可以继续操作，所以说明transid是严格递增的，查询一行数据时只查找transid最大的数据就可以了。

答：不是，你想这个序列：

A启动，更新id=1，

B 启动更新id=2，提交；

A更新id=2，再提交；

438、引用问题：

今天重新看了一下这章您的修改地方，有个地方不明白

落在黄色区域未提交事务集合部分怎么还要分类，低水位+高水位不就是这个数组了吗，之前说，这个数组是记录事务启动瞬间，所有已经启动还未提交的事务ID，那不应该是未提交的事务吗，不就应该是不可读的吗?

之前说的是启动时会获取一个最大row trx_id，所有大于这个id都不认，这个id肯定是已经提交了的事务的才对啊，这个id不才应该是数组的高水位吗，这里有点懵了。

作者回复：你设计一个“比低水位大，但是在当前事务启动前，就已经提交了的例子。

无法理解，在当前事务启动前就已经提交了，不是已经不活跃了吗，怎么还在快照数组中

答：在当前事务启动前就已经提交了，不在数组中，但是比低水位高。

你想下这种场景，按照时间顺序：

事务A启动 ，假设trx_id是1；

事务B启动，假设trx_id是2；

事务B提交；事务C启动，这时候C的低水位是1，因为A还没提交。所对于C的事务数组里是没有2的，但是“2>低水位（也就是1）”。

439、高水位不是当前事务自己吗? 当前事务启动瞬间 它的线程号应该是最大的啊，“当前系统里面已经创建过的事务 ID 的最大值加 1 记为高水”为什么不是?

答：事务启动时申请自己的trxid，但是创建视图列表的时候，是“稍微晚一点”的，这里有时间差，中间可能有别的事务也提交啦。

440、“在实现上， InnoDB 为每个事务构造了一个数组，用来保存这个事务启动瞬间，当前正在“活跃”的所有事务 ID。“活跃”指的就是，启动了但还没提交。”
“如果落在黄色部分，那就包括两种情况
a. 若 row trx_id 在数组中，表示这个版本是由还没提交的事务生成的，不可见；
b. 若 row trx_id 不在数组中，表示这个版本是已经提交了的事务生成的，可见”
这两段话有点矛盾，前一段定义了“活跃”事务是启动了但还没提交，后面一段又说不在数组中是已经提交了的事务生成，那么黄色段是个区间还是活跃事务数组呢？从后面解释来理解是个区间。
答：这两个概念的关系是：视图数组是活跃事务ID的集合，这个集合里，最小的事务id记为Min_id, 最大的事务id记为Max_id黄色区间就是 [Min_id, Max_id] 这个区间。
PS：Min_id就是低水位。

441、undo log 有缓冲吗。像redo log那样。
答：Undo log的写入策略和数据是一样的（WAL）。

442、评论区有人说高水位就是当前trx_id, 我也是这样认为，看你回复的是可能高水位会略大一些，我想了很久，除了文章中提到的高水位是最大trx_id + 1之外，还是没想明白这个略大是出现在什么场景下，老师能给举个例子吗？
答：1. 事务A申请到trx_id
2. 事务B启动，申请到 trx_id
3. 事务C启动，申请到 trx_id
4. 事务A创建视图数组

443、对于row trx_id， 当事务启动的时候，会生成保存所有活跃事务id的数组，以及生成高水位。按稳重高水位定义，当前系统里面已经创建过的事务 ID 的最大值加 1 记为高水位，而事务id是按申请顺序严格递增的， 那是不是可以理解为“高水位 = 当前启动事务id + 1”？
答：不是，因为在计算高水位之前，可能别的事务也创建出来了。

444、原文：“若 row trx_id 不在数组中，表示这个版本是已经提交的事务生成，数据可见”

疑惑之处：既然是已提交的事务，该事务又怎么会处于黄色区域呢?

答：事务A启动，创建事务A的视图；事务B启动，更新，提交；这时候在事务A的视图数组里面，没有B，但是B已经提交了。

445、在RR级别下，事务获取到的一致性视图数组是静态的么，高水位是当前创建的最大事务id+1，可否认为是当前事务的id+1（事务创建的时候，视图数组和高水位难道不是和事务创建的时候一起确定的么）？在RC级别下，事务获取的一致性视图是动态的，事务中可能出现多次查询，期间会产生别的事务，所以高水位只能是当前创建的最大事务id+1而不能是当前事务id+1。

答：RR和RC在一致性视图上的差别是：RR的视图是在事务启动的第一个语句创建的，之后事务存续期间都不变；RC是在每个语句开始执行的时候，都创建一个视图，每个视图只管自己一个语句。

446、只读事物不分配trx_id,那么事务A为什么 事务Id为100啊?

答：只读事务不分配id，是5.6以后的优化；其实也不是不分配id，只是不分配自增的id，随机分配的那个也是事务id的。这里简化为同一个机制（等同于都是按照老版本来说），比较便于理解哈。

447、若 row trx_id 不在数组中，表示这个版本是已经提交了的事务生成的，可见。这句话不明白，能解释一下吗?

答：事务提交后，他的trx_id就会从数组中拿走。

448、主键id也是唯一索引吧? 那我们的新增操作如何利用 change buffer呢?

答：所以主键索引用不上，都是对于那些二级索引的才有效。 一个insert语句要操作所有索引的嘛，收益在二级索引。

449、commit对change buffer的影响是什么？是仅仅改变redolog记录的提交状态吗?

答：好问题，是的。对于WAL 机制来说，change buffer就是数据的一种，在commit的时候处理机制是和数据页一样的

450、问题一：change buffer相当于推迟了更新操作，那对并发控制相关的是否有影响，比如加锁？我一直以为加锁需要把具体的数据页读到内存中来，才能加锁，然而并不是?

问题二：purge行为之后应该不会再产生redo log了吧？从应用开发的角度看，还是由数据库保证唯一好。

答：1. 锁是一个单独的数据结构，如果数据页上有锁，change buffer 在判断“是否能用”的时候，就会认为否

2. 是这样的，这个问题你分成两步来考虑。第一步，merge其实是从磁盘读数据页到内存，然后应用，这一步都是更新的内存，同时写redolog。

现在内存变成脏页了，跟磁盘数据不一样。之后就走刷脏页的流程。刷脏页也不用写。

是否使用唯一索引，这个要看业务有没有保证，和性能是否有问题。

451、之前第二章说redo-log是优先记录的，想问下既然redo-log也记录了charge-buffer的操作，为什么还需要写入系统表空间?

答：内存不够的时候会淘汰的（还是跟数据的机制一样）。

452、INSERT INTO .. ON DUPLICATE KEY UPDATE TID = 2,...

能详情解释下这个SQL的执行过程吗?

答：插入数据，无关没有主键冲突就插入，如果有，就对冲突的第一行修改它的TID值。

453、业务要求数据库中某个字段是唯一的，当请求并发的时候，除了在数据库层吗设置成唯一索引，还有别的办法吗?

答：要看你的唯一主键怎么来的。比如有一个唯一主键的全局产生器。

454、老师回复：“其实是恢复change huffer的内容，但是不怕丢失，因为change buffer也确保不会丢哈”这个我明白，因为redo控制它不会丢失，但是change buffer 进行purge合并的时候不会记录redo了，这个时候如果形成的脏数据还没落盘然后就异常宕机了，redo恢复的应该还是change buffer没合并之前的数据吧，应该不会直接恢复成合并后的脏数据吧

答：进行merge合并的时候会记redolog的，这样如果之后异常重启了就会通过崩溃恢复，恢复回来。

455、正常运行中的实例，数据写入后的最终落盘，是从redo log更新过去还是从buffer pool呢?

答：Buffer pool， 其实redo log并没有直接写磁盘数据页的能力，因为它没有整页的数据。

456、数据更新时,写入change buffer,立马读取这一条 不是应该直接从内存页读取吗,有必要把整个页读入内存吗,而且innoDB怎么知道应该加载哪一页到内存,我这一条记录并没有在数据页上有任何标示位啊。

答：1. 就是数据页没在内存，才能用上change buffer。

2. B+树有序的。能找到（你想，数据库一开始启动的时候，要找一个磁盘上的记录是怎么找到，一样的过程）。

457、前提：

1.order表包含Id，orderId，channelId三个字段，其中id为自增主键，orderId和channelId为联合非唯一索引（idx_order_channel），

事务隔离级别为Repeatable Read

2.表中包含三条数据：

id　OrderId　channelId

1678　JY005　1005

1679　JY010　1047

1680　JY015　1023

3.由于idx_order_channel为非唯一索引，下单时需要按照orderId_channelId删除数据，再插入数据。

现在有两个同时下单的请求，执行的事务如下：

事务1：begin;delete from order where orderId = 'JY007' and channelId = 1007;insert... ('JY007',1007);commit;

事务2：begin;delete from order where orderId = 'JY008' and channelId = 1025;insert... ('JY008',1025);commit;

请问这种情况会发生死锁吗？会有抢夺锁的情况发生不?

答：会死锁，有间隙锁gap lock冲突。

58、系统表空间跟数据表空间这两个概念各是什么意思?

答：系统表空间就是用来放系统信息的，比如数据字典什么的，对应的磁盘文件是ibdata1，

数据表空间就是一个个的表数据文件，对应的磁盘文件就是 表名.ibd。

459、不知道理解的对不对，比如update t set name=xxx where k=xxx,其中k是二级索引,如果是以下几种情况:

1.对于k是二级普通索引的情况,如果索引的数据页不在内存中,则写入到change buffer中,后续某些条件下(比如查找),会将该索引数据页读入内存,然后再回表更新记录的数据页中的name字段为xxx?

2.对于k是二级普通索引的情况,如果索引的数据页在内存中,然后回表更新记录数据页中的name字段为xxx?

3.对于k是二级唯一索引,如果索引的数据页不在内存中,将该索引数据页读入内存,然后回表更新记录数据页中的name为xxx?

答：如果用where里面用k索引的话，还是要把这一页读出来的，所以k这个页用不上change buffer，但是如果name也建了普通索引，那么更新name值这个行为可能被放入change buffer。

460、在pool里做更新，然后merge到磁盘，那么行锁是什么时候加上的?

答：更新之前加的了，行锁没有存在数据页里哦。

461、关于change buffer 主要节省的则是随机读磁盘的 IO消耗这个点，我的理解是如果没有change buffer机制，那么在执行更新后（写入redolog），读取数据的时候需要从次磁盘随机读取redolog合并到数据中，主要减少的是这部分消耗?

答：不是，如果没有change buffer, 执行更新的“当时那一刻”，就要求从磁盘把数据页读出来（这个操作是随机读）。Chsnge buffer省了这个。

462、可不可以这样理解，change buffer在merge时是将内存中的数据更新到最新，而redo log是将磁盘中的数据更新到最新？在内存中的数据失效之前，redo log是不是必须要将数据写盘，有这样一种机制来保障吗?

答：第一个理解很对，第二个，内存中的数据*淘汰*之前，需要写盘。

这里两个说明：

1）内存无所谓失效，就是被LRU淘汰了，数据页才会移出内存

2） 这时候不需要redolog做什么的，内存写到磁盘，直接就写了

463、随机io写，和顺序写有什么区别呢？为什么顺序写更加高效呢？而且这里的随机，顺序，指的是什么呢?

答：顺序写比随机写高效，是磁盘的机制决定的。顺序就是下一次写从上一次结束的位置继续。

464、如果更新后马上就要查询，使用change buffer就有点得不偿失，那这个更新后多长时间查询，change buffer才能真正提升效率呢？有个大概的范围吗？或者跟哪些因素有关？

答：读写比例吧。一般是二级索引多的、主要是insert数据场景的库，用change buffer效果好。

465、change buffer记录的是新增或修改的数据，那么merge操作是针对二级索引还是数据。如果说因为唯一索引需要确保数据唯一，那么普通索引应当也是需要的，毕竟插入操作避免不了要带去主键（或系统生成）。

答：普通索引不需要哦，就是说插入的时候，操作主键ID索引的变更确实不能用change buffer，但是普通索引可以。

466、 关于主键索引和二级索引数据同步的问题不太明白:

id = 1, k = 5; id = 2, k = 5

执行delete from T where k = 5;

假设k对应的page页不在change_buffer中, 这时候会在change_buffer中记录 delete from T where k = 5;

1、主键索引是本来就不能用上change buffer机制的，那么这时候是直接在主键对应的B+tree上删除 id in (1, 2)的数据? (随即写?)

2、如果执行delete from T where id = 1; 这时候会用到change_buffer, 在change_buffer中做标记delete from page2 where id = 1 ？ 类似这样吗? 如果这样 会首先把id=1的page页读到内存中吧(否则不知道删除的k = ?), page2 这个是怎么知道的? 会去K对应的B-Tree中找吗?

答：1. 是的，但是不是in(1,2), 都是一个个回表的删除的

2. 你说的Page2 是放普通索引的page吗？流程是这样，id=1主键索引取出来，读到k值，再去k索引树上删。如果发现page2 在内存就直接删除。如果发现不在，就往chang buffer中记删除动作。还是要在k索引找的，不找不知道在不在内存里。

467、普通索引的更新记录会放到change buffer中，这个过程也会存到redo log ，redo log会怎么处理这条记录？在往磁盘更新的时候忽略掉吗?

答：这条记录如果没有碰上崩溃恢复啥的，就看chepoint推进到它的时候，有没有写了盘，如果没有，就把change buffer 内容写盘（其实跟数据是一样的策略）。

468、场景如下：

假设表t有id，k 两列。id是主键，k是二级索引。

有数据(3,300)，(4,400)。(3,300)存放在数据块page3,(4,400)存放在数据块page4。这两行数据均不在buffer pool中。

现在做update操作，修改(3,300)为(3,400)，这时会把修改操作写入到到change buffer中；

并在change buffer merge之前做查询操作：select k from t where k=400;

因为数据不在buffer pool中，会到磁盘上读取数据，根据索引会读取到page4，并把page4数据加载到buffer pool中，然后应用change buffer时，发现遗漏了page3，再重新去取page3。

请问这个过程的推导是否正确，是否会出现这种多次IO的情况

答：按照你的描述，应该是假设“做update操作，修改(3,300)为(3,400)”这个操作，是要在page3删除这个记录，然后往page4插入一行对吧?

这两个操作都可能被放到change buffer里面。接下来如果查询是 where k=400， 就只会访问page 4，page 4从磁盘读入内存的时候，做merge操作，返回结果里面有两条满足k=400的记录，就完成了。这个查询过程跟page3没关系，不会发现“page 3遗漏”这个逻辑哦。

469、添加数据时主键问题。 看您在评论区中回复：insert的时候，写主键是肯定不能用change buffer了。 那请问什么情况下insert能用到？因为表中，基本都有ID主键，那insert语句是不是没有用上change buffer的情况了。

答： 一个insert语句会更新主键索引和其他普通索引，更新普通索引的时候，就有可能可以用上change buffer了。

470、针对同一个字段，建了唯一索引，还需要建普通索引吗?

答：尝试对同一个字段建了唯一索引和普通索引，跑SQL时发现两个索引都用上了，不是两个索引都用上了，是只用了一个，如果建了唯一索引，就不用再同样的建一个普通索引了。

471、由于mvcc机制，更新数据需要记录undo log，此时应该把要更新的数据页读入buffer里，如果记了change log 不读内存，之前还未提交的事务如何能拿到需要的数据呢?

答：如果能用上change buffer, 那么就不更新这个数据页，不需要记这个页的undo 了。

472、不是太明白读数据时是怎么判断当前要读的数据是否在change buffer中存在待merge的数据?

答：有个hash表，读出来的时候顺手判断。

473、“更新一个数据页时，在不影响数据一致性的前提下，innodb会将这些更新操作缓存在change buffer中。”请问具体有哪些操作呢？我想了一个场景，非唯一索引的更新操作，需要判断该行数据是否存在，这样还是需要将磁盘中的数据页读取到内存中呢。

答：不用判断，这一行一定存在，（除非出bug了）。

474、在没有索引的情况下，cahnge buffer会起作用吗？我觉得是会。

答：不会没有索引意味着就只有一个主键索引。

475、数据读入内存是需要占用 buffer pool 的，所以这种方式还能够避免占用内存，提高内存利用率。后面又说change是占用pool内存的，那到底占不占用buffer pool的内存?

答：1. change buffer本身是占用内存的；

2. 但是chage buffer本身只是记录了“更新过程”，远远比数据页（一个16k）小。相比于把数据页读入内存，这个方式还是省了内存的。

476、什么是数据表空间和系统表空间?

答：比如有一个表a，那么a.ibd 就是数据表空间；“表a的表结构”这个信息存在ibdata1的系统表空间里。

477、update操作不是先读后写吗？如果是先读的话，不是应该把数据已经读到内存了吗？那这样的话直接更新内存不就好了，为什么还要写change buffer?

答：change buffer就是为了避免先读后写，因为读有随机io的消耗。

478、文中讲change buffer中存的内容是“在某个数据页更新什么”，但是在update/insert时，确定这条记录更新/插入在哪个数据页，不也是有一个查找的过程么？（肯定有一个一层层查找的过程，会路过很多数据页啊）为了确定在哪个数据页操作而遍历过的数据页也会读进内存作缓存吗?

答：是的，查找过程避不开，但是二级索引树的非叶子节点没多少，主要在磁盘上的还是叶子节点。

479、【对于唯一索引来说，需要将数据页读入内存，判断到没有冲突，插入这个值，语句执行结束】那么这里是只读【索引页】呢，还是会连带【数据页】一起读入内存?

答：如果是判断唯一索引的，就只读【索引页】就可以的，

但是因为primary key也有唯一约束，所以这个”索引“也需要判断冲突。而主键索引上其实就是数据了。

480、建了唯一索引 changebuffer不会生效.然后要查询的数据也不在内存中 这时候 是不是就要合并redolog的数据 才能得到正确的结果了（假设这时候redolog还没刷盘）
答：这两个概念有点混淆了哦。如果数据没有刷盘，就一定会在内存中，查询过程不会需要去读redolog的（否则这个速度就太慢了）。

481、我用存储过程插入100000条数据特别慢，后来我set autocommit=0,每1000条才commit，这样就快了。我想不出来这是为什么。
答：Redo log 和 binlog刷盘次数少了，你把100000个事务变成了100个事务。

482、redo log 是实时写入磁盘的吗？是不是还有一层所谓的“redo log buffer”？
答：Redolog buffer是在事务执行过程中，先把要写入的内容在内存里存起来，在commit阶段，一次性写入redolog file

483、我的理解是由于B是查找(50000,100000),由于B+树有序,通过二分查找找到b=50000的值,从50000往右扫描,一条一条回表查数据,在执行器上做where a(1,1000)的筛选,然后做判断是否够不够limit的数,够就结束循环。由于这里b(50000,100000)必然不存在a(1,1000),所以需要扫描5W行左右.但是如果把a改为(50001,51000),扫描行数没有变。那么是因为优化器给的扫描行数有问题还是执行器没有结束循环？为什么不结束循环?
(好像rows能直观展示limit起作用,必须在执行器上过滤数据,不能在索引上过滤数据,不知道为什么这样设计)。
答：是的，加了limit 1 能减少扫描多少行，其实优化器也不确定，【得执行才知道】，所以显示的时候还是按照“最多可能扫多少行”来显示。

484、对于枚举属性（像：类型、标记这种只有少量值的字段）字段，按照文中的意思这样的字段建立索引的话区分度很低，基数也很小。那适合在这样的字段上建索引吗？我能想到的不合适的理由是：行级锁的范围会很大。
答：这是一个原因，还有就是区分度不好。业务上要尽量避免只有这一个条件的语句（如果实在有，那也没办法还是要建）。

485、假如要查 A in () AND B in (), 怎么建索引?

答：where A in (a,b,c) AND B in (x,y,z)会转成

(A=a and B=x) or (A=a and B=y) or (A=a and B=z) or

(A=b and B=x) or (A=b and B=y) or (A=b and B=z) or

(A=c and B=x) or (A=c and B=y) or (A=c and B=z)

486、下面这条SQL order_id 和 user_id 都是 int 类型，都加了索引，在看 EXPLAIN 的时候执行只使用了 order_id 索引， 为什么 user_id 索引没有采用呢? 如果WHERE 条件后面都有索引是否都会执行、还是优化器会选择最有效率都一个索引执行? 将两个调整成组合索引也没有效果, 如果 force index(user_id) 则全表扫描。

SELECT count(1) FROM A a WHERE EXISTS(SELECT 1 FROM B b WHERE b.order_id = a.order_id AND b.user_id = a.user_id);

答：只能用一个索引，如果两个都用就是merge-index算法，一般优化器很少采用。

487、在评估前缀索引长度的时候，如果是新表的话，表中还没有数据，这个时候没法估计区分度，只能按照业务先去设计，后续再优化吗？还有其他什么评估办法吗?

答：跟业务负责人确定数据分布和查询模式。其实这个不止是评估前缀索引长度时需要，整个表的索引该怎么设计，都是应该事先根据数据分布和查询来决定的。

488、char 和 varchar 可以设置长度，这个长度是干什么的，对于不同字符集又有什么影响?

答：char（N）表示“最长存N，但是如果字符串小于N，用空格补到N”，

varchar（N）表示“最长存N，如果字符串小于N，按照实际长度来存”。

489、“内存不够用了，要先将脏页写到磁盘“和“redo log 写满了，要 flush 脏页”可以理解为一个脏页本身占用内存，释放内存需要将脏页写入到磁盘才能释放。而redo log写满只有当redo log对应的脏页flush到磁盘上才能释放对应空间。有几个问题：

1、“内存不够用了，要先将脏页写到磁盘“redo log对应的空间会释放嘛？“redo log 写满了，要 flush 脏页”对应的内存页会释放嘛?

2、将脏页flush到磁盘上是直接将脏页数据覆盖到对应磁盘上的数据？还是从磁盘上取到数据后取根据redo log记录进行更新后再写入到磁盘?

3、redo log是怎么记录对应脏页是否已经flush了？如果断电了重启导致内存丢失，前面几章说通过redo log进行数据恢复那redo log又怎么去释放空间?

答：1. Redolog 的空间是循环使用的，无所谓释放。 对应的内存页会变成干净页。但是等淘汰的时候才会逐出内存；

2. 好问题，前者；

3. 不用记，重启了就从checkpoint 的位置往后扫。 如果已经之前刷过盘的, 不会重复应用redi log。

490、当内存不够用了，要将脏页写到磁盘，会有一个数据页淘汰机制（最久不使用），假设淘汰的是脏页，则此时脏页所对应的redo log的位置是随机的，当有多个不同的脏页需要刷，则对应的redo log可能在不同的位置，这样就需要把redo log的多个不同位置刷掉，这样对于redo log的处理不是就会很麻烦吗？（合并间隙，移动位置？）

另外，redo log的优势在于将磁盘随机写转换成了顺序写，如果需要将redo log的不同部分刷掉（刷脏页），不是就在redo log里随机读写了么?

答：其实由于淘汰的时候，刷脏页过程不用动redo log文件的。这个有个额外的保证，是redo log在“重放”的时候，如果一个数据页已经是刷过的，会识别出来并跳过。

491、我理解 redolog 和 undolog 是在有数据更新时同时写的，redolog 作为 crash-safe 机制，每次全部读入做恢复就好。但是像在做脏页flush 或者 mvcc版本回退的时候，难道是按照涉及的数据行来定位日志行的吗？如果是，是怎么做到这种“定位”的呢？如果不是，InnoDB 的机制又是怎样的?

答：Flush 和 mvcc不能放一起说哈，不一样的。每一行存了一个位置可以直接找到undo log, 这个MVCC的基础，Redolog里面记录了“这是哪个页面的修改”。

492、innodb是如何知道一个页是不是脏页的，是有标记位还是通过redolog的ckeckpoint来确定的?

答：每个数据页头部有LSN，8字节，每次修改都会变大。对比这个LSN跟checkpoint 的LSN，比checkpoint小的一定是干净页。

493、您提到通过对比数据页的 LSN 和 checkpoint 的 LSN，比checkpoint LSN 小的就是干净页，那在重放 redo log 时是如何跳过那些由于内存不足而刷脏的页面的呢？刷脏的时候会将 LSN 重置为0之类的?

答：判断一下发现是干净页就跳过。不是重置为0，是把磁盘上数据页的LSN也一起改掉了。

494、在一个事务回滚操作里如果有DDL操作会怎样。例如如下步骤：1.开始事务及设置手动提交；2.修改表Aid为2的行；3.动态创建一张表B；4.修改表C里id为5的行；6.提交事务及回滚代码编写。如果在步骤5时发生异常数据库会怎样做?

答：第三步会自动提交事务的。如果你是autocommit=1的话，其实4也是单独事务提交了。

495、系统内存不够 和 缓冲池内存的关系是 ？ 缓冲池可以无限扩大内存将系统内存沾满么？

答：不能扩大，buffer pool size 设定好就只能用这么多，不会再多吃系统内存。

496、用fio测了一下，iops 在1500左右，MySQL的innodb_io_capacity的缺省值为200，但还有一个参数是innodb_io_capacity_max的参数，值是2000。这种情况下，系统算正常吗？max参数是怎么算出来的?

答： Max只是用来控制 innodb_io_capacity的值不能设置超过这个值，真正生效的还是innodb_io_capacity。

497、访问某条记录时，存储引擎是如何判断这条记录所在的数据页是否在内存当中，这个查内存机制是如何实现的?

答：每个页面有编号的。拿着编号去内存看，没有，就去磁盘。

498、WAL 将随机写转换成顺序写，这里顺序写是指 redo log 的顺序写，还是还会把多个脏页调整成顺序写。如果是会把脏页调整成顺序写，这个是操作系统本来就有的功能，还是 MySQL 实现的。（操作系统学得一般:()

答：这两个顺序写都是WAL机制的收益，刷脏页是Innodb 实现的，没有依赖操作系统。

499、我们需要考虑的是内存脏页flush 速率，但是如何避免出现擦黑板，对应的是数据库的 check point 移动造成的整个系统无法更新呢?

答：就是io_capacit不要设置太小。如果是磁盘性能问题导致checkpoinnt推进太慢，就得换磁盘了。

500、每个数据页上都有Lsn，数据库运行时数据页上的lsn应该是千差万别的吧？若是数据库正常关闭，然后重新启动，这是所有数据页的lsn是一致的？还是保持数据库关闭时千差万别的?

答：每个数据页的lsn不同的，表示“更新这个page的最后一个lab”，不需要相同。