


1222模拟面试解析

来自： 小6互联网求职面试

 马听

2023年12月26日 12:10



扫码加入
查看更多优质内容

1 MySQL 5.6 5.7 8.0分别有哪些特点

MySQL 5.6，引入了GTID，引入了基于库的并行复制，引入了ICP，MRR等特性，Undo Log可保存在独立表空间中

MySQL 5.7，增加了组复制、基于WRITESET的并行复制

，原生支持JSON类型，支持动态调整很多全局参数（innodb_buffer_pool_size、max_connections、connect_timeout、max_user_connections等）、支持临时表设置为InnoDB存储引擎

MySQL 8.0，增加了事务性数据字典、大表秒级加列、原子DDL、角色管理、通过 set persist 命令将全局变量的修改持久化到配置文件中、hash join、额外端口、自增主键的持久化

2 半同步复制和增强半同步复制的区别

after commit的大致流程

SQL解析

存储引擎处理：执行SQL操作

写Redo Log（Prepare）阶段：事务的更改被准备好（prepare），但还没有提交到数据库；

写binlog：记录操作的Binlog；

把Binlog里的内容传给从库的中继日志；

Redo Log提交：事务在InnoDB存储引擎中被提交。这时，更改正式应用到数据库文件中；

从库发送 ACK：从库收到 binlog 后，发送给主库一个 ACK，表示收到了

主库确认提交：主库收到这个 ACK 以后，才向客户端返回 commit 成功；

增强半同步复制after_sync的大致流程：

SQL解析

存储引擎处理：执行SQL操作，但不提交事务

写Redo Log（Prepare）阶段：事务的更改被准备好（prepare），但还没有提交到数据库；

写binlog

把Binlog里的内容传给从库的中继日志

等待从库确认：等待从库成功接收binlog的返回ACK信号

Redo Log提交：在收到从库的 ACK 信号后，事务在InnoDB存储引擎中被提交。这时，更改正式应用到数据库文件中；

反馈至客户端：确认提交后向客户端返回成功信息

区别

after commit侧重于事务在本地数据库的提交，而after sync（增强半同步复制）则确保了主从数据库之间的数据一致性

after sync特别适用于对数据完整性有严格要求的场景，比如金融场景。

3 MDL的作用是什么？

避免了在结构更改过程中数据不一致或损坏的风险

当一个事务正在读取一个表时，MDL 会阻止其他事务对该表执行结构更改，保证语句执行期间的数据一致性

防止Binlog乱序的情况出现

MySQL 5.5.3之前，没有MDL锁存在的问题。比如下面这个实验：

步骤	session1	session2
1	begin;	
2	insert into t ...;	drop table t;
3	commit;	

而落到MySQL的Binlog

就是

drop table t;

begin;

insert into t ...;

commit;

就出现了这样位置错乱的Binlog，如果传到从库，就会导致复制中断。

4 在一个事务中，查询了某张表的数据，事务没提交，其他会话再执行这张表的DDL操作（比如drop table 操作），会出现什么情况？

DDL会等待

原因：

所有DML操作会在表上加一个MDL读锁

所有的DDL操作都会在表上加一个MDL写锁

读锁和写锁之间相互阻塞

即同一个表上的dml和ddl之间互相阻塞。

写锁和写锁之间互相阻塞

即两个session不能对表同时做表定义变更，需要串行操作。

读锁和读锁之间不会产生阻塞

也就是增删改查不会因为metadata lock产生阻塞，可以并发执行，日常工作中大家看到的dml之间的锁等待，是InnoDB行锁引起的，和MDL无关。这个锅他不背。

5 从库延迟，有哪些处理办法？

调整从库的innodb_flush_log_at_trx_commit和sync_binlog，暂时不要设置为双一

开启并行复制

优化查询，避免大事务

增加从库资源

我们有时也会遇到这种场景，比如某张表比较大，延迟经常是这张表导致的，从库上读取数据时又用不上这张表，就可以考虑把大表单独创建一个从库进行复制。然后在原来的从库忽略这张表的复制，业务查询原来的从库就基本没延迟了

6 xtrabackup的备份和恢复原理

备份原理

（1）先记录日志序列号（LSN），然后复制InnoDB数据文件。同时，运行一个后台进程来监视Redo Log，如果Redo Log有修改，则从中复制。

（2）加备份锁（使用lock instance for backup语句，8.0才有）。

- (3) 备份完事务引擎的数据和日志后，锁定MyISAM和其他非InnoDB表。
- (4) 复制非事务引擎的表数据文件。
- (5) 查询GTID信息和Binlog位点（使用select server_uuid, local, replication, storage_engines from performance_schema.log_status语句）。
- (6) 停止复制Redo Log。
- (7) 释放锁（使用unlock instance语句）。
- (8) 复制ib_buffer_pool。
- (9) 备份完成。

恢复原理

- (1) 首先是prepare
- (2) 模拟MySQL进行崩溃恢复，将Redo Log回放到数据文件中
- (3) 等恢复完成
- (4) 重建redo log

7 在数据备份过程中遇到过哪些问题？

根据实际情况讲

比如：

备份时间过长，就需要考虑增量备份

备份机器磁盘满了（这种就需要考虑怎么规避这种情况，比如预估当天备份量，看是否有足够空间存放备份文件，不足则提前告警）

8 mysqldump参数--master-data几个值的区别

- 1，则在备份文件中写入change replication source to语句或者change master to语句（恢复时会直接执行）
- 2，则会在备份文件中写入注释的change master语句
- 0，则不会记录change master

9 MGR是怎样判断主库故障的？

MGR故障检测

节点互相探测

MGR的节点之间会互相发送检测消息，当节点A在给定的时间内没有接收到节点B的消息时，就会把节点B标记为可疑节点。

多数节点认为某个节点出现故障

如果集群中有多数节点认为B节点可疑，那么小组就会认定B节点确实出现故障

驱逐故障节点

确定某个节点故障之后，就会吧节点驱逐出集群，随后节点状态会变成error，并设置为只读。

判断group_replication_autorejoin_tries参数设置的值

如果值不为0

比如为3，那就意味着成员自动进行3次重新加入组的尝试，每次尝试间隔5分钟。

在自动重连接尝试期间和之间，成员保持超级只读模式并且不接受写操作，但是仍然可以对成员进行读操作。如果达到指定的尝试次数后，该成员没有重新加入，会判断group_replication_exit_state_action设置的值。

如果值为0

则判断group_replication_exit_state_action的值，但是这之后，需要手动干预把成员加会组中

设置为read_only时，会把这个实例的super_read_only设置为on；

设置为offline_mode时，会把这个实例切换到离线模式

设置为abort_server时，将关闭MySQL。

10 MGR选主逻辑是怎样的？

在单主模式下，如果主出现故障，则会考虑下面的因素选择新主。

(1) 考虑的第一个因素是哪个节点运行的是最低的MySQL版本。如果所有节点都运行MySQL 8.0.17或更高版本，那么组节点将按照发布的补丁版本进行排序。如果所有节点运行MySQL 8.0.16或更低版本，那么组节点将按照其发布的主要版本排序，并忽略补丁版本。低版本优先考虑将高版本同步到低版本，高版本可能有一些新特性无法在从库正常回放，导致同步出现问题。

(2) 如果有多个节点版本一样，则要考虑的第二个因素是每个节点的权重，由 group_replication_member_weight 参数指定。如果运行 MySQL 5.7，则group_replication_member_weight参数不可用，将不考虑这个因素。

group_replication_member_weight参数指定一个范围为0~100的数字。值越大，权重越大。

(3) 如果前面两个因素都一样，则需要考虑每个节点生成的UUID的顺序，如果指定了server_uuid系统变量，则选择UUID排序最靠前的节点作为主节点。

在多主模式下，如果一个节点出现故障，连接到它的客户端可以重定向或将故障转移到处于读/写模式的任何其他节点。Group Replication本身并不处理客户端故障转移，因此需要使用中间件框架，如MySQL Route。

11 MGR可以设置哪些一致性级别？

可以通过配置group_replication_consistency参数来配置一致性级别，具体可配置的值及含义如下。

EVENTUAL

事务提交后会广播到集群的多数节点，然后节点检查是否有冲突，如果没有冲突，则事务在本地提交，其他节点异步处理，可能导致读取到稍旧的数据

BEFORE_ON_PRIMARY_FAILOVER

在主节点故障时，必须等待新主处理完待处理的事务，才能开始响应业务的读写请求，这样可以保证业务读写请求不会读取到旧数据

BEFORE

一个事务会等待之前的事务执行完后再开始执行，确保读取到的数据是最新的。

AFTER

写事务会等待其更改在所有其他节点应用后才提交，保证后续事务读取已写入或其他节点上最新的值。对只读事务没有影响

BEFORE_AND_AFTER

会等待之前的事务执行完后才开始执行新事物，并等到事务在所有节点应用后才提交，确保读取和提交都具有强一致性。

12 主从复制遇到过什么问题？

当主有问题切换之后，发现新主和老主有一条数据主键冲突，除了主键值一样，其他字段的数据不同

这种情况，需要跟研发确定，主键有没有业务意义（不过一般建议主键是不要有业务意义的），如果没有业务意义，那就把从库的这行数据用mysqldump备份成SQL语句之后，

在删除这行记录，如果配置的是主主架构，建议删除的时候忽略Binlog，不然这个删除操作还是会同步到主库

然后从库在重启复制，stop slave;start slave;一般复制就会恢复了；

然后再把从库备份的记录导入到主库，这个时候不加主键，主键让他自增。

然后两条记录都写入到数据库里面了。只是有一条记录其他字段一样，主键变了。

当然，为了避免这种情况出现，建议keepalived+主主的架构，使用跳主键的策略。主 1、3、5，从2、4、6。

13 日常工作中遇到过比较棘手的MySQL问题

则个结合自己实际情况讲

我们之前也讨论过这个问题，有几位嘉宾讲了自己的思路：

https://wx.zsxq.com/dweb2/index/topic_detail/811244415458412

14 假如现在新上线了一套MySQL 8.0，你会调整和优化哪些参数？

(1) 缓冲池大小

innodb_buffer_pool_size,

一般建议设置机器内存的60%–80%

(2) 最大连接数

max_connections

结合实际业务设置；

(3) binlog相关参数

一定要保证Binlog开启的：

log-bin = /data/mysql/binlog/mysql-bin,

Binlog格式：

binlog_format = row,

Binlog大小：max_binlog_size=1G,

Binlog自动过期时间：expire_logs_days

(4) 双一参数

sync_binlog和innodb_flush_log_at_trx_commit,

数据重要的，设置为双一，

数据相对不重要，比如日志数据库，性能优先的，设置为100、2；

(5) 开启gtid：

gtid_mode=on

和enforce_gtid_consistency=on

(6) 采用独立表空间

innodb_file_per_table = 1

(7) 不区分大小写

lower_case_table_names = 1

(8) 每秒刷新脏页的数量

innodb_io_capacity

(9) 重做日志大小

innodb_log_file_size，一般建议2G