Mysql 连环20问

苏三说技术 今天

以下文章来源于悟空聊架构 ， 作者悟空聊架构

悟空聊架构
用故事讲解分布式、架构。 《 JVM 性能调优实战》专栏作者， 《Spring Cloud 实战 ...

大家好，我是苏三

一、说下 MySQL 的 redo log 和 binlog?

#每天一道面试题# 35

#悟空拧螺丝# 2021-08-16

（1）MySQL 分两层：Server 层和引擎层。区别如下：

Server 层：主要做的是 MySQL 功能层面的事情。Server 层也有自己的日志，称为 binlog（归档日志）

引擎层：负责存储相关的具体事宜。redo log 是 InnoDB 引擎特有的日志。

（2）redo log 是物理日志，记录的是“在某个数据页上做了什么修改”；binlog 是逻辑日志，记录的是这个语句的原始逻辑，比如“给 ID=2 这一行的 c 字段加 1 ”。

（3）redo log 是循环写的，空间固定会用完；

（4）binlog 是可以追加写入的。“追加写”是指 binlog 文件写到一定大小后会切换到下一个，并不会覆盖以前的日志。

二、说说建立索引的优势、负面影响和原则?

#每天一道面试题# 37

#悟空拧螺丝# 2021-08-18

（1）索引的优势?

检索速度：快速访问数据表中的特定信息，提高检索速度。

唯一性：创建唯一性索引，保证数据库表中每一行数据的唯一性。

加速连接：加速表和表之间的连接。

减少分组和排序的时间：使用分组和排序进行数据检索时，可以显著减少查询中分组和排序的时间。

（2）索引的负面影响?

耗时：创建索引和维护索引需要耗费时间，这个时间随着数据量的增加而增加。

占空间：索引需要占用物理空间，不光是表需要占用数据空间，每个索引也需要占用物理空间。

维护速度：当对表进行增、删、改、的时候索引也要动态维护，这样就降低了数据的维护速度。

（3）为数据表建立索引的原则有哪些?

在最频繁使用的、用以缩小查询范围的字段上建立索引。

在频繁使用的、需要排序的字段上建立索引。

（4）什么情况下不适合建立索引?

对于查询中很少涉及的列或者重复值比较多的列，不宜建立索引。

对于一些特殊的数据类型，不宜建立索引，比如文本字段（text）。

三、说下 MySQL中的索引有哪些分类?

#每天一道面试题# 40

#悟空拧螺丝# 2021-08-21

MySQL的所有列类型都可以被索引。

MyISASM和InnoDB类型的表默认创建的都是BTREE索引;

MySQL中的索引是在存储引擎层中实现的，而不是在服务器层实现的。所以每种存储引擎的索引都不一定完全相同，也不是所有的存储引擎都支持所有的索引类型。

MySQL目前提供了以下几种索引。

1）BTREE索引：最常见的索引类型，大部分引擎都支持BTREE索引，例如MyISASM、InnoDB、MEMORY等。

2）HASH索引：只有MEMORY和NDB引擎支持，适用于简单场景。

3）RTREE索引（空间索引）：空间索引是MyISAM的一个特殊索引类型，主要用于地理空间数据类型，通常使用较少。

4）FULLTEXT（全文索引）：全文索引也是MyISAM的一个特殊索引类型，主要用于全文索引，InnoDB从MySQL 5.6版本开始提供对全文索引的支持。

四、说下使用索引的推荐原则有哪些?

#每天一道面试题# 41

#悟空拧螺丝# 2021-08-22

（1）最适合索引的列是出现在WHERE子句中的列，或连接子句中指定的列，而不是出现在SELECT关键字后的选择列表中的列。

（2）使用唯一索引。唯一性索引的值是唯一的，可以更快速的通过该索引来确定某条记录。

（3）不要过度索引。因为每个索引都要占用额外的磁盘空间，并降低写操作的性能，增加维护成本。在修改表的内容时，索引必须进行更新，有时也可能需要重构，因此，索引越多，维护索引所花的时间也就越长。

（4）为经常需要排序、分组和联合操作的字段建立索引。

（5）删除不再使用或者很少使用的索引。

（6）利用最左原则。mysql建立多列索引（联合索引）有最左前缀的原则，即最左优先，如：

如果有一个2列的索引(col1,col2),则已经对(col1)、(col1,col2)上建立了索引；

如果有一个3列索引(col1,col2,col3)，则已经对(col1)、(col1,col2)、(col1,col2,col3)上建立了索引。

五、说下 MySQL 覆盖索引?

#每天一道面试题# 44

#悟空拧螺丝# 2021-08-26

概念：如果一个索引包含（或者说覆盖了）所有满足查询所需要的数据，那么就称这类索引为覆盖索引（Covering Index）。在 MySQL 中，可以通过使用 explain 命令输出的

Extra 列来判断是否使用了索引覆盖查询。若使用了索引覆盖查询，则 Extra 列包含“Using index””字符串。

大白话解释：select 的数据列只用从索引中就能够取得，不必从数据表中读取，换句话说查询列要被所使用的索引覆盖。

优点:

1）覆盖索引能有效地提高查询性能，因为覆盖索引只需要读取索引而不需要再回表读取数据。MySQL查询优化器在执行查询前会判断是否有一个索引能执行覆盖查询。

2）索引项通常比记录要小，所以MySQL会访问更少的数据。

补充:

不是所有类型的索引都可以成为覆盖索引。覆盖索引必须要存储索引的列，而哈希索引、空间索引和全文索引等都不存储索引列的值，所以MySQL只能使用B-Tree索引做覆盖索引

六、说下联合索引的各种匹配规则?

#每天一道面试题# 48

#悟空拧螺丝# 2021-09-01

（1）等值匹配

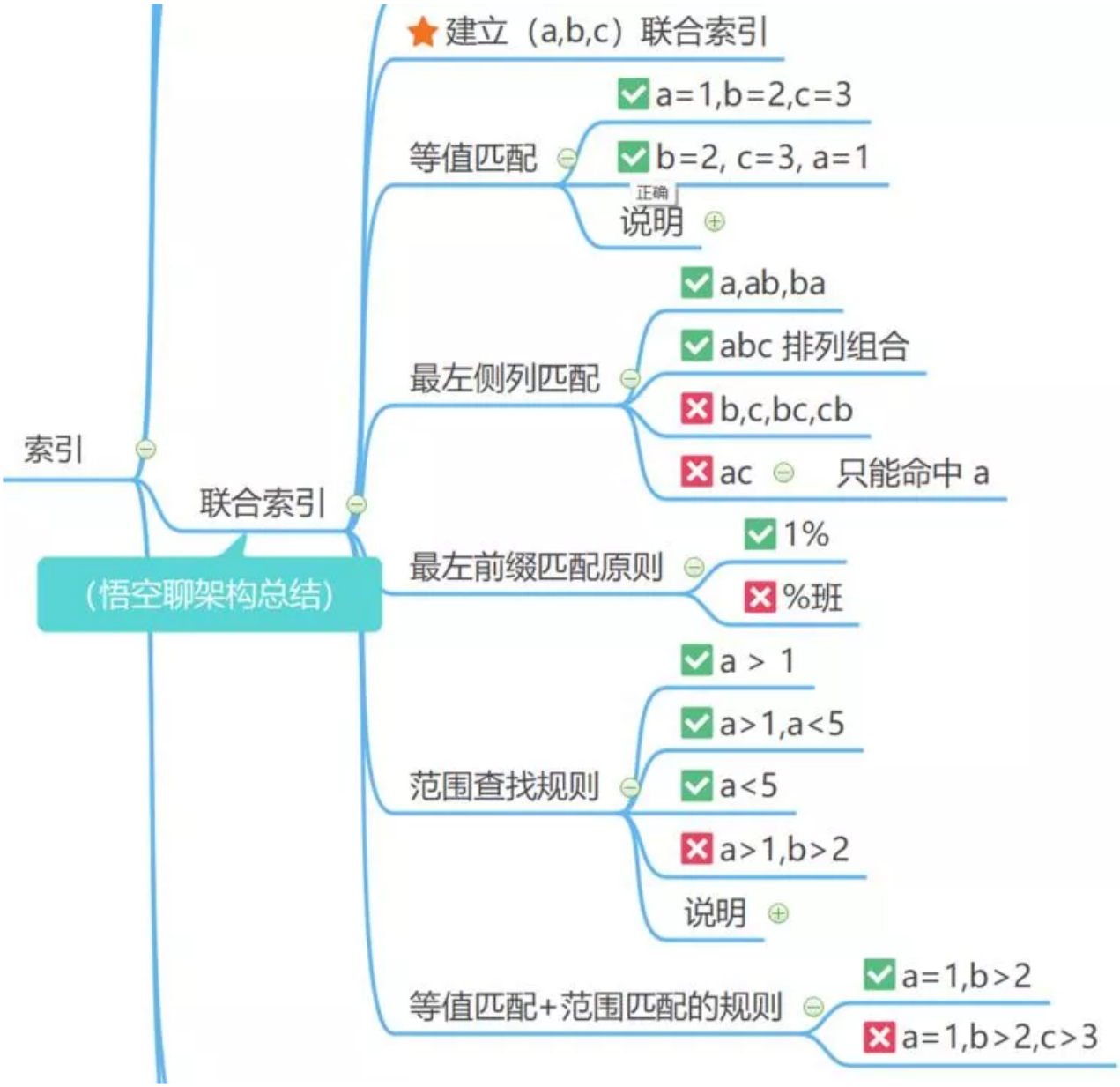
（2）最左侧列匹配

（3）最左前缀匹配原则

（4）范围查找规则

（5）等值匹配+范围匹配的规则

比如建立了 (a、b、c) 联合索引，那么有以下几种情况可以用到索引：



（对勾✓表示索引全命中）

七、说下 MySQL 回表?

#每天一道面试题# 49

#悟空拧螺丝# 2021-09-02

回表查询：先到普通索引上定位主键值，再到聚集索引上定位行记录，它的性能较扫一遍索引树低（一般情况下）。

详细说明：

一般我们自己建的索引不管是单列索引还是联合索引，都称为普通索引，相对应的另外一种就是聚簇索引。每个普通索引就对应着一颗独立的索引B+树，索引 B+ 树的节点仅仅包含了索引里的几个字段的值以及主键值。

根据索引树按照条件找到了需要的数据，仅仅是索引里的几个字段的值和主键值，如果用 select * 则还需要很多其他的字段，就得走一个回表操作，根据主键再到主键的聚簇索引里去找，聚簇索引的叶子节点是数据页，找到数据页里才能把一行数据的所有字段值提取出来。

假设 select * from table order by a,b,c 的语句，（table 有 abcdef 6 个字段），首先得从联合索引的索引树里按照顺序 a、b、c 取出来所有数据，接着对每一条数据都根据主键到聚簇索引的查找，其实性能不高。

有时候 MySQL 引擎会觉得用了既用了联合索引和聚簇索引来查找指定的字段，太慢了，那不不如直接全表扫描得了，只用聚集索引就行。

聚簇（聚集）索引补充：

只有一个聚集索引，聚簇索引的叶子节点存储行记录。根据聚簇索引的 key 查找是非常快的。

（1）如果表定义了主键，则主键就是聚集索引；

（2）如果表没有定义PK，则第一个 not NULL unique 列是聚集索引；

（3）否则，InnoDB 会创建一个隐藏的 row-id 作为聚集索引。

八、说下 MySQL 最左匹配原则知道吗?

#每天一道面试题# 46

#悟空拧螺丝# 2021-08-28

最左优先，以最左边的为起点任何连续的索引都能匹配上。同时遇到范围查询(>、<、between、like)就会停止匹配。

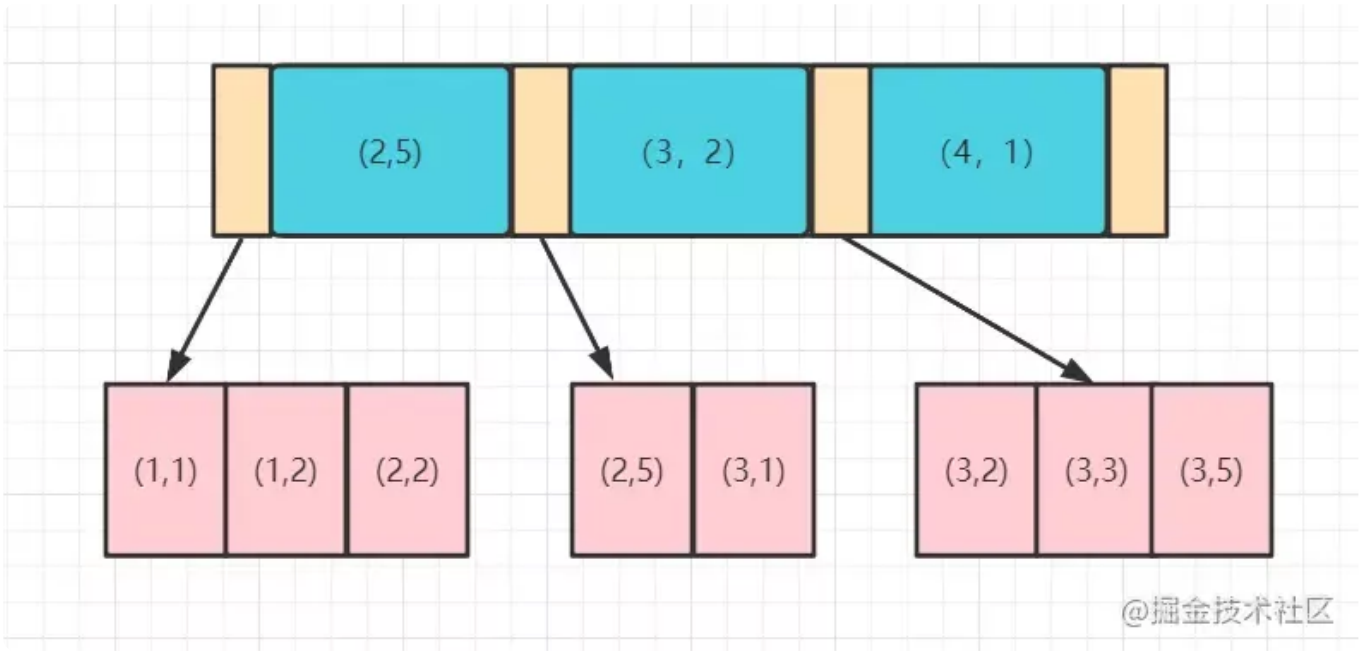
比如有联合索引 [a、b、c]，where 过滤条件中哪些排列组合可以用到索引？（比如这种：where a=xxx b=xxx and c=xxx)

以下排列组合都会走索引：a、ab、ac、ba、ca、abc、acb、bac、bca、cab、cba。必须有一个 a，排列组合中的顺序会被优化器优化，所以不用关心顺序。

以下排列组合不会走索引：b、c、bc、cb。因为没有 a。

关于范围查询：a=xxx and b<10 and b > 5 and c =xxx，c 字段用不到索引，因为 b 是一个范围查询，遇到范围查询就停止了。

最左匹配原则的原理：我们都知道索引的底层是一颗 B+ 树，那么联合索引当然还是一颗 B+ 树，只不过联合索引的健值数量不是一个，而是多个。构建一颗B+树只能根据一个值来构建，因此数据库依据联合索引最左的字段来构建B+树。例子：假如创建一个（a,b)的联合索引，那么它的索引树是这样的，如下图所示：



可以看到 a 的值是有顺序的，1，1，2，2，3，3，3，3。b 的值是没有顺序的1，2，2，5，1，2，3，5。

所以 b = 2 这种查询条件没有办法利用索引，因为联合索引首先是按a排序的，b是无序的。

同时我们还可以发现在a值相等的情况下，b值又是按顺序排列的，但是这种顺序是相对的。所以最左匹配原则遇上范围查询就会停止，剩下的字段都无法使用索引。例如 a=1 and b=2 a,b 字段都可以使用索引，因为在 a 值确定的情况下 b 是相对有序的，而 a>1 and b=2，a 字段可以匹配上索引，但 b 值不可以，因为a的值是一个范围，在这个范围中b是无序的。

九、说下你在使用索引上遇到的一些问题?

#365天学习打卡# 65

#阳光下的喵# 2021-08-29

索引的出现是为了提高查询效率，但是使用索引也存在一些常见的思维误区:

用索引 和 用索引快速查询 是有区别的，查询SQL经常提到全表扫描效率低，这个全表扫描默认指主键索引全表扫描，实际操作中非聚簇索引也存在全表扫描；

覆盖索引简单点理解就是查询的条件和要查询的值都在同一颗索引树上这种现象称为覆盖索引可以通过执行计划查看，和索引类型是不是唯一索引，关联索引没有联系；覆盖索引是经常用来做索引优化的一种手段(覆盖索引也存在慢操作）；

最左前缀 - 关联索引时候使用，这个最左指的不是字段上的最左，可以是字符串级别的；比如 name，age 是关联索引 name 用模糊查询也是最左前缀；索引下推(Icp) - 5.6以后引入的机制，关联索引使用，不过 ICP 在explain执行索引计划时候 可能没遇到 ICP，也会显示 using index condition；

回表 - 查完索引要去主键索引树上查一下，多了一次磁盘 io；

小问题:

“

person 表中有 id、name、age、gender四个属性id为主键索引，name，age组合为联合索引；这条SQL属于 回表、覆盖索引、最左前缀、索引下推中的那些场景？

”

```
select id，name，age from person where name like '赵%';
```

参考答案：

① 没有回表！因为关联索引会存储主键id。

② 查询列的数据都能在关联索引所在的索引树上查到，所以满足覆盖索引的条件；

③ 字符串可以通过前n个字符检索数据进行索引前缀查询，所以也满足最左前缀；

④ 不满足索引下推，索引下推是5.6引入的优化机制，为了减少回表次数；没有发生回表操作也就不存在索引下推；

十、说下索引条件下推（ICP）？

#365天学习打卡# 66

#阳光下的喵# 2021-08-30

首先声明一点索引下推不是只存在关联索引中，普通索引也可以执行索引下推；一般提到索引下推条件反射的都会先想到关联索引查询；

Using index condition 可以理解为 ICP 的必要不充分条件；即执行计划Extra中出现 Using index condition，但是SQL语句却不一定发生索引下推，using index condition 我理解的是“可以下推”，也就是说“在执行逻辑上可以下推，但不一定非要执行索引下推这个操作”；

表结构：person 表中有四个字段，id，name，age，gender，其中 id为主键，name，age 关联索引；

```
-----------------------------------------------------
    id              name          age             gender
-----------------------------------------------------
    1                 a           1
    2                ab           2
    3               abc           3
    4              abcd           4
    5                 b           5
    6                bc           6
    7               bcd           7
    8              bcde           8
    9                 c           9
    10               cd           10
    11              cde           11
    12             cdef           12
    13                d           13
    14                a           14
    15               ab           15
-----------------------------------------------------
```

举个索引下推的简单例子：

存在两条记录 它们name都是 ab，age 值分别是 2，15；

```
select * FROM person WHERE name like 'ab%' and age = 2;
```

这条sql在回表时只查询一条数据，查找到所有name是 ab 的记录以后，又作了age=2的判断才进行回表查询；5.6以前是查到所有name是 ab 的两条记录以后直接回表然后和age做对比。

十一、说下 MyISAM 和 InnoDB 的区别?

#每天一道面试题# 38

#悟空拧螺丝# 2021-08-19

MySQL 支持多种存储引擎， MyISAM 和 InnoDB 存储引擎只是其中的两种。

MyISAM 存储引擎:
5.5.8 版本之前的默认引擎，支持全文检索、压缩、空间函数等。不支持事务和行级锁，所以一般用于有大量查询少量插入的场景来使用，而且 MyISAM 不支持外键，并且索引和数据是分开存储的。

InnoDB 存储引擎:

5.5.8 版本之后的默认引擎，实现了 SQL 标准的四种隔离级别，默认为 REPEATABLE 级别。基于聚簇索引建立的，和 MyISAM 相反它支持事务、外键，并且通过 MVCC 来支持高并发，索引和数据存储在一起。

十二、说下 MySQL 的 Buffer Pool 的工作原理?

#每天一道面试题# 39

#悟空拧螺丝# 2021-08-20

MySQL 先把磁盘里面的数据加载到 Buffer Pool 中，增删改都是基于 Buffer Pool 里面的内存数据进行操作的，内存的效率比 IO 高很多倍。改了内存数据后，再定期刷新到磁盘。

Buffer Pool 有三大双端链表：free、flush、lru 链表。

- free 主要指向空闲缓存页。
- flush 指向已修改的缓存页。
- lru 指向被修改的缓存页，并根据最近最少使用的规则进行排序。

流程:

一边不停地加载数据到缓存页里去，一边不停地查询和修改缓存数据，然后free链表中的缓存页不停地减少↓，flush链表中的缓存页不停地增加↑，lru链表中的缓存页不停的在增加↑和移动←→。另外一边，后台线程不停的把 lru 链表的冷数据区域的缓存页以及 flush 链表的缓存页，刷入磁盘中来清空缓存 页，然后flush链表和lru链表中的缓存页在减少↓，free链表中的缓存页在增加↑。

十三、说下 InnoDB 存储引擎中的锁?

#每天一道面试题# 47

#悟空拧螺丝# 2021-08-31

行级锁：共享锁（S Lock），允许事务读一行数据。排他锁（X Lock），允许事务删除或更新一行数据。

表级锁：意向共享锁（IS Lock），事务想要获得一张表中某几行的共享锁。意向排他锁（IX Lock），事务想要获得一张表中某几行的排他锁。

	共享锁（S）	排他锁（X）	意向共享锁（IS）	意向排他锁（IX）
共享锁（S）	兼容	冲突	兼容	冲突
排他锁（X）	冲突	冲突	冲突	冲突
意向共享锁（IS）	兼容	冲突	兼容	兼容
意向排他锁（IX）	冲突	冲突	兼容	兼容

补充:

1、若将上锁的对象看成一颗树，那么对最下层的对象上锁，就是对最细粒度的对象进行上锁，首先就需要对粗粒度的对象上锁，

比如需要给某记录上 X 锁，那么就需要先对数据库 A、表、页 上意向锁 IX，最后对对记录上 X 锁。2、默认读操作不加锁，走 MVCC 多版本控制机制。

3、为什么要有意向锁？如果加了行锁，肯定之前就会给加上意向锁，有其他事务想要锁住表，先看有没有表级意向锁，这样就不用到记录上看有没有 S 或 X 锁，优点就是快，省了很多步骤。

举个生活中的例子:

图书馆有很多层，每一层有很多房间，二楼的 201 房间正在装修，不能进入，相当于给 201 房间加了排他锁，然后在二楼门口立了一个警示牌：二楼有人正在装修（相当于加了一个意向排他锁）。到图书馆关门的点了，管理员开始检查各楼栋是否还有人，发现二楼有个警示牌，呀，还有人在装修啊，暂时就不能关门了。这里把 201 房间当做某行记录，二楼当做表。装修比作事务 1，管理员要关闭图书馆比作事务 2。

文绉绉解释：

IS、IX锁是表级锁，它们的提出仅仅为了在之后加表级别的S锁和X锁时可以快速判断表中的记录是否被上锁，以避免用遍历的方式来查看表中有没有上锁的记录。就是说当对一个行加锁之后，如果有打算给行所在的表加一个表锁，必须先看看该表的行有没有被加锁，否则就会出现冲突。IS锁和IX锁就避免了判断表中行有没有加锁时对每一行的遍历。直接查看表有没有意向锁就可以知道表中有没有行锁。收起

十四、说下 MySQL 中的 MVCC 机制?

#每天一道面试题# 50

#悟空拧螺丝# 2021-09-03

MySQL 中有四种隔离级别，Read Repeatable （RR）级别可以防止脏读、不可重复读、幻读问题。Read Committed （RC）级别解决了脏读问题。

那它是怎么做到的呢？就是利用了 MVCC 多版本控制机制。而且可以实现 读-写，写-读不冲突。

本解答尽量通俗易懂：

多版本

就是有多个某行记录更新后的版本，然后将这些版本从上到下串起来。有点像串糖葫芦，这个就是版本链。

比如说银行转账记录，将多次对账户的修改都串起来了。记录里面有是哪个事务做的转账记录，最后值等于多少。

（1）账户 A = 初始值 200元，事务 id = 40 ->

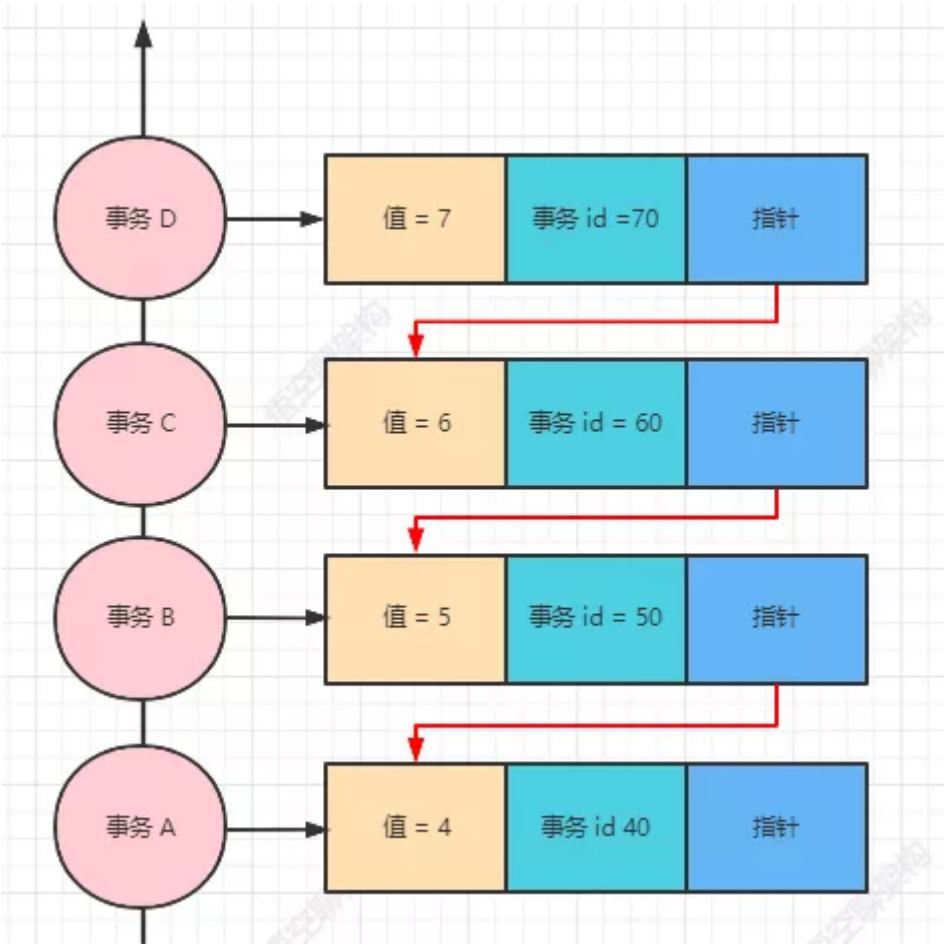
（2）账户 A = 初始值 200元 + 100 元 = 300 元，事务 id = 51 ->

（3）账户 A = 300 元 + 50 元 = 350 元，事务 id = 59 ->

（4）账户 A = 350 元 - 30 元 = 320 元，事务 id = 72

在 MySQL 就是利用 undo log 日志将这些串起来的。

如下图所示，undolog 的版本串起来长这样：



控制

用自身的事务 id 和其他地方存的事务 id 进行比较，看是否符合读取版本链上的条件，如果符合，读取后就返回了。怎么控制的呢？利用 ReadView。ReadView 其实也不难理解，就是对当前活跃事务的一个统计。然后 MySQL 利用这个数据统计 + 版本链上的事务 id 来进行比较，获得某个可读到的版本。

ReadView

保证你只能读到事务开启之前，别的事务提交的值，或者自己提交的值。其他情况下无法读取到其他事务提交的值，避免了脏读。

ReadView 生成时机?

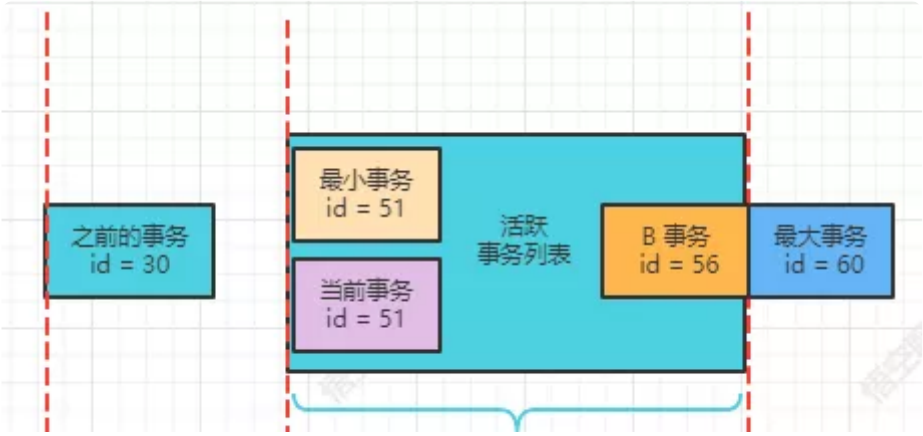
每个事务执行查询时都会生成自己事务的 ReadView。RC 级别是每次查询都会重新生成一份，RR 级别是事务中的 ReadView 都不变。

ReadView 里面有四个重要的属性：

m_ids 事务列表：有哪些事务在MySQL里执行还没提交的；min_trx_id 最小事务 id：m_ids 列表中最小的值 ；max_trx_id 最大事务 id：下一个要生成的事务id，就是最大事务id；creator_trx_id：当前事务的 id。

这四个属性怎么用的呢?

比如说事务 A 用来查询，事务 B 用来更新，它俩都开启了事务，也都还没有提交，对应的事务 id 分别为 51 和 59，那么 ReadView 就长这样：



活跃事务列表就是 [51，59]。最小事务 id = 51。最大事务 id = 59+1 = 60。当前事务 id = 51。

事务 A 首先拿着这几个属性值，到版本链上一个一个比较版本上的事务 id，符合条件就返回。比较又分三种情况：

- 1、如果版本上的事务 id < 最小值 51，说明这个行记录在这些活跃的事务创建前就已经提交了，这个行记录的版本对于当前事务 A 是可见的，就返回了。
- 2、如果版本上的事务 id >= 最大值 60，则说明提交的事务是 ReadView 生成之后创建的，这个版本也是不可读的，就接着往下找。
- 3、如果版本上的事务 id 在在最小和最大值之间，就进行下一步判断：
  - 3.1、如果在这个列表 [51，59] 里面，说明提交的事务是和 A 事务差不多时间开启的事务，被 ReadView 记录在列表里面了。这种事务提交的版本也是不可读的，就接着往下找。（避免了脏读）
  - 3.2、如果不在这个列表 [51，59] 里面，说明事务已提交了，是可以读取的，读到了就返回。

RC 的读已提交怎么做到的?

我们说 RC 隔离级别下，事务 A 下次查询时，就可以读到其他事务提交的数据了（读已提交），但是根据上面的3.1 的情况来看，事务 A 是读取不到事务 B 提交的呀?

这就需要在 A 查询时重新生成一个 ReadView 了，来看下重新生成的长啥样：

活跃事务列表就是 [51]。

最小事务 id = 51。

最大事务 id = 60。注意：这是 MySQL 下一个要生成的事务 id，不是指活跃事务中的最大事务 id。

当前事务 id = 51。

看到了吗?

事务 B 的 事务 id 59 不在活跃事务列表啦！但是又是小于最大事务 id 60 的。这就符合 3.2 的情况啦，可以读到这个版本了。

那下次 事务 A 再次查询时，又会生成一个 ReadView，可以读到其他事务提交的数据，这个数据和上次的数据很有可能不一样，也就是说不能保证每次读到的数据一样的，这就是不可重复读。RR 的可重复读怎么做到的?

它和 RC 不同的地方在于，事务 A 查询时，是不会重新生成 ReadView 的，也就是说 B 提交的事务读取不到的，那就顺着版本链继续找呗。找着找着就只能读事务 A 自己提交的，或者事务开启之前，其他事务提交的，那么事务 A 每次查询都是读到一样的数据啦，但是读取的都不是最新的数据，这就是可重复读，避免了读取数据不一致的情况。

注意：不管其他事务怎么修改数据，事务 A 生成的 ReadView 是不会改变的，基于这个 ReadView 看到的值都是一样的!

RR 的幻读是怎么避免的?

比如 A 执行范围查询：select * from table where age > 10，查到了一条数据 X。然后事务 C 72 插入了一条数据，事务 A 再次查询时，可以查到两条数据 X 和 Y。但是 Y 的版本链上事务 id 等于 72，大于最大事务 id 60，说明是事务 A 发起查询后，当然是不可读到的了，所以事务 A 还是只能读到数据 X。

小结

通过版本链 + ReadView 做到了这些事情：避免了 RR 隔离级别下的脏读、不可重复度、幻读问题。避免了 RC 隔离级别下的脏读问题，实现了读取已提交数据的功能。

十五、说下执行计划?

#每天一道面试题# 51

在生产过程中，经常会遇到因为SQL语句导致的性能瓶颈问题，这时候就需要我们去优化SQL的执行效率；EXPLAIN语句的各个输出项指标可以帮助我们有针对性的提升查询语句的性能。

执行计划重要指标解释(id主键，name,age 联合索引, a 普通索引，b普通列)：

type 字段 =>>

ALL - 最牛的优化结果这里就不展开介绍了

range - 索引范围查询；

ref - 二级索引等值查询；

const - 主键/唯一索引等值查询；

index_merge - 多个索引查询；

index - 覆盖索引且需要扫描全部的索引；

eq_ref - 被连接表主键/唯一引等值查询；

possible_keys 字段 =>> 可能用到的索引(备胎）；

key 字段 =>> 实际用到的索引(转正)；

key_len 字段 =>> 实际使用索引的最大长度

ref 字段 =>> 等值查询有一个常数/列值

const - 索引等值查询；

[DB].[table].[column] - 被调用表索引列等值查询；

Extra字段 =>>

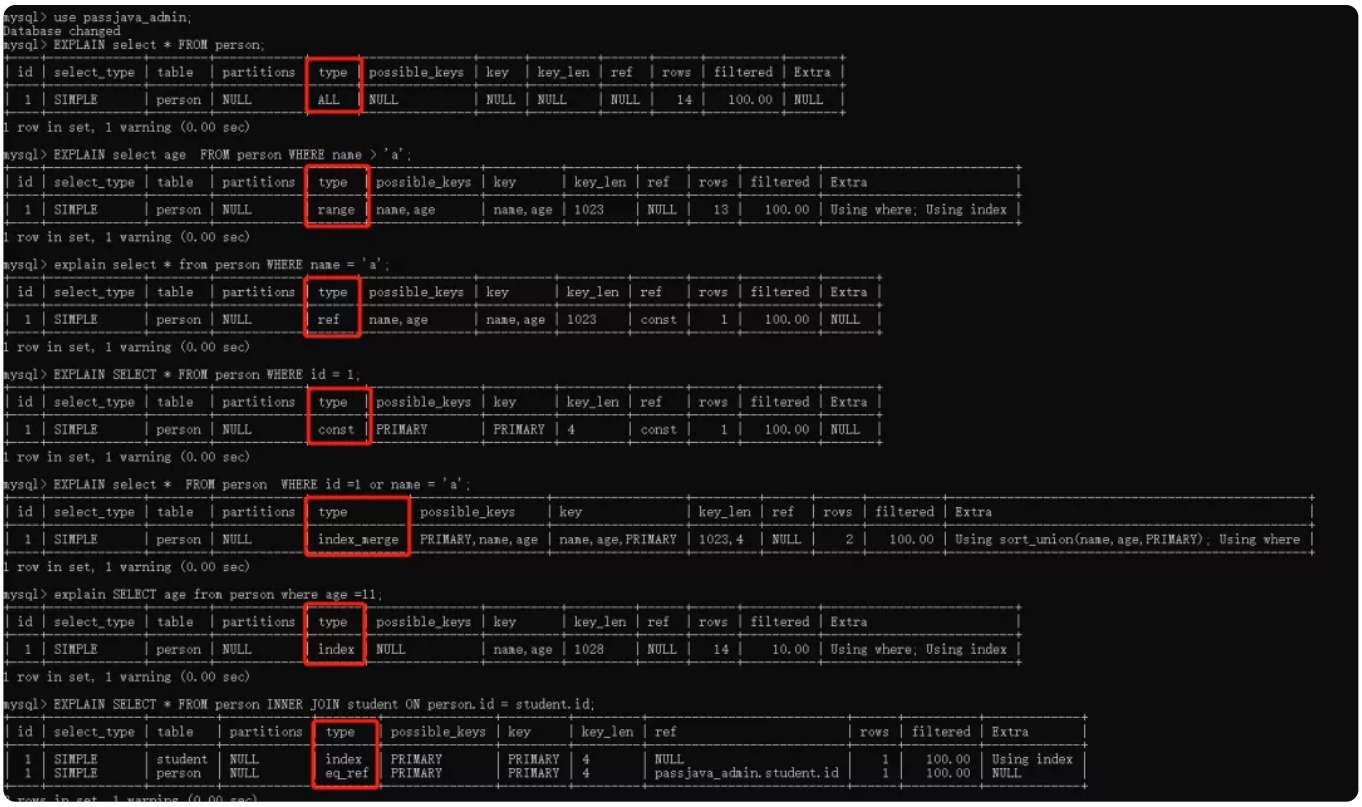
Impossible WHERE - where false;

Using index - 覆盖索引;

Using index condition - 索引下推;

Using where - 顺序扫描，where条件查询。

下面几张图说明了具体出现的场景：



```
mysql> use passjava_admin;
Database changed
mysql> EXPLAIN select * FROM person;
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table  | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | person | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   14 |   100.00 | NULL  |
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN select age  FROM person WHERE name > 'a';
+----+-------------+--------+------------+-------+---------------+----------+---------+------+------+----------+--------------------------+
| id | select_type | table  | partitions | type  | possible_keys | key      | key_len | ref  | rows | filtered | Extra                    |
+----+-------------+--------+------------+-------+---------------+----------+---------+------+------+----------+--------------------------+
|  1 | SIMPLE      | person | NULL       | range | name,age      | name,age | 1023    | NULL |   13 |   100.00 | Using where; Using index |
+----+-------------+--------+------------+-------+---------------+----------+---------+------+------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from person WHERE name = 'a';
+----+-------------+--------+------------+------+---------------+----------+---------+-------+------+----------+-------+
| id | select_type | table  | partitions | type | possible_keys | key      | key_len | ref   | rows | filtered | Extra |
+----+-------------+--------+------------+------+---------------+----------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | person | NULL       | ref  | name,age      | name,age | 1023    | const |    1 |   100.00 | NULL  |
+----+-------------+--------+------------+------+---------------+----------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM person WHERE id = 1;
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table  | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | person | NULL       | const | PRIMARY       | PRIMARY | 4       | const |    1 |   100.00 | NULL  |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN select *  FROM person  WHERE id =1 or name = 'a';
+----+-------------+--------+------------+-------------+-----------------+-----------------+---------+------+------+----------+-----------------------------------------------------+
| id | select_type | table  | partitions | type        | possible_keys   | key             | key_len | ref  | rows | filtered | Extra                                               |
+----+-------------+--------+------------+-------------+-----------------+-----------------+---------+------+------+----------+-----------------------------------------------------+
|  1 | SIMPLE      | person | NULL       | index_merge | PRIMARY,name,age | name,age,PRIMARY | 1023,4  | NULL |    2 |   100.00 | Using sort_union(name,age,PRIMARY); Using where |
+----+-------------+--------+------------+-------------+-----------------+-----------------+---------+------+------+----------+-----------------------------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain SELECT age from person where age =11;
+----+-------------+--------+------------+-------+---------------+----------+---------+------+------+----------+--------------------------+
| id | select_type | table  | partitions | type  | possible_keys | key      | key_len | ref  | rows | filtered | Extra                    |
+----+-------------+--------+------------+-------+---------------+----------+---------+------+------+----------+--------------------------+
|  1 | SIMPLE      | person | NULL       | index | NULL          | name,age | 1028    | NULL |   14 |    10.00 | Using where; Using index |
+----+-------------+--------+------------+-------+---------------+----------+---------+------+------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM person INNER JOIN student ON person.id = student.id;
+----+-------------+---------+------------+--------+---------------+---------+---------+-------------------------+------+----------+-------------+
| id | select_type | table   | partitions | type   | possible_keys | key     | key_len | ref                     | rows | filtered | Extra       |
+----+-------------+---------+------------+--------+---------------+---------+---------+-------------------------+------+----------+-------------+
|  1 | SIMPLE      | student | NULL       | index  | PRIMARY       | PRIMARY | 4       | NULL                    |    1 |   100.00 | Using index |
|  1 | SIMPLE      | person  | NULL       | eq_ref | PRIMARY       | PRIMARY | 4       | passjava_admin.student.id |    1 |   100.00 | NULL        |
+----+-------------+---------+------------+--------+---------------+---------+---------+-------------------------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

```
mysql> EXPLAIN SELECT * FROM person WHERE FALSE;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra           |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------+
|  1 | SIMPLE      | NULL  | NULL       | NULL | NULL          | NULL | NULL    | NULL | NULL |     NULL | Impossible WHERE |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select a from person WHERE a = 11;
+----+-------------+--------+------------+------+---------------+------+---------+-------+------+----------+-------------+
| id | select_type | table  | partitions | type | possible_keys | key  | key_len | ref   | rows | filtered | Extra       |
+----+-------------+--------+------------+------+---------------+------+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | person | NULL       | ref  | a             | a    | 5       | const |    2 |   100.00 | Using index |
+----+-------------+--------+------------+------+---------------+------+---------+-------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN select *  FROM person WHERE name like 'a%' and age = 7;
+----+-------------+--------+------------+-------+---------------+----------+---------+------+------+----------+-----------------------+
| id | select_type | table  | partitions | type  | possible_keys | key      | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+--------+------------+-------+---------------+----------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | person | NULL       | range | name,age      | name,age | 1028    | NULL |    5 |    10.00 | Using index condition |
+----+-------------+--------+------------+-------+---------------+----------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> EXPLAIN select *  FROM person where b=1;
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table  | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | person | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   14 |    10.00 | Using where |
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> EXPLAIN select id  FROM person WHERE a = 1 and name like 'a%';
+----+-------------+--------+------------+------+---------------+------+---------+-------+------+----------+-------------+
| id | select_type | table  | partitions | type | possible_keys | key  | key_len | ref   | rows | filtered | Extra       |
+----+-------------+--------+------------+------+---------------+------+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | person | NULL       | ref  | name,age,a    | a    | 5       | const |    1 |    35.71 | Using where |
+----+-------------+--------+------------+------+---------------+------+---------+-------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN select id  FROM person WHERE a = 1;
+----+-------------+--------+------------+------+---------------+------+---------+-------+------+----------+-------------+
| id | select_type | table  | partitions | type | possible_keys | key  | key_len | ref   | rows | filtered | Extra       |
+----+-------------+--------+------------+------+---------------+------+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | person | NULL       | ref  | a             | a    | 5       | const |    1 |   100.00 | Using index |
+----+-------------+--------+------------+------+---------------+------+---------+-------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM person INNER JOIN student ON person.id = student.id;
+----+-------------+---------+------------+--------+---------------+---------+---------+-------------------------+------+----------+-------------+
| id | select_type | table   | partitions | type   | possible_keys | key     | key_len | ref                     | rows | filtered | Extra       |
+----+-------------+---------+------------+--------+---------------+---------+---------+-------------------------+------+----------+-------------+
|  1 | SIMPLE      | student | NULL       | index  | PRIMARY       | PRIMARY | 4       | NULL                    |    1 |   100.00 | Using index |
|  1 | SIMPLE      | person  | NULL       | eq_ref | PRIMARY       | PRIMARY | 4       | passjava_admin.student.id |    1 |   100.00 | NULL        |
+----+-------------+---------+------------+--------+---------------+---------+---------+-------------------------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

十六、说下主从复制原理?

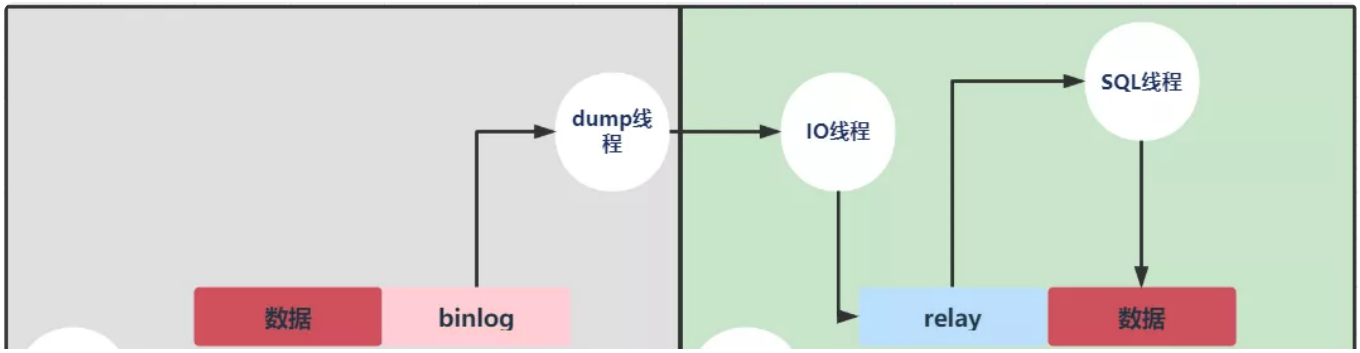
#每天一道面试题# 52

主从复制主要有以下流程：

1. master服务器将数据的改变记录到binlog中；

2. slave服务器会在一定时间间隔内对master 的binlog进行检查，如果发生改变，则开始一个I/OThread请求读取master中binlog；

3. 同时主节点为每个I/O线程启动一个dump线程，用于向其发送二进制事件，并保存至从节点本地的中继日志中，从节点将启动SQL线程从中继日志中读取二进制日志，在本地重放，使得其数据和主节点的保持一致，最后I/OThread和SQLThread将进入睡眠状态，等待下一次被唤醒；



大白话解释:

从库会生成两个线程,一个I/O线程,一个SQL线程;

I/O线程会去请求主库的binlog,并将得到的binlog写到本地的relay-log(中继日志)文件中;

主库会生成一个dump线程,用来给从库I/O线程传binlog;

SQL线程,会读取relay log文件中的日志,并解析成sql语句逐一执行;

主从复制存在数据丢失问题的解决方案：在使用过程中需要开启半同步复制；

主从复制的使用场景主要有以下两种：HA、读写分离。

高可用(HA)架构：

MySQL 的高可用由互为主从的MySQL构成，平时只有主库提供服务，备库不提供服务。当主库停止服务时，服务自动切换到备库。

- MHA管理工具 MHA存在Manager、Node两种节点；Manager节点通过探测Node节点去判断 MySQL运行是否正常，如果发现 Master故障。
- 就把他的一个Slave提升为Master，然后剩余Slave都挂到新的Master。
- LVS+Keepalived

  Keepalived可以进行检查心跳和动态漂移；当Master节点出现异常主服务所在keepalived会发出通知，然后slave节点的keepalived通知从节点切换为master。

读写分离架构：

高并发下读写分离会出现数据延迟问题。

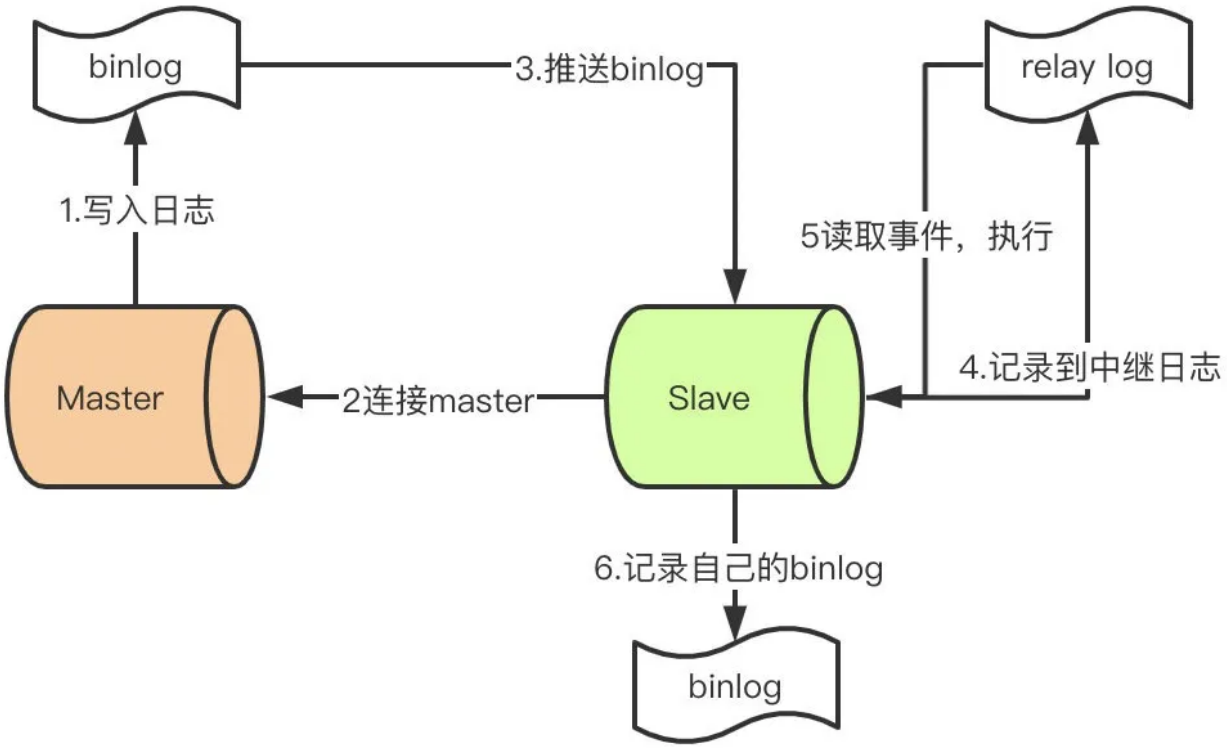
解决方案如下：

- 分库分表；
- 开启并行复制；
- 在业务逻辑上避免；

补充资料：

（以下MySQL 主从同步的原理的补充资料来自艾小仙）

1. master提交完事务后，写入binlog。
2. slave连接到master，获取binlog。
3. master创建dump线程，推送binglog到slave。
4. slave启动一个IO线程读取同步过来的master的binlog，记录到relay log中继日志中。
5. slave再开启一个sql线程读取relay log事件并在slave执行，完成同步。
6. slave记录自己的binglog。



由于mysql默认的复制方式是异步的，主库把日志发送给从库后不关心从库是否已经处理，这样会产生一个问题就是假设主库挂了，从库处理失败了，这时候从库升为主库后，日志就丢失了。由此产生两个概念。

全同步复制

主库写入binlog后强制同步日志到从库，所有的从库都执行完成后才返回给客户端，但是很显然这个方式的话性能会受到严重影响。

半同步复制

和全同步不同的是，半同步复制的逻辑是这样，从库写入日志成功后返回ACK确认给主库，主库收到至少一个从库的确认就认为写操作完成。

十七、说下 SQL 标准的事务隔离级别?

读未提交（read uncommitted）是指，一个事务还没提交时，它做的变更就能被别的事务看到。

读提交（read committed）是指，一个事务提交之后，它做的变更才会被其他事务看到。

可重复读（repeatable read）是指，一个事务执行过程中看到的数据，总是跟这个事务在启动时看到的数据是一致的。当然在可重复读隔离级别下，未提交变更对其他事务也是不可见的。

串行化（serializable ），顾名思义是对于同一行记录，“写”会加“写锁”，“读”会加“读锁”。当出现读写锁冲突的时候，后访问的事务必须等前一个事务执行完成，才能继续执行。

隔离级别	是否读取未提交的行	是否不可重复读	是否丢失更新	是否幻读	共享锁持续时间	是否持有范围锁
未提交读 READ UNCOMMITTED	Y	Y	Y	Y	当前语句	N
已提交读 READ						

已提交读 READ COMMITTED	N	Y	Y	Y	当前语句	N
可重复读REPEATABLE READ	N	N	N	Y	事务开始到事务完成	N
可序列化SERIALZABLE	N	N	N	N	事务开始到事务完成	

十八、说说生成唯一 ID 的雪花算法是怎么样的?

每天一道面试题 16 悟空拧螺丝 2021-07-28

`snowflake` （雪花算法）：Twitter 开源的分布式 id 生成算法，64 位的 long 型的 id，分为 4 部分：

snowflake 算法

- 1 bit：不用，统一为 0
- 41 bits：毫秒时间戳，可以表示 69 年的时间。
- 10 bits：5 bits 代表机房 id，5 个 bits 代表机器 id。最多代表 32 个机房，每个机房最多代表 32 台机器。
- 12 bits：同一毫秒内的 id，最多 4096 个不同 id，自增模式

优点：

- 毫秒数在高位，自增序列在低位，整个ID都是趋势递增的。
- 不依赖数据库等第三方系统，以服务的方式部署，稳定性更高，生成ID的性能也是非常高的。
- 可以根据自身业务特性分配bit位，非常灵活。

缺点：

- 强依赖机器时钟，如果机器上时钟回拨（可以搜索 **2017 年闰秒 7:59:60**），会导致发号重复或者服务会处于不可用状态。

十九、说下 MySQL 内部的 XA 分布式事务?

#每天一道面试题# 42

#悟空拧螺丝# 2021-08-23

XA是X/Open DTP组织（X/Open DTP group）定义的两阶段提交协议。

MySQL本身的插件式架构导致在其内部需要使用XA事务，此时MySQL即是协调者，也是参与者。内部XA事务发生在存储引擎与插件之间或者存储引擎与存储引擎之间。例如，不同的存储引擎之间是完全独立的，因此当一个事务涉及两个不同的存储引擎时，就必须使用内部XA事务。由于只在单机上工作，所以被称为内部XA。

最为常见的内部XA事务存在于二进制日志（Binlog）和InnoDB存储引擎之间。由于复制的需要，因此，目前绝大多数的数据库都开启了Binlog功能。

在事务提交时，先写二进制日志，再写InnoDB存储引擎的重做日志。对上述两个操作的要求也是原子的，即二进制日志和重做日志必须同时写入。若二进制日志先写了，而在写入InnoDB存储引擎时发生了宕机，那么Slave可能会接收到Master传过去的二进制日志并执行，最终导致了主从不一致的情况发生。

为了解决这个问题，MySQL数据库在Binlog与InnoDB存储引擎之间采用XA事务。当事务提交时，InnoDB存储引擎会先做一个PREPARE操作，将事务的Xid写入，接着进行Binlog的写入。如果在Binlog存储引擎提交前，MYSQL数据库宕机了，那么MySQL数据库在重启后会先检查准备的UXID事务是否已经提交，若没有，则在存储引擎层再进行一次提交操作

二十、说下 MySQL 的外部 XA 事务?

#每天一道面试题# 43 #悟空拧螺丝# 2021-08-24

（1）XA 事务是什么

XA是X/Open DTP组织（X/Open DTP group）定义的两阶段提交协议。

分布式事务通常采用 2PC 协议，全称 Two Phase Commitment Protocol（两阶段提交协议）。该协议主要为了解决在分布式数据库场景下，所有节点间数据一致性的问题。分布式事务通过2PC协议将提交分成两个阶段：

阶段一为准备（prepare）阶段。即所有的参与者准备执行事务并锁住需要的资源。参与者 ready 时，向transaction manager 报告已准备就绪。

阶段二为提交阶段（commit）。当 transaction manager 确认所有参与者都 ready 后，向所有参与者发送 commit命令。

缺点：第一个阶段会锁定资源，等待其他参与者 Ready，在高并发场景下，会严重影响系统的吞吐量。

（2）MySQL 的外部 XA

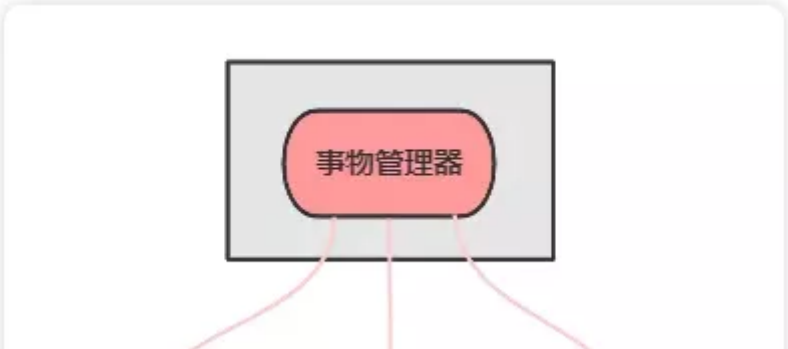
MySQL在执行分布式事务（外部XA）的时候，MySQL服务器相当于XA“事务资源管理器”，

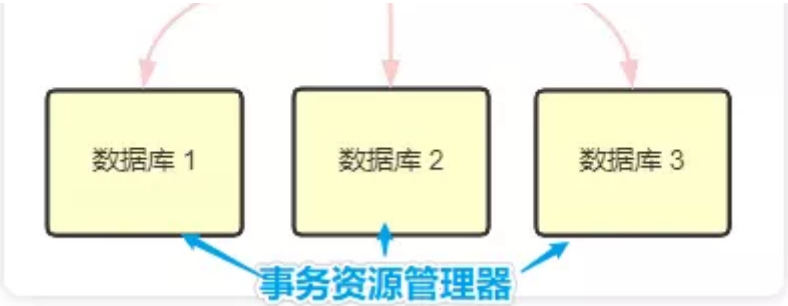
与MySQL连接的客户端相当于“事务管理器”，比如 Java 应用程序。

事务管理器负责协调多个数据库的事务，先问问各个数据库准备好了吗？如果准备好了，则在数据库执行操作，如果任一数据库没有准备，则回滚事务。

MySQL支持 XASTART/END/PREPARE/COMMIT 这些SQL语句，通过使用这些命令可以完成分布式事务的状态转移。

内部XA事务用于同一实例下跨多引擎事务，而外部XA事务用于跨多 MySQL 实例的分布式事务，需要应用层作为协调者。应用层负责决定提交还是回滚。





XA 方案

苏三说技术
作者就职于知名互联网公司，掘金月度优秀作者，从事开发、架构和部分管理工作。实...
43篇原创内容

公众号

喜欢此内容的人还喜欢

Kafka 精妙的高性能设计（下篇）

武哥漫谈IT

一文讲清，MySQL事务隔离级别

南山的架构笔记

详解事务、隔离级别、悲观锁和乐观锁

涛歌依旧