Mysql优化建议

- 对查询进行优化，应尽量避免全表扫描，首先应考虑在 WHERE 及 ORDER BY 涉及的列上建立索引。
- 应尽量避免在 WHERE 子句中对字段进行 NULL 值判断，创建表时 NULL 是默认值，但大多数时候应该使用 NOT NULL，或者使用一个特殊的值，如 0，-1 作为默认值。
- 应尽量避免在 WHERE 子句中使用 != 或 <> 操作符。MySQL 只有对以下操作符才使用索引：<，<=，=，>，>=，BETWEEN，IN，以及某些时候的 LIKE。、
- 应尽量避免在 WHERE 子句中使用 OR 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，可以使用 UNION 合并查询：select id from t where num=10 union all select id from t where num=20。
- IN 和 NOT IN 也要慎用，否则会导致全表扫描。对于连续的数值，能用 BETWEEN 就不要用 IN：select id from t where num between 1 and 3。IN肯定会走索引，但是当IN的取值范围较大时会导致索引失效，走全表扫描。
- 下面的查询也将导致全表扫描：select id from t where name like‘%abc%’ 或者select id from t where name like‘%abc’若要提高效率，可以考虑全文检索。而select id from t where name like‘abc%’才用到索引。like只有在右面才会走索引
- 如果在 WHERE 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然 而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：

  select id from t where num=@num

  -- 可以改为强制查询使用索引：

  select id from t with(index(索引名)) where num=@num
- 应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。
- 很多时候用 EXISTS 代替 IN 是一个好的选择：select num from a where num in(select num from b)。用下面的语句替换：select num from a where exists(select 1 from b where num=a.num)。
- 索引固然可以提高相应的 SELECT 的效率，但同时也降低了 INSERT 及 UPDATE 的效。因为 INSERT 或 UPDATE 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过 6 个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。

Mysql的事务隔离级别

- read uncommitted：读未提交

  产生的问题：脏读、不可重复读、幻读
- read committed：读已提交 （Oracle）

  产生的问题：不可重复读、幻读
- repeatable read：可重复读 （MySQL默认）

  产生的问题：幻读
- serializable：串行化

  可以解决所有的问题

  注意：隔离级别从小到大安全性越来越高，但是效率越来越低，并发性越来越低

数据库查询隔离级别：select @@tx_isolation;

数据库设置隔离级别：set global transaction isolation level 级别字符串;

mysql事务隔离时的锁表和锁行操作

1、事务隔离级别为读提交时，写数据只会锁住相应的行

2、事务隔离级别为可重复读时，如果检索条件有索引（包括主键索引）的时候，默认加锁方式是next-key 锁；如果检索条件没有索引，更新数据时会锁住整张表。一个间隙被事务加了锁，其他事务是不能在这个间隙插入记录的，这样可以防止幻读。

3、事务隔离级别为串行化时，读写数据都会锁住整张表

mysql事务隔离级别产生的问题

1. 脏读：事务A读取了事务B更新的数据，然后B回滚操作，那么A读取到的数据是脏数据
2. 不可重复读(虚读)：事务 A 多次读取同一数据，事务 B 在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果 不一致。锁行。
3. 幻读：一个事务操作(DML)数据表中所有记录，另一个事务添加了一条数据，则第一个事务查询不到自己的修改。针对于多条数据。锁表。

mysql事务四大特性

1. 原子性：是不可分割的最小操作单位，要么同时成功，要么同时失败。
2. 持久性：当事务提交或回滚后，数据库会持久化的保存数据。
3. 隔离性：多个事务之间。相互独立。
4. 一致性：事务操作前后，数据总量不变

Mysql分类

DDL(Data Definition Language)数据定义语言 用来定义数据库对象：数据库，表，列等。关键字：create, drop,alter 等 DML(Data Manipulation Language)数据操作语言 用来对数据库中表的数据进行增删改。关键字：insert, delete, update 等 DQL(Data Query Language)数据查询语言 用来查询数据库中表的记录(数据)。关键字：select, where 等 DCL(Data Control Language)数据控制语言(了解) 用来定义数据库的访问权限和安全级别，及创建用户。关键字：GRANT， REVOKE 等

数据库的三范式

数据库总共5个范式，目前是满足前三个范式即可。

第一范式(确保每列保持原子性)：表中的字段不可再次拆分。比如地址，如果频繁的访问地址的省市。我们将地址进行拆分存储即可。这样才满足第一范式。

第二范式（确保表中每列都和主键相关）：如果说我们有一个用户表和一个权限表。如果说我们用用户的id和角色的id作为主键，也是可以进行存储的。这样的问题就是，其中的任意一个信息，并不是和主键完全相关的。如果将用户id和角色id分别拆分为两个表。然后用一个中间表关联，这样是符合第二范式的。

第三范式(确保每列都和主键列直接相关,而不是间接相关)：第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关。比如在设计一个订单数据表的时候，可以将客户编号作为一个外键和订单表建立相应的关系。而不可以在订单表中添加关于客户其它信息（比如姓名、所属公司等）的字段。如下面这两个表所示的设计就是一个满足第三范式的数据库表。如果冗余了就不满足第三范式。

Mysql聚集索引

聚集索引定义了表中数据的物理存储顺序，索引顺序和物理顺序一致。InnoDB聚集索引的叶子节点存储行记录，因此InnoDB必须要有且只有一个聚集索引。一个表只能有一个聚集索引。Innodb的存储索引是基于B+tree。聚集索引既存储了索引，也存储了行值。它的数据是存储在索引的叶子页（leaf pages）。

如果一个主键被定义了，那么这个主键就是作为聚集索引

如果没有主键被定义，那么该表的第一个唯一非空索引被作为聚集索引

如果没有主键也没有合适的唯一索引，那么innodb内部会生成一个隐藏的主键作为聚集索引，这个隐藏的主键是一个6个字节的列，改列的值会随着数据的插入自增。

Innodb中的每张表都会有一个聚集索引，而聚集索引又是以物理磁盘顺序来存储的，自增主键会把数据自动向后插入，避免了插入过程中的聚集索引排序问题。聚集索引的排序，必然会带来大范围的数据的物理移动，这里面带来的磁盘IO性能损耗是非常大的。 而如果聚集索引上的值可以改动的话，那么也会触发物理磁盘上的移动，于是就可能出现page分裂，表碎片横生。

某些特殊的情况也是可以自己指定一些非自增主键为聚集索引的。如：

- 当数据量大，但长时间不会被更新的；
- 新生成的数据的索引本来就是按照自增的顺序增加的等等。

Mysql的非聚集索引

除了聚簇索引的其他索引都是非聚簇索引，叶节点指向表中的聚集索引的id，记录的物理顺序与逻辑顺序没有必然的联系。

每个表只能有一个聚簇索引，因为一个表中的记录只能以一种物理顺序存放。但是，一个表可以有不止一个非聚簇索引。实际上，对每个表你最多可以建立249个非聚簇索引。非聚簇索引需要大量的硬盘空间和内存。另外，虽然非聚簇索引可以提高从表中取数据的速度，它也会降低向表中插入和更新数据的速度。每当你改变了一个建立了非聚簇索引的表中的数据时，必须同时更新索引。因此你对一个表建立非聚簇索引时要慎重考虑。如果你预计一个表需要频繁地更新数据，那么不要对它建立太多非聚簇索引。另外，如果硬盘和内存空间有限，也应该限制使用非聚簇索引的数量

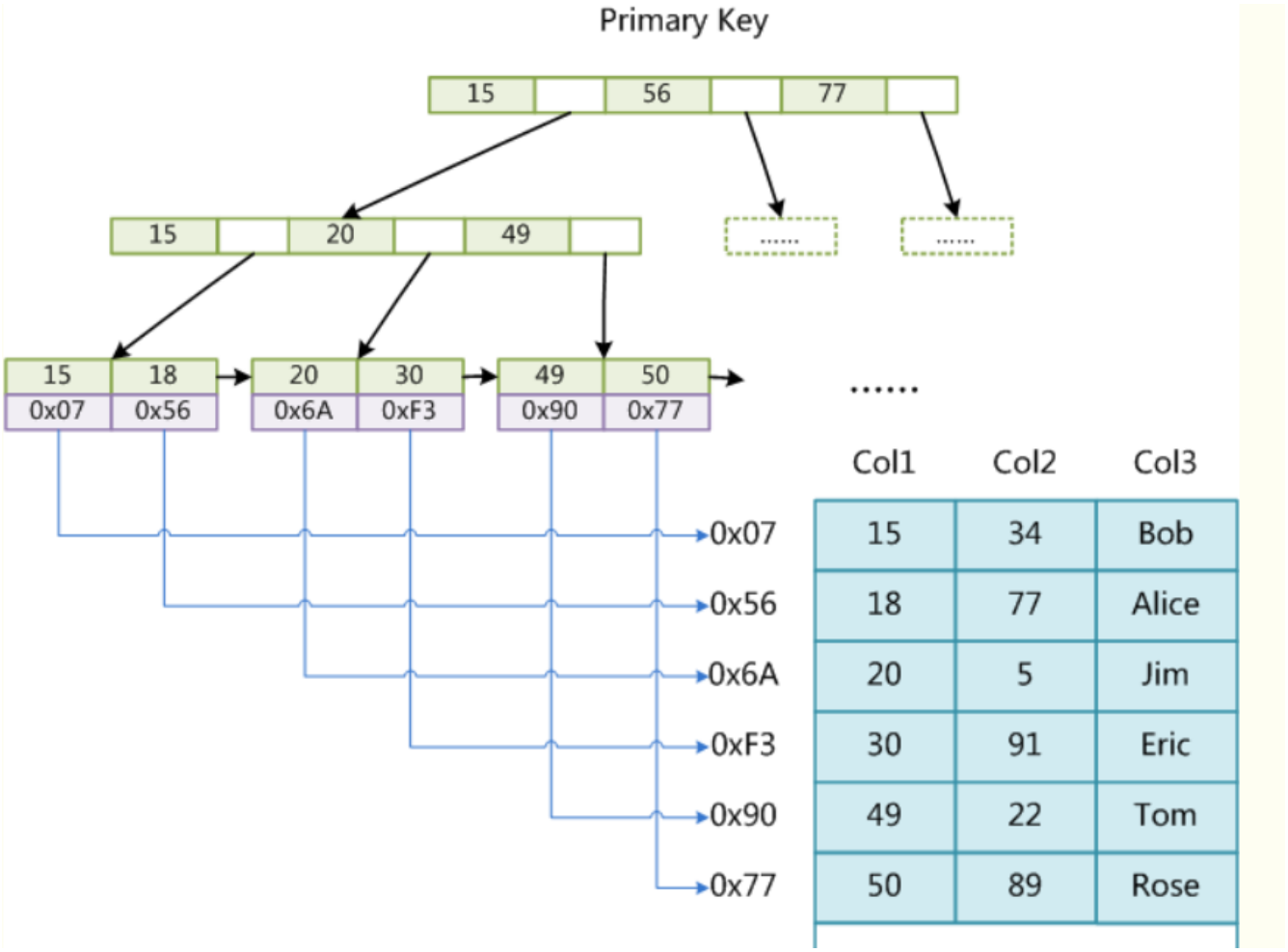
Mysql的回表查询

除了聚集索引以外，其他建立索引的方式都是非聚集索引，就是普通索引，二级索引。二级索引要进行回表。二级索引存储的并不是本身的数据，而是聚集索引中的主键值。第一次查询，找到主键值，再通过主键值找到真正的数据。

mysql的innodb和myisam索引的区别

myisam索引实现

MyIsam的索引是B+树形式的非聚集索引，叶节点的data域存放的是数据记录的地址。



二级索引和主键索引没有区别，只是允许了重复。

MyISAM中索引检索的算法为首先按照B+Tree搜索算法搜索索引，如果指定的Key存在，则取出其data域的值，然后以data域的值为地址，读取相应数据记录。

innodb索引实现

innodb索引同样是b+树索引。innodb表数据文件本身就是索引结构，树的叶节点data域保存了完整的数据记录，这种索引叫做聚集索引。二级索引也是B+树，不过二级索引的data节点存入的是数据的主键，然后通过回表查到数据。如果主键定义的比较大，二级索引就会比较大。

myisam和innodb区别

存储结构（主索引 / 辅助索引）

InnoDB的数据文件本身就是主索引文件。而MyISAM的主索引和数据是分开的。

InnoDB的辅助索引data域存储相应记录主键的值而不是地址。而MyISAM的辅助索引和主索引没有多大区别。

innoDB是聚簇索引，数据挂在逐渐索引之下。

主键

MyISAM允许没有任何索引和主键的表存在，索引都是保存行的地址

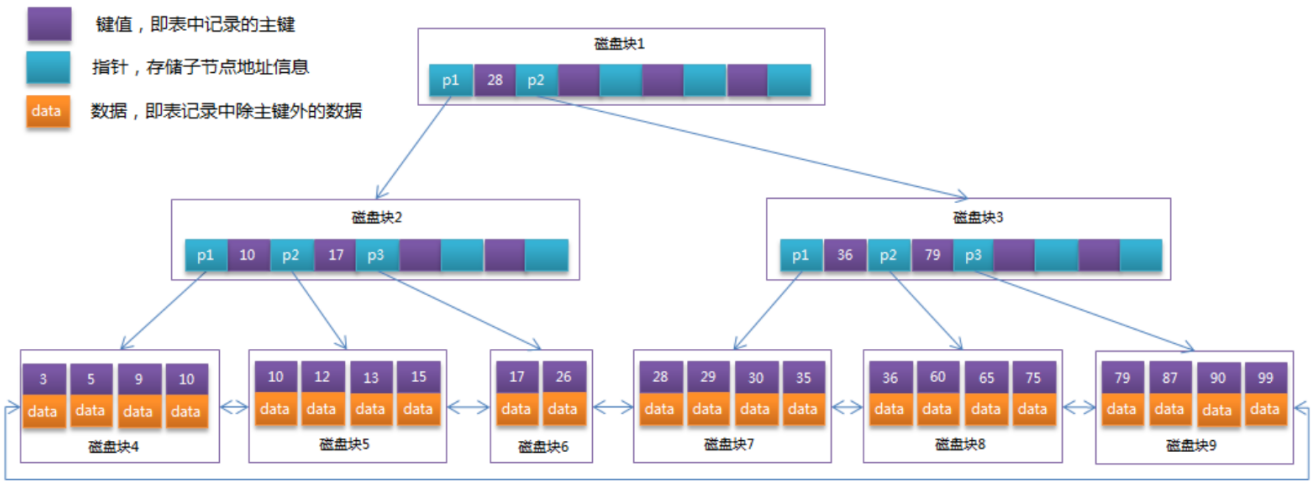
InnoDB如果没有设定主键或非空唯一索引，就会自动生成一个6字节的主键，数据是主索引的一部分，附加索引保存的是主索引的值

mysql的索引覆盖

就是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。比如你查询一个name字段，但是这个name正好被建立了索引，那么这个数据你不用去找到数据表，直接从索引中就获取到了。这就是索引覆盖。explain的输出结果Extra字段为Using index时，能够触发索引覆盖。平常我们通过建立联合索引来实现索引覆盖。

mysql索引介绍

mysql的索引采用的是B+树的形式，B+树是一种在叶子节点存储数据的结构，在非叶子节点只存储指针索引的结构。非叶子节点不存储数据使得B+树可以存储更多的关键字，减少了树的深度，每次io读入的关键字加大，减少了io次数。



当查找数据的时候，首先将磁盘块1加载到内存，二分查找找到区间，发生一次磁盘io。查找的时间忽略不计。找到后，再次往下找数据。

mysql索引失效的情况

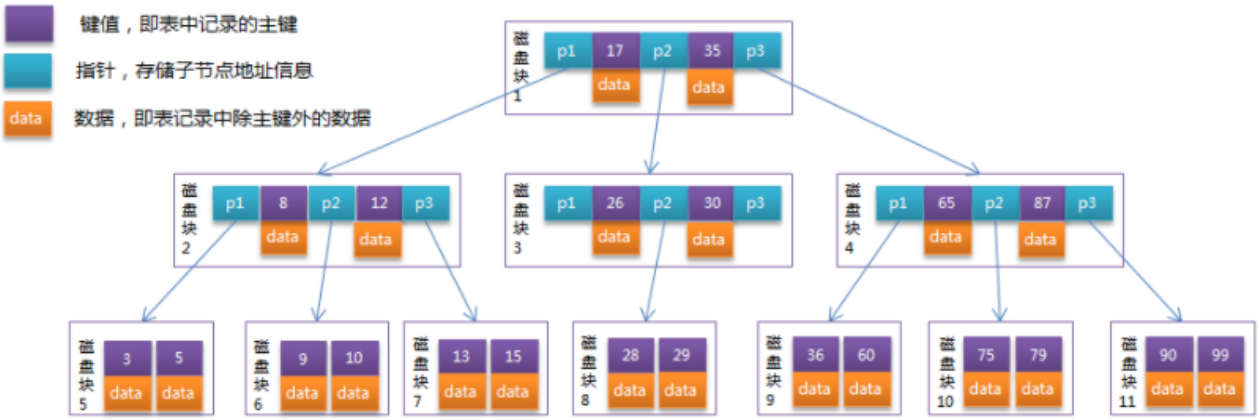
1、在where后使用or，导致索引失效（尽量少用or） 2、使用like ，like查询是以%开头，以%结尾不会失效 3、不符合最左原则 4、如果列类型是字符串，那一定要在条件中将数据使用引号引用起来,否则不使用索引 5、使用in导致索引失效，in的数据不挨着 6、使用mysql内部函数导致索引失效,可能会导致索引失效。 7、如果MySQL估计使用索引比全表扫描更慢，则不使用索引 8、B-tree索引 is null不会走,is not null会走 9、联合索引 is not null 只要在建立的索引列（不分先后）都会走, is null时 必须要和建立索引第一列一起使用,当建立索引第一位置条件是is null 时,其他建立索引的列可以是is null（但必须在所有列 都满足is null的时候）,或者=一个值； 当建立索引的第一位置是=一个值时,其他索引列可以是任何情况（包括is null =一个值）,以上两种情况索引都会走。其他情况不会走。

B+树和B-树的区别

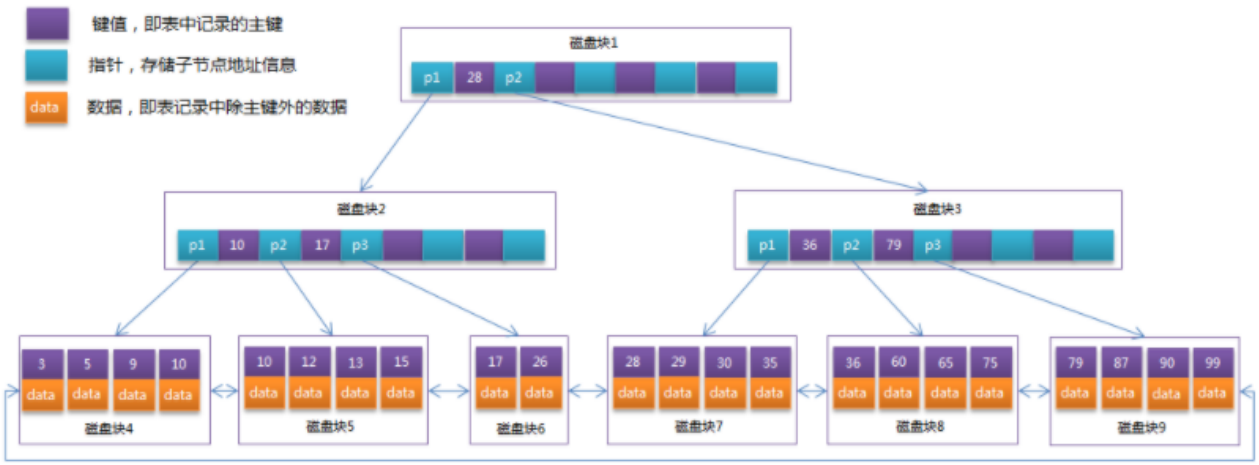
B树是平衡多叉树，平衡树查询效率高。平衡树就是任意子树的高度差不超过1。平衡是通过找到位置后，进行分裂实现。

在B+Tree中，所有数据记录节点都是按照键值大小顺序存放在同一层的叶子节点上，而非叶子节点上只存储key值信息，这样可以大大加大每个节点存储的key值数量，降低B+Tree的高度。

B树：



B+树：



Mysql监控指标

1、QPS：数据库每秒处理的请求数量

```
show global status where variable_name in ('Queries', 'uptime');
```

QPS = (Queries2 -Queries1) / (uptime2 - uptime1)

2、TPS：数据库每秒处理的事务数量

```
show global status where variable_name in ('com_insert' , 'com_delete' , 'com_update',
'uptime');
```

事务数TC ≈'com_insert' , 'com_delete' , 'com_update'

TPS ≈ (TC2 -TC1) / (uptime2 - uptime1)

3、并发数：数据库实例当前并行处理的会话数量

```
show global status like 'Threads_running';
```

4、连接数：连接到数据库会话的数量

```
show global status like 'Threads_connected';
```

5、缓存命中率：查询命中缓存的比例

innodb缓冲池查询总数：

```
show global status like 'innodb_buffer_pool_read_requests';
```

innodb从磁盘查询数：

```
show global status like 'innodb_buffer_pool_reads';
```

生产中配置报警阈值：innodb_buffer_pool_read_requests /(innodb_buffer_pool_read_requests + innodb_buffer_pool_reads) > 0.95

6、可用性：数据库是否可以正常对外服务

周期性连接数据库并执行 **select @@version;**

7、阻塞：当前阻塞的会话数

```
select waiting_pid as '被阻塞线程',
    waiting_query as '被阻塞SQL',
     blocking_pid as '阻塞线程',
     blocking_query as '阻塞SQL',
     wait_age as '阻塞时间',
     sql_kill_blocking_query as '建议操作'
from sys.innodb_lock_waits
where(unix_timestamp()-unix_timestamp(wait_started))>阻塞秒数
```

8、慢查询：慢查询情况

开启慢查询日志。my.inf

```
slow_query_log=on
slow_query_log_file=存放目录
long_query_time=0.1秒
log_queries_not_using_indexes=on
```

项目中mysql数据量多大？分表如何实现的？

现在数据库中最大的表是200w的数据。是存储企业每日上传的能耗数据。是一个增量变化表，未来会更大。这方面就是采取了表分区的方式，有的表采取的是水平分表的方式。分区是为了处理复杂查询的情况。水平分表十分不好处理，shard-jdbc不支持复杂的查询。

分布式事务解决方案

可靠消息最终一致性方案

TCC事务补偿方案

最大努力通知型

可靠消息最终一致性方案的实现

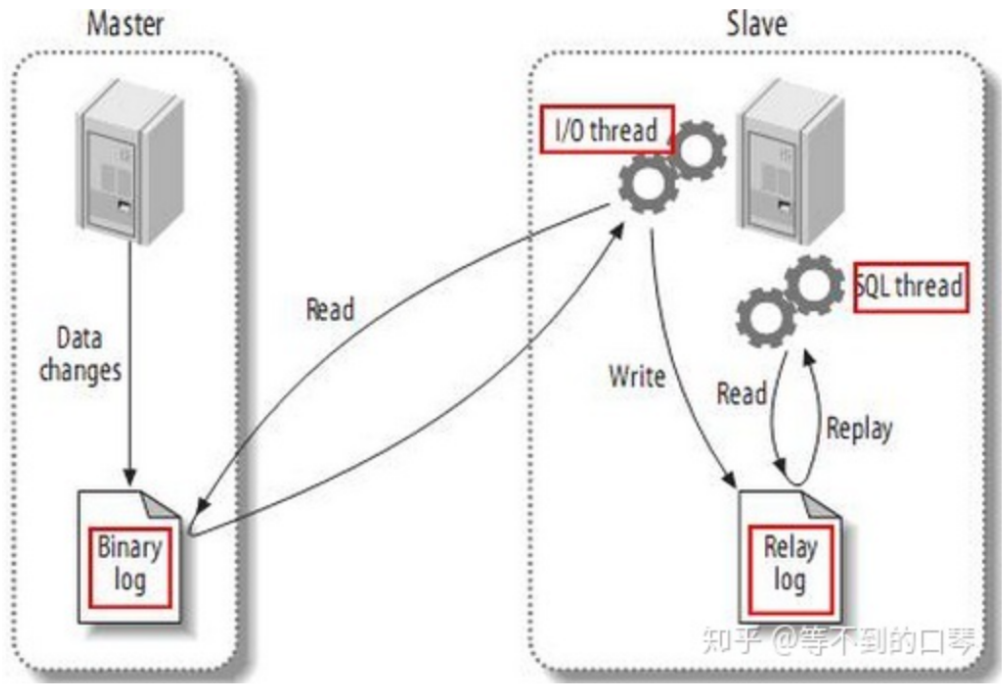
https://www.cnblogs.com/bluemiaomiao/p/11216380.html

mysql主从同步原理

（1）master服务器将数据的改变记录二进制binlog日志，当master上的数据发生改变时，则将其改变写入二进制日志中；

（2）slave服务器会在一定时间间隔内对master二进制日志进行探测其是否发生改变，如果发生改变，则开始一个I/OThread请求master二进制事件

（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。



mysql主从同步延迟的原因和解决办法

原因

当主服务器有大并发的更新操作, 但是从服务器的里面读取binlog 的线程仅有一个， 当某个SQL在从服务器上执行的时间稍长或者由于某个SQL要进行锁表就会导致，主服务器的SQL大量积压，未被同步到从服务器里。这就导致了主从不一致， 也就是主从延迟。

解决办法

我们知道因为主服务器要负责更新操作， 他对安全性的要求比从服务器高， 所有有些设置可以修改，比如sync_binlog=1，innodb_flush_log_at_trx_commit = 1 之类的设置，而slave则不需要这么高的数据安全，完全可以讲sync_binlog设置为0或者关闭binlog，innodb_flushlog， innodb_flush_log_at_trx_commit 也 可以设置为0来提高sql的执行效率 这个能很大程度上提高效率。另外就是使用比主库更好的硬件设备作为slave。通过show slave status进行查看Seconds_Behind_Master就是我们的延迟时间。

主从放到同一个交换机网络下。

直接禁用slave端的binlog。

参数

sync_binlog：这个参数是对于MySQL系统来说是至关重要的，他不仅影响到Binlog对MySQL所带来的性能损耗，而且还影响到MySQL中数据的完整性。对于“sync_binlog”参数的各种设置的说明如下：

sync_binlog=0，当事务提交之后，MySQL不做fsync之类的磁盘同步指令刷新binlog_cache中的信息到磁盘，而让Filesystem自行决定什么时候来做同步，或者cache满了之后才同步到磁盘。

sync_binlog=n，当每进行n次事务提交之后，MySQL将进行一次fsync之类的磁盘同步指令来将binlog_cache中的数据强制写入磁盘。

innodb_flush_log_at_trx_commit

可以定义mysql的事务提交方式

mysql写文件有2块缓存。一块是自己定义在内存的log buffer, 另一个是磁盘映射到内存的os cache。

mysql可以 调用 flush 主动将log buffer 刷新到磁盘内存映射，也可以调用 fsync 强制操作系同步磁盘映射文件到磁盘。还可以同时调用 flush + fsync, 将缓存直接落盘。

innodb_flush_log_at_trx_commit = 0 就是每秒调用 flush + fsync ，定时器自己维护。

innodb_flush_log_at_trx_commit = 1 就是实时调用 flush + fsync 没法批处理，性能很低。

innodb_flush_log_at_trx_commit = 2 就是实时flush ,定时 fsync 交给OS维护定时器。

–logs-slave-updates

从服务器从主服务器接收到的更新不记入它的二进制日志。

mysql主从同步存在的问题

1、主机宕机后，数据可能丢失。

2、从库只有一个thread，复制会延迟。

mysql的全复制、半复制、异步复制

异步复制

mysql默认的就是异步复制。主库在执行完客户端提交的事务后会立即将结果返回给客户端，并不关心从库是否已经接收并处理。主库将事务 Binlog 事件写入到 Binlog 文件中，此时主库只是通知 Dump 线程发送这些新的 Binlog，然后主库就会继续处理提交操作，并不保证这些 Binlog 传到任何一个从库节点上。这样就会存在一个问题，如果主库出现故障，此时主库已经提交的事务可能并没有传到从库上，可能导致数据丢失。

全复制

当主库提交事务之后，所有的从库节点必须收到、APPLY并且提交这些事务，然后主库线程才能继续做后续操作。因为需要等待所有从库执行完该事务才能返回，所以全同步复制的性能必然会收到严重的影响。

半同步复制

介于异步复制和全同步复制之间，主库在执行完客户端提交的事务后不是立刻返回给客户端，而是等待至少一个从库接收到并写到 relay log 中才返回给客户端。相对于异步复制，半同步复制提高了数据的安全性，同时它也造成了一定程度的延迟。

半同步复制的出现，就是为了保证在任何时刻主备数据一致的问题。相对于异步复制，半同步复制要求执行的每一个事务，都要求至少有一个备库成功接收后，才返回给用户。实现原理也很简单，主库本地执行完毕后，等待备库的响应消息（包含最新备库接收到的binlog（file，pos）），接收到备库响应消息后，再返回给用户，这样一个事务才算真正完成。在主库实例上，有一个专门的线程（ack_receiver）接收备库的响应消息，并以通知机制告知主库备库已经接收的日志，可以继续执行。

安装一个插件，然后设置mysql的参数即可。

mysql半同步复制的特点

（1）从库会在连接到主库时告诉主库，它是不是配置了半同步。 （2）如果半同步复制在主库端开启，并且至少有一个半同步复制的从库节点，那么此时主库的事务线程在提交时会被阻塞并等待，结果有两种可能：（a）至少一个从库节点通知它已经收到了所有这个事务的Binlog事件；（b）一直等待直到超过配置的某一个时间点为止，此时，半同步复制将自动关闭，转换为异步复制。 （3）从库节点只有在接收到某一个事务的所有 Binlog，将其写入到 Relay Log 文件之后，才会通知对应主库上面的等待线程。 （4）如果在等待过程中，等待时间已经超过了配置的超时时间，没有任何一个从节点通知当前事务，那么此时主库会自动转换为异步复制，当至少一个半同步从节点赶上来时，主库便会自动转换为半同步方式的复制。 （5）半同步复制必须是在主库和从库两端都开启时才行，如果在主库上没打开，或者在主库上开启了而在从库上没有开启，主库都会使用异步方式复制。

mysql中int(1)和int(10)的区别

从占用空间上来看，没有什么区别，都是四个字节。后面的数字只是一种显示宽度，只有在补充0的时候，才能看到区别。不足位数的会进行补0。默认的int，当有符号的时候是一个11位的值。有一位要存储正负号。当无符号的时候，默认是10位。当你写小于0的时候，就会给你改为默认。11位是因为2的32次方，正好是10位，加上符号11位。最大位数为255。

mysql中tinyint的默认位数

tinyint默认占用一个字节。他后面的位数和int一样，只有补0的时候，才能展示作用。有符号的时候，默认位数是4，无符号时候，默认为3。-128~127。最大位数为255

mysql的explain有哪些列

id（sql的执行顺序标识），selcet_type（select子句的类型），table（显示该列关于哪张表，有别名的时候展示的是别名），type（就是我们常看走不走索引的一列），possible_key（哪些列可能走索引，查询涉及的列有索引就会展示，但最后不一定走那个），key（mysql中实际走索引的列），key_len（走索引的时候，实际列中数据的最大长度），ref（表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值），rows（表示MySQL根据表统计信息及索引选用情况，估算的找到所需的记录所需要读取的行数），Extra（该列包含MySQL解决查询的详细信息），partitions（分区，落入了哪些分区），filtered（通过表条件过滤出的行数的百分比估计值）