01 MySQL 基础篇

1.说一下 MySQL 执行一条查询语句的内部执行过程？

- 客户端先通过连接器连接到 MySQL 服务器。
- 连接器权限验证通过之后，先查询是否有查询缓存，如果有缓存（之前执行过此语句）则直接返回缓存数据，如果没有缓存则进入分析器。
- 分析器会对查询语句进行语法分析和词法分析，判断 SQL 语法是否正确，如果查询语法错误会直接返回给客户端错误信息，如果语法正确则进入优化器。
- 优化器是对查询语句进行优化处理，例如一个表里面有多个索引，优化器会判别哪个索引性能更好。
- 优化器执行完就进入执行器，执行器就开始执行语句进行查询比对了，直到查询到满足条件的所有数据，然后进行返回。

2.MySQL 提示“不存在此列”是执行到哪个节点报出的？

此错误是执行到分析器阶段报出的，因为 MySQL 会在分析器阶段检查 SQL 语句的正确性。

3.MySQL 查询缓存的功能有何优缺点？

MySQL 查询缓存功能是在连接器之后发生的，它的优点是效率高，如果已经有缓存则会直接返回结果。查询缓存的缺点是失效太频繁导致缓存命中率比较低，任何更新表操作都会清空查询缓存，因此导致查询缓存非常容易失效。

4.如何关闭 MySQL 的查询缓存功能？

MySQL 查询缓存默认是开启的，配置 querycachetype 参数为 DEMAND（按需使用）关闭查询缓存，MySQL 8.0 之后直接删除了查询缓存的功能。

5.MySQL 的常用引擎都有哪些？

MySQL 的常用引擎有 InnoDB、MyISAM、Memory 等，从 MySQL 5.5.5 版本开始 InnoDB 就成为了默认的存储引擎。

6.MySQL 可以针对表级别设置数据库引擎吗？怎么设置？

可以针对不同的表设置不同的引擎。在 create table 语句中使用 engine=引擎名（比如 Memory）来设置此表的存储引擎。完整代码如下：

```
create table student(
   id int primary key auto_increment,
   username varchar(120),
   age int
) ENGINE=Memory
```

7.常用的存储引擎 InnoDB 和 MyISAM 有什么区别？

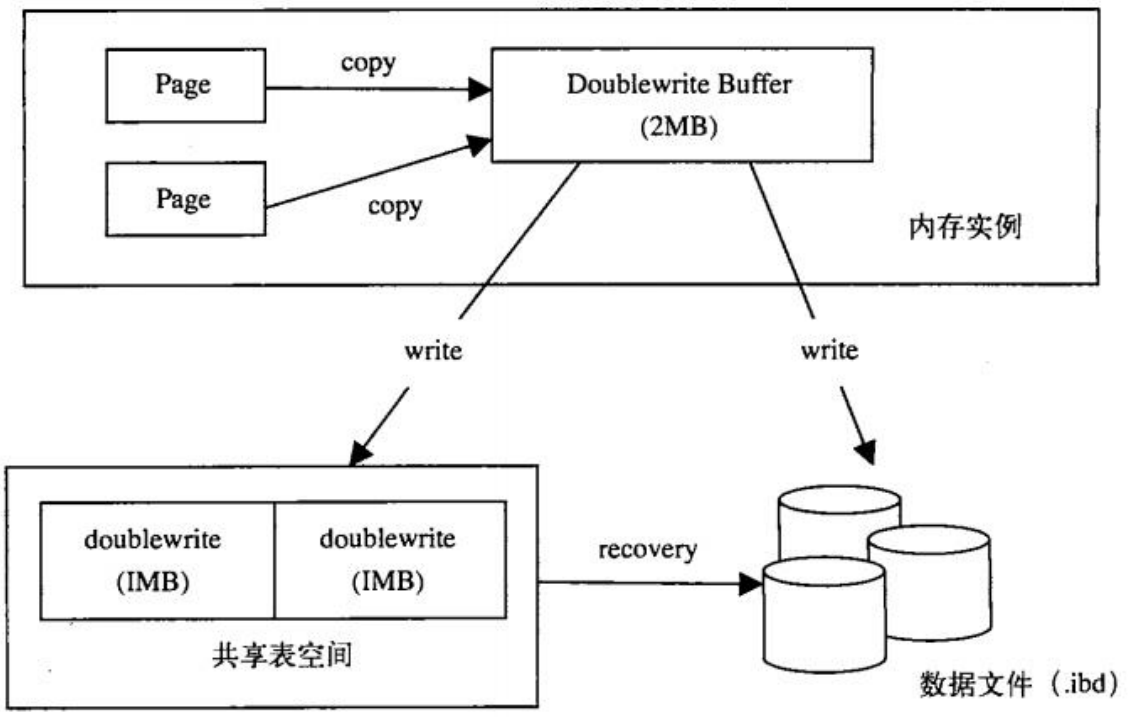
InnoDB 和 MyISAM 最大的区别是 InnoDB 支持事务，而 MyISAM 不支持事务，它们主要区别如下：

- InnoDB 支持崩溃后安全恢复，MyISAM 不支持崩溃后安全恢复；
- InnoDB 支持行级锁，MyISAM 不支持行级锁，只支持到表锁；
- InnoDB 支持外键，MyISAM 不支持外键；
- MyISAM 性能比 InnoDB 高；
- MyISAM 支持 FULLTEXT 类型的全文索引，InnoDB 不支持 FULLTEXT 类型的全文索引，但是 InnoDB 可以使用 sphinx 插件支持全文索引，并且效果更好；
- InnoDB 主键查询性能高于 MyISAM。

8.InnoDB 有哪些特性？

1）插入缓冲(insert buffer)：对于非聚集索引的插入和更新，不是每一次直接插入索引页中，而是首先判断插入的非聚集索引页是否在缓冲池中，如果在，则直接插入，否则，先放入一个插入缓冲区中。好似欺骗数据库这个非聚集的索引已经插入到叶子节点了，然后再以一定的频率执行插入缓冲和非聚集索引页子节点的合并操作，这时通常能将多个插入合并到一个操作中，这就大大提高了对非聚集索引执行插入和修改操作的性能。

2）两次写(double write)：两次写给 InnoDB 带来的是可靠性，主要用来解决部分写失败(partial page write)。doublewrite 有两部分组成，一部分是内存中的 doublewrite buffer ，大小为 2M，另外一部分就是物理磁盘上的共享表空间中连续的 128 个页，即两个区，大小同样为 2M。当缓冲池的作业刷新时，并不直接写硬盘，而是通过 memcpy 函数将脏页先拷贝到内存中的 doublewrite buffer，之后通过 doublewrite buffer 再分两次写，每次写入 1M 到共享表空间的物理磁盘上，然后马上调用 fsync 函数，同步磁盘。如下图所示



avatar

3）自适应哈希索引(adaptive hash index)：由于 InnoDB 不支持 hash 索引，但在某些情况下 hash 索引的效率很高，于是出现了 adaptive hash index 功能， InnoDB 存储引擎会监控对表上索引的查找，如果观察到建立 hash 索引可以提高性能的时候，则自动建立 hash 索引。

9.一张自增表中有三条数据，删除了两条数据之后重启数据库，再新增一条数据，此时这条数据的 ID 是几？

如果这张表的引擎是 MyISAM，那么 ID=4，如果是 InnoDB 那么 ID=2（MySQL 8 之前的版本）。

10.MySQL 中什么情况会导致自增主键不能连续？

以下情况会导致 MySQL 自增主键不能连续：

- 唯一主键冲突会导致自增主键不连续；
- 事务回滚也会导致自增主键不连续。

11.InnoDB 中自增主键能不能被持久化？

自增主键能不能被持久化，说的是 MySQL 重启之后 InnoDB 能不能恢复重启之前的自增列，InnoDB 在 8.0 之前是没有持久化能力的，但 MySQL 8.0 之后就把自增主键保存到 redo log（一种日志类型，下文会详细讲）中，当 MySQL 重启之后就会从 redo log 日志中恢复。

12.什么是独立表空间和共享表空间？它们的区别是什么？

共享表空间：指的是数据库的所有的表数据，索引文件全部放在一个文件中，默认这个共享表空间的文件路径在 data 目录下。 **独立表空间**：每一个表都将会生成以独立的文件方式来进行存储。 共享表空间和独立表空间最大的区别是如果把表放再共享表空间，即使表删除了空间也不会删除，所以表依然很大，而独立表空间如果删除表就会清除空间。

13.如何设置独立表空间？

独立表空间是由参数 innodbfileper_table 控制的，把它设置成 ON 就是独立表空间了，从 MySQL 5.6.6 版本之后，这个值就默认是 ON 了。

14.如何进行表空间收缩？

使用重建表的方式可以收缩表空间，重建表有以下三种方式：

- alter table t engine=InnoDB
- optmize table t
- truncate table t

15.说一下重建表的执行流程？

- 建立一个临时文件，扫描表 t 主键的所有数据页；
- 用数据页中表 t 的记录生成 B+ 树，存储到临时文件中；
- 生成临时文件的过程中，将所有对 t 的操作记录在一个日志文件（row log）中；
- 临时文件生成后，将日志文件中的操作应用到临时文件，得到一个逻辑数据上与表 t 相同的数据文件；
- 用临时文件替换表 t 的数据文件。

16.表的结构信息存在哪里？

表结构定义占有的存储空间比较小，在 MySQL 8 之前，表结构的定义信息存在以 .frm 为后缀的文件里，在 MySQL 8 之后，则允许把表结构的定义信息存在系统数据表之中。

17.什么是覆盖索引？

覆盖索引是指，索引上的信息足够满足查询请求，不需要再回到主键上去取数据。

18.如果把一个 InnoDB 表的主键删掉，是不是就没有主键，就没办法进行回表查询了？

可以回表查询，如果把主键删掉了，那么 InnoDB 会自己生成一个长度为 6 字节的 rowid 作为主键。

19.执行一个 update 语句以后，我再去执行 hexdump 命令直接查看 ibd 文件内容，为什么没有看到数据有改变呢？

可能是因为 update 语句执行完成后，InnoDB 只保证写完了 redo log、内存，可能还没来得及将数据写到磁盘。

20.内存表和临时表有什么区别？

- 内存表，指的是使用 Memory 引擎的表，建表语法是 create table … engine=memory。这种表的数据都保存在内存里，系统重启的时候会被清空，但是表结构还在。除了这两个特性看上去比较“奇怪”外，从其他的特征上看，它就是一个正常的表。
- 而临时表，可以使用各种引擎类型 。如果是使用 InnoDB 引擎或者 MyISAM 引擎的临时表，写数据的时候是写到磁盘上的。

21.并发事务会带来哪些问题？

- 脏读
- 修改丢失
- 不可重复读
- 幻读

22.什么是脏读和幻读？

脏读是一个事务在处理过程中读取了另外一个事务未提交的数据；幻读是指同一个事务内多次查询返回的结果集不一样（比如增加了或者减少了行记录）。

23.为什么会出现幻读？幻读会带来什么问题？

因为行锁只能锁定存在的行，针对新插入的操作没有限定，所以就有可能产生幻读。 幻读带来的问题如下：

- 对行锁语义的破坏；
- 破坏了数据一致性。

24.如何避免幻读？

使用间隙锁的方式来避免出现幻读。间隙锁，是专门用于解决幻读这种问题的锁，它锁的了行与行之间的间隙，能够阻塞新插入的操作 间隙锁的引入也带来了一些新的问题，比如：降低并发度，可能导致死锁。

25.如何查看 MySQL 的空闲连接？

在 MySQL 的命令行中使用 `show processlist;` 查看所有连接，其中 Command 列显示为 Sleep 的表示空闲连接，如下图所示：

```
mysql> show processlist;
+----+-----------------+---------------------+------+---------+-------+------------------------+------------------+
| Id | User            | Host                | db   | Command | Time  | State                  | Info             |
+----+-----------------+---------------------+------+---------+-------+------------------------+------------------+
|  4 | event_scheduler | localhost           | NULL | Daemon  | 25930 | Waiting on empty queue | NULL             |
| 12 | root            | localhost           | NULL | Query   |     0 | starting               | show processlist |
| 13 | root            | 222.90.210.42:29695 | NULL | Sleep   |     6 |                        | NULL             |
+----+-----------------+---------------------+------+---------+-------+------------------------+------------------+
3 rows in set (0.00 sec)
```

avatar

26.MySQL 中的字符串类型都有哪些？

MySQL 的字符串类型和取值如下：

类型	取值范围
CHAR(N)	0~255
VARCHAR(N)	0~65536
TINYBLOB	0~255
BLOB	0~65535
MEDUIMBLOB	0~167772150
LONGBLOB	0~4294967295
TINYTEXT	0~255

类型	取值范围
TEXT	0~65535
MEDIUMTEXT	0~167772150
LONGTEXT	0~4294967295
VARBINARY(N)	0~N 个字节的变长字节字符集
BINARY(N)	0~N 个字节的定长字节字符集

27.VARCHAR 和 CHAR 的区别是什么？分别适用的场景有哪些？

VARCHAR 和 CHAR 最大区别就是，VARCHAR 的长度是可变的，而 CHAR 是固定长度，CHAR 的取值范围为 1-255，因此 VARCHAR 可能会造成存储碎片。由于它们的特性决定了 CHAR 比较适合长度较短的字段和固定长度的字段，如身份证号、手机号等，反之则适合使用 VARCHAR。

28.MySQL 存储金额应该使用哪种数据类型？为什么？

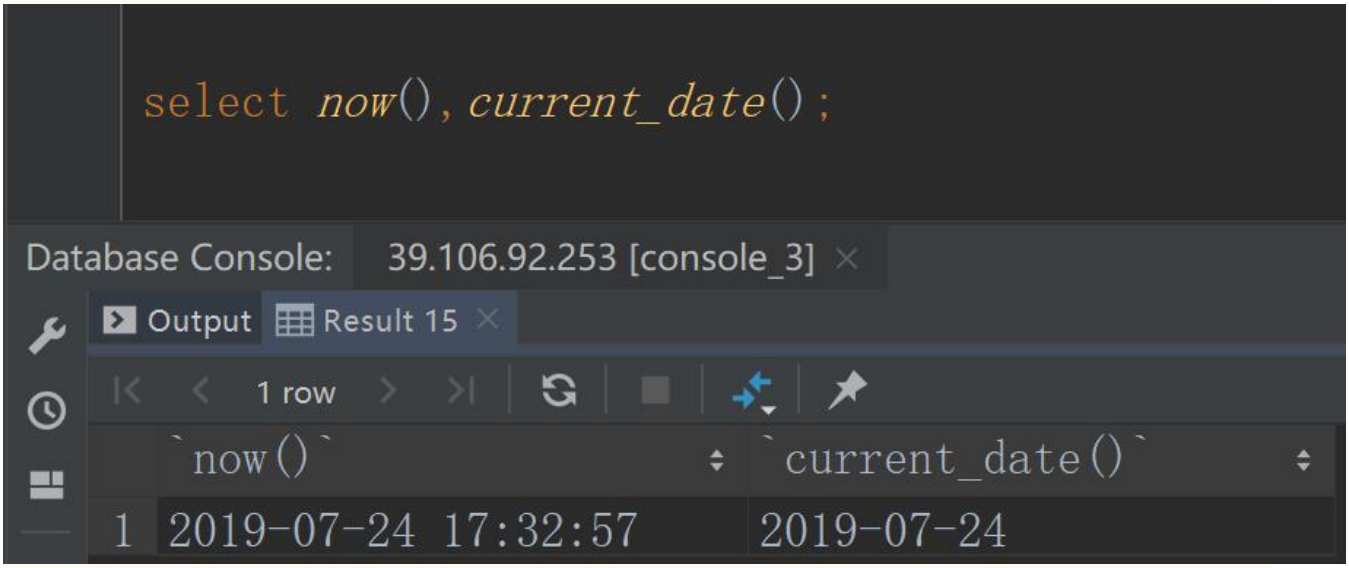
MySQL 存储金额应该使用 `decimal`，因为如果存储其他数据类型，比如 `float` 有导致小数点后数据丢失的风险。

29.limit 3,2 的含义是什么？

去除前三条数据之后查询两条信息。

30.now() 和 current_date() 有什么区别？

now() 返回当前时间包含日期和时分秒，current_date() 只返回当前时间，如下图所示：



avatar

31.如何去重计算总条数？

使用 distinct 去重，使用 count 统计总条数，具体实现脚本如下：

```
select count(distinct f) from t
```

32.lastinsertid() 函数功能是什么？有什么特点？

lastinsertid() 用于查询最后一次自增表的编号，它的特点是查询时不需要不需要指定表名，使用 `select last_insert_id()` 即可查询，因为不需要指定表名所以它始终以最后一条自增编号为主，可以被其它表的自增编号覆盖。比如 A 表的最大编号是 10，lastinsertid() 查询出来的值为 10，这时 B 表插入了一条数据，它的最大编号为 3，这个时候使用 lastinsertid() 查询的值就是 3。

33.删除表的数据有几种方式？它们有什么区别？

删除数据有两种方式：delete 和 truncate，它们的区别如下：

- delete 可以添加 where 条件删除部分数据，truncate 不能添加 where 条件只能删除整张表；
- delete 的删除信息会在 MySQL 的日志中记录，而 truncate 的删除信息不被记录在 MySQL 的日志中，因此 detele 的信息可以被找回而 truncate 的信息无法被找回；

- truncate 因为不记录日志所以执行效率比 delete 快。

delete 和 truncate 的使用脚本如下：

delete from t where username='redis'; truncate table t;

34.MySQL 中支持几种模糊查询？它们有什么区别？

MySQL 中支持两种模糊查询：regexp 和 like，like 是对任意多字符匹配或任意单字符进行模糊匹配，而 regexp 则支持正则表达式的匹配方式，提供比 like 更多的匹配方式。 regexp 和 like 的使用示例如下： select * from person where uname like '%SQL%';> select from person where uname regexp '.SQL*.';

35.MySQL 支持枚举吗？如何实现？它的用途是什么？

MySQL 支持枚举，它的实现方式如下：

```
create table t(

    sex enum('boy','grid') default 'unknown'

);
```

枚举的作用是预定义结果值，当插入数据不在枚举值范围内，则插入失败，提示错误 `Data truncated for column 'xxx' at row n` 。

36.count(column) 和 count(*) 有什么区别？

count(column) 和 count() 最大区别是统计结果可能不一致，count(column) 统计不会统计列值为 null 的数据，而 count() 则会统计所有信息，所以最终的统计结果可能会不同。

37.以下关于 count 说法正确的是？

A. count 的查询性能在各种存储引擎下的性能都是一样的。 B. count 在 MyISAM 比 InnoDB 的性能要低。 C. count 在 InnoDB 中是一行一行读取，然后累计计数的。D. count 在 InnoDB 中存储了总条数，查询的时候直接取出。

答：C

38.为什么 InnoDB 不把总条数记录下来，查询的时候直接返回呢？

因为 InnoDB 使用了事务实现，而事务的设计使用了多版本并发控制，即使是在同一时间进行查询，得到的结果也可能不相同，所以 InnoDB 不能把结果直接保存下来，因为这样是不准确的。

39.能否使用 show table status 中的表行数作为表的总行数直接使用？为什么？

不能，因为 show table status 是通过采样统计估算出来的，官方文档说误差可能在 40% 左右，所以 show table status 中的表行数不能直接使用。

40.以下哪个 SQL 的查询性能最高？

A. select count(*) from t where time>1000 and time<4500 B. show table status where name='t' C. select count(id) from t where time>1000 and time<4500 D. select count(name) from t where time>1000 and time<4500

答：B 题目解析：因为 show table status 的表行数是估算出来，而其他的查询因为添加了 where 条件，即使是 MyISAM 引擎也不能直接使用已经存储的总条数，所以 show table status 的查询性能最高。

41.InnoDB 和 MyISAM 执行 select count(*) from t，哪个效率更高？为什么？

MyISAM 效率最高，因为 MyISAM 内部维护了一个计数器，直接返回总条数，而 InnoDB 要逐行统计。

42.在 MySQL 中有对 count(*) 做优化吗？做了哪些优化？

count(*) 在不同的 MySQL 引擎中的实现方式是不相同的，在没有 where 条件的情况下：

- MyISAM 引擎会把表的总行数存储在磁盘上，因此在执行 count(*) 的时候会直接返回这个这个行数，执行效率很高；
- InnoDB 引擎中 count(*) 就比较麻烦了，需要把数据一行一行的从引擎中读出来，然后累计基数。

但即使这样，在 InnoDB 中，MySQL 还是做了优化的，我们知道对于 count() 这样的操作，遍历任意索引树得到的结果，在逻辑上都是一样的，因此，MySQL 优化器会找到最小的那颗索引树来遍历，这样就能在保证逻辑正确的前提下，尽量少扫描数据量，从而优化了 count() 的执行效率。

43.在 InnoDB 引擎中 count(*)、count(1)、count(主键)、count(字段) 哪个性能最高？

count(字段)<count(主键 id)<count(1)≈count(*) 题目解析：

- 对于 count(主键 id) 来说，InnoDB 引擎会遍历整张表，把每一行的 id 值都取出来，返回给 server 层。server 层拿到 id 后，判断是不可能为空的，就按行累加。
- 对于 count(1) 来说，InnoDB 引擎遍历整张表，但不取值。server 层对于返回的每一行，放一个数字“1”进去，判断是不可能为空的，按行累加。
- 对于 count(字段) 来说，如果这个“字段”是定义为 not null 的话，一行行地从记录里面读出这个字段，判断不能为 null，按行累加；如果这个“字段”定义允许为 null，那么执行的时候，判断到有可能是 null，还要把值取出来再判断一下，不是 null 才累加。
- 对于 count(*) 来说，并不会把全部字段取出来，而是专门做了优化，不取值，直接按行累加。

所以最后得出的结果是：count(字段)<count(主键 id)<count(1)≈count(*)。

44.MySQL 中内连接、左连接、右连接有什么区别？

- 内连（inner join）— 把匹配的关联数据显示出来；

- 左连接（left join）— 把左边的表全部显示出来，右边的表显示出符合条件的数据；
- 右连接（right join）— 把右边的表全部显示出来，左边的表显示出符合条件的数据；

45.什么是视图？如何创建视图？

视图是一种虚拟的表，具有和物理表相同的功能，可以对视图进行增、改、查操作。视图通常是一个表或者多个表的行或列的子集。 视图创建脚本如下：

```
create view vname as

select column_namesfrom table_name

where condition
```

46.视图有哪些优点？

- 获取数据更容易，相对于多表查询来说；
- 视图能够对机密数据提供安全保护；
- 视图的修改不会影响基本表，提供了独立的操作单元，比较轻量。

47.MySQL 中“视图”的概念有几个？分别代表什么含义？

MySQL 中的“视图”概念有两个，它们分别是：

- MySQL 中的普通视图也是我们最常用的 view，创建语法是 create view …,它的查询和普通表一样；
- InnoDB 实现 MVCC（Multi-Version Concurrency Control）多版本并发控制时用到的一致性读视图，它没有物理结构，作用是事务执行期间定于可以看到的数据。

48.使用 delete 误删数据怎么找回？

可以用 Flashback 工具通过闪回把数据恢复回来。

49.Flashback 恢复数据的原理是什么？

Flashback 恢复数据的原理是是修改 binlog 的内容，拿回原库重放，从而实现数据找回。

02 MySQL 索引篇

1.什么是索引？

索引是一种能帮助 MySQL 提高查询效率的数据结构。

2.索引分别有哪些优点和缺点？

索引的优点如下：

- 快速访问数据表中的特定信息，提高检索速度。
- 创建唯一性索引，保证数据表中每一行数据的唯一性。
- 加速表与表之间的连接。
- 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。

索引的缺点：

- 虽然提高了的查询速度，但却降低了更新表的速度，比如 update、insert，因为更新数据时，MySQL 不仅要更新数据，还要更新索引文件；
- 建立索引会占用磁盘文件的索引文件。

使用索引注意事项：

- 使用短索引，短索引不仅可以提高查询速度，更能节省磁盘空间和 I/O 操作；
- 索引列排序，MySQL 查询只使用一个索引，因此如果 where 子句中已经使用了索引的话，那么 order by 中的列是不会使用索引的，因此数据库默

认排序可以符合要求的情况下，不要进行排序操作；尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引；

- like 语句操作，一般情况下不鼓励使用 like 操作，如果非使用不可， 注意 like "%aaa%" 不会使用索引，而- - like "aaa%"可以使用索引；
- 不要在列上进行运算；
- 不适用 NOT IN 和 <> 操作。

3.以下 SQL 有什么问题？该如何优化？

```
select * from t where f/2=100;
```

该 SQL 会导致引擎放弃索引而全表扫描，尽量避免在索引列上计算。可改为：

```
select * from t where f=100*2;
```

4.为什么 MySQL 官方建议使用自增主键作为表的主键？

因为自增主键是连续的，在插入过程中尽量减少页分裂，即使要进行页分裂，也只会分裂很少一部分；并且自增主键也能减少数据的移动，每次插入都是插入到最后，所以自增主键作为表的主键，对于表的操作来说性能是最高的。

5.自增主键有哪些优缺点？

优点：

数据存储空间很小；

性能最好；

减少页分裂。

缺点：

数据量过大，可能会超出自增长取值范围；

无法满足分布式存储，分库分表的情况下无法合并表；

主键有自增规律，容易被破解；

综上所述：是否需要使用自增主键，需要根据自己的业务场景来设计。如果是单表单库，则优先考虑自增主键，如果是分布式存储，分库分表，则需要考虑数据合并的业务场景来做数据库设计方案。

6.索引有几种类型？分别如何创建？

MySQL 的索引有两种分类方式：逻辑分类和物理分类。 按照逻辑分类，索引可分为：

- 主键索引：一张表只能有一个主键索引，不允许重复、不允许为 NULL；
- 唯一索引：数据列不允许重复，允许为 NULL 值，一张表可有多个唯一索引，但是一个唯一索引只能包含一列，比如身份证号码、卡号等都可以作为唯一索引；
- 普通索引：一张表可以创建多个普通索引，一个普通索引可以包含多个字段，允许数据重复，允许 NULL 值插入；
- 全文索引：让搜索关键词更高效的一种索引。

按照物理分类，索引可分为：

- 聚集索引：一般是表中的主键索引，如果表中没有显示指定主键，则会选择表中的第一个不允许为 NULL 的唯一索引，如果还是没有的话，就采用 Innodb 存储引擎为每行数据内置的 6 字节 ROWID 作为聚集索引。每张表只有一个聚集索引，因为聚集索引的键值的逻辑顺序决定了表中相应行的物理顺序。聚集索引在精确查找和范围查找方面有良好的性能表现（相比于普通索引和全表扫描），聚集索引就显得弥足珍贵，聚集索引选择还是要慎重的（一般不会让没有语义的自增 id 充当聚集索引）；
- 非聚集索引：该索引中索引的逻辑顺序与磁盘上行的物理存储顺序不同（非主键的那一列），一个表中可以拥有多个非聚集索引。

各种索引的创建脚本如下：

```
-- 创建主键索引 alter table t add primary key add (`id`);

-- 创建唯一索引 alter table t add unique (`username`);

-- 创建普通索引 alter table t add index index_name (`username`);

-- 创建全文索引 alter table t add fulltext (`username`);
```

7.主索引和唯一索引有什么区别？

- 主索引不能重复且不能为空，唯一索引不能重复，但可以为空；
- 一张表只能有一个主索引，但可以有多个唯一索引；
- 主索引的查询性能要高于唯一索引。

8.在 InnDB 中主键索引为什么比普通索引的查询性能高？

因为普通索引的查询会多执行一次检索操作。比如主键查询 select * from t where id=10 只需要搜索 id 的这棵 B+ 树，而普通索引查询 select * from t where f=3 会先查询 f 索引树，得到 id 的值之后再去搜索 id 的 B+ 树，因为多执行了一次检索，所以执行效率就比主键索引要低。

9.什么叫回表查询？

普通索引查询到主键索引后，回到主键索引树搜索的过程，我们称为回表查询。

参考 SQL：

```
mysql> create table T(

id int primary key,

k int not null,

name varchar(16),index (k))engine=InnoDB;
```

如果语句是 select * from T where ID=500，即主键查询方式，则只需要检索主键 ID 字段。

```
mysql>  select * from T where ID=500;

+-----+---+-------+| id  | k | name  |

+-----+---+-------+| 500 | 5 | name5 |

+-----+---+-------+
```

如果语句是 select * from T where k=5，即普通索引查询方式，则需要先搜索 k 索引树，得到 ID 的值为 500，再到 ID 索引树搜索一次，这个过程称为回表查询。

```
mysql> select * from T where k=5;

+-----+---+-------+| id  | k | name  |

+-----+---+-------+| 500 | 5 | name5 |

+-----+---+-------+
```

也就是说，基于非主键索引的查询需要多扫描一棵索引树。因此，我们在应用中应该尽量使用主键查询。

10.如何查询一张表的所有索引？

SHOW INDEX FROM T 查询表 T 所有索引。

11.MySQL 最多可以创建多少个索引列？

MySQL 中最多可以创建 16 个索引列。

12.以下 like 查询会使用索引的是哪一个选项？为什么？

A.like '%A%' B.like '%A' C.like 'A%' D.以上都不是 答：C 题目解析：like 查询要走索引，查询字符不能以通配符（%）开始。

13.如何让 like %abc 走索引查询？

我们知道如果要让 like 查询要走索引，查询字符不能以通配符（%）开始，如果要让 like %abc 也走索引，可以使用 REVERSE() 函数来创建一个函数索引，查询脚本如下：

```
select * from t where reverse(f) like reverse('%abc');
```

14.MySQL 联合索引应该注意什么？

联合索引又叫复合索引，MySQL 中的联合索引，遵循最左匹配原则，比如，联合索引为 key(a,b,c)，则能触发索引的搜索组合是 a|ab|abc 这三种查询。

15.联合索引的作用是什么？

联合索引的作用如下：

- 用于多字段查询，比如，建了一个 key(a,b,c) 的联合索引，那么实际等于建了 `key(a)、key(a,b)、key(a,b,c)`等三个索引，我们知道，每多一个索引，就会多一些写操作和占用磁盘空间的开销，尤其是对大数据量的表来说，这可以减少一部分不必要的开销；
- 覆盖索引，比如，对于联合索引 key(a,b,c) 来说，如果使用 SQL：`select a,b,c from table where a=1 and b = 1`，就可以直接通过遍历索引取得数据，而无需回表查询，这就减少了随机的 IO 操作，减少随机的 IO 操作，可以有效的提升数据库查询的性能，是非常重要的数据库优化手段之一；
- 索引列越多，通过索引筛选出的数据越少。

16.什么是最左匹配原则？它的生效原则有哪些？

最左匹配原则也叫最左前缀原则，是 MySQL 中的一个重要原则，说的是索引以最左边的为起点任何连续的索引都能匹配上，当遇到范围查询（>、<、between、like）就会停止匹配。 生效原则来看以下示例，比如表中有一个联合索引字段 index(a,b,c)：

- where a=1 只使用了索引 a；

- where a=1 and b=2 只使用了索引 a,b；
- where a=1 and b=2 and c=3 使用 a,b,c；
- where b=1 or where c=1 不使用索引；
- where a=1 and c=3 只使用了索引 a；
- where a=3 and b like 'xx%' and c=3 只使用了索引 a,b。

17.列值为 NULL 时，查询会使用到索引吗？

在 MySQL 5.6 以上的 InnoDB 存储引擎会正常触发索引。但为了兼容低版本的 MySQL 和兼容其他数据库存储引擎，不建议使用 NULL 值来存储和查询数据，建议设置列为 NOT NULL，并设置一个默认值，比如 0 和空字符串等，如果是 datetime 类型，可以设置成 1970-01-01 00:00:00 这样的特殊值。

18.以下语句会走索引么？

```
select * from t where year(date)>2018;
```

不会，因为在索引列上涉及到了运算。

19.能否给手机号的前 6 位创建索引？如何创建？

可以，创建方式有两种：

```
alter table t add index index_phone(phone(6));create index index_phone on t(phone
(6));
```

20.什么是前缀索引？

前缀索引也叫局部索引，比如给身份证的前 10 位添加索引，类似这种给某列部分信息添加索引的方式叫做前缀索引。

21.为什么要用前缀索引？

前缀索引能有效减小索引文件的大小，让每个索引页可以保存更多的索引值，从而提高了索引查询的速度。但前缀索引也有它的缺点，不能在 order by 或者 group by 中触发前缀索引，也不能把它们用于覆盖索引。

22.什么情况下适合使用前缀索引？

当字符串本身可能比较长，而且前几个字符就开始不相同，适合使用前缀索引；相反情况下不适合使用前缀索引，比如，整个字段的长度为 20，索引选择性为 0.9，而我们对前 10 个字符建立前缀索引其选择性也只有 0.5，那么我们需要继续加大前缀字符的长度，但是这个时候前缀索引的优势已经不明显，就没有创建前缀索引的必要了。

23.什么是页？

页是计算机管理存储器的逻辑块，硬件及操作系统往往将主存和磁盘存储区分割为连续的大小相等的块，每个存储块称为一页。主存和磁盘以页为单位交换数据。数据库系统的设计者巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次磁盘 IO 就可以完全载入。

24.索引的常见存储算法有哪些？

- 哈希存储法：以 key、value 方式存储，把值存入数组中使用哈希值确认数据的位置，如果发生哈希冲突，使用链表存储数据；
- 有序数组存储法：按顺序存储，优点是可以使用二分法快速找到数据，缺点是更新效率，适合静态数据存储；
- 搜索树：以树的方式进行存储，查询性能好，更新速度快。

25.InnoDB 为什么要使用 B+ 树，而不是 B 树、Hash、红黑树或二叉树？

因为 B 树、Hash、红黑树或二叉树存在以下问题：

- B 树：不管叶子节点还是非叶子节点，都会保存数据，这样导致在非叶子节点中能保存的指针数量变少（有些资料也称为扇出），指针少的情况下要保存大量数据，只能增加树的高度，导致 IO 操作变多，查询性能变低；
- Hash：虽然可以快速定位，但是没有顺序，IO 复杂度高；
- 二叉树：树的高度不均匀，不能自平衡，查找效率跟数据有关（树的高度），并且 IO 代价高；

- 红黑树：树的高度随着数据量增加而增加，IO 代价高。

26.为什么 InnoDB 要使用 B+ 树来存储索引？

B+Tree 中的 B 是 Balance，是平衡的意思，它在经典 B Tree 的基础上进行了优化，增加了顺序访问指针，在 B+Tree 的每个叶子节点增加一个指向相邻叶子节点的指针，就形成了带有顺序访问指针的 B+Tree，这样就提高了区间访问性能：如果要查询 key 为从 18 到 49 的所有数据记录，当找到 18 后，只需顺着节点和指针顺序遍历就可以一次性访问到所有数据节点，极大提到了区间查询效率（无需返回上层父节点重复遍历查找减少 IO 操作）。

索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上，这样的话，索引查找过程中就要产生磁盘 IO 消耗，相对于内存存取，IO 存取的消耗要高几个数量级，所以索引的结构组织要尽量减少查找过程中磁盘 IO 的存取次数，从而提升索引效率。综合所述，InnDB 只有采取 B+ 树的数据结构存储索引，才能提供数据库整体的操作性能。

27.唯一索引和普通索引哪个性能更好？

- 对于查询操作来说：普通索引和唯一索引的性能相近，都是从索引树中进行查询；
- 对于更新操作来说：唯一索引要比普通索引执行的慢，因为唯一索引需要先将数据读取到内存中，再在内存中进行数据的唯一效验，所以执行起来要比普通索引更慢。

28.优化器选择查询索引的影响因素有哪些？

优化器的目的是使用最小的代价选择最优的执行方案，影响优化器选择索引的因素如下：

- 扫描行数，扫描的行数越少，执行代价就越少，执行效率就会越高；
- 是否使用了临时表；
- 是否排序。

29.MySQL 是如何判断索引扫描行数的多少？

MySQL 的扫描行数是通过索引统计列（cardinality）大致得到并且判断的，而索引统计列（cardinality）可以通过查询命令 show index 得到，索引扫描行数的多少就是通过这个值进行判断的。

30.MySQL 是如何得到索引基数的？它准确吗？

MySQL 的索引基数并不准确，因为 MySQL 的索引基数是通过采样统计得到的，比如 InnoDb 默认会有 N 个数据页，采样统计会统计这些页面上的不同值得到一个平均值，然后除以这个索引的页面数就得到了这个索引基数。

31.MySQL 如何指定查询的索引？

在 MySQL 中可以使用 force index 强行选择一个索引，具体查询语句如下：

```
select * from t force index(index_t)
```

32.在 MySQL 中指定了查询索引，为什么没有生效？

我们知道在 MySQL 中使用 force index 可以指定查询的索引，但并不是一定会生效，原因是 MySQL 会根据优化器自己选择索引，如果 force index 指定的索引出现在候选索引上，这个时候 MySQL 不会在判断扫描的行数的多少直接使用指定的索引，如果没在候选索引中，即使 force index 指定了索引也是不会生效的。

33.以下 or 查询有什么问题吗？该如何优化？

```
select * from t where num=10 or num=20;
```

答：如果使用 or 查询会使 MySQL 放弃索引而全表扫描，可以改为：

```
select * from t where num=10 union select * from t where num=20;
```

34.以下查询要如何优化？

表中包含索引：

```
KEY mid (mid)

KEY begintime (begintime)

KEY dg (day,group)
```

使用以下 SQL 进行查询：

```
select f from t where day='2010-12-31' and group=18 and begintime<'2019-12-31 12:14:
28' order by begintime limit 1;
```

答：此查询理论上是使用 dg 索引效率更高，通过 explain 可以对比查询扫描次数。由于使用了 order by begintime 则使查询放弃了 dg 索引，而使用 begintime 索引，从侧面印证 order by 关键字会影响查询使用索引，这时可以使查询强制使用索引，改为以下 SQL：

```
select f from t use index(dg) where day='2010-12-31' and group=18 and begintime< '201
9-12-31 12:14:28' order by begintime limit 1;
```

35.MySQL 会错选索引吗？

MySQL 会错选索引，比如 k 索引的速度更快，但是 MySQL 并没有使用而是采用了 v 索引，这种就叫错选索引，因为索引选择是 MySQL 的服务层的优化器来自动选择的，但它在复杂情况下也和人写程序一样出现缺陷。

36.如何解决 MySQL 错选索引的问题？

- 删除错选的索引，只留下对的索引；
- 使用 force index 指定索引；
- 修改 SQL 查询语句引导 MySQL 使用我们期望的索引，比如把 order by b limit 1 改为 order by b,a limit 1 语义是相同的，但 MySQL 查询的时候会考虑使用 a 键上的索引。

37.如何优化身份证的索引？

在中国因为前 6 位代表的是地区，所以很多人的前六位都是相同的，如果我们使用前缀索引为 6 位的话，性能提升也并不是很明显，但如果设置的位数过长，那么占用的磁盘空间也越大，数据页能放下的索引值就越少，搜索效率也越低。针对这种情况优化方案有以下两种：

- 使用身份证倒序存储，这样设置前六位的意义就很大了；
- 使用 hash 值，新创建一个字段用于存储身份证的 hash 值。

03 MySQL 锁篇

1.什么是锁？MySQL 中提供了几类锁？

锁是实现数据库并发控制的重要手段，可以保证数据库在多人同时操作时能够正常运行。MySQL 提供了全局锁、行级锁、表级锁。其中 InnoDB 支持表级锁和行级锁，MyISAM 只支持表级锁。

2.什么是死锁？

是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象,若无外力作用,它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的过程称为死锁。

死锁是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象,若无外力作用,它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的过程称为死锁。

3.常见的死锁案例有哪些？

将投资的钱拆封几份借给借款人，这时处理业务逻辑就要把若干个借款人一起锁住 select * from xxx where id in (xx,xx,xx) for update。

批量入库，存在则更新，不存在则插入。解决方法 insert into tab(xx,xx) on duplicate key update xx='xx'。

4.如何处理死锁？

对待死锁常见的两种策略：

通过 innodblockwait_timeout 来设置超时时间，一直等待直到超时；

发起死锁检测，发现死锁之后，主动回滚死锁中的某一个事务，让其它事务继续执行。

5.如何查看死锁？

使用命令 `show engine innodb status` 查看最近的一次死锁。

InnoDB Lock Monitor 打开锁监控，每 15s 输出一次日志。使用完毕后建议关闭，否则会影响数据库性能。

6.如何避免死锁？

- 为了在单个 InnoDB 表上执行多个并发写入操作时避免死锁，可以在事务开始时通过为预期要修改的每个元祖（行）使用 SELECT … FOR UPDATE 语句来获取必要的锁，即使这些行的更改语句是在之后才执行的。
- 在事务中，如果要更新记录，应该直接申请足够级别的锁，即排他锁，而不应先申请共享锁、更新时再申请排他锁，因为这时候当用户再申请排他锁时，其他事务可能又已经获得了相同记录的共享锁，从而造成锁冲突，甚至死锁
- 如果事务需要修改或锁定多个表，则应在每个事务中以相同的顺序使用加锁语句。在应用中，如果不同的程序会并发存取多个表，应尽量约定以相同的顺序来访问表，这样可以大大降低产生死锁的机会
- 通过 SELECT … LOCK IN SHARE MODE 获取行的读锁后，如果当前事务再需要对该记录进行更新操作，则很有可能造成死锁。
- 改变事务隔离级别。

7.InnoDB 默认是如何对待死锁的？

InnoDB 默认是使用设置死锁时间来让死锁超时的策略，默认 innodblockwait_timeout 设置的时长是 50s。

8.如何开启死锁检测？

设置 innodbdeadlockdetect 设置为 on 可以主动检测死锁，在 Innodb 中这个值默认就是 on 开启的状态。

9.什么是全局锁？它的应用场景有哪些？

全局锁就是对整个数据库实例加锁，它的典型使用场景就是做全库逻辑备份。这个命令可以使整个库处于只读状态。使用该命令之后，数据更新语句、数据定义语句、更新类事务的提交语句等操作都会被阻塞。

10.什么是共享锁？

共享锁又称读锁 (read lock)，是读取操作创建的锁。其他用户可以并发读取数据，但任何事务都不能对数据进行修改（获取数据上的排他锁），直到已释放所有共享锁。当如果事务对读锁进行修改操作，很可能会造成死锁。

11.什么是排它锁？

排他锁 exclusive lock（也叫 writer lock）又称写锁。

若某个事物对某一行加上了排他锁，只能这个事务对其进行读写，在此事务结束之前，其他事务不能对其进行加任何锁，其他进程可以读取,不能进行写操作，需等待其释放。

排它锁是悲观锁的一种实现，在上面悲观锁也介绍过。

若事务 1 对数据对象 A 加上 X 锁，事务 1 可以读 A 也可以修改 A，其他事务不能再对 A 加任何锁，直到事物 1 释放 A 上的锁。这保证了其他事务在事物 1 释放 A 上的锁之前不能再读取和修改 A。排它锁会阻塞所有的排它锁和共享锁。

12.使用全局锁会导致什么问题？

如果在主库备份，在备份期间不能更新，业务停摆，所以更新业务会处于等待状态。

如果在从库备份，在备份期间不能执行主库同步的 binlog，导致主从延迟。

13.如何处理逻辑备份时，整个数据库不能插入的情况？

如果使用全局锁进行逻辑备份就会让整个库成为只读状态，幸好官方推出了一个逻辑备份工具 MySQLdump 来解决了这个问题，只需要在使用 MySQLdump 时，使用参数 -single-transaction 就会在导入数据之前启动一个事务来保证数据的一致性，并且这个过程是支持数据更新操作的。

14.如何设置数据库为全局只读锁？

使用命令 `flush tables with read lock`（简称 FTWRL）就可以实现设置数据库为全局只读锁。

15.除了 FTWRL 可以设置数据库只读外，还有什么别的方法？

除了使用 FTWRL 外，还可以使用命令 set global readonly=true 设置数据库为只读。

16.FTWRL 和 set global readonly=true 有什么区别？

FTWRL 和 set global readonly=true 都是设置整个数据库为只读状态，但他们最大的区别就是，当执行 FTWRL 的客户端断开之后，整个数据库会取消只读，而 set global readonly=true 会一直让数据处于只读状态。

17.如何实现表锁？

MySQL 里标记锁有两种：表级锁、元数据锁（meta data lock）简称 MDL。表锁的语法是 lock tables t read/write。

可以用 unlock tables 主动释放锁，也可以在客户端断开的时候自动释放。lock tables 语法除了会限制别的线程的读写外，也限定了本线程接下来的操作对象。

对于 InnoDB 这种支持行锁的引擎，一般不使用 lock tables 命令来控制并发，毕竟锁住整个表的影响面还是太大。

MDL：不需要显式使用，在访问一个表的时候会被自动加上。

MDL 的作用：保证读写的正确性。

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。

读锁之间不互斥，读写锁之间，写锁之间是互斥的，用来保证变更表结构操作的安全性。

MDL 会直到事务提交才会释放，在做表结构变更的时候，一定要小心不要导致锁住线上查询和更新。

18.悲观锁和乐观锁有什么区别？

顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会 block 直到它拿到锁。正因为如此，悲观锁需要耗费较多的时间，另外与乐观锁相对应的，悲观锁是由数据库自己实现了的，要用的时候，我们直接调用数据库的相关语句就可以了。

说到这里，由悲观锁涉及到的另外两个锁概念就出来了，它们就是共享锁与排它锁。共享锁和排它锁是悲观锁的不同的实现，它俩都属于悲观锁的范畴。

乐观锁是用数据版本（Version）记录机制实现，这是乐观锁最常用的一种实现方式。何谓数据版本？即为数据增加一个版本标识，一般是通过为数据库表增加一个数字类型的 version 字段来实现。当读取数据时，将 version 字段的值一同读出，数据每更新一次，对此 version 值加 1。当我们提交更新的时候，判断数据库表对应记录的当前版本信息与第一次取出来的 version 值进行比对，如果数据库表当前版本号与第一次取出来的 version 值相等，则予以更新，否则认为是过期数据。

比如： 1、数据库表三个字段，分别是 id、value、version `select id,value,version from t where id=#{id}` 2、每次更新表中的 value 字段时，为了防止发生冲突，需要这样操作

```
update t

set value=2,version=version+1

where id=#{id} and version=#{version}
```

19.乐观锁有什么优点和缺点？

因为没有加锁所以乐观锁的优点就是执行性能高。它的缺点就是有可能产生 ABA 的问题，ABA 问题指的是有一个变量 V 初次读取的时候是 A 值，并且在准备赋值的时候检查到它仍然是 A 值，会误以为没有被修改会正常的执行修改操作，实际上这段时间它的值可能被改了其他值，之后又改回为 A 值，这个问题被称为 ABA 问题。

20.InnoDB 存储引擎有几种锁算法？

- Record Lock — 单个行记录上的锁；
- Gap Lock — 间隙锁，锁定一个范围，不包括记录本身；
- Next-Key Lock — 锁定一个范围，包括记录本身。

21.InnoDB 如何实现行锁？

行级锁是 MySQL 中粒度最小的一种锁，他能大大减少数据库操作的冲突。

INNODB 的行级锁有共享锁（S LOCK）和排他锁（X LOCK）两种。共享锁允许事物读一行记录，不允许任何线程对该行记录进行修改。排他锁允许当前事物删除或更新一行记录，其他线程不能操作该记录。

共享锁：SELECT … LOCK IN SHARE MODE，MySQL 会对查询结果集中每行都添加共享锁，前提是当前线程没有对该结果集中的任何行使用排他锁，否则申请会阻塞。

排他锁：select * from t where id=1 for update，其中 id 字段必须有索引，MySQL 会对查询结果集中每行都添加排他锁，在事物操作中，任何对记录的更新与删除操作会自动加上排他锁。前提是当前没有线程对该结果集中的任何行使用排他锁或共享锁，否则申请会阻塞。

22.优化锁方面你有什么建议？

- 尽量使用较低的隔离级别。
- 精心设计索引， 并尽量使用索引访问数据， 使加锁更精确， 从而减少锁冲突的机会。
- 选择合理的事务大小，小事务发生锁冲突的几率也更小。
- 给记录集显示加锁时，最好一次性请求足够级别的锁。比如要修改数据的话，最好直接申请排他锁，而不是先申请共享锁，修改时再请求排他锁，这样容易产生死锁。
- 不同的程序访问一组表时，应尽量约定以相同的顺序访问各表，对一个表而言，尽可能以固定的顺序存取表中的行。这样可以大大减少死锁的机会。
- 尽量用相等条件访问数据，这样可以避免间隙锁对并发插入的影响。
- 不要申请超过实际需要的锁级别。
- 除非必须，查询时不要显示加锁。 MySQL 的 MVCC 可以实现事务中的查询不用加锁，优化事务性能；MVCC 只在 COMMITTED READ（读提交）和 REPEATABLE READ（可重复读）两种隔离级别下工作。
- 对于一些特定的事务，可以使用表锁来提高处理速度或减少死锁的可能。

04 MySQL 日志篇

1. MySQL 有哪些重要的日志文件？

MySQL 中的重要日志分为以下几个： ① 错误日志：用来记录 MySQL 服务器运行过程中的错误信息，比如，无法加载 MySQL 数据库的数据文件，或权限不正确等都会被记录在此，还有复制环境下，从服务器进程的信息也会被记录进

错误日志。默认情况下，错误日志是开启的，且无法被禁止。默认情况下，错误日志是存储在数据库的数据文件目录中，名称为 hostname.err，其中 hostname 为服务器主机名。在 MySQL 5.5.7 之前，数据库管理员可以删除很长时间之前的错误日志，以节省服务器上的硬盘空间， MySQL 5.5.7 之后，服务器将关闭此项功能，只能使用重命名原来的错误日志文件，手动冲洗日志创建一个新的，命令为：

```
mv hostname.err  hostname.err.old mysqladmin flush-logs
```

② 查询日志：查询日志在 MySQL 中被称为 general log(通用日志)，查询日志里的内容不要被“查询日志”误导，认为里面只存储 select 语句，其实不然，查询日志里面记录了数据库执行的所有命令，不管语句是否正确，都会被记录，具体原因如下:

- insert 查询为了避免数据冲突，如果此前插入过数据，当前插入的数据如果跟主键或唯一键的数据重复那肯定会报错；
- update 时也会查询因为更新的时候很可能会更新某一块数据；
- delete 查询，只删除符合条件的数据；

因此都会产生日志，在并发操作非常多的场景下，查询信息会非常多，那么如果都记录下来会导致 IO 非常大，影响 MySQL 性能，因此如果不是在调试环境下，是不建议开启查询日志功能的。

查询日志的开启有助于帮助我们分析哪些语句执行密集，执行密集的 select 语句对应的数据是否能够被缓存，同时也可以帮助我们分析问题，所以，我们可以根据自己的实际情况来决定是否开启查询日志。

查询日志模式是关闭的，可以通过以下命令开启查询日志：

```
set global generallog=1 set global logoutput='table';
```

general_log=1 为开启查询日志，0 为关闭查询日志，这个设置命令即时生效，不用重启 MySQL 服务器。

③ 慢日志：慢查询会导致 CPU、IOPS、内存消耗过高，当数据库遇到性能瓶颈时，大部分时间都是由于慢查询导致的。开启慢查询日志，可以让 MySQL 记录下查询超过指定时间的语句，之后运维人员通过定位分析，能够很好的优化数据库性能。默认情况下，慢查询日志是不开启的，只有手动开启了，慢查询才会被记录到慢查询日志中。使用如下命令记录当前数据库的慢查询语句：

```
set global slowquerylog='ON';
```

使用 set global slowquerylog='ON' 开启慢查询日志，只是对当前数据库有效，如果 MySQL 数据库重启后就会失效。所以如果要永久生效，就要修改配置文件 my.cnf，设置 slowquerylog=1 并重启 MySQL 服务器。

④ redo log（重做日志）：为了最大程度的避免数据写入时，因为 IO 瓶颈造成的性能问题，MySQL 采用了这样一种缓存机制，先将数据写入内存中，再批量把内存中的数据统一刷回磁盘。为了避免将数据刷回磁盘过程中，因为掉电或系统故障带来的数据丢失问题，InnoDB 采用 redo log 来解决此问题。

⑤ undo log（回滚日志）：用于存储日志被修改前的值，从而保证如果修改出现异常，可以使用 undo log 日志来实现回滚操作。 undo log 和 redo log 记录物理日志不一样，它是逻辑日志，可以认为当 delete 一条记录时，undo log 中会记录一条对应的 insert 记录，反之亦然，当 update 一条记录时，它记录一条对应相反的 update 记录，当执行 rollback 时，就可以从 undo log 中的逻辑记录读取到相应的内容并进行回滚。undo log 默认存放在共享表空间中，在 ySQL 5.6 中，undo log 的存放位置还可以通过变量 innodbundodirectory 来自定义存放目录，默认值为“.”表示 datadir 目录。

⑥ bin log（二进制日志）：是一个二进制文件，主要记录所有数据库表结构变更，比如，CREATE、ALTER TABLE 等，以及表数据修改，比如，INSERT、UPDATE、DELETE 的所有操作，bin log 中记录了对 MySQL 数据库执行更改的所有操作，并且记录了语句发生时间、执行时长、操作数据等其它额外信息，但是它不记录 SELECT、SHOW 等那些不修改数据的 SQL 语句。

binlog 的作用如下：

- 恢复（recovery）：某些数据的恢复需要二进制日志。比如，在一个数据库全备文件恢复后，用户可以通过二进制日志进行 point-in-time 的恢复；
- 复制（replication）：其原理与恢复类似，通过复制和执行二进制日志使一台远程的 MySQL 数据库（一般称为 slave 或者 standby）与一台 MySQL 数据库（一般称为 master 或者 primary）进行实时同步；
- 审计（audit）：用户可以通过二进制日志中的信息来进行审计，判断是否有对数据库进行注入攻击。

除了上面介绍的几个作用外，binlog 对于事务存储引擎的崩溃恢复也有非常重要的作用，在开启 binlog 的情况下，为了保证 binlog 与 redo 的一致性，MySQL 将采用事务的两阶段提交协议。当 MySQL 系统发生崩溃时，事务在存储引擎内部的状态可能为 prepared（准备状态）和 commit（提交状态）两种，对于 prepared 状态的事务，是进行提交操作还是进行回滚操作，这时需要参考 binlog，如果事务在 binlog 中存在，那么将其提交；如果不在 binlog 中存在，那么将其回滚，这样就保证了数据在主库和从库之间的一致性。

binlog 默认是关闭状态，可以在 MySQL 配置文件（my.cnf）中通过配置参数 log-bin = [base-name] 开启记录 binlog 日志，如果不指定 base-name，则默认二进制日志文件名为主机名，并以自增的数字作为后缀，比如：mysql-bin.000001，所在目录为数据库所在目录（datadir）。

通过以下命令来查询 binlog 是否开启：

```
show variables like 'log_%';
```

```
mysql> show variables like 'log_%';
+----------------------------------------+-----------------------------------------+
| Variable_name                          | Value                                   |
+----------------------------------------+-----------------------------------------+
| log_bin                                | ON                                      |
| log_bin_basename                       | /var/lib/mysql/binlog                   |
| log_bin_index                          | /var/lib/mysql/binlog.index             |
| log_bin_trust_function_creators        | OFF                                     |
| log_bin_use_v1_row_events              | OFF                                     |
| log_error                              | /var/log/mysqld.log                     |
| log_error_services                     | log_filter_internal; log_sink_internal  |
| log_error_suppression_list             |                                         |
| log_error_verbosity                    | 2                                       |
| log_output                             | FILE                                    |
| log_queries_not_using_indexes          | OFF                                     |
| log_slave_updates                      | ON                                      |
| log_slow_admin_statements              | OFF                                     |
| log_slow_extra                         | OFF                                     |
| log_slow_slave_statements              | OFF                                     |
| log_statements_unsafe_for_binlog       | ON                                      |
| log_throttle_queries_not_using_indexes | 0                                       |
| log_timestamps                         | UTC                                     |
+----------------------------------------+-----------------------------------------+
18 rows in set (0.00 sec)
```

binlog 格式分为: STATEMENT、ROW 和 MIXED 三种：

-

STATEMENT 格式的 binlog 记录的是数据库上执行的原生 SQL 语句。这种格式的优点是简单，简单地记录和执行这些语句，能够让主备保持同步，在主服务器上执行的 SQL 语句，在从服务器上执行同样的语句。另一个好处是二进制日志里的时间更加紧凑，所以相对而言，基于语句的复制模式不会使用太多带宽，同时也节约磁盘空间。并且通过 mysqlbinlog 工具容易读懂其中的内容。缺点就是同一条 SQL 在主库和从库上执行的时间可能稍微或很大不相同，因此在传输的二进制日志中，除了查询语句，还包括了一些元数据信息，如当前的时间戳。即便如此，还存在着一些无法被正确复制的 SQL。比如，使用 INSERT INTO TB1 VALUE(CUURENT_DATE()) 这一条使用函数的语句插入的数据复制到当前从服务器上来就会发生变化，存储过程和触发器在使用基于语句的复制模式时也可能存在问题；另外一个问题就是基于语句的复制必须是串行化的，比如：InnoDB 的 next-key 锁等，并不是所有的存储引擎都支持基于语句的复制；

ROW 格式是从 MySQL 5.1 开始支持基于行的复制，也就是基于数据的复制，基于行的更改。这种方式会将实际数据记录在二进制日志中，它有其自身的一些优点和缺点，最大的好处是可以正确地复制每一行数据，一些语句可以被更加有效地复制，另外就是几乎没有基于行的复制模式无法处理的场景，对于所有的

SQL 构造、触发器、存储过程等都能正确执行；它的缺点就是二进制日志可能会很大，而且不直观，所以，你不能使用 mysqlbinlog 来查看二进制日志，也无法通过看二进制日志判断当前执行到那一条 SQL 语句。现在对于 ROW 格式的二进制日志基本是标配了，主要是因为它的优势远远大于缺点，并且由于 ROW 格式记录行数据，所以可以基于这种模式做一些 DBA 工具，比如数据恢复，不同数据库之间数据同步等；

MIXED 也是 MySQL 默认使用的二进制日志记录方式，但 MIXED 格式默认采用基于语句的复制，一旦发现基于语句的无法精确的复制时，就会采用基于行的复制。比如用到 UUID()、USER()、CURRENTUSER()、ROWCOUNT() 等无法确定的函数。

-

2.redo log 和 binlog 有什么区别？

redo log（重做日志）和 binlog（归档日志）都是 MySQL 的重要的日志，它们的区别如下：

- 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 ”。
- redo log 是 InnoDB 引擎特有的；binlog 是 MySQL 的 Server 层实现的，所有引擎都可以使用。
- redo log 是循环写的，空间固定会用完；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。

最开始 MySQL 里并没有 InnoDB 引擎，MySQL 自带的引擎是 MyISAM，但是 MyISAM 没有 crash-safe 的能力，binlog 日志只能用于归档。而 InnoDB 是另一个公司以插件形式引入 MySQL 的，既然只依靠 binlog 是没有 crash-safe 能力的，所以 InnoDB 使用另外一套日志系统，也就是 redo log 来实现 crash-safe 能力。

3.什么是 crash-safe？

crash-safe 是指发生宕机等意外情况下，服务器重启后数据依然不会丢失的情况。

4.什么是脏页和干净页？

MySQL 为了操作的性能优化，会把数据更新先放入内存中，之后再统一更新到磁盘。当内存数据和磁盘数据内容不一致的时候，我们称这个内存页为脏页；内存数据写到磁盘后，内存的数据和磁盘上的内容就一致了，我们称为“干净页”。

5.什么情况下会引发 MySQL 刷脏页（flush）的操作？

- 内存写满了，这个时候就会引发 flush 操作，对应到 InnoDB 就是 redo log 写满了；
- 系统的内存不足了，当需要新的内存页的时候，就会淘汰一些内存页，如果淘汰的是脏页这个时候就会触发 flush 操作；
- 系统空闲的时候，MySQL 会同步内存中的数据到磁盘也会触发 flush 操作；
- MySQL 服务关闭的时候也会刷脏页，触发 flush 操作。

6.MySQL 刷脏页的速度很慢可能是什么原因？

在 MySQL 中单独刷一个脏页的速度是很快的，如果发现刷脏页的速度很慢，说明触发了 MySQL 刷脏页的“连坐”机制，MySQL 的“连坐”机制是指当 MySQL 刷脏页的时候如果发现相邻的数据页也是脏页也会一起刷掉，而这个动作可以一直蔓延下去，这就是导致 MySQL 刷脏页慢的原因了。

7.如何控制 MySQL 只刷新当前脏页？

在 InnoDB 中设置 innodbflushneighbors 这个参数的值为 0，来规定 MySQL 只刷当前脏页，MySQL 8 这个值默认是 0。

8.MySQL 的 WAL 技术是解决什么问题的？

A.防止误删除，找回数据用的 B.容灾恢复，为了还原异常数据用的 C.事务处理，为了数据库的稳定性 D.为了降低 IO 成本 答：D 题目解析：WAL 技术的全称是 Write Ahead Logging（中文：预写式日志），是先写日志，再写磁盘的方式，因为每次更新都写磁盘的话 IO 成本很高，所以才有了 WAL 技术。

9.为什么有时候会感觉 MySQL 偶尔卡一下？

如果偶尔感觉 MySQL 卡一下，可能是 MySQL 正在刷脏页，正在把内存中的更新操作刷到磁盘中。

10.redo log 和 binlog 是怎么关联的?

它们有一个共同的数据字段，叫 XID。崩溃恢复的时候，会按顺序扫描 redo log：

如果碰到既有 prepare、又有 commit 的 redo log，就直接提交；

如果碰到只有 parepare、而没有 commit 的 redo log，就拿着 XID 去 binlog 找对应的事务。

11.MySQL 怎么知道 binlog 是完整的?

statement 格式的 binlog，完整的标识是最后有 COMMIT 关键字。

row 格式的 binlog，完整的标识是最后会有一个 XID event 关键字。

12.MySQL 中可不可以只要 binlog，不要 redo log?

不可以，binlog 没有崩溃恢复的能力。

13.MySQL 中可不可以只要 redo log，不要 binlog?

不可以，原因有以下两个：

- redo log 是循环写不能保证所有的历史数据，这些历史数据只能在 binlog 中找到；
- binlog 是高可用的基础，高可用的实现原理就是 binlog 复制。

14.为什么 binlog cache 是每个线程自己维护的，而 redo log buffer 是全局共用的？

因为 binlog 是不能“被打断的”，一个事务的 binlog 必须连续写，因此要整个事务完成后，再一起写到文件里。而 redo log 并没有这个要求，中间有生成的日志可以写到 redo log buffer 中，redo log buffer 中的内容还能“搭便车”，其他事务提交的时候可以被一起写到磁盘中。

15.事务执行期间，还未提交，如果发生 crash，redo log 丢失，会导致主备不一致呢？

不会，因为这时候 binlog 也还在 binlog cache 里，没发给备库，crash 以后 redo log 和 binlog 都没有了，从业务角度看这个事务也没有提交，所以数据是一致的。

16.在 MySQL 中用什么机制来优化随机读/写磁盘对 IO 的消耗？

redo log 是用来节省随机写磁盘的 IO 消耗，而 change buffer 主要是节省随机读磁盘的 IO 消耗。redo log 会把 MySQL 的更新操作先记录到内存中，之后再统一更新到磁盘，而 change buffer 也是把关键查询数据先加载到内存中，以便优化 MySQL 的查询。

17.以下说法错误的是？

A.redo log 是 InnoDB 引擎特有的，它的固定大小的 B.redo log 日志是不全的，只有最新的一些日志，这和它的内存大小有关 C.redo log 可以保证数据库异常重启之后，数据不丢失 D.binlog 是 MySQL 自带的日志，它能保证数据库异常重启之后，数据不丢失 答：D 题目解析：binlog 是 MySQL 自带的日志，但它并不能保证数据库异常重启之后数据不丢失。

18.以下说法正确的是？

A.redo log 日志是追加写的，后面的日志并不会覆盖前面的日志 B.binlog 日志是追加写的，后面的日志并不会覆盖前面的日志 C.redo log 和 binlog 日志都是追加写的，后面的日志并不会覆盖前面的日志 D.以上说法都正确 答：B 题目解析：binlog 日志是追加写的，后面的日志并不会覆盖前面的日志，redo log 日志是固定大小的，后面的日志会覆盖前面的日志。

19.有没有办法把 MySQL 的数据恢复到过去某个指定的时间节点？怎么恢复？

可以恢复，只要你备份了这段时间的所有 binlog，同时做了全量数据库的定期备份，比如，一天一备，或者三天一备，这取决于你们的备份策略，这个时候你就可以把之前备份的数据库先还原到测试库，从备份的时间点开始，将备份的 binlog 依次取出来，重放到你要恢复数据的那个时刻，这个时候就完成了数据到指定节点的恢复。比如，今天早上 9 点的时候，你想把数据恢复成今天早上 6:00:00 的状态，这个时候你可以先取出今天凌晨（00:01:59）备份的数据库文件，还原到测试库，再从 binlog 文件中依次取出 00:01:59 之后的操作信息，重放到 6:00:00 这个时刻，这就完成了数据库的还原。

20.MySQL 命令和内置函数篇

21.如何用命令行方式连接 MySQL 数据库？

使用 `mysql -u用户名 -p密码`；输入用户名和密码就可以正常进入数据库连接了，实例如下：

```
mysql -uroot -p123456;
```

其中，用户名为 root，密码为 123456。

22.关于命令 `mysql -h 127.0.0.1 -uroot -P 3307 -p3307` 以下说法错误的是？

A.-h 和 -P 可以省略 B.-u 和用户名之间不能有空格 C.-p 和密码之间不能用空格 D.小写 -p 对应的是用户密码，大写 -P 对应的是 MySQL 服务器的端口

答：B 题目解析：-p 和密码之间不能用空格，否则空格会被识别为密码的一部分，提示密码错误。-u 和用户名之间可以有空格。

23.如何创建用户？并给用户授权？

创建用户使用关键字：`CREATE USER` ，授权使用关键字： `GRANT` ，具体实现脚本如下：

```
-- 创建用户 laowang

create user 'laowang'@'localhost' identified by '123456';

-- 授权 test 数据库给 laowang

grant all on test.* to 'laowang'@'localhost'
```

24.如何修改 MySQL 密码？

使用如下命令，修改密码：

mysqladmin -u 用户名 -p 旧密码 password 新密码;

注意：刚开始 root 没有密码，所以 -p 旧密码一项就可以省略了。

25.如何使用 SQL 创建数据库，并设置数据库的编码格式？

创建数据库可使用关键字： `CREATE DATABASE` ，设置编码格式使用关键字：`CHARSET` ，具体 SQL 如下：

```
create database learndb default charset utf8 collate utf8_general_ci;
```

26.如何修改数据库、表的编码格式？

使用 `alter` 关键字设置库或表的编码格式即可，具体代码如下：

> mysql> alter database dbname default character set utf8; mysql> alter table t default character set utf8;

27.如何使用 SQL 创建表？

创建表的 SQL 如下：

> create table t(tid int not null autoincrement, tname char(50) not null, tage int null default 18, primary key(t_id))engine=innodb;

其中：

- auto_increment：表示自增；
- primary key：用于指定主键；
- engine：用于指定表的引擎。

28.在 MySQL 命令行中如何查看表结构信息？

使用 `desc 表名` 查看表结构信息，示例信息如下：

```
mysql> desc person;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| uname | varchar(255) | YES  |     | NULL    |                |
| age   | int(11)      | YES  |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

使用 `desc person;` 查看表 `person` 的结构信息。

29.如何使用 SQL 查看已知表的建表脚本？

查看已知表的建表脚本，命令如下：

mysql> show create table 表名;

效果如下图所示：

```
mysql> show create table person;
+--------+---------------------------------------------------------------
----------------------------------------------------------------------+
| Table  | Create Table
                                                                       |
+--------+---------------------------------------------------------------
----------------------------------------------------------------------+
| person | CREATE TABLE `person` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `uname` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8 |
+--------+---------------------------------------------------------------
----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

30.如何使用 SQL 语句更新表结构？

更新表结构信息可以使用 alter table 子句，如，为表增加一列的脚本如下：alter

alter table t add name char(20);

如果要重命名表名，使用如下命令：

rename table new_t to t;

31.MySQL 有哪些删除方式？有什么区别？

MySQL 有三种删除方式： 1）删除表数据：

delete from t;

2）删除数据，保留表结构：

```
truncate table t;
```

3）删数据和表结构：

```
drop table t;
```

它们的区别如下：

- delete 可以有条件的删除，也可以回滚数据，删除数据时进行两个动作：删除与备份，所以速度很慢；
- truncate 删除所有数据，无条件选择删除，不可回滚，保留表结构；
- drop：删除数据和表结构 删除速度最快。

32.如何开启和关闭 MySQL 服务？

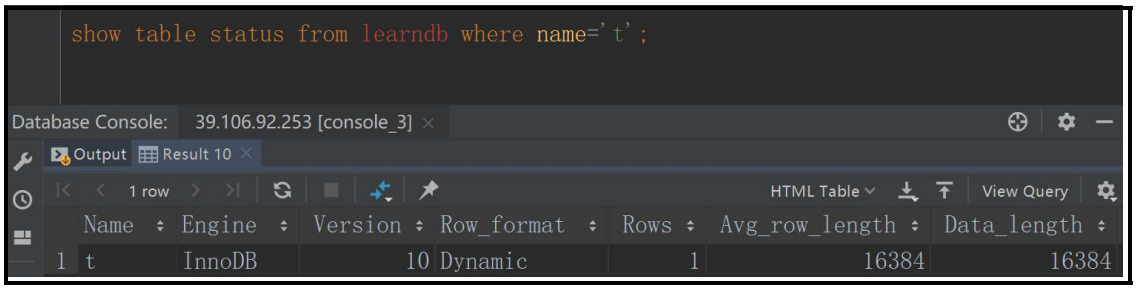
使用 `systemctl stop mysqld` 停止 MySQL 服务，使用 `systemctl start mysqld` 启动 MySQL 服务。

33.如何查询当前 MySQL 安装的版本号？

使用 `SELECT VERSION();` 可以查询当前连接的 MySQL 的版本号。

34.如何查看某张表的存储引擎？

可使用 `show table status from db where name='t';` 查询数据库 db 中表 t 的所有信息，其中 `Engine` 列表示表 t 使用的存储引擎，如下图所示：



35.如何查看当前数据库增删改查的执行次数统计？

使用以下命令行查看：

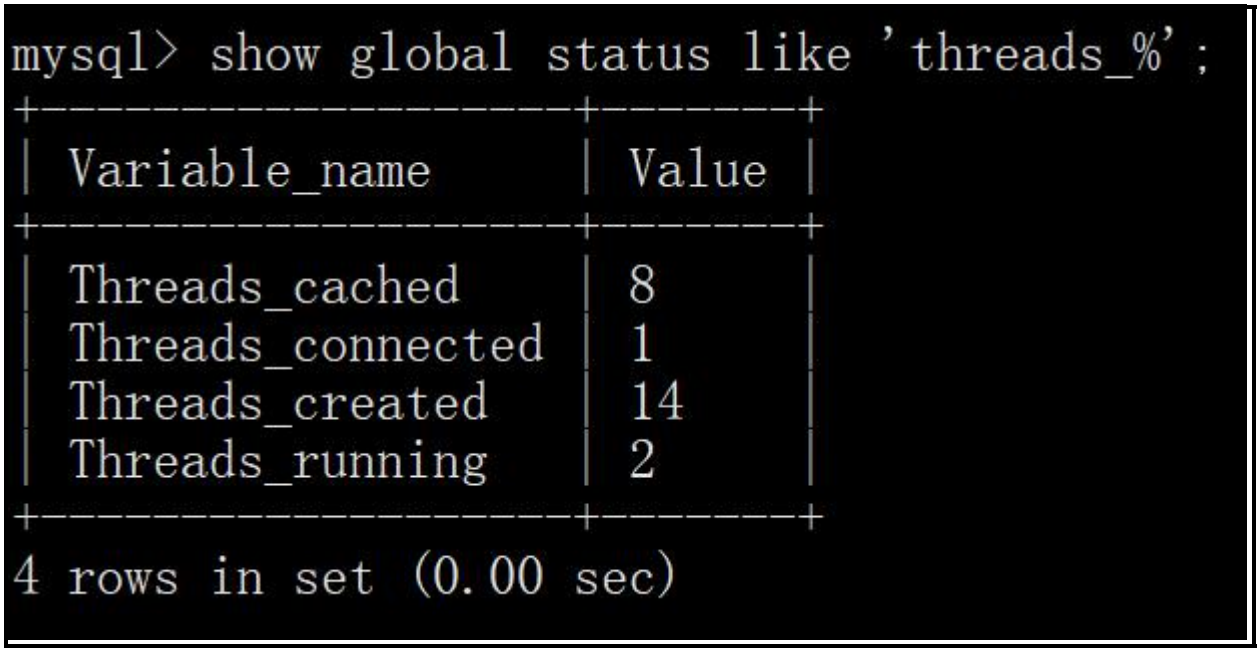
```
mysql> show global status where variablename
in('comselect','cominsert','comdelete','comupdate');
+---------------+-------+ | Variablename | Value | +---------------+-------+ |
Comdelete | 0 | | Cominsert | 1 | | Comselect | 40 | | Comupdate | 0 |
+---------------+-------+
```

36.如何查询线程连接数？

使用如下命令：

```
mysql> show global status like 'threads_%';
```

执行效果如下图所示：



其中：

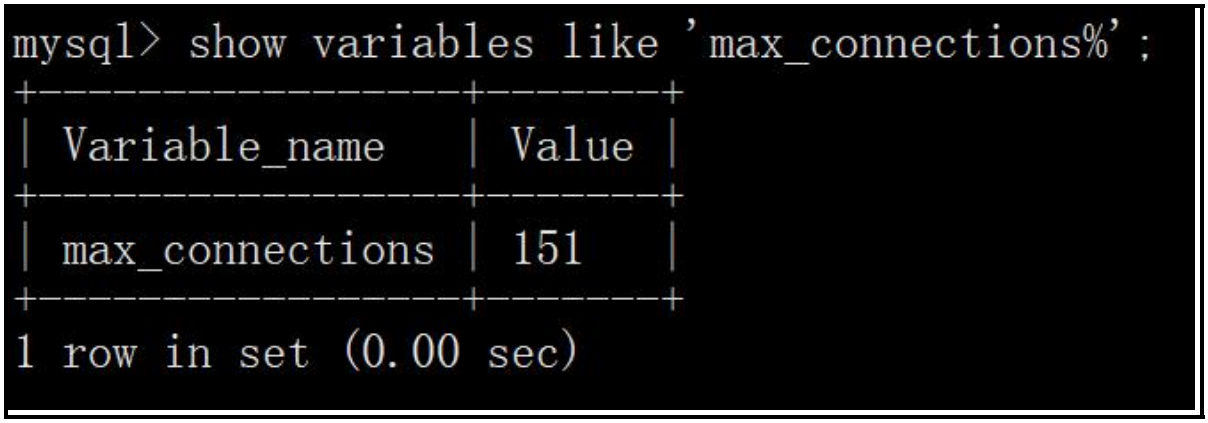
- Threads_cached：代表当前此时此刻线程缓存中有多少空闲线程；
- Threads_connected：代表当前已建立连接的数量，因为一个连接就需要一个线程，所以也可以看成当前被使用的线程数；
- Threads_created：代表从最近一次服务启动，已创建线程的数量；
- Threads_running：代表当前激活的（非睡眠状态）线程数。

37.如何查看 MySQL 的最大连接数？能不能修改？怎么修改？

查询 MySQL 最大连接数，使用如下命令：

```
mysql> show variables like 'max_connections%';
```

此命令输出的结果如下：

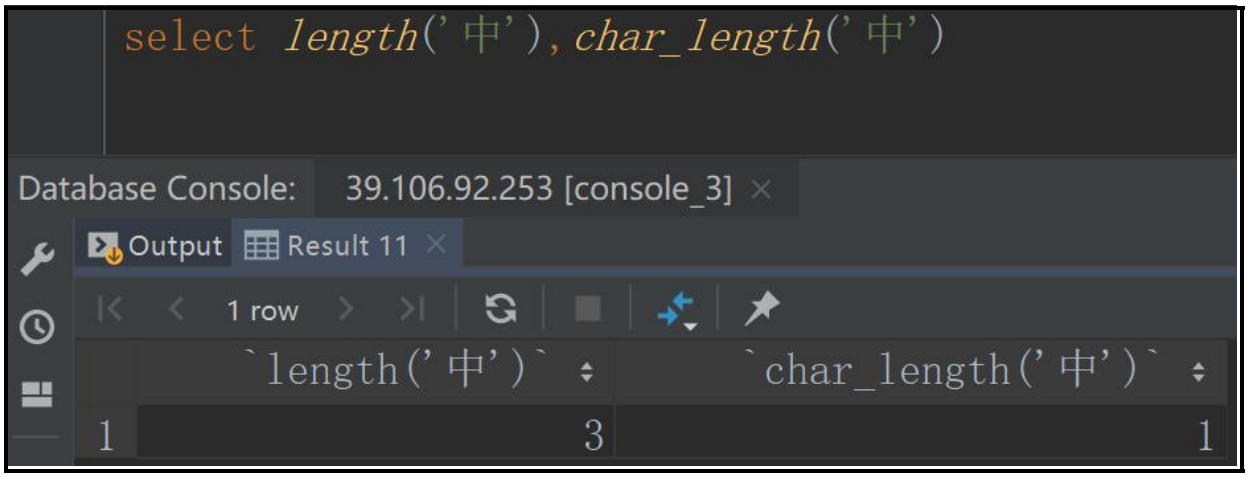


可以修改 MySQL 的最大连接数，可以在 MySQL 的配置文件 my.cnf 里修改最大连接数，通过修改 maxconnections 的值，然后重启 MySQL 就会生效，如果 my.ini 文件中没有找到 maxconnections，可自行添加 max_connections 的设置，内容如下：

```
max_connections=200
```

38.CHAR_LENGTH 和 LENGTH 有什么区别？

CHARLENGTH 是字符数，而 LENGTH 是字节数。它们在不同编码下，值是不相同的，比如对于 UTF-8 编码来说，一个中文字的 LENGTH 为 1，而 CHARLENGTH 通常等于 3，如下图所示：



39.UNION 和 UNION ALL 的用途是什么？有什么区别？

UNION 和 UNION ALL 都是用于合并数据集的，它们的区别如下：

- 去重：UNION 会对结果进行去重，UNION ALL 则不会进行去重操作；
- 排序：UNION 会对结果根据字段进行排序，而 UNION ALL 则不会进行排序；
- 性能：UNION ALL 的性能要高于 UNION。

40.以下关于 WHERE 和 HAVING 说法正确的是？

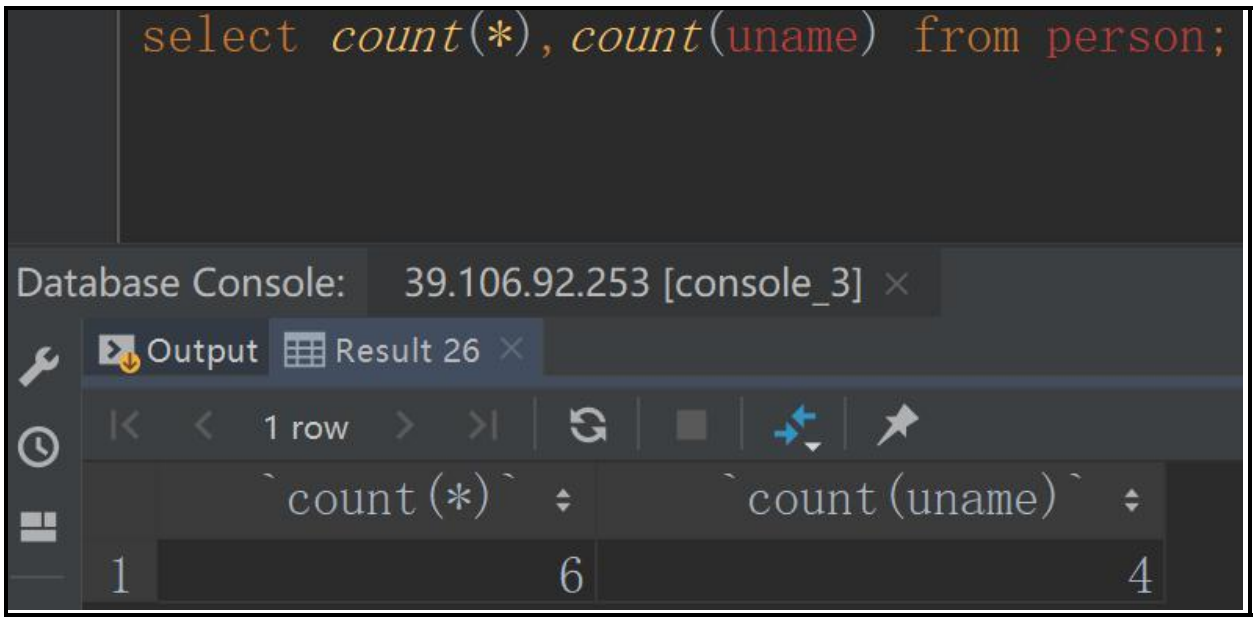
A.任何情况 WHERE 和 HAVING 都可以相互替代 B.GROUP BY 前后都可以使用 WHERE C.使用 SELECT X FROM T HAVING Y>20 查询报错 D.使用 SELECT X FROM T WHERE Y>20 查询报错 答：C，HAVING 非报错用法是 `SELECT X,Y FROM T HAVING Y>20` 。

41.空值和 NULL 的区别是什么？

空值表示字段的值为空，而 NULL 则表示字段没有值，它们的区别如下：

- 空值不占用空间，NULL 值是未知的占用空间；
- 空值判断使用 `=''` 或 `<>''` 来判断，NULL 值使用 `IS NULL` 或 `IS NOT NULL` 来判断；
- 使用 COUNT 统计某字段时，如果是 NULL 则会忽略不统计，而空值则会算入统计之内。

比如，其中字段 `name` 有两个 `NULL` 值和一个空值，查询结果如图：



42.MySQL 的常用函数有哪些？

- sum(field) – 求某个字段的和值；
- count(*) – 查询总条数；
- min(field) – 某列中最小的值；
- max(field) – 某列中最大的值；
- avg(field) – 求平均数；
- current_date() – 获取当前日期；
- now() – 获取当前日期和时间；
- concat(a, b) – 连接两个字符串值以创建单个字符串输出；
- datediff(a, b) – 确定两个日期之间的差异，通常用于计算年龄。

05 MySQL 性能优化

1.MySQL 性能指标都有哪些？如何得到这些指标？

MySQL 的性能指标如下：

① **TPS（Transaction Per Second）** 每秒事务数，即数据库每秒执行的事务数。

MySQL 本身没有直接提供 TPS 参数值，如果我们想要获得 TPS 的值，只有我们自己计算了，可以根据 MySQL 数据库提供的状态变量，来计算 TPS。

需要使用的参数：

- Com_commit ：表示提交次数，通过命令 *show global status like 'Com_commit';* 获取；
- Com_rollback：表示回滚次数，通过命令 *show global status like 'Com_rollback';* 获取。

我们定义第一次获取的 Comcommit 的值与 Comrollback 值的和为 c_r1，时间为 t1；

第二次获取的 Comcommit 的值与 Comrollback 值的和为 cr2，时间为 t2，t1 与 t2 单位为秒。 那么 TPS = (cr2 - c_r1) / (t2 - t1) 算出来的就是该 MySQL 实例在 t1 与 t2 生命周期之间的平均 TPS。

② **QPS（Query Per Second）** 每秒请求次数，也就是数据库每秒执行的 SQL 数量，包含 INSERT、SELECT、UPDATE、DELETE 等。 QPS = Queries / Seconds Queries 是系统状态值—总查询次数，可以通过 *show status like 'queries';* 查询得出，如下所示：

```
mysql> show status like 'queries';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Queries       | 743   |
+---------------+-------+
1 row in set (0.00 sec)
```

Seconds 是监控的时间区间，单位为秒。 比如，采样 10 秒内的查询次数，那么先查询一次 Queries 值（Q1），等待 10 秒，再查询一次 Queries 值（Q2），那么 QPS 就可以通过，如下公式获得：

```
QPS = (Q2 - Q1) / 10
```

③ **IOPS（Input/Output Operations per Second）** 每秒处理的 I/O 请求次数。

IOPS 是判断磁盘 I/O 能力的指标之一，一般来讲 IOPS 指标越高，那么单位时间内能够响应的请求自然也就越多。理论上讲，只要系统实际的请求数低于 IOPS 的能力，就相当于每一个请求都能得到即时响应，那么 I/O 就不会是瓶颈了。

注意：IOPS 与磁盘吞吐量不一样，吞吐量是指单位时间内可以成功传输的数据数量。

可以使用 iostat 命令，查看磁盘的 IOPS，命令如下：

```
yum install sysstat iostat  -dx 1 10
```

执行效果如下图所示：

```
[root@iZ2ze0nc5n41zomzyqtksmZ soft]# iostat  -dx 1 10
Linux 3.10.0-862.14.4.el7.x86_64 (iZ2ze0nc5n41zomzyqtksmZ)      09/20/2019      _x86_64_        (2 CPU)

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.11    0.01    0.29     0.07     2.62    18.15     0.00    6.08    2.37    6.15   0.40   0.01

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     3.00    0.00    2.00     0.00    20.00    20.00     0.00    0.50    0.00    0.50   0.50   0.10

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
```

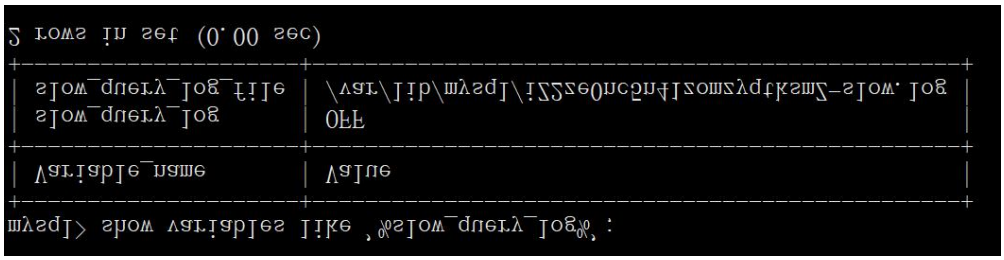
IOPS = r/s + w/s 其中：

- r/s：代表每秒读了多少次；
- w/s：代表每秒写了多少次。

2.什么是慢查询？

慢查询是 MySQL 中提供的一种慢查询日志，它用来记录在 MySQL 中响应时间超过阀值的语句，具体指运行时间超过 longquerytime 值的 SQL，则会被记录到慢查询日志中。longquerytime 的默认值为 10，意思是运行 10S 以上的语句。默认情况下，MySQL 数据库并不启动慢查询日志，需要我们手动来设置这个参数，如果不是调优需要的话，一般不建议启动该参数，因为开启慢查询日志会给 MySQL 服务器带来一定的性能影响。慢查询日志支持将日志记录写入文件，也支持将日志记录写入数据库表。

使用 *mysql> show variables like '%slow_query_log%';* 来查询慢查询日志是否开启，执行效果如下图所示：



slowquerylog 的值为 OFF 时，表示未开启慢查询日志。

3.如何开启慢查询日志？

开启慢查询日志，可以使用如下 MySQL 命令：

```
mysql> set global slowquerylog=1
```

不过这种设置方式，只对当前数据库生效，如果 MySQL 重启也会失效，如果要永久生效，就必须修改 MySQL 的配置文件 my.cnf，配置如下：

```
slowquerylog =1 slowquerylogfile=/tmp/mysqlslow.log
```

4.如何定位慢查询？

使用 MySQL 中的 explain 分析执行语句，比如：

```
explain select * from t where id=5;
```

如下图所示：

```
mysql> explain select * from person where id=5;
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| 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)
```

其中：

- id — 选择标识符。id 越大优先级越高，越先被执行。
- select_type — 表示查询的类型。
- table — 输出结果集的表
- partitions — 匹配的分区
- type — 表示表的连接类型
- possible_keys — 表示查询时，可能使用的索引
- key — 表示实际使用的索引
- key_len — 索引字段的长度
- ref— 列与索引的比较
- rows — 大概估算的行数
- filtered — 按表条件过滤的行百分比
- Extra — 执行情况的描述和说明

其中最重要的就是 type 字段，type 值类型如下：

- all — 扫描全表数据
- index — 遍历索引
- range — 索引范围查找
- index_subquery — 在子查询中使用 ref
- uniquesubquery — 在子查询中使用 eqref
- refornull — 对 null 进行索引的优化的 ref
- fulltext — 使用全文索引
- ref — 使用非唯一索引查找数据
- eq_ref — 在 join 查询中使用主键或唯一索引关联
- const — 将一个主键放置到 where 后面作为条件查询， MySQL 优化器就能把这次查询优化转化为一个常量，如何转化以及何时转化，这个取决于优化器，这个比 eq_ref 效率高一点

5.MySQL 的优化手段都有哪些？

MySQL 的常见的优化手段有以下五种：

① 查询优化

- 避免 SELECT *，只查询需要的字段。
- 小表驱动大表，即小的数据集驱动大的数据集，比如，当 B 表的数据集小于 A 表时，用 in 优化 exist，两表执行顺序是先查 B 表，再

查 A 表，查询语句：select * from A where id in (select id from B) 。

- 一些情况下,可以使用连接代替子查询,因为使用 join 时,MySQL 不会在内存中创建临时表。

② 优化索引的使用

- 尽量使用主键查询，而非其他索引，因为主键查询不会触发回表查询。
- 不做列运算，把计算都放入各个业务系统实现
- 查询语句尽可能简单，大语句拆小语句，减少锁时间
- 不使用 select * 查询
- or 查询改写成 in 查询
- 不用函数和触发器
- 避免 %xx 查询
- 少用 join 查询
- 使用同类型比较，比如 '123' 和 '123'、123 和 123
- 尽量避免在 where 子句中使用 != 或者 <> 操作符，查询引用会放弃索引而进行全表扫描
- 列表数据使用分页查询，每页数据量不要太大
- 用 exists 替代 in 查询
- 避免在索引列上使用 is null 和 is not null
- 尽量使用主键查询
- 避免在 where 子句中对字段进行表达式操作

- 尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型

③ 表结构设计优化

- 使用可以存下数据最小的数据类型。
- 使用简单的数据类型，int 要比 varchar 类型在 MySQL 处理简单。
- 尽量使用 tinyint、smallint、mediumint 作为整数类型而非 int。
- 尽可能使用 not null 定义字段，因为 null 占用 4 字节空间。
- 尽量少用 text 类型，非用不可时最好考虑分表。
- 尽量使用 timestamp，而非 datetime。
- 单表不要有太多字段，建议在 20 个字段以内。

④ 表拆分

当数据库中的数据非常大时，查询优化方案也不能解决查询速度慢的问题时，我们可以考虑拆分表，让每张表的数据量变小，从而提高查询效率。 **a）垂直拆分**：是指数据表列的拆分，把一张列比较多的表拆分为多张表，比如，用户表中一些字段经常被访问，将这些字段放在一张表中，另外一些不常用的字段放在另一张表中，插入数据时，使用事务确保两张表的数据一致性。垂直拆分的原则：

- 把不常用的字段单独放在一张表；
- 把 text，blob 等大字段拆分出来放在附表中；
- 经常组合查询的列放在一张表中。

b）水平拆分：指数据表行的拆分，表的行数超过 200 万行时，就会变慢，这时可以把一张的表的数据拆成多张表来存放。

通常情况下，我们使用取模的方式来进行表的拆分，比如，一张有 400W 的用户表 users，为提高其查询效率我们把其分成 4 张表 users1，users2，users3，users4，然后通过用户 ID 取模的方法，同时查询、更新、删除也是通过取模的方法来操作。

⑤ 读写分离

一般情况下对数据库而言都是"读多写少"，换言之，数据库的压力多数是因为大量的读取数据的操作造成的，我们可以采用数据库集群的方案，使用一个库作为主库，负责写入数据；其他库为从库，负责读取数据。这样可以缓解对数据库的访问压力。

6.MySQL 常见读写分离方案有哪些？

MySQL 常见的读写分离方案，如下列表：

1）应用层解决方案 可以通过应用层对数据源做路由来实现读写分离，比如，使用 SpringMVC + MyBatis，可以将 SQL 路由交给 Spring，通过 AOP 或者 Annotation 由代码显示的控制数据源。 优点：路由策略的扩展性和可控性较强。 缺点：需要在 Spring 中添加耦合控制代码。

2）中间件解决方案 通过 MySQL 的中间件做主从集群，比如：Mysql Proxy、Amoeba、Atlas 等中间件都能符合需求。 优点：与应用层解耦。

缺点：增加一个服务维护的风险点，性能及稳定性待测试，需要支持代码强制主从和事务。

7.介绍一下 Sharding-JDBC 的功能和执行流程？

Sharding-JDBC 在客户端对数据库进行水平分区的常用解决方案，也就是保持表结构不变，根据策略存储数据分片，这样每一片数据被分散到不同的表或者库中，Sharding-JDBC 提供以下功能：

- 分库分表
- 读写分离
- 分布式主键生成

Sharding-JDBC 的执行流程：当业务代码调用数据库执行的时候，先触发 Sharding-JDBC 的分配规则对 SQL 语句进行解析、改写之后，才会对改写的 SQL 进行执行和结果归并，然后返回给调用层。

8.什么是 MySQL 多实例？如何配置 MySQL 多实例？

MySQL 多实例就是在同一台服务器上启用多个 MySQL 服务，它们监听不同的端口，运行多个服务进程，它们相互独立，互不影响的对外提供服务，便于节约服务器资源与后期架构扩展。 多实例的配置方法有两种：

- 一个实例一个配置文件，不同端口；
- 同一配置文件(my.cnf)下配置不同实例，基于 MySQL 的 d_multi 工具。

9.怎样保证确保备库无延迟？

通常保证主备无延迟有以下三种方法：

- 每次从库执行查询请求前，先判断 secondsbehindmaster 是否已经等于 0。如果还不等于 0 ，那就必须等到这个参数变为 0 才能执行查询请求，secondsbehindmaster 参数是用来衡量主备延迟时间的长短；
- 对比位点确保主备无延迟。MasterLogFile 和 ReadMasterLogPos，表示的是读到的主库的最新位点，RelayMasterLogFile 和 ExecMasterLog_Pos，表示的是备库执行的最新位点；
- 对比 GTID 集合确保主备无延迟。AutoPosition=1 ，表示这对主备关系使用了 GTID 协议；RetrievedGtidSet，是备库收到的所有日志的 GTID 集合；ExecutedGtid_Set，是备库所有已经执行完成的 GTID 集合。

06 MySQL 开放性问题

1.有一个超级大表，如何优化分页查询？

超级大表的分页优化分有以下两种方式：

- 数据库层面优化：利用子查询优化超多分页场景，比如：SELECT a.* FROM 表 1 a, (select id from 表 1 where 条件 LIMIT 100000,20) b where a.id=b.id ，先快速定位需要获取的 id 段，然后再关联查询。MySQL 并不是跳

过 offset 行，而是取 offset+N 行，然后返回放弃前 offset 行，返回 N 行，那当 offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行 SQL 改写，利用子查询先快速定位需要获取的 id 段，然后再关联查询，就是对分页进行 SQL 改写的具体实现；

- 程序层面优化：可以利用缓存把查询的结果缓存起来，这样再下一次查询的时候性能就非常高了。

2.线上修改表结构有哪些风险？

线上修改表结构有可能 MySQL 服务器阻塞，因为在执行 DML（select、update、delete、insert）操作时，会给表增加一个元数据锁，这个元数据锁是为了保证在查询期间表结构不会被修改，而执行修改表结构时，必须要等待元数据锁完成之后才能执行，这就可能造成数据库服务器的阻塞。

在 MySQL 5.6 开始提供了 online ddl 功能，允许一些 DDL（create table/view/index/syn/cluster）语句和 DML 语句并发，在 5.7 版本对 online ddl 又有了增强，这使得大部分 DDL 操作可以在线进行，详见：https://dev.mysql.com/doc/refman/5.7/en/innodb-create-index-overview.html，这使得在线上修改表结构的风险变的更大，如果在业务开发过程中必须在线修改表结构，可以参考以下方案：

- 尽量在业务量小的时间段进行；
- 查看官方文档，确认要做的表修改可以和 DML 并发，不会阻塞线上业务；

- 推荐使用 percona 公司的 pt-online-schema-change 工具，该工具被官方的 online ddl 更为强大，它的基本原理是：通过 insert…select… 语句进行一次全量拷贝，通过触发器记录表结构变更过程中产生的增量，从而达到表结构变更的目的。比如，要对 A 表进行变更，它的主要流程为：

1）创建目的表结构的空表 A_new；

2）在 A 表上创建触发器，包括增、删、改触发器；

3）通过 insert…select…limit N 语句分片拷贝数据到目的表；

4）Copy 完成后，将 A_new 表 rename 到 A 表。

3.查询长时间不返回可能是什么原因？应该如何处理？

查询速度慢的原因很多，常见如下几种： 1）查询字段没有索引或者没有触发索引查询，没有触发索引查询的情况如下： 不会使用索引的情况如下：

- 以 % 开头的 like 查询不会使用 b-tree 索引；
- 数据类型出现隐式转换时不会使用索引，比如，某列是 varchar 类型，却使用了 columnname=1 的查询语句，这是不会使用索引，正确触发索引的查询语句为：columnname='1' ；
- 不符合最左前缀原则；

- 如果查询条件有 or 分割，or 前面的使用索引，or 后面的未使用索引，则不会使用索引，因为即使 or 之前的使用了索引，但是 or 之后的也需要全表查询，索引就忽略索引，直接全表查询；
- 如果 MySQL 认为使用索引会比全表查询更慢，则不会使用索引。

2）I/O 压力大，读取磁盘速度变慢。 3）内存不足 4）网络速度慢 5）查询出的数据量过大，可以采用多次查询或其他的方法降低数据量 6）死锁，一般碰到这种情况的话，大概率是表被锁住了，可以使用 *show processlist;* 命令，看看 SQL 语句的状态，再针对不同的状态做相应的处理。

```
mysql> show processlist;
+-----+-----------------+-----------+------+---------+---------+-----------------------+------------------+
| Id  | User            | Host      | db   | Command | Time    | State                 | Info             |
+-----+-----------------+-----------+------+---------+---------+-----------------------+------------------+
|   4 | event_scheduler | localhost | NULL | Daemon  | 5008858 | Waiting on empty queue | NULL             |
| 697 | root            | localhost | NULL | Query   |       0 | starting              | show processlist |
+-----+-----------------+-----------+------+---------+---------+-----------------------+------------------+
2 rows in set (0.00 sec)
```

其中，当 State 列值为 Locked 时，表示被锁定。 其它关于查看死锁的命令： a）查看当前的事务：

```
select * from informationschema.innodbtrx;
```

b）查看当前锁定的事务：

```
select * from informationschema.innodblocks;
```

c）查看当前等锁的事务

```
select * from informationschema.innodblock_waits;
```

以上问题的解决方案如下：

1）正确创建和使用索引。 2）把数据、日志、索引放到不同的 IO 设备上，减少主数据库的 IO 操作。更换 MySQL 的磁盘为固态硬盘，以提高磁盘的

IO 性能。 3）升级内存，更换更大的内存。 4）提升网速，升级带宽。 5）用 Profiler 来跟踪查询，得到查询所需的时间，找出有问题的 SQL 语句，优化 SQL。 6）查询时值返回需要的字段。 7）设置死锁的超时时间，限制和避免死锁消耗过多服务器的资源。 8）尽量少用视图，它的效率低，对视图操作比直接对表操作慢,可以用存储过程来代替视图。不要用视图嵌套，嵌套视图增加了寻找原始数据的难度。

4.MySQL 主从延迟的原因有哪些？

主从延迟可以根据 MySQL 提供的命令判断，比如，在从服务器使用命令： *show slave status;*，其中 SecondsBehindMaster 如果为 0 表示主从复制状态正常。 导致主从延迟的原因有以下几个：

- 主库有大事务处理；
- 主库做大量的增、删、改操作；
- 主库对大表进行字段新增、修改或添加索引等操作；
- 主库的从库太多，导致复制延迟。从库数量一般 3-5 个为宜，要复制的节点过多，导致复制延迟；
- 从库硬件配置比主库差，导致延迟。查看 Master 和 Slave 的配置，可能因为从库的配置过低，执行时间长，由此导致的复制延迟时间长；
- 主库读写压力大，导致复制延迟；
- 从库之间的网络延迟。主从库网卡、网线、连接的交换机等网络设备都可能成为复制的瓶颈，导致复制延迟，另外跨公网主从复制很容易导致主从复制延迟。

5.如何保证数据不被误删？

保证数据不被误删的方法如下列表：

- 权限控制与分配（数据库和服务器权限）
- 避免数据库账号信息泄露，在生产环境中，业务代码不要使用明文保存数据库连接信息；
- 重要的数据库操作，通过平台型工具自动实施，减少人工操作；
- 部署延迟复制从库，万一误删除时用于数据回档，且从库设置为 read-only；
- 确认备份制度及时有效；
- 启用 SQL 审计功能，养成良好 SQL 习惯；
- 启用 sqlsafeupdates 选项，不允许没 where 条件的更新/删除；
- 将系统层的 rm 改为 mv；
- 线上不进行物理删除，改为逻辑删除（将 row data 标记为不可用）；
- 启用堡垒机，屏蔽高危 SQL；
- 降低数据库中普通账号的权限级别；
- 开启 binlog，方便追溯数据。

6.MySQL 服务器 CPU 飙升应该如何处理？

使用 *show full processlist;* 查出慢查询，为了缓解数据库服务器压力，先使用 kill 命令杀掉慢查询的客户端，效果如下：

```
mysql> show full processlist;
+----+-----------------+---------------------+---------+---------+--------+------------------------+----------------------+
| Id | User            | Host                | db      | Command | Time   | State                  | Info                 |
+----+-----------------+---------------------+---------+---------+--------+------------------------+----------------------+
|  4 | event_scheduler | localhost           | NULL    | Daemon  | 164972 | Waiting on empty queue | NULL                 |
| 25 | root            | localhost           | learndb | Query   |      0 | starting               | show full processlist |
| 28 | root            | 111.21.173.130:65451 | learndb | Sleep   |    340 |                        | NULL                 |
+----+-----------------+---------------------+---------+---------+--------+------------------------+----------------------+
3 rows in set (0.00 sec)
```

然后再去项目中找到执行慢的 SQL 语句进行修改和优化。

7.MySQL 毫无规律的异常重启，可能产生的原因是什么？该如何解决？

可能是积累的长连接导致内存占用太多，被系统强行杀掉导致的异常重启，因为在 MySQL 中长连接在执行过程中使用的临时内存对象，只有在连接断开的时候才会释放，这就会导致内存不断飙升，解决方案如下：

- 定期断开空闲的长连接；
- 如果是用的是 MySQL 5.7 以上的版本，可以定期执行 mysqlresetconnection 重新初始化连接资源，这个过程会释放之前使用的内存资源，恢复到连接刚初始化的状态。

8.如何实现一个高并发的系统？

这道面试题涉及的知识点比较多，主要考察的是面试者的综合技术能力。高并发系统的设计手段有很多，主要体现在以下五个方面。

1）前端优化

① 静态资源缓存：将活动页面上的所有可以静态的元素全部静态化，尽量减少动态元素；通过 CDN、浏览器缓存，来减少客户端向服务器端的数据请求。

② 禁止重复提交：用户提交之后按钮置灰，禁止重复提交。 ③ 用户限流：在某一时间段内只允许用户提交一次请求，比如，采取 IP 限流。

2）中间层负载分发

可利用负载均衡，比如 nginx 等工具，可以将并发请求分配到不同的服务器，从而提高了系统处理并发的能力。 nginx 负载分发的五种方式：

① 轮询（默认） 每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器不能正常响应，nginx 能自动剔除故障服务器。 ② 按权重（weight） 使用 weight 参数，指定轮询几率，weight 和访问比率成正比，用于后端服务器性能不均的情况，配置如下：

```
upstream backend {
    server 192.168.0.14 weight=10;
    server 192.168.0.15 weight=10;
}
```

③ IP 哈希值（ip_hash） 每个请求按访问 IP 的哈希值分配，这样每个访客固定访问一个后端服务器，可以解决 session 共享的问题，配置如下：

```
upstream backend {
    ip_hash;
    server 192.168.0.14:88;
```

```
    server 192.168.0.15:80;
}
```

④ 响应时间（fair） 按后端服务器的响应时间来分配请求，响应时间短的优先分配，配置如下：

```
upstream backend {
    fair;
    server server1.com;
    server server2.com;
}
```

⑤ URL 哈希值（url_hash） 按访问 url 的 hash 结果来分配请求，和 IP 哈希值类似。

```
upstream backend {
    hash $request_uri;
    server server1.com;
    server server2.com;
}
```

3）控制层（网关层）

限制同一个用户的访问频率，限制访问次数，防止多次恶意请求。

4）服务层

① 业务服务器分离：比如，将秒杀业务系统和其他业务分离，单独放在高配服务器上，可以集中资源对访问请求抗压。 ② 采用 MQ（消息队列）缓存请求：MQ 具有削峰填谷的作用，可以把客户端的请求先导流到 MQ，程序在从 MQ 中进行消费（执行请求），这样可以避免短时间内大量请求，导致服务器程序无法响应的问题。③ 利用缓存应对读请求，比如，使用 Redis 等缓存，利用 Redis 可以分担数据库很大一部分压力。

5）数据库层

① 合理使用数据库引擎 ② 合理设置事务隔离级别，合理使用事务 ③ 正确使用 SQL 语句和查询索引 ④ 合理分库分表 ⑤ 使用数据库中间件实现数据库读写分离 ⑥ 设置数据库主从读写分离