### 1、MySQL查询字段区不区分大小写

不区分。

如何解决需要区分大小写的场景？

例如登录用户为admin，此时填写ADMIN也能登录，如果用户名需要区分大小写，你的做法是什么？

#### 方案一

MySQL默认的字符检索策略：utf\_general\_ci，表示不区分大小写。可以使用utf\_general\_cs，表示区分大小写。也可以使用utf\_bin，表示二进制比较，同样也区分大小写。

注意：在Mysql5.6.10版本中，不支持utf8\_genral\_cs。

创建表时，直接设置表的collate属性为utf8\_general\_cs或者utf8\_bin；如果已经创建表，则直接修改字段的Collation属性为utf8\_general\_cs或者utf8\_bin。

-- 创建表：

CREATE TABLE testt ( id INT PRIMARY KEY, NAME VARCHAR ( 32 ) NOT NULL ) ENGINE = INNODB COLLATE = utf8\_bin;

-- 修改表结构的Collation属性

ALTER TABLE TABLENAME MODIFY COLUMN COLUMNNAME VARCHAR ( 50 ) BINARY CHARACTER

SET utf8 COLLATE utf8\_bin DEFAULT NULL;

#### 方案二

直接修改sql语句，在要查询的字段前面加上binary关键字。

-- 在每一个条件前加上binary关键字

SELECT

\*

FROM

USER

WHERE

BINARY username = 'admin'

AND BINARY PASSWORD = 'admin';

-- 将参数以binary('')包围

SELECT

\*

FROM

USER

WHERE

username LIKE BINARY ( 'admin' )

AND PASSWORD LIKE BINARY ( 'admin' );

### 2、MySQL innodb的事务与日志的实现方式

#### 日志种类

* 错误日志：
* 查询日志：
* 慢查询日志：
* 二进制日志：
* 中继日志：
* 事务日志：

#### 事务的4种隔离级别

* 读未提交(RU)
* 读已提交(RC)
* 可重复读(RR)
* 串行

事务是如何通过日志来实现的？

事务日志是通过redo和innodb的存储引擎日志缓冲（Innodb log buffer）来实现的，当开始一个事务的时候，会记录该事务的lsn(log sequence number)号；

当事务执行时，会往InnoDB存储引擎的日志的日志缓存里面插入事务日志；

当事务提交时，必须将存储引擎的日志缓冲写入磁盘（通过innodb\_flush\_log\_at\_trx\_commit来控制），也就是写数据前，需要先写日志。这种方式称为“预写日志方式”

### 3、MySQL binlog的几种日志录入格式以及区别

Statement：每一条会修改数据的sql都会记录在binlog中。

优点：不需要记录每一行的变化，减少了binlog日志量，节约了IO，提高性能。

相比row能节约多少性能 与日志量，这个取决于应用的SQL情况，正常同一条记录修改或者插入row格式所产生的日志量还小于Statement产生的日志量，但是考虑到如果带条件的update操作，以及整表删除，alter表等操作，ROW格式会产生大量日志，因此在考虑是否使用ROW格式日志时应该根据应用的实际情况，其所 产生的日志量会增加多少，以及带来的IO性能问题。

缺点：由于记录的只是执行语句，为了这些语句能在slave上正确运行，因此还必须记录每条语句在执行的时候的一些相关信息，以保证所有语句能在slave得到和在master端执行时候相同的结果。

另外mysql 的复制,像一些特定函数功能，slave可与master上要保持一致会有很多相关问题(如sleep()函数， last\_insert\_id()，以及user-defined functions(udf)会出现问题).

使用以下函数的语句也无法被复制：

LOAD\_FILE()

UUID()

USER()

FOUND\_ROWS()

SYSDATE() (除非启动时启用了 --sysdate-is-now 选项)

同时在INSERT …SELECT 会产生比 RBR 更多的行级锁

Row:不记录sql语句上下文相关信息，仅保存哪条记录被修改。

优点：binlog中可以不记录执行的sql语句的上下文相关的信息，仅需要记录那一条记录被修改成什么了。

所以rowlevel的日志内容会非常清楚的记录下 每一行数据修改的细节。而且不会出现某些特定情况下的存储过程，或function，以及trigger的调用和触发无法被正确复制的问题

缺点：所有的执行的语句当记录到日志中的时候，都将以每行记录的修改来记录，这样可能会产生大量的日志内容。

比如一条update语句，修改多条记录，则binlog中每一条修改都会有记录，这样造成binlog日志量会很大，特别是当执行alter table之类的语句的时候，由于表结构修改，每条记录都发生改变，那么该表每一条记录都会记录到日志中。

Mixedlevel: 以上两种level的混合使用。

一般的语句修改使用statment格式保存binlog，如一些函数，statement无法完成主从复制的操作，则采用row格式保存binlog，MySQL会根据执行的每一条具体的sql语句来区分对待记录的日志形式，也就是在Statement和Row之间选择一种。

新版本的MySQL中对row level模式也被做了优化，并不是所有的修改都会以row level来记录，像遇到表结构变更的时候就会以statement模式来记录。至于update或者delete等修改数据的语句，还是会记录所有行的变更。