1. MySQL InnoDB、Mysaim的特点?

a. InnoDB：

1. 支持事务处理
2. 支持外键
3. 支持行锁
4. 不支持FULLTEXT类型的索引（在Mysql5.6已引入）
5. 不保存表的具体行数，扫描表来计算有多少行
6. 对于AUTO_INCREMENT类型的字段，必须包含只有该字段的索引
7. DELETE 表时，是一行一行的删除
8. InnoDB 把数据和索引存放在表空间里面
9. 跨平台可直接拷贝使用
10. 表格很难被压缩

b. MyISAM：

1. 不支持事务，回滚将造成不完全回滚，不具有原子性
2. 不支持外键
3. 支持全文搜索
4. 保存表的具体行数,不带where时，直接返回保存的行数
5. DELETE 表时，先drop表，然后重建表
6. MyISAM 表被存放在三个文件 。frm 文件存放表格定义。 数据文件是MYD (MYData) 。 索引文件是MYI (MYIndex)引伸
7. 跨平台很难直接拷贝
8. AUTO_INCREMENT类型字段可以和其他字段一起建立联合索引
9. 表格可以被压缩

c. 选择：因为MyISAM相对简单所以在效率上要优于InnoDB.如果系统读多，写少。对原子性要求低。那么MyISAM最好的选择。且MyISAM恢复速度快。可直接用备份覆盖恢复。如果系统读少，写多的时候，尤其是并发写入高的时候。InnoDB就是首选了。两种类型都有自己优缺点，选择那个完全要看自己的实际类弄。

2. 行锁，表锁；乐观锁，悲观锁?

a. 行锁：数据库表中某一行被锁住。

b. 表锁：整个数据库表被锁住。

c. 乐观锁：顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，具体实现是给表增加一个版本号的字段，在执行update操作时比较该版本号是否与当前数据库中版本号一致，如一致，更新数据，反之拒绝。

d. 悲观锁：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。读数据的时候会上锁，直到update完成才释放锁，使用悲观锁要注意不要锁住整个表。

3. 数据库隔离级别是什么？有什么作用?

1. ISOLATION*READ*UNCOMMITTED 这是事务最低的隔离级别，它允许另外一个事务可以看到这个事务未提交的数据。这种隔离级别会产生脏读，不可重复读和幻读。

2. ISOLATION*READ*COMMITTED 保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据。这种事务隔离级别可以避免脏读出现，但是可能会出现不可重复读和幻读。

3. ISOLATION*REPEATABLE*READ 这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻读。它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免不可重复读。

4. ISOLATION_SERIALIZABLE 这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。除了防止脏读，不可重复读外，还避免了幻读。

隔离级别	脏读（Dirty Read）	不可重复读（NonRepeatable Read）	幻读（Phantom Read）
未提交读（Read uncommitted）	可能	可能	可能
已提交读（Read committed）	不可能	可能	可能
可重复读（Repeatable read）	不可能	不可能	可能
可串行化（Serializable ）	不可能	不可能	不可能

- 未提交读(Read Uncommitted)：允许脏读，也就是可能读取到其他会话中未提交事务修改的数据
- 提交读(Read Committed)：只能读取到已经提交的数据。Oracle等多数数据库默认都是该级别 (不重复读)
- 可重复读(Repeated Read)：可重复读。在同一个事务内的查询都是事务开始时刻一致的，InnoDB默认级别。在SQL标准中，该隔离级别消除了不可重复读，但是还存在幻象读
- 串行读(Serializable)：完全串行化的读，每次读都需要获得表级共享锁，读写相互都会阻塞

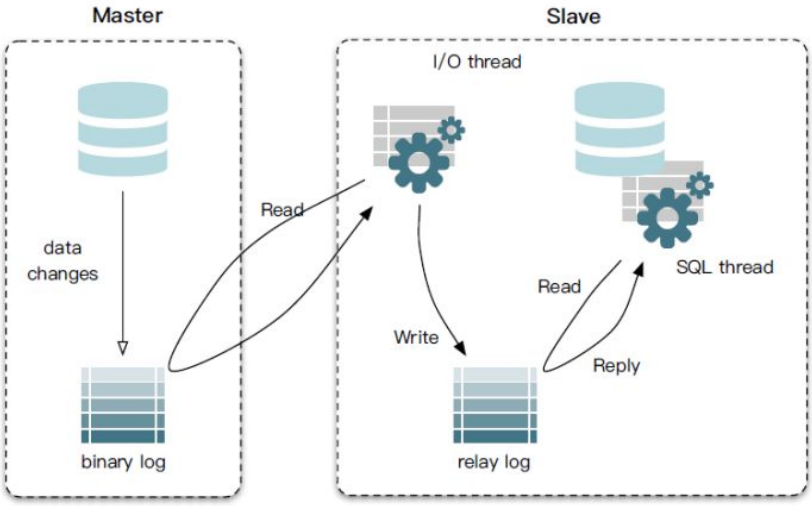
4. MySQL主备同步的基本原理。

mysql主备复制实现分成三个步骤：

1、master将改变记录到二进制日志(binary log)中（这些记录叫做二进制日志事件，binary log events，可以通过show binlog events进行查看）；

2、slave将master的binary log events拷贝到它的中继日志(relay log)；

3、slave重做中继日志中的事件，将改变反映它自己的数据。



5. select * from table t where size > 10 group by size order by size的sql语句执行顺序?

sql语句执行顺序如下：

where -> group by -> having -> select -> orderby

6. 如何优化数据库性能（索引、分库分表、批置操作、分页算法、升级硬盘SSD、业务优化、主从部署）

1、选择合适的数据库引擎，合理使用索引

2、分页获取数据，只获取需要的字段

3、优化业务逻辑，减少数据库IO

4、分库分表

5、部署主从数据库

6、升级硬件

7. SQL什么情况下不会使用索引（不包含，不等于，函数）

a. select * 可能导致不走索引；

b. 空值会导致不走索引，因为hashset不能存空值；

c. 索引列有函数运算，不走索引，可以在索引列建立一个函数的索引。

d. 隐式转换可能导致不走索引；

e. 表的数据库小或者需要选择大部分数据，不走索引；

f. !=或者<>可能导致不走索引；

g. 字符型的索引列会导致优化器认为需要扫描索引大部分数据且聚簇因子很大，最终导致弃用索引扫描而改用全表扫描方式

h. like '%liu' 百分号在前不走索引；

i. not in, not exist不走索引；

8. 一般在什么字段上建索引（过滤数据最多的字段）

1、表的主键、外键必须有索引；

2、数据量超过300的表应该有索引；

3、经常与其他表进行连接的表，在连接字段上应该建立索引；

4、经常出现在Where子句中的字段，特别是大表的字段，应该建立索引；

5、索引应该建在选择性高的字段上；

6、索引应该建在小字段上，对于大的文本字段甚至超长字段，不要建索引；

9. 如何从一张表中查出name字段不包含"XYZ"的所有行?

```
select * from table where name not like '%XYZ%';
```

10. HRedis, RDB和A0Ff如何做高可用、集群

11. 如何解决高并发减库存问题

消息队列，异步处理，减库存加锁

12. mysql存储引擎中索引的实现机制；

https://blog.csdn.net/debug_zhang/article/details/52168552

13. 数据库事务的几种粒度；

a. 表锁定：对整个表的锁定。

b. 行锁定：只锁定进行更改的行，例如：insert，update，delete，都隐式采用行锁定。

c. 数据库锁机制可分为多种粒度的： 数据库，表，页面，行

d. 粒度越大，DBMS管理越容易，但是实现并发处理的能力就越差，表，页面，行

14. mysql调优：

a. explain select语句；

b. 当只要一条数据时使用limit 1；

c. 为搜索字段建索引；

d. 避免select *；

e. 字段尽量使用not null；

f. 垂直分割；

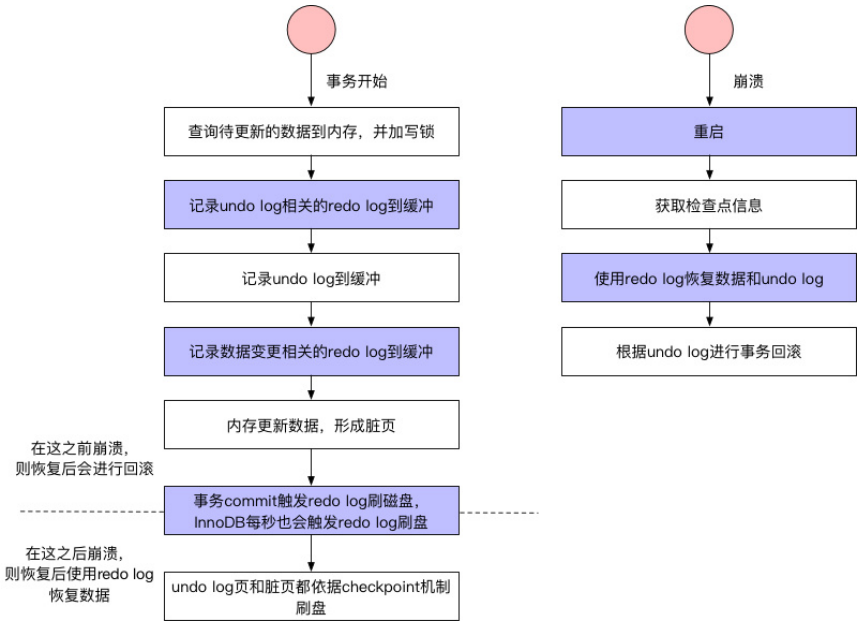
g. 拆分大的delete和insert语句：delete和insert会锁表；

h. 分表分库分区。

15. 说说事务的四种特性（ACID）？

- **原子性（Atomicity）**
  原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
- **一致性（Consistency）**
  事务前后数据的完整性必须保持一致。（比如转账）
- **隔离性（Isolation）**
  事务的隔离性是指多个用户并发访问数据库时，一个用户的事务不能被其它用户的事务所干扰，多个并发事务之间数据要相互隔离。
- **持久性（Durability）**
  持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响。

16. innodb如何实现mysql的事务?



事务进行过程中，每次sql语句执行，都会记录undo log和redo log，然后更新数据形成脏页，然后redo log按照时间或者空间等条件进行落盘，undo log和脏页按照checkpoint进行落盘，落盘后相应的redo log就可以删除了。此时，事务还未COMMIT，如果发生崩溃，则首先检查checkpoint记录，使用相应的redo log进行数据和undo log的恢复，然后查看undo log的状态发现事务尚未提交，然后就使用undo log进行事务回滚。事务执行COMMIT操作时，会将本事务相关的所有redo log都进行落盘，只有所有redo log落盘成功，才算COMMIT成功。然后内存中的数据脏页继续按照checkpoint进行落盘。如果此时发生了崩溃，则只使用redo log恢复数据。

17. 让你设计一个索引，你会怎么设计?

mysql默认存储引擎innodb只显式支持B树索引，对于频繁访问的表，innodb会透明建立自适应hash索引，即在B树索引基础上建立hash索引，可以显著提高查找效率，对于客户端是透明的，不可控制的，隐式的。

18.