1.数据库三范式是什么?

1. 第一范式（1NF）：字段具有原子性,不可再分。(所有关系型数据库系统都满足第一范式数据库表中的字段都是单一属性的，不可再分)
2. 第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。要求数据库表中的每个实例或行必须可以被惟一地区分。通常需要为表加上一个列，以存储各个实例的惟一标识。这个惟一属性列被称为主关键字或主键。
3. 满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。 >所以第三范式具有如下特征： >>1. 每一列只有一个值 >>2. 每一行都能区分。 >>3. 每一个表都不包含其他表已经包含的非主关键字信息。

2.有哪些数据库优化方面的经验?

1. 用 PreparedStatement， 一般来说比 Statement 性能高：一个 sql 发给服务器去执行，涉及步骤：语法检查、语义分析， 编译，缓存。
2. 有外键约束会影响插入和删除性能，如果程序能够保证数据的完整性，那在设计数据库时就去掉外键。
3. 表中允许适当冗余，譬如，主题帖的回复数量和最后回复时间等
4. UNION ALL 要比 UNION 快很多，所以，如果可以确认合并的两个结果集中不包含重复数据且不需要排序时的话，那么就使用 UNION ALL。 >>UNION 和 UNION ALL 关键字都是将两个结果集合并为一个，但这两者从使用和效率上来说都有所不同。 >1. 对重复结果的处理：UNION 在进行表链接后会筛选掉重复的记录，Union All 不会去除重复记录。 >2. 对排序的处理：Union 将会按照字段的顺序进行排序；UNION ALL 只是简单的将两个结果合并后就返回。

3.请简述常用的索引有哪些种类?

1. 普通索引：即针对数据库表创建索引
2. 唯一索引：与普通索引类似，不同的就是：MySQL 数据库索引列的值必须唯一，但允许有空值
3. 主键索引：它是一种特殊的唯一索引，不允许有空值。一般是在建表的时候同时创建主键索引
4. 组合索引：为了进一步榨取 MySQL 的效率，就要考虑建立组合索引。即将数据库表中的多个字段联合起来作为一个组合索引。

4.以及在 mysql 数据库中索引的工作机制是什么？

数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用 B 树及其变种 B+树

5.MySQL 的基础操作命令:

1. MySQL 是否处于运行状态:Debian 上运行命令 service mysql status，在 RedHat 上运行命令 service mysqld status
2. 开启或停止 MySQL 服务 :运行命令 service mysqld start 开启服务；运行命令 service mysqld stop 停止服务
3. Shell 登入 MySQL: 运行命令 mysql -u root -p
4. 列出所有数据库:运行命令 show databases;
5. 切换到某个数据库并在上面工作:运行命令 use database*name; 进入名为 database*name 的数据库
6. 列出某个数据库内所有表: show tables;
7. 获取表内所有 Field 对象的名称和类型 :describe table_name;

6.mysql 的复制原理以及流程。

Mysql 内建的复制功能是构建大型，高性能应用程序的基础。将 Mysql 的数据分布到多个系统上去，这种分布的机制，是通过将 Mysql 的某一台主机的数据复制到其它主机（slaves）上，并重新执行一遍来实现的。 * 复制过程中一个服务器充当主服务器，而一个或多个其它服务器充当从服务器。主服务器将更新写入二进制日志文件，并维护文件的一个索引以跟踪日志循环。这些日志可以记录发送到从服务器的更新。 当一个从服务器连接主服务器时，它通知主服务器在日志中读取的最后一次成功更新的位置。从服务器接收从那时起发生的任何更新，然后封锁并等待主服务器通知新的更新。 过程如下 1. 主服务器把更新记录到二进制日志文件中。 2. 从服务器把主服务器的二进制日志拷贝到自己的中继日志（replay log）中。 3. 从服务器重做中继日志中的时间，把更新应用到自己的数据库上。

7.mysql 支持的复制类型?

1. 基于语句的复制： 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。MySQL 默认采用基于语句的复制，效率比较高。 一旦发现没法精确复制时，会自动选着基于行的复制。
2. 基于行的复制：把改变的内容复制过去，而不是把命令在从服务器上执行一遍. 从 mysql5.0 开始支持
3. 混合类型的复制: 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。

8.mysql 中 myisam 与 innodb 的区别?

1. 事务支持 > *MyISAM：强调的是性能，每次查询具有原子性,其执行数度比 InnoDB 类型更快，但是不提供事务支持。* > InnoDB：提供事务支持事务，外部键等高级数据库功能。 具有事务(commit)、回滚(rollback)和崩溃修复能力(crash recovery capabilities)的事务安全(transaction-safe (ACID compliant))型表。
2. InnoDB 支持行级锁，而 MyISAM 支持表级锁. >> 用户在操作 myisam 表时，select，update，delete，insert 语句都会给表自动加锁，如果加锁以后的表满足 insert 并发的情况下，可以在表的尾部插入新的数据。

3. InnoDB 支持 MVCC, 而 MyISAM 不支持
4. InnoDB 支持外键，而 MyISAM 不支持
5. 表主键 > *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者非空唯一索引，就会自动生成一个 6 字节的主键(用户不可见)，数据是主索引的一部分，附加索引保存的是主索引的值。
6. InnoDB 不支持全文索引，而 MyISAM 支持。
7. 可移植性、备份及恢复 > *MyISAM：数据是以文件的形式存储，所以在跨平台的数据转移中会很方便。在备份和恢复时可单独针对某个表进行操作。* > InnoDB：免费的方案可以是拷贝数据文件、备份 binlog，或者用 mysqldump，在数据量达到几十 G 的时候就相对痛苦了
8. 存储结构 > *MyISAM：每个 MyISAM 在磁盘上存储成三个文件。第一个文件的名字以表的名字开始，扩展名指出文件类型。.frm 文件存储表定义。数据文件的扩展名为.MYD (MYData)。索引文件的扩展名是.MYI (MYIndex)。* > InnoDB：所有的表都保存在同一个数据文件中（也可能是多个文件，或者是独立的表空间文件），InnoDB 表的大小只受限于操作系统文件的大小，一般为 2GB。

9.mysql 中 varchar 与 char 的区别以及 varchar(50)中的 50 代表的涵义？

1. varchar 与 char 的区别: char 是一种固定长度的类型，varchar 则是一种可变长度的类型.
2. varchar(50)中 50 的涵义 ： 最多存放 50 个字节
3. int（20）中 20 的涵义: int(M)中的 M indicates the maximum display width (最大显示宽度)for integer types. The maximum legal display width is 255.

10.MySQL 中 InnoDB 支持的四种事务隔离级别名称，以及逐级之间的区别？

1. Read Uncommitted（读取未提交内容） >> 在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。读取未提交的数据，也被称之为脏读（Dirty Read）。
2. Read Committed（读取提交内容） >> 这是大多数数据库系统的默认隔离级别（但不是 MySQL 默认的）。它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变。这种隔离级别也支持所谓的不可重复读（Nonrepeatable Read），因为同一事务的其他实例在该实例处理其间可能会有新的 commit，所以同一 select 可能返回不同结果。
3. Repeatable Read（可重读） >> 这是 MySQL 的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读（Phantom

Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的"幻影" 行。InnoDB 和 Falcon 存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control 间隙锁）机制解决了该问题。注：其实多版本只是解决不可重复读问题，而加上间隙锁（也就是它这里所谓的并发控制）才解决了幻读问题。

4. Serializable（可串行化） >> 这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。

11.表中有大字段 X（例如：text 类型），且字段 X 不会经常更新，以读为为主，将该字段拆成子表好处是什么？

如果字段里面有大字段（text,blob)类型的，而且这些字段的访问并不多，这时候放在一起就变成缺点了。 MYSQL 数据库的记录存储是按行存储的，数据块大小又是固定的（16K），每条记录越小，相同的块存储的记录就越多。此时应该把大字段拆走，这样应付大部分小字段的查询时，就能提高效率。当需要查询大字段时，此时的关联查询是不可避免的，但也是值得的。拆分开后，对字段的 UPDAE 就要 UPDATE 多个表了

12.MySQL 中 InnoDB 引擎的行锁是通过加在什么上完成（或称实现）的？

InnoDB 行锁是通过给索引上的索引项加锁来实现的，这一点 MySQL 与 Oracle 不同，后者是通过在数据块中对相应数据行加锁来实现的。InnoDB 这种行锁实现特点意味着：只有通过索引条件检索数据，InnoDB 才使用行级锁，否则，InnoDB 将使用表锁！

13.MySQL 中控制内存分配的全局参数，有哪些？

1. Key*buffer*size： > * key*buffer*size 指定索引缓冲区的大小，它决定索引处理的速度，尤其是索引读的速度。通过检查状态值 Key*read*requests 和 Key*reads*，*可以知道 key*buffer*size 设置是否合理。比例 key*reads /key*read*requests 应该尽可能的低，至少是 1:100，1:1000 更好（上述状态值可以使用 SHOW STATUS LIKE 'key*read%'获得*）。 > * *key*buffer*size 只对 MyISAM 表起作用。即使你不使用 MyISAM 表，但是内部的临时磁盘表是 MyISAM 表，也要使用该值。可以使用检查状态值 created*tmp*disk*tables 得知详情。对于 1G 内存的机器，如果不使用 MyISAM 表，推荐值是 16M（8-64M） > * key*buffer*size 设置注意事项 >>>1. 单个 key*buffer 的大小不能超过 4G，如果设置超过 4G，就有可能遇到下面 3 个 bug: >>>>> http://bugs.mysql.com/bug.php?id=29446
 >>>>> http://bugs.mysql.com/bug.php?id=29419
 >>>>> http://bugs.mysql.com/bug.php?id=5731
 >>>2. 建议 key*buffer 设置为物理内存的 1/4(针对 MyISAM 引擎)，甚至是物理内存的 30%~40%，如果 key*buffer*size 设置太大，

系统就会频繁的换页，降低系统性能。因为 MySQL 使用操作系统的缓存来缓存数据，所以我们得为系统留够足够的内存；在很多情况下数据要比索引大得多。 >>>3. 如果机器性能优越，可以设置多个 key*buffer,分别让不同的 key*buffer 来缓存专门的索引

2. innodb*buffer*pool_size > *表示缓冲池字节大小，InnoDB 缓存表和索引数据的内存区域。mysql 默认的值是 128M。最大值与你的 CPU 体系结构有关，在 32 位操作系统，最大值是 4294967295 (2^32-1) ，在 64 位操作系统，最大值为 18446744073709551615 (2^64-1)。* > 在 32 位操作系统中，CPU 和操作系统实用的最大大小低于设置的最大值。如果设定的缓冲池的大小大于 1G，设置 innodb*buffer*pool*instances 的值大于 1.* > *数据读写在内存中非常快, innodb*buffer*pool*size 减少了对磁盘的读写。 当数据提交或满足检查点条件后才一次性将内存数据刷新到磁盘中。然而内存还有操作系统或数据库其他进程使用，一般设置 buffer pool 大小为总内存的 3/4 至 4/5。 若设置不当，内存使用可能浪费或者使用过多。 对于繁忙的服务器，buffer pool 将划分为多个实例以提高系统并发性，减少线程间读写缓存的争用。buffer pool 的大小首先受 innodb*buffer*pool_instances 影响，当然影响较小。

3. query*cache*size > *当 mysql 接收到一条 select 类型的 query 时，mysql 会对这条 query 进行 hash 计算而得到一个 hash 值，然后通过该 hash 值到 query cache 中去匹配，如果没有匹配中，则将这个 hash 值存放在一个 hash 链表中，同时将 query 的结果集存放进 cache 中，存放 hash 值的链表的每一个 hash 节点存放了相应 query 结果集在 cache 中的地址，以及该 query 所涉及到的一些 table 的相关信息；如果通过 hash 值匹配到了一样的 query，则直接将 cache 中相应的 query 结果集返回给客户端。如果 mysql 任何一个表中的任何一条数据发生了变化，便会通知 query cache 需要与该 table 相关的 query 的 cache 全部失效，并释放占用的内存地址。* > query cache 优缺点 >> 1. query 语句的 hash 计算和 hash 查找带来的资源消耗。mysql 会对每条接收到的 select 类型的 query 进行 hash 计算然后查找该 query 的 cache 是否存在，虽然 hash 计算和查找的效率已经足够高了，一条 query 所带来的消耗可以忽略，但一旦涉及到高并发，有成千上万条 query 时，hash 计算和查找所带来的开销就的重视了； >> 2. query cache 的失效问题。如果表变更比较频繁，则会造成 query cache 的失效率非常高。表变更不仅仅指表中的数据发生变化，还包括结构或者索引的任何变化； >> 3. 对于不同 sql 但同一结果集的 query 都会被缓存，这样便会造成内存资源的过渡消耗。sql 的字符大小写、空格或者注释的不同，缓存都是认为是不同的 sql（因为他们的 hash 值会不同）； >> 4. 相关参数设置不合理会造成大量内存碎片，相关的参数设置会稍后介绍。

4. read*buffer*size >是 MySQL 读入缓冲区大小。对表进行顺序扫描的请求将分配一个读入缓冲区，MySQL 会为它分配一段内存缓冲区。read*buffer*size 变量控制这一缓冲区的大小。如果对表的顺序扫描请求

非常频繁，并且你认为频繁扫描进行得太慢，可以通过增加该变量值以及内存缓冲区大小提高其性能。

14.若一张表中只有一个字段 VARCHAR(N)类型，utf8 编码，则 N 最大值为多少(精确到数量级即可)?

由于 utf8 的每个字符最多占用 3 个字节。而 MySQL 定义行的长度不能超过 65535，因此 N 的最大值计算方法为：(65535-1-2)/3。减去 1 的原因是实际存储从第二个字节开始，减去 2 的原因是因为要在列表长度存储实际的字符长度，除以 3 是因为 utf8 限制：每个字符最多占用 3 个字节。

15. [SELECT *] 和[SELECT 全部字段]的 2 种写法有何优缺点?

1. 前者要解析数据字典，后者不需要
2. 结果输出顺序，前者与建表列顺序相同，后者按指定字段顺序。
3. 表字段改名，前者不需要修改，后者需要改
4. 后者可以建立索引进行优化，前者无法优化
5. 后者的可读性比前者要高

16.HAVNG 子句 和 WHERE 的异同点?

1. 语法上：where 用表中列名，having 用 select 结果别名
2. 影响结果范围：where 从表读出数据的行数，having 返回客户端的行数
3. 索引：where 可以使用索引，having 不能使用索引，只能在临时结果集操作
4. where 后面不能使用聚集函数，having 是专门使用聚集函数的。

17.MySQL 当记录不存在时 insert,当记录存在时 update，语句怎么写？

INSERT INTO table (a,b,c) VALUES (1,2,3) ON DUPLICATE KEY UPDATE c=c+1;

18.MySQL 的 insert 和 update 的 select 语句语法

`SQL insert into student (stuid,stuname,deptid) select 10,'xzm',3 from student where stuid > 8;

update student a inner join student b on b.stuID=10 set a.stuname=concat(b.stuname, b.stuID) where a.stuID=10 ; `