前言

> 前言面试是你进入公司的第一个关卡，如同高考一般，但是面试不是高考，高考只有一次，而面试则可以有N次，只要有合适的面试邀请，你就可以继续尝试，不要害怕失败；假如公司没有选择你，不是你不优秀，只是你不符合他们的要求罢了，**结束后总结下面试中不足的地方，及时调整好心态，准备下个面试才是你最正确的做法。**

面试地图

> 俗话说：“将军不打没准备的仗”，参加高考也是需要寒窗苦读12年甚至更久，而求职面试呢，也是需要提前准备的，主要分如下几个方面来准备：

1. 准备好你的个人简历
2. 自我介绍
3. PHP技术面试要点：
    - 提前准备好怎么在面试官面前表述自己的项目（体现用到的技术与亮点/难点）
    - 简历上自己的列出的专业技能，要看一下相关知识点，因为面试官会从你的这些技能中问你问题
    - MySQL数据库方面，比如事务处理、索引、锁、查询优化、表结构设计等
    - PHP进阶方面，例如MVC、OOP、算法、设计模式、redis等
    - Javascript、Ajax、jQuery等前端知识，会Vue或者Node.js一般会有加分呢
    - 丰富GitHub项目、博客以及自己活跃的社区主页信息等
4. 人事面试要点：
    - 你之前所在公司的离职原因
    - 谈谈你对加班的看法
    - 你为什么会选择来我们公司
    - 你未来的职业规划以及发展方向
    - 对自己的一个评价
    - 你的业余爱好，最近在看什么书
    - 对你的经历提出一些问题
    - 你还有什么问题要问我
    - 你在之前，有去过那些公司面试？接下来还约了其他的面试吗
    - 你的期望薪资以及要求
5. 面试当天

分项准备

个人简历

> 简历是递给HR的名片，好的可以收获面试通知 简历也是面试时，面试官发问的一个来源，一定要对简历上所写的认真思考，可能会涉及的问题一定要多准备。

作为程序猿如何打造一份HR眼中漂亮的简历？

- 简历的脉络要清晰，重点要突出。

> 简历分为几部分，前后次序的安排

- 简历篇幅不要太长，尽量精简，突出重点。一般2-3页为佳
- 重要部分要放在第一页，Hr筛选的时候第一时间就会翻到这些信息
- 简历可以用Markdown书写，转换为PDF或Word文件打印

个人信息

把必须的重要信息写出来就可以了，如电话、邮箱、博客地址、Github、期望薪资。其他比如出生年月、户籍可有可无的信息可以不填。

个人经历

按照时间的倒叙理出自己职业生涯所经历的几家公司，时间-公司-职责一定要清楚。最好放在第一页

个人技能

这部分列出你所掌握的技术栈和一些解决方案。

项目经验

列出几个能代表自己能力的项目，3-4个就可以，小项目不用写。多了Hr一般不会看，技术面试时面试官一般会挑1-2个项目让你讲解以体现你的能力。

项目经验中有几点需要说明：

1. 项目名-开始参与和结束时间
2. 项目简介
3. 自己负责的功能模块
4. 项目用到了哪些技术栈和解决方案

教育经历

学校-专业-学历-入学时间和毕业时间。

2.自我介绍

自我介绍应该围绕简历相关信息，流畅自然,条例清晰。 需要补充的是，提前了解该公司以及该岗位相关信息，了解这个行业，公司，以及岗位。

> 示例： 面试官您好！我叫郑华毕业于中国地质大学（武汉），专业地球信息科学与技术，是地质学与信息科学相结合的一门学科，毕业后从事过几年相关工作，后转学PHP，在重庆候维网络科技有限公司工作，主要参与的项目有威付宝聚合支付系统的开发维护，全业盟分润系统的开发，期间还完成一套数码商城系统。

可参看文末面试时如何优雅地自我介绍？

3.PHP技术面试要点

PHP面试总结

我的复习

linux

crontab 定时任务

> 通过crontab 命令，我们可以在固定的间隔时间执行指定的系统指令或 shell script脚本。时间间隔的单位可以是分钟、小时、日、月、周及以上的任意组合。这个命令非常适合周期性的日志分析或数据备份等工作。

```
```
* * * * * echo "hello" #每1分钟执行hello
3,15 * * * * myCommand #每小时第三分钟和第十五分钟执行
3,15 8-11 * * * myCommand# 在上午8点到11点的第3和第15分钟执行
3,15 8-11 */2 * * myCommand #每隔两天的上午8点到11点的第3和第15分钟执行
30 21 * * * /etc/init.d/smb restart #每晚的21:30重启smb
0 23 * * 6 /etc/init.d/smb restart #每星期六的晚上11 : 00 pm重启smb
```
```

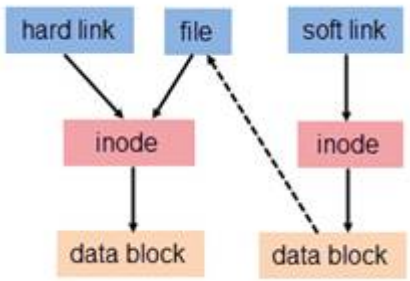
软连接&硬连接

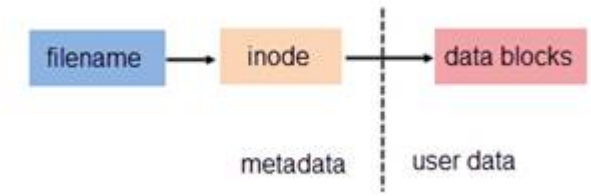
> 我们文件数据存储在硬盘上，硬盘最小的单位是扇区，一个扇区是512个字节。操作系统读取硬盘每次读取4Kb。成为一个块。文件都存在块中。同时操作系统还会有一个区域记录文件的基本信息。这个基本信息成为文件的元信息。

软连接:直接可以理解为windows下的快捷方式，通过一个地址指向一个文件，如果该文件的名字被修改或者该文件被删除，那么快捷方式也就失效，即该快捷方式也就找不到对应的文件，软连接的限制很少。**软连接本身是一个文件，内容是连接目标地址。**

硬连接:linux操作系统为每个文件分配一个编号叫节点索引(inode Index)，建立硬连接实际上是**对于同一个文件起了多个"名字/路径"**，即硬连接的文件被删除或者名字被修改后其连接的文件依然可以访问这个文件。

> 使用差异：硬连接只能连接文件，不能跨分区（？）





shell

待编辑

LNMP

待编辑

Awk

待编辑

Sed

待编辑

同步异步与阻塞非阻塞

前者在乎返回结果的时机、后者在乎过程中能否离开做其他事 会响的开水壶之比喻（水开后会自动关闭） 阻塞和非阻塞，同步和异步

Mysql

语句分类

DDL 数据定义语言;DQL 数据查询语言;DML 数据操纵语句;DCL 数据控制语句。

MySQL常用的三种引擎

1.MyISAM存储引擎

特点：不支持事务、也不支持外键、只支持表级锁，优势是访问速度快。

适用：对事务完整性没有要求或以select，insert为主的应用基本上可以用这个引擎来创建表

支持3种不同的存储格式，分别是：静态表；动态表；压缩表

MyISAM的表存储成3个文件。文件的名字与表名相同。拓展名为frm、MYD、MYI。备份，恢复相对Innodb方便。

其实，frm文件存储表的结构；MYD文件存储数据，是MYData的缩写；MYI文件存储索引，是MYIndex的缩写。

myisam表锁

mysql的表锁有两种模式：表共享读锁和表独占写锁。

- myISAM表的读操作，不会阻塞其他用户对同一个表的读请求，但会阻塞对同一个表的写请求。
- myISAM表的写操作，会阻塞其他用户对同一个表的读和写操作。
- myISAM表的读、写操作之间、以及写操作之间是串行的。
- MyISAM存储引擎的读锁和写锁是互斥的，读写操作室串行的，那么如果读写两个进程同时请求同一张表，Mysql将会使**写进程先获得锁**。不仅仅如此，即使读请求先到达锁等待队列，写锁后到达，写锁也会先执行。因为mysql因为写请求比读请求更加重要。**这也正是MyISAM不适合含有大量更新操作和查询操作应用的原因。**
- 使用locak tables 给表加锁时候，必须同时给所有涉及到的表加锁，因为加锁之后，当前会话，就不能操作没有加锁的表。

```
lock table t1 read, t2 read;
select count(t1.id1) as 'sum' from t1;
select count(t2.id1) as 'sum' from t2;
unlock tables;
```

  - 查询表级锁争用情况: `show status like 'table%';`

table_locks_waited锁定等待时间越长，则说明存在较严重的表级别锁争用情况。

2.InnoDB存储引擎

特点：支持外键约束、支持事务、支持行级锁，缺点是对比MyISAM引擎，写的处理效率会差一些，并且会占用更多的磁盘空间以保留数据和索引。

适用：并发读写操作较多的项目，对事物完整性有要求的项目，行锁大幅度提高了多用户并发操作的性能。

InnoDB中表的表结构存储在.frm文件中（我觉得是frame的缩写吧）。数据和索引存储在innodb_data_home_dir和innodb_data_file_path定义的表空间中。

innodb锁模式

- 查看InnoDB行锁争用情况:`show status like 'innodb_row_lock%';`
- Mysql行锁是针对索引加的锁，不是针对记录加的锁。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，在实际开发中应当注意。
- innoDB实现了以先两种类型的行锁：

  共享锁（S）：允许一个事务去读一行，阻止其他事务获取相同数据集的排他锁。

  排他锁（X）：允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的共享读锁和排他写锁。

  先两种意向表锁：

  意向同享锁

  意向排他锁

  如果一个事务请求的锁模式与当前的锁模式兼容，innodb就将请求的锁授予该事务；反之，如果两者不兼容，该事务就要等待锁释放。**意向锁是Innodb自动加的**，不需要用户干预。对于UPDATE、DELETE、INSERT语句，Innodb会自动给涉及的数据集加排他锁（X）；对于普通SELECT语句，Innodb不会加任何锁。

  一些细节上的差别：

  InnoDB中不保存表的 具体行数，也就是说，执行select count() from table时（无where条件），InnoDB要扫描一遍整个表来计算有多少行，但是MyISAM只要简单的读出保存好的行数即可。

  对于AUTO_INCREMENT类型的字段，InnoDB中必须包含只有该字段的索引，但是在MyISAM表中，可以和其他字段一起建立联合索引。

  DELETE FROM table时，InnoDB不会重新建立表，而是一行一行的删除。

  LOAD TABLE FROM MASTER操作对InnoDB是不起作用的，解决方法是首先把InnoDB表改成MyISAM表，导入数据后再改成InnoDB表，但是对于使用的额外的InnoDB特性（例如外键）的表不适用。

  另外，InnoDB表的行锁也不是绝对的，如果在执行一个SQL语句时MySQL不能确定要扫描的范围，InnoDB表同样会锁全表。

3.MEMORY存储引擎

Memory存储引擎使用存在于内存中的内容来创建表。每个memory表只实际对应一个磁盘文件，格式是.frm。

memory类型的表访问非常的快，因为它的数据是放在内存中的，并且默认使用HASH索引，但是一旦服务关闭，表中的数据就会丢失掉。

MEMORY存储引擎的表可以选择使用BTREE索引或者HASH索引，两种不同类型的索引有其不同的使用范围。

应用：Memory类型的存储引擎主要用于哪些内容变化不频繁的代码表，或者作为统计操作的中间结果表，便于高效地对中间结果进行分析并得到最终的统计结果。对存储引擎为memory的表进行更新操作要谨慎，因为数据并没有实际写入到磁盘中，所以一定要对下次重新启动服务后如何获得这些修改后的数据有所考虑。

乐观锁和悲观锁

无论是悲观锁还是乐观锁，都是人们定义出来的概念，可以认为是**处理并发控制的两种思想。**其实不仅仅是数据库系统中有乐观锁和悲观锁的概念，像memcache、hibernate、tair等都有类似的概念。所以，**不要把乐观并发控制和悲观并发控制狭义的理解为DBMS中的概念**，更**不要把他们和数据中提供的锁机制（行锁、表锁、排他锁、共享锁）混为一谈。**其实，在DBMS中，悲观锁正是利用数据库本身提供的锁机制来实现的。

悲观锁，正如其名，它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度(悲观)，因此，在整个数据处理过程中，将数据处于锁定状态。 悲观锁的实现，往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）。**悲观并发控制实际上是"先取锁再访问"的保守策略**，为数据处理的安全提供了保证。

乐观锁（ Optimistic Locking ） 相对悲观锁而言，乐观锁假设认为**数据一般情况下不会造成冲突**，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则让返回用户错误的信息，让用户决定如何去做。乐观锁并不会使用数据库提供的锁机制。一般的**实现乐观锁的方式就是记录数据版本。**

数据库事务的四个特性ACID

原子性【Atomicity】

原子性指的指的就是这个操作，要么全部成功，要么全部失败回滚。不存在其他的情况。

一致性（Consistency）

一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

举个例子。就是A和B的钱是1000元，A给你100元，无论最后双方转了多少次，总的钱一定是1000元。不会应为服务器意外，导致数据不一致。

隔离性（Isolation）

隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离

持久性（Durability）

持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

并发操作几个问题

在事务的并发操作中可能会出现更新丢失，脏读，不可重复读，幻读。

更新丢失

一个事务的更新覆盖了另一个事务的更新。

事务A：向银行卡存钱100元。事务B：向银行卡存钱200元。A和B同时读到银行卡的余额，分别更新余额，后提交的事务B覆盖了事务A的更新。更新丢失本质上是写操作的冲突，解决办法是一个一个地写。

脏读

脏读是指在一个事务处理过程里读取了另一个未提交的事务中的数据。

比如A转账给B100元，然后还没有提交成功，这个时候，你用B用手机付款，这个事务读取到这个100了。然后就行了扣款，B读到的这个数据就是脏数据。因为A没有提交，可能会撤销。

不可重复读

不可重复读是指在对于数据库中的某个数据，一个事务范围内多次查询却返回了不同的数据值，这是由于在查询间隔，被另一个事务修改并提交了。不可重复读一般是update操作

举例：比如A事务 读取了一个记录，然后此时B事务修改了这个记录会提交了，A事务再进行读取的时候就会跟之前的记录不一样。 会产生我们说的ABA问题。

幻读

幻读是事务非独立执行时发生的一种现象。例如事务T1对一个表中所有的行的某个数据项做了从“1”修改为“2”的操作，这时事务T2又对这个表中插入了一行数据项，而这个数据项的数值还是为“1”并且提交给数据库。而操作事务T1的用户如果再查看刚刚修改的数据，会发现还有一行没有修改，其实这行是从事务T2中添加的，就好像产生幻觉一样，这就是发生了幻读。 幻读一般是insert操作

举例：程序员某一天去消费，花了2千元，然后他的妻子去查看他今天的消费记录（全表扫描FTS，妻子事务开启），看到确实是花了2千元，就在这个时候，程序员花了1万买了一部电脑，即新增INSERT了一条消费记录，并提交。当妻子打印程序员的消费记录清单时（妻子事务提交），发现花了1.2万元，似乎出现了幻觉，这就是幻读。

四种隔离级别

数据库事务的隔离级别有4种，由低到高分别为Read uncommitted 、Read committed 、Repeatable read 、Serializable 。

Read uncommitted 未提交读

最低的隔离，就是可以读取到未提交的数据。会产生**脏读。**

A给B转账1000元。开启事务，还没提交。 此时B查看自己余额多了1000元。但是突然A输错了密码。放弃了转款，回滚。此时这个B看到1000元就是一个脏数据，因为事务没有提交完成。

Read committed 提交读

就是一个事务要等另一个事务提交后才能读取数据。

比如A的卡里有1000元。去请吃饭花付钱需要花900元。然后此时他媳妇B取出了500.当结账的时候，就会出现付款失败。出现这种情况，就是因为出现了B取钱的时候，没有等待A刷卡这个事务完成，而自己的事务开始读取和提交了数据。

Repeatable read 重复读

重复读，就是在开始读取数据（事务开启）时，不再允许修改操作。MySQL默认的事务隔离级别是Rr 重复读

比如上面的例子。当A进行付款的时候，B取款的操作就应该等待，不允许进行修改。

Serializable 序列化

Serializable 是最高的事务隔离级别，在该级别下，事务串行化顺序执行，可以避免脏读、不可重复读与幻读。但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用。

序列化就是将并行的操作。改成顺序执行。

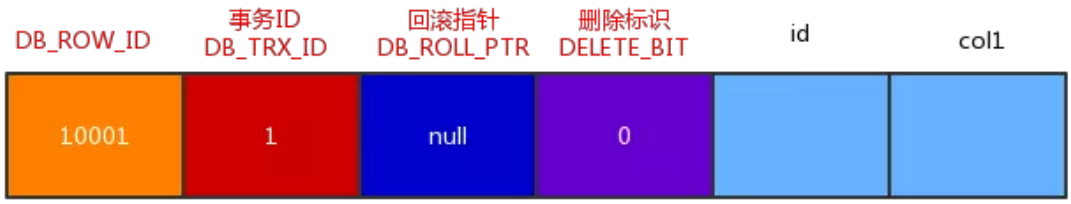
MCVV多版本并发控制

并发事务解决方案：

- 锁机制（悲观锁及乐观锁）-------解决**写写冲突**问题
- MVCC多版本并发控制-------解决**读写冲突**问题

MVCC是通过保存数据在某个时间点的快照来实现的。不管事务执行多长时间，事务看到的数据都是一致的。具体方法是在每行数据中包含一些隐藏字段。具体如下：

- 数据存储



DB_TRX_ID：用来标识最近一次对本行记录做修改的事务的标识符，即最后一次修改本行记录的事务id。delete操作在内部来看是一次update操作，更新行中的删除标识位DELELE_BIT。

DB_ROLL_PTR：指向当前数据的**undo log**记录，回滚数据通过这个指针来寻找记录被更新之前的内容信息。

DB_ROW_ID：包含一个随着新行插入而单调递增的行ID, 当由innodb自动产生聚集索引时，聚集索引会包括这个行ID的值，否则这个行ID不会出现在任何索引中。

DELELE_BIT：用于标识该记录是否被删除。

- 数据操作

insert

创建一条记录，DB_TRX_ID为当前事务ID，DB_ROLL_PTR为NULL。

delete

将当前行的DB_TRX_ID设置为当前事务ID，DELELE_BIT设置为1。

update 复制一行，新行的DB_TRX_ID为当前事务ID，DB_ROLL_PTR指向上个版本的记录，事务提交后DB_ROLL_PTR设置为NULL。

select

1、只查找创建早于当前事务ID的记录，确保当前事务读取到的行都是事务之前就已经存在的，或者是由当前事务创建或修改的；

2、行的DELETE BIT为1时，查找删除晚于当前事务ID的记录，确保当前事务开始之前，行没有被删除。

读操作分成两类：**快照读**和**当前读。**

MVCC下事务读操作就不需要加共享锁了。

提交读和重复读解决可重复读的实现机制就是MVCC

解决幻读的实现机制是MVCC和区间锁

Mysql索引

> 索引的目的在于提高查询效率，可以类比字典，如果要查“mysql”这个单词，我们肯定需要定位到m字母，然后从下往下找到y字母，再找到剩下的sql。如果没有索引，那么你可能需要把所有单词看一遍才能找到你想要的，如果我想找到m开头的单词呢？或者ze开头的单词呢？是不是觉得如果没有索引，这个事情根本无法完成？

主要有两种结构：**B+tree和hash.**

Hash索引： Hash 索引结构的特殊性，其检索效率非常高，索引的检索可以一次定位，不像B-Tree 索引需要从根节点到枝节点，最后才能访问到页节点这样多次的IO访问，所以**Hash 索引的查询效率要远高于 B-Tree 索引。**因为hash算法是基于等值计算的，所以对于“like”等**范围查找hash索引无效**，不支持；因为是散列存储，对于**排序也很吃力**；

Mysql常见索引

主键索引、唯一索引、普通索引、全文索引、组合索引

PRIMARY KEY（主键索引） ALTER TABLE `table_name` ADD PRIMARY KEY (`column`)

UNIQUE(唯一索引) ALTER TABLE `table_name` ADD UNIQUE (`column`)

INDEX(普通索引) ALTER TABLE `table_name` ADD INDEX index_name (`column`)

FULLTEXT(全文索引) ALTER TABLE `table_name` ADD FULLTEXT (`column`)

组合索引 ALTER TABLE `table_name` ADD INDEX index_name (`column1`, `column2`, `column3`)

聚集索引和辅助索引、覆盖索引

- 聚集索引（主键索引）

  innodb存储引擎是索引组织表，即表中的数据按照主键顺序存放。而聚集索引就是按照每张表的主键构造颗B+树，同时叶子节点中存放的即为整张表的记录数据

  聚集索引的叶子节点称为数据页，数据页，数据页！重要的事说三遍。聚集索引的这个特性决定了索引组织表中的数据也是索引的一部分。

- 辅助索引（二级索引）--- 非主键索引

叶子节点=键值+书签。Innodb存储引擎的书签就是相应行数据的主键索引值

- 覆盖索引

  如果查询的列恰好是索引的一部分，那么查询只需要在索引文件上进行，不需要进行到磁盘中找数据，若果查询的列不是索引的一部分则要到磁盘中找数据

  使用explain，可以通过输出的extra列来判断，对于一个索引覆盖查询，显示为**using index**,MySQL查询优化器在执行查询前会决定是否有索引覆盖查询

数据库优化

- **服务器优化**

  max_allowed_packet：要足够大，以适应比较大的SQL查询，对性能没有太大影响，主要是避免出现packet错误。 max_connections：server允许的最大连接。太大的话会出现out of memory。 table_cache：MySQL在同一时间保持打开的table的数量。打开table开销比较大。一般设置为512。 query_cache_size： 用于缓存查询的内存大小。 datadir：mysql存放数据的根目录，和安装文件分开在不同的磁盘可以提高一点性能。

  存储引擎：根据业务需要选择主要使用的有MyISAM和InnoDB。

  innodb_buffer_pool_size(配置innodb缓冲池的大小)：这是InnoDB最重要的设置，对InnoDB性能有决定性的影响。默认的设置只有8M，所以默认的数据库设置下面InnoDB性能很差。在只有InnoDB存储引擎的数据库服务器上面，可以设置60-80%的内存。更精确一点，在内存容量允许的情况下面设置比InnoDB tablespaces大10%的内存大小。

  innodb_buffer_pool_instances 可以控制缓冲池的个数，增加个数可以增加并发性。

  innodb_flush_log_at_trx_commit = 1

#关键参数，0代表大约每秒写入到日志并同步到磁盘，数据库故障会丢失1秒左右事务数据。1为每执行一条SQL后写入到日志并同步到磁盘，I/O开销大，执行完SQL要等待日志读写，效率低。2代表只把日志写入到系统缓存区，再每秒同步到磁盘，效率很高，如果服务器故障，才会丢失事务数据。对数据安全性要求不是很高的推荐设置2，性能高，修改后效果明显。 innodb_file_per_table = OFF

#默认是共享表空间，共享表空间idbdata文件不断增大，影响一定的I/O性能。推荐开启独立表空间模式，每个表的索引和数据都存在自己独立的表空间中，可以实现单表在不同数据库中移动。 innodb_log_buffer_size = 8M

#日志缓冲区大小，由于日志最长每秒钟刷新一次，所以一般不用超过16M

- **数据库优化**
  - 数据类型

    最基本的优化之一就是使表在磁盘上占据的空间尽可能小。这能带来性能非常大的提升，因为数据小，磁盘读入较快，并且在查询过程中表内容被处理所占用的内存更少。同时，在更小的列上建索引，索引也会占用更少的资源。 可以使用下面的技术可以使表的性能更好并且使存储空间最小：

    1.使用正确合适的类型，不要将数字存储为字符串。

    2.尽可能地使用最有效(最小)的数据类型。MySQL有很多节省磁盘空间和内存的专业化类型。

    3. 如果可能，声明列为NOT NULL。有利于更高效的利用索引，减少存储空间。
  - 查询语句的优化
    - 不要查询不需要的列，不要在多表关联返回全部的列，不要select *
    - 尽量使用**关联查询来替代子查询。**
    - union代替临时表
  - 索引使用

    > 对查询进行优化，就是为了避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。尽量使用索引优化。如果不使用索引。mysql则使用临时表或者文件排序。

    - 应尽量**避免**在 where 子句中对字段进行**null 值判断、使用 !=或<> 操作符、前缀%、对字段进行表达式及函数操作**，否则将导致引擎放弃使用索引而进行全表扫描。
    - 尽量选择区分度高的列作为索引。
    - 尽量的扩展索引，不要新建索引。比如表中已经有a的索引，现在要加(a,b)的索引，那么只需要修改原来的索引即可
    - 最左前缀匹配原则，非常重要的原则，mysql会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，比如a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d的顺序可以任意调整。
    - 查询时使用匹配的类型。例如select * from a where id=5， 如果这里id是字符类型，同时有index，这条查询则使用不到index，会做全表扫描，速度会很慢。正确的应该是 ... where id=”5” ，加上引号表明类型是字符。
    - 优化分页查询，最简单的就是利用**覆盖索引**扫描。

EXPLAIN分析SQL

- 慢查询优化基本步骤：
  1. 先运行看看是否真的很慢，注意设置SQL_NO_CACHE
  2. where条件单表查，锁定最小返回记录表。这句话的意思是把查询语句的 where都应用到表中返回的记录数最小的表开始查起，单表每个字段分别查询，看哪个字段的区分度最高

3. explain查看执行计划，是否与1预期一致（从锁定记录较少的表开始查询）
4. order by limit 形式的sql语句让排序的表优先查
5. 了解业务方使用场景
6. 加索引时参照建索引的几大原则
7. 观察结果，不符合预期继续从0分析

几个慢查询案例 下面几个例子详细解释了如何分析和优化慢查询

参考：

MySQL数据库锁机制之MyISAM引擎表锁和InnoDB行锁详解

数据库事务和MVCC多版本并发控制

5分钟带你读懂事务隔离性与隔离级别

数据结构之哈希表

MYSQL索引：对聚簇索引和非聚簇索引的认识

MySQL 覆盖索引

PHP

参考信息：

1. 一个 1年工作经验的 PHP 程序员是如何被面试官虐的？
2. 一份简历修改指南
3. 面试时如何优雅地自我介绍？
4. PHP面试总结
5. []
6. 金三银四，如何征服面试官，拿到Offer
7. 技术面试中，当面试官「套路」你时，怎么「反套路」回去？
8. 金三银四跳槽季，拿上攻略有底气！