# Practice 1

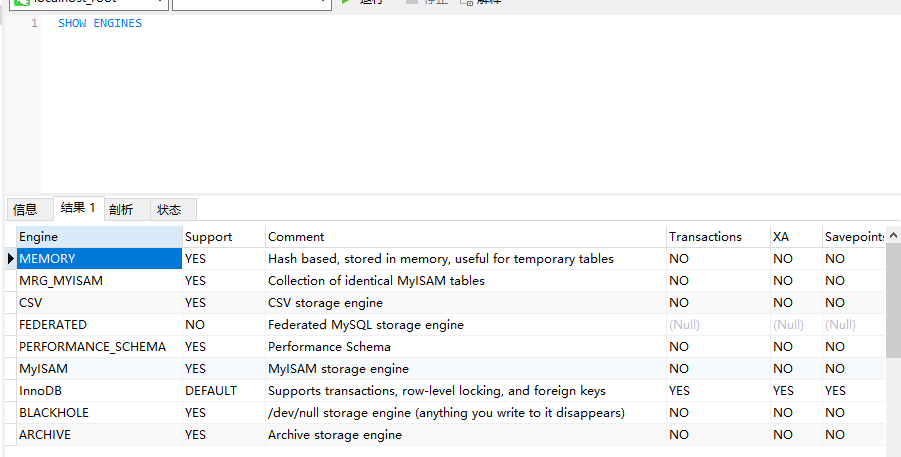
## Q: 打开你手头的数据库，去了解这个数据库的版本演变、 SQL引擎、事务处理级别划分、以及体系结构。

### mysql 版本历史演变

* **1995年**，MySQL 1.0发布，仅供内部使用。
* **1996年**，MySQL 3.11.1发布，直接跳过了MySQL 2.x版本。
* 1999年，MySQL AB公司成立。同年，发布MySQL 3.23，该版本集成了Berkeley DB存储引擎。该引擎由Sleepycat公司开发，支持事务。在集成该引擎的过程中，对源码进行了改造，为后续可插拔式存储引擎架构奠定了基础。
* **2000年**，ISAM升级为MyISAM存储引擎。同年，MySQL基于GPL协议开放源码。
* 2002年，MySQL 4.0发布，集成了后来大名鼎鼎的InnoDB存储引擎。该引擎由Innobase公司开发，支持事务，支持行级锁，适用于OLTP等高并发场景。
* 2005年，MySQL 5.0发布，开始支持游标，存储过程，触发器，视图，XA事务等特性。同年，Oracle收购Innobase公司。
* 2008， Sun以10亿美金收购MySQL AB。同年，发布MySQL 5.1，其开始支持定时器（Event scheduler），分区，基于行的复制等特性。
* 2009，Oracle以74亿美金收购Sun公司。
* 2010，MySQL 5.5发布，其包括如下重要特性及更新。 InnoDB代替MyISAM成为MySQL默认的存储引擎。 多核扩展，能更充分地使用多核CPU。 InnoDB的性能提升，包括支持索引的快速创建，表压缩，I/O子系统的性能提升，PURGE操作从主线程中剥离出来，Buffer Pool可拆分为多个Instances。 半同步复制。 引入utf8mb4字符集，可用来存储emoji表情。 引入metadata locks（元数据锁）。 分区表的增强，新增两个分区类型：RANGE COLUMNS和LIST COLUMNS。 MySQL企业版引入线程池。 可配置IO读写线程的数量（innodb*read*io*threads，innodb*write*io*threads）。在此之前，其数量为1，且不可配置。 引入innodb*io*capacity选项，用于控制脏页刷新的数量。
* 2013，MySQL 5.6发布，其包括如下重要特性及更新。 GTID复制。 无损复制。 延迟复制。 基于库级别的并行复制。 mysqlbinlog可远程备份binlog。 对TIME, DATETIME和TIMESTAMP进行了重构，可支持小数秒。DATETIME的空间需求也从之前的8个字节减少到5个字节。 Online DDL。ALTER操作不再阻塞DML。 可传输表空间（transportable tablespaces）。 统计信息的持久化。避免主从之间或数据库重启后，同一个SQL的执行计划有差异。 全文索引。 InnoDB Memcached plugin。 EXPLAIN可用来查看DELETE，INSERT，REPLACE，UPDATE等DML操作的执行计划，在此之前，只支持SELECT操作。 分区表的增强，包括最大可用分区数增加至8192，支持分区和非分区表之间的数据交换，操作时显式指定分区。 Redo Log总大小的限制从之前的4G扩展至512G。 Undo Log可保存在独立表空间中，因其是随机IO，更适合放到SSD中。但仍然不支持空间的自动回收。 可dump和load Buffer pool的状态，避免数据库重启后需要较长的预热时间。 InnoDB内部的性能提升，包括拆分kernel mutex，引入独立的刷新线程，可设置多个purge线程。 优化器性能提升，引入了ICP，MRR，BKA等特性，针对子查询进行了优化。 可以说，MySQL 5.6是MySQL历史上一个里程碑式的版本，这也是目前生产上应用得最广泛的版本。
* 2015，MySQL 5.7发布，其包括如下重要特性及更新。 组复制 InnoDB Cluster 多源复制 增强半同步（AFTER*SYNC） 基于WRITESET的并行复制。 在线开启GTID复制。 在线设置复制过滤规则。 在线修改Buffer pool的大小。 在同一长度编码字节内，修改VARCHAR的大小只需修改表的元数据，无需创建临时表。 可设置NUMA架构的内存分配策略（innodb*numa\_interleave）。 透明页压缩（Transparent Page Compression）。 UNDO表空间的自动回收。 查询优化器的重构和增强。 可查看当前正在执行的SQL的执行计

### Mysql引擎

好多引擎。。。



#### innodb

InnoDB是事务型数据库的首选引擎，支持事务安全表（ACID），支持行锁定和外键，InnoDB是默认的MySQL引擎。InnoDB主要特性有：

1、InnoDB给MySQL提供了具有提交、回滚和崩溃恢复能力的事物安全（ACID兼容）存储引擎。InnoDB锁定在行级并且也在SELECT语句中提供一个类似Oracle的非锁定读。这些功能增加了多用户部署和性能。在SQL查询中，可以自由地将InnoDB类型的表和其他MySQL的表类型混合起来，甚至在同一个查询中也可以混合

2、InnoDB是为处理巨[**大数据**](http://lib.csdn.net/base/hadoop)量的最大性能设计。它的CPU效率可能是任何其他基于磁盘的关系型数据库引擎锁不能匹敌的

3、InnoDB存储引擎完全与MySQL服务器整合，InnoDB存储引擎为在主内存中缓存数据和索引而维持它自己的缓冲池。InnoDB将它的表和索引在一个逻辑表空间中，表空间可以包含数个文件（或原始磁盘文件）。这与MyISAM表不同，比如在MyISAM表中每个表被存放在分离的文件中。InnoDB表可以是任何尺寸，即使在文件尺寸被限制为2GB的[**操作系统**](http://lib.csdn.net/base/operatingsystem)上

4、InnoDB支持外键完整性约束，存储表中的数据时，每张表的存储都按主键顺序存放，如果没有显示在表定义时指定主键，InnoDB会为每一行生成一个6字节的ROWID，并以此作为主键

5、InnoDB被用在众多需要高性能的大型数据库站点上

InnoDB不创建目录，使用InnoDB时，MySQL将在MySQL数据目录下创建一个名为ibdata1的10MB大小的自动扩展数据文件，以及两个名为ib*logfile0和ib*logfile1的5MB大小的日志文件

#### **MyISAM存储引擎**

MyISAM基于ISAM存储引擎，并对其进行扩展。它是在Web、数据仓储和其他应用环境下最常使用的存储引擎之一。MyISAM拥有较高的插入、查询速度，但**不支持事物**。MyISAM主要特性有：

1、大文件（达到63位文件长度）在支持大文件的文件系统和操作系统上被支持

2、当把删除和更新及插入操作混合使用的时候，动态尺寸的行产生更少碎片。这要通过合并相邻被删除的块，以及若下一个块被删除，就扩展到下一块自动完成

3、每个MyISAM表最大索引数是64，这可以通过重新编译来改变。每个索引最大的列数是16

4、最大的键长度是1000字节，这也可以通过编译来改变，对于键长度超过250字节的情况，一个超过1024字节的键将被用上

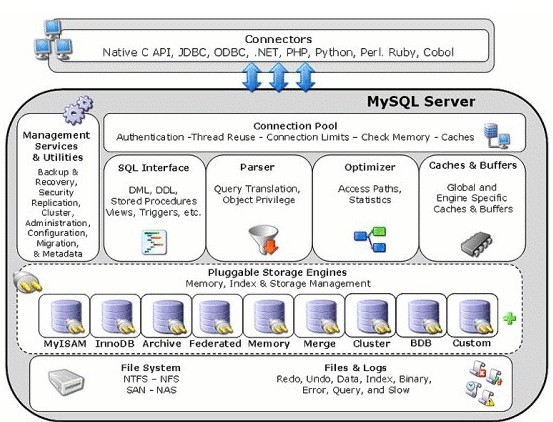
5、BLOB和TEXT列可以被索引

6、NULL被允许在索引的列中，这个值占每个键的0~1个字节

### mysql 事务处理级别划分



### mysql体系结构



# Practice 2-1

## Q: 对大多数码农而言，数据库锁机制好像都是自动和透明实现的，那么深入了解每个数据库的锁机制实现细节，对码农编码有什么影响嘛？

平时没有什么问题，但是一旦出现关于锁的异常，如果不懂锁的机制就难以定位问题。

## Q: 根据这节课Oracle的锁机制特征的分析，你尝试去了解一下MySQL、SQLServer这些其它数据库锁机制实现的特征

### InnoDB锁算法

#### 记录锁(Record-Lock)

记录锁是锁住记录的，这里要说明的是这里锁住的是索引记录，而不是我们真正的数据记录。

* 如果锁的是非主键索引，会在自己的索引上面加锁之后然后再去主键上面加锁锁住.
* 如果没有表上没有索引(包括没有主键)，则会使用隐藏的主键索引进行加锁。
* 如果要锁的列没有索引，则会进行全表记录加锁。

#### 间隙锁

间隙锁顾名思义锁间隙，不锁记录。锁间隙的意思就是锁定某一个范围，间隙锁又叫gap锁，其不会阻塞其他的gap锁，但是会阻塞插入间隙锁，这也是用来防止幻读的关键。

#### next-key锁

这个锁本质是记录锁加上gap锁。在可重复读隔离级别下(InnoDB默认)，Innodb对于行的扫描锁定都是使用此算法，但是如果查询扫描中有唯一索引会退化成只使用记录锁。为什么呢? 因为唯一索引能确定行数，而其他索引不能确定行数，有可能在其他事务中会再次添加这个索引的数据会造成幻读。

#### MVCC

MVCC，多版本并发控制技术。在InnoDB中，在每一行记录的后面增加两个隐藏列，记录创建版本号和删除版本号。通过版本号和行锁，从而提高数据库系统并发性能。

在MVCC中，对于读操作可以分为两种读:

* 快照读:读取的历史数据，简单的select语句，不加锁，MVCC实现可重复读，使用的是MVCC机制读取undo中的已经提交的数据。所以它的读取是非阻塞的。
* 当前读:需要加锁的语句,update,insert,delete,select...for update等等都是当前读。

在RR隔离级别下的快照读，不是以begin事务开始的时间点作为snapshot建立时间点，而是以第一条select语句的时间点作为snapshot建立的时间点。以后的select都会读取当前时间点的快照值。

# Practice in class 2-2

## Q: 你对你常用的关系数据库关系系统中，去寻找一些针对优化的工具，去尝试使用一些性能的分析和监控工具（查看数据库官方Reference，首先使用官方的命令和工具）

### mysqlsla

https://www.cnblogs.com/ding2016/p/9755468.html

Mysql日志分析工具

mysqlsla -lt slow --sort t\_sum --top 20 mysql-slow.log > /tmp/select.log  
// 查询记录最多的20个sql语句，并写到select.log中去。  
mysqlsla -lt slow -sf "+select" -top 100 mysql-slow.log >/tmp/sql\_select.log  
// 统计慢查询文件为mysql-slow.log中的所有select的慢查询sql，并显示执行时间最长的100条sql，并写到sql\_select.log中去  
mysqlsla -lt slow -sf "+select,update" -top 100 -sort c\_sum -db mydata mysql-slow.log >/tmp/sql\_num.log  
// 统计慢查询文件为mysql-slow.log的数据库为mydata的所有select和update的慢查询sql，并查询次数最多的100条sql，并写到sql\_num.sql中去

### mysqlreport

https://blog.csdn.net/dy*252/article/details/6717408?utm*medium=distribute.pc*relevant.none-task-blog-BlogCommendFromBaidu-1.control&depth*1-utm*source=distribute.pc*relevant.none-task-blog-BlogCommendFromBaidu-1.control

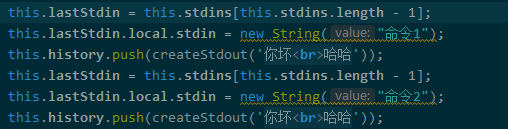
进行MySQL的配置优化，首先必须找出MySQL的性能瓶颈所在；而SHOW STATUS输出的报告正是用来计算性能瓶颈的参考数据。mysqlreport不像SHOW STATUS那样简单的罗列数据，而是对这些参考数据加以融合计算，整理成一个个优化参考点，然后DBA就可以根据这个优化参考点的值以及该点的衡量标准，进行对应调整。

## Q: 关于把数据库当成黑盒使用的错误，其实也会在你学习软件开发中遇到类似的问题，比如，对操作系统的黑盒化，比如对某些开发框架的黑盒化等等，请你思考一下，你的学习过程中，还能找到类似的例子嘛？

写前端时用到了vue-command,安装之后效果如下。想完成如下需求，



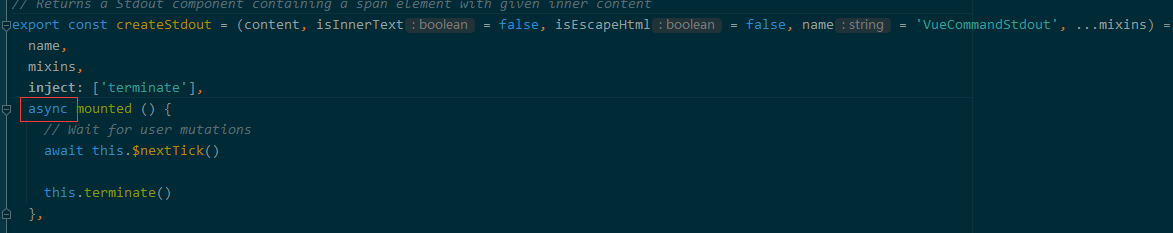
于是写了如下代码：



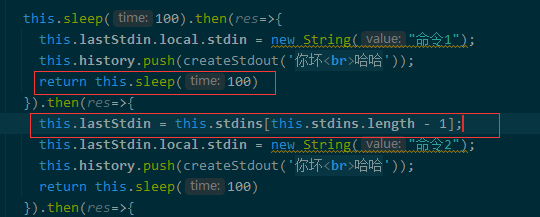
但是只能得到下面的效果



查看源码后发现，createStdout是一个异步的代码，期间最后一个标准输入不会更新，导致下一条的lastStdin还是指向的上一个，甚至于上上一个的标准输入，这样此次做的修改就会被应用到错误的地方。



不改变源码的情况，解决办法想到了让程序短暂休眠，但也只是权益之策。



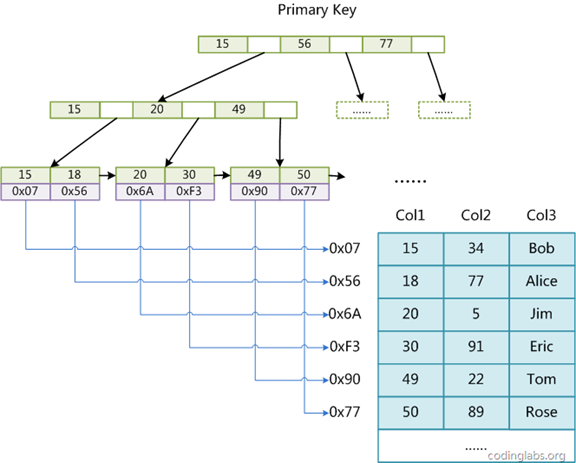
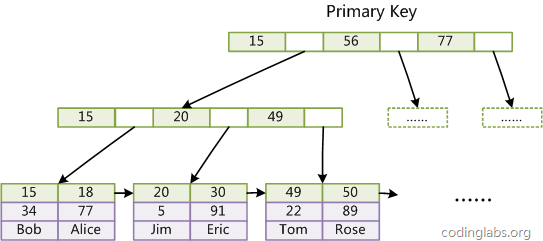
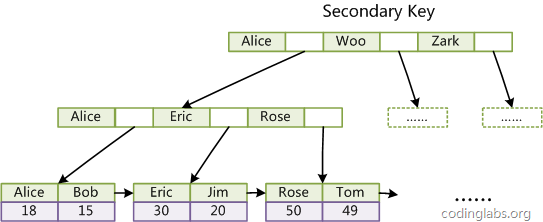
# Practice 3.1

那么不管怎么样，但，它至少能够提升查询效率不是吗？

对于B+树索引，不少数据库都有自己的处理方式，比如， MySQL中不同的存储引擎使用了  
不同的方式把索引保存到磁盘上，他们会影响性能。  
• MyISAM使用前缀压缩以减少索引，而InnoDB不会压缩索引，（有啥差别？）  
• MyISAM索引按照行存储的物理位置引用被索引的行，但是InnDB按照主键值引用行，（有啥差别?）

请有兴趣的同学尝试去看一下你所用数据库索引的官方参考

答：

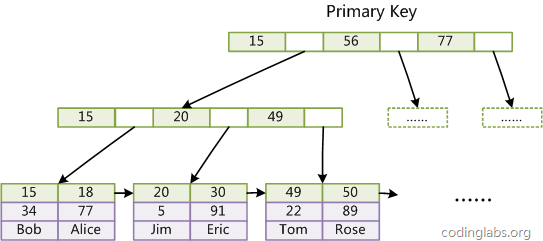
* 一定程度上能提高查询效率，但是如果滥用，可能代价会更大。如对于小型表，大部分情况下全表扫描更有效。
* MyISAM使用前缀压缩减少索引的大小，从而让更多的索引可以放进内存中，这在某些情况下能极大的提高性能。MyISAM压缩的方法是：先完全保存索引块中的第一个值，然后将其他值和第一个值进行比较得到相同前缀的字节数和剩余的不同后缀部分，然后把这部分存储起来。例如：索引块中第一个值是“perform”,第二个值是“performance”，那么第二个值得前缀压缩后存储的是类似“7，ance”这样的形势。但是代价是，可能某些操作会变得慢，如无法在索引块中使用二分查找，而只能从头扫描。
* MyISAM引擎使用B+树作为索引结果，叶节点的data域存放的是数据记录的地址。如下图
* 
* **在MyISAM中，主索引和辅助索引在结构上没有任何区别，只是主索引要求key是唯一的，而辅助索引的key可以重复。**
* InnoDB表数据文件本身就是一个索引结构，树的叶节点data域保存了完整的数据记录，这种索引叫做**聚集索引**。如下图
* 
* InnoDB的辅助索引：如图所示
* 
* **聚集索引这种实现方式使得按主键的搜索十分高效，但是辅助索引搜索需要检索两遍索引：首先检索辅助索引获得主键，然后用主键到主索引中检索获得记录。**不建议使用过长的字段作为主键，因为所有辅助索引都引用主索引，过长的主索引会令辅助索引变得过大。用非单调的字段作为主键在InnoDB中不是个好主意，因为InnoDB数据文件本身是一颗B+Tree，非单调的主键会造成在。插入新记录时数据文件为了维持B+Tree的特性而频繁的分裂调整，十分低效，而使用自增字段作为主键则是一个很好的选择。
* 既然，使用复合键索引，在select子句中，如果所有字段都在复合键索引所包括的字段之中的时候，查询可以只使用索引不使用表
* 那么，为什么不可以针对表T（x,y,z）这样的表，直接构建一个索引index（x,y,z），这样所有对这个表的访问就可以直接使用索引不使用表了，这会不会大幅度地提升查询效率呢？

答：不一定能大幅度提高查询效率，对于where x=\*\* and y =\*\*, where x=\*\* where x=\*\* and y =\*\* and z=\*\*可以提高查询效率，但是，对于非前缀查询，如where y=\*\*等，还是会全表扫描。

# Practice 3.2

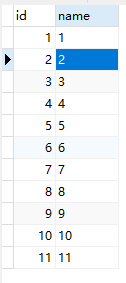
* 请研究你手上使用的数据库，比如， MySQL or Oracle，请研究数据库管理系统提供的其它索引形式，并阅读相关的文档
* 了解最常使用的3种索引的名称、结构、适用范围、不适用的范围、有益的例子、错误使用的例子等等，以及自己试一下。

答：【mysql】

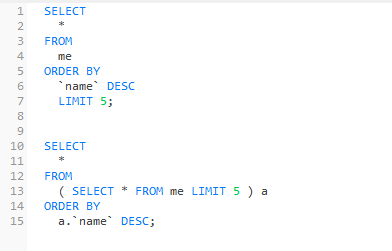
* B-Tree索引，实际上是B+Tree, 结构如图：
* 
* B-tree索引适合：全键值，键值范围或者键前缀查找。如where key = 1, where key >= 100, where key1=1
* 如果索引列有多个，不能跳过索引中的列，如索引（a,b,c）则不能where a = 1 and c =1。
* 如果查询中有某个列的范围查询，则其右边所有列都无法使用索引优化查找。
* Hash索引
* Hash索引基于hash表实现，只有精确匹配索引**所有列**查询才有效。
* 使用限制：
  + Hash索引并不是按照索引值顺序存储的，无法用于排序
  + Hash索引不支持部分索引列匹配查找，如在（a，b）上建立索引，如果查询只有数据列A，则无法使用该索引。
  + 只支持等值查询
* 空间数据索引
* 可以用于地理数据的存储，这类数据无需前缀查询，空间索引会从所有维度来索引数据。

# Practice 4.1

• Oracle的rownum是一个非常讨厌的SQL方言，但它是Oracle数据库中唯一的限定返回行数的函数，其它数据库也有类似的方言  
• DB2使用FETCH FIRST子句  
• MySQL和PostgreSQL使用LIMIT子句  
• SQL Server使用TOP关键字  
• 请你用你手上常用的数据库试一下本课程那个限定返回行数查询的例子，看看有没有Oracle出现的问题  
• 如果你是用Oracle，你试一下，你能通过rownum=5，来返回第5行记录嘛？

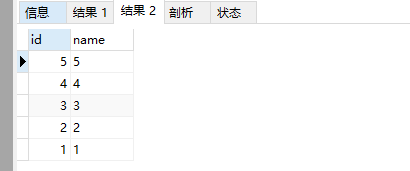


如图：查询前5条数据，并且按照名字排序 sql如图：



结果如下：





显然正确的sql应该是第二条，

* 你还有什么方法（自己遇到的，或者查询技术资料、论坛等等资源）能够在数据库应用方面，照顾好用户的情绪？欢迎你的分享。

比如，在微服务架构中，有的时候用户需要查询统计数据，可能需要调用其他微服务的接口，通过调用接口的形式来调用远程应用的数据库进行查询操作时延可能会比较大，我们可以使用canal在本地创建一个相同的数据库，并将远程数据库的数据定期同步到本地数据库，这样以来，用户调用本地服务查询统计等信息的效率将大幅度提升。