操作系统

1. 操作系统的四大特性：
   a) 并发性
   b) 共享性
   c) 虚拟性
   d) 不确定性
2. 请叙述一下并发和并行两个概念的区别？
   a) 并行是指两个或多个事件在同一时刻发生，并发是指两个或多个事件在同一时间间隔内发生。
3. 什么是进程？什么是线程？
   a) “进程是一个**可并发执行**的，具有**独立功能**的程序关于某个数据集合的一次执行过程，也是**操作系统进行资源分配和调度的独立单位**”。**进程是资源分配的基本单位。**
   b) 线程也称为轻量级进程(LWP)，是程序执行流量的最小单位，它是进程的一个实体，是系统独立调度和分派处理机的基本单位。**线程是操作系统调度的最小单位。**
4. 进程和线程的区别？从调度、并发性、拥有资源和系统开销四个方面来比较：
   a) 调度。在引入线程的操作系统中，把线程作为调度和分派 CPU 的基本单位，把进程作为资源分配的基本单位，显著提高了并发程度。**由于系统调度的基本单位是线程，所以每个进程至少创建一个线程，否则无法被调度。**
   b) 并发性。多线程可以提高服务的质量，在一个线程阻塞时，还有其他线程提供服务。
   c) 拥有资源。进程是拥有资源的独立单位，线程自己不拥有系统资源，而是共享进程的资源(包括代码段、数据段即系统资源等)。
   d) 系统开销。**进程切换的开销远远大于线程切换的开销，进程的切换需要保存很多现场，但线程只需要保存和设置少量的寄存器内容，不涉及存储器管理方面的操作。**
5. 进程的特征：

a) 动态性：进程是动态产生和动态消亡的，有其生存周期。

b) 并发性：一个进程可以与其他进程一起向前推进。

c) 独立性：一个进程是一个相对完整的调度单位。

d) 异步性：每个进程都已相对独立、不可预知的速度向前推进。

e) 结构性：**为了控制和管理进程，系统为每个进程设立一个进程控制块(PCB)。**

6. 进程有几种状态？

a) 就绪状态

b) 运行状态：单 CPU 环境下，系统处于运行状态的进程最多只有一个。

c) 阻塞状态

7. 进程的组成：

a) 程序：

b) 数据集合：进程独有

c) **进程控制块(PCB)**：它和进程一一对应，PCB 是操作系统能感知进程存在的唯一标识，**操作系统正是通过管理 PCB 来管理进程的。**系统创建进程时，为每个进程分配 PCB，进程执行完成后，系统释放 PCB，进程也随之消亡。

8. 进程的切换：

a) 进程上下文包含用户级上下文、系统级上下文、寄存器上下文

b) 进程上下文切换是核心态的切换，不发生在用户态。

c) **用户态到核心态之间的转变是 CPU 模式的改变。模式切换不同于进程切换，并不引起进程状态的改变。**

9. 为什么会提出线程这个概念？进程的缺点是什么？

a) 进程切换开销大

b) 进程通信代价大

c) 进程之间并发性粒度粗，并发度不高

d) 不适合并行计算和分布式并行计算的要求

e) 不适合客户-服务器计算的要求

f) **操作系统中引入进程的目的是为了使多个程序并发执行，改善资源的利用率以提高系统的吞吐量。**

10. 线程的分类：

a) 用户级线程 – ULT，优点是线程切换不需要系统状态的转换，每个进程可以使用专门的调度算法来调度线程，不需要依赖操作系统底层的内核。缺点是父进程阻塞会导致线程全都阻塞，不能真正的并行。**Java 的线程就是一种用户级的线程。**

b) 内核级线程 – KLT，**优点是同一个进程内多个线程可以并行执行，缺点是线程状态转换时内核态的任务，通常很慢。**

c) 混合式线程 – 上两种的结合，有良好的效果。

11. 进程控制块的作用？它是如何描述进程动态性质的？

a) PCB 是系统感知进程存在的唯一标志，是进程动态特性的集中反映，和进程一一对应，操作系统通过管理 PCB 来管理进程。

b) 进程控制块包含进程描述信息，控制信息，和资源管理信息三类。这些信息的变化反映进程的动态性质。

12. 操作系统内核都包括哪些内容？

a) 一是支撑功能，包括中断处理，时钟管理和原语操作等，二是资源管理功能，包括进程管理，存储器管理和设备管理等。

13. 处理机调度的三个分类？

a) 高级调度 – 作业调度

b) 中级调度 – 交换调度(内存和硬盘之间的交换)

c) 低级调度 – 进程调度(操作系统的核心)

d) 高级调度发生在创建新进程时，它决定一个进程能否被创建，或者是创建后能否被设置成就绪状态，以参与竞争处理器资源；中级调度反映到进程状态上就是挂起和解除挂起，它根据系统的当前符合情况决定停留在主存中的进程数；低级调度则是决定哪一个就绪进程或线程占有 CPU 运行。

14. 批作业调度算法

a) 先来先服务

b) 最短作业优先

c) 响应比高者优先

d) 优先级算法

15. 进程调度的方式：

a) 非剥夺方式(非抢占式方式)：优点是简单、易实现，系统开销小。缺点是不太灵活，难以满足紧迫任务必须立即执行的要求。实时系统不宜采用这种调度方式。

b) 可剥夺方式(抢占式方式)：优先权更高的进程优先执行。

16. 进程调度的算法：

a) 先来先服务算法

b) 最短优先算法

c) **优先级算法：优先级高的先执行。可以采用剥夺或非剥夺，剥夺方式更能反映优先的特点，但是会造成无穷阻塞和饥饿现象。**

d) 轮转算法：基本思想是系统把所有就绪进程按先来先服务的原则排成一个队列，且规定一个较小的时间单元，称为时间量或时间片，按时间片把 CPU 轮流分配给进入就绪队列的第一个进程使用，当进程的时间片使用完后，产生一个时钟中断，剥夺该进程的执行，将它送到就绪队列的队尾，等待下次调度。**轮转算法专门为分时系统设计。**

e) 多级队列调度算法：根据不同的进程，分配到不同的队列中，实行不同的调度算法。一般按照优先级分成多个队列，高优先级的任务先做。

f) 多级反馈队列调度算法：按优先级分配队列，但是优先级越高分配的时间片越小，如果没做完，自动移动到下一级队列继续做。

g) 实时调度算法

17. 操作系统多任务的抢占机制是怎么实现的？

18. 哪种权限许可用户进入一个文件系统的目录？Read，Execute，Write，Access Controll

19. Linux32 位系统，应用程序最多能分配的内存大小？

20. sleep 和 wait 的区别？

21. Win32 下线程的基本模式？

22. 进程之间的关系？

a) 竞争关系：由于进程之间不知道彼此的存在，而使用了同一份资源，就会造成竞争。**资源竞争会出现饥饿和死锁。**

b) 协作关系

23. 什么是临界资源？什么是临界区？

a) **临界资源：把一次只允许一个进程使用的资源成为临界资源。**(**独占性**，如打印机，卡片输出机等)

b) **临界区：把每个进程中访问临界资源的那段代码从概念上分离出来，将其称为临界区。即临界区是指对临界资源实时操作的程序的代码段。**

c) **相关临界区：并发进程中涉及相同临界资源的临界区。**相关临界区必须互斥执行。

24. 什么是进程互斥？

a) 进程互斥是解决进程间竞争关系(间接制约关系)的手段。**指任何时刻不允许两个以上的共享该资源的并发进程同时进入临界区，这种现象称为互斥。**

b) **相关临界区的管理原则：互斥、空闲让进、有限等待。**

25. 进程同步的概念？进程同步指两个或多个进程为了合作完成同一个任务，在执行速度或某些确定的时序点上必须相互协调，**即一个进程的执行依赖于另一个进程的消息，当一个进程到达了某一个确定点而没有得到合作伙伴发来的已完成消息时必须等待，知道该消息到达被唤醒后，才能继续向前推进。**

26. 进程同步和互斥的关系？

a) **进程的互斥实际上是进程同步的一种特殊情况**，即主次使用互斥共享资源，也是对进程使用资源次序上的一种协调。**进程的互斥和同步统称为进程同步。**

b) 进程的互斥是进程间共享资源的使用权，**这种竞争没有固定的必然联系**，哪个进程竞争到资源的使用权，该资源就归哪个进程使用，直到它不再需要使用时才归还资源；而进程同步中，**所涉及的共享资源的并发进程间有一种必然的联系，当进程必须同步时，即使无进程在使用共享资源，尚未得到同步消息的进程也不能去使用该资源。**

27. 信号量机制(PV 操作)：

a) **信号量机制的实现原理是两个或多个进程可以利用彼此间收发的简单信号来实现正确的并发执行，一个进程在收到一个指令信号前，会被迫在一个确定的或者需要的地方停下来，从而保持同步或互斥。**

b) 用信号量机制解决进程的同步和互斥问题有如下三个步骤：

i. 分析进程之间的制约关系

ii. 设置信号量

iii. 实施 P、V 操作

28. **同步的实现机制：**

a) **临界区：通过多线程的串行化来访问公共资源或者一段代码，速度快，适合控制数据访问。**

b) 互斥量：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限，因为互斥对象只有一个，所以可以保证公共资源不会同时被多个线程访问。

c) 信号量：允许多个线程同时访问同一资源，但是需要限制同一时刻访问此资源的最大线程数目。信号量对象对线程的同步方式与前面几种方法不同，信号允许多个线程同时使用共享资源，这与操作系统 PV 操作相似。

d) 事件(信号)：通过通知操作的方式保持多线程同步，还可以方便的实现多线程的优先级比较的操作。

29. 经典的同步问题：

a) 生产者消费者问题

b) 读者-写者问题（读者优先：信号量+读进程计数器 rc；弱写者优先：信号量+读进程计数器 rc+排队信号量 read；强写者优先：信号量+读进程计数器 rc+排队信号量 read+写优先信号量 write_first）

c) 哲学家就餐问题：是在多个线程之间共享多个资源时会不会导致死锁或饥饿的典型模型。解决方案：

i. 每个哲学家取得手边的两个叉子才能吃面，即仅当一个哲学家左右两边的叉子都可用时，才允许他拿叉子，否则一个叉子也不取。

ii. 偶数号哲学家先取手边的叉子，奇数号哲学家先取右手边的叉子。

d) 嗜睡理发师问题

30. 进程通信的方式：

a) 共享存储：消息缓冲

b) 消息传递：信箱

c) 管道通信

31. 产生死锁的原因？

a) 进程竞争资源引起的死锁

b) 进程推进顺序不当产生死锁

32. 产生死锁的条件

a) 互斥条件：同时只能有一个进程持有资源

b) 请求和保持条件：一个进程请求资源得不到满足时，不释放占有的资源

c) 不剥夺条件：任何一个进程不能抢夺其他进程占有的资源

d) 循环等待条件：存在一个循环等待链，链中每个进程已获得资源，并分别等待前一个进程持有的资源。

33. 处理死锁的方法：

a) 死锁预防：破坏产生死锁条件的任何一个或多个，如静态资源分配策略(2)和按序分配资源策略(4)。

b) 死锁避免：采用银行家算法，每次分配都查看能否找到一种资源分配方法，使得已有的进程可以顺利完成任务，如果有，则分配，否则不分配。

c) 死锁检测和解除：用软件来检查有进程和资源构成的有向图是否存在一个或多个回路。

34. 分页存储管理、段式存储管理和段页式存储管理的基本思想：

a) (分页) 利用分页存储管理，允许把一个作业存放到若干个不相邻的内存区域中，减少大碎片。

b) (分段) 段式存储管理支持用户的分段观点，以段为单位进行存储空间的分配。分段存储管理的引入，主要为了方便编程、信息共享和信息保护(有利于程序的运行)。

c) (段页式) 段页式存储管理的基本原理是先将整个主存划分成大小相等的存储块(页框)，把用户程序分段，接着为每一段进行分页。

35. 分页和分段的区别？

a) 分页是信息的物理单位，与源程序的逻辑结构无关，用户不可见，分页的目的主要是为了减少碎片，提高主存的利用率。分段是信息的逻辑单位，由源程序的逻辑结构来决定，目的是更好地满足用户的需求。

b) 页的大小固定且由系统确定，而段的长度不固定，由用户程序决定。

c) 分页的作业地址空间是一维的(线性地址空间)，分段的作业地址空间是二维的(段名和段内地址)。

36. 缓存的局部性原理：根据研究，在较短的时间内，程序的执行会局限于某一个部分，则可以根据当前程序运行的位置，推测可能执行的程序，预先加载，来达到缓存的目的。(虚拟内存的实现)

37. RAID 技术：

a) RAID 1：两个磁盘互相备份，安全性最好，但磁盘利用率 50%，最低。

b) RAID 2：采用汉明码做出错校验，按位交叉存取，用于大数据的读写，但冗余信息开销大，已被淘汰。

c) **RAID 3：位交织奇偶校验，使用一个磁盘做奇偶校验，数据分段存储在其余磁盘中，一旦有损坏，可以利用奇偶校验来重建数据，但校验盘损坏则没救，磁盘利用率 n-1。**

d) RAID 4：块交织奇偶校验，按块存取，可以单独对某个盘进行操作，一次操作只涉及数据盘和校验盘，不适合随机分散的小数据

e) **RAID 5：块交织分布式奇偶校验，同样以数据校验位来保证数据的安全，不同于校验盘，它将数据段的校验位交互存放于各个硬盘，则任何一个硬盘损坏，都可以根据其他硬盘上的校验位来重建损坏的数据，磁盘利用率 n-1.**

38. SPOOLing 系统？
    a) 在内存和硬盘中间建立缓冲区，在内存写入硬盘的过程中，先写入缓冲，等到 CPU 空闲时，才从缓冲区写入硬盘。
39. 同步和异步有什么不同？各自的优势？
40. 什么是线程？线程的基本状态？
41. synchronized 和 Lock 的异同？
42. 什么是序列化？什么是持久化？什么是串行化？transient 的用法？哪些字段需要标记 transient？
43. synchronized 关键字的用法？
44. 什么是守护线程？举一个守护线程的例子？

计算机网络

1. OSI 模型及其各层次的作用？TCP/IP 模型各层次及协议？
2. 数据链路层和 MAC 层(介质访问控制层)？
3. 计算机网络协议、接口和服务的概念？
4. 数据链路层：流量控制、可靠传输和滑动窗口机制。
5. 数据链路层设备：网桥和局域网交换机
6. 网络层的功能：异构网络互联、路由与转发、拥塞控制
7. 路由算法：距离-向量路由算法、链路状态路由算法、层次路由

8. IPv4 内容：分组、NAT、子网划分和子网掩码
9. ARP 协议、DHCP 协议与 ICMP 协议
10. 网络层设备：路由器的组成和功能，路由表和路由转发
11. 流量控制和拥塞控制的区别？
12. 传输层的功能？传输层的寻址与端口？套接字？
13. TCP 协议和 UDP 协议？Java 实现？UDP 首部？
14. TCP 段、TCP 连接管理、TCP 可靠传输、TCP 流量控制和拥塞控制
15. TCP 的拥塞控制：慢开始、拥塞避免、快重传、快恢复。
16. 应用层:P2P 模型？应用层协议：DNS、FTP、EMAIL、MIME、STMP、POP3、HTTP。。
17. CGI？域名解析过程？

MySQL 数据库

1. INSERT 嵌套 SELECT
2. MySQL 数据类型，int 类型长度超出了也不会对数据产生影响，和 SQL MODE 有关。
3. 一个表中最多只能有一个自增长列。
4. DDL、DML、DCL、DQL 语句？
    a) DDL（数据定义语言）：create、drop、alter
    b) DML（数据操纵语言）：insert、delete、update、select
    c) DCL（数据控制语言）：grant、revoke
5. MySQL 的其他数据库：
    a) information_schema 数据库存储了数据库对象信息，比如用户表信息、列信息、权限信息、字符集信息、分区信息等。
    b) cluster 存储了系统的集群信息
    c) mysql 存储了系统的用户权限信息
    d) test 是系统自动创建的测试数据库，任何用户都可以使用。
6. 内连接查询即为多表查询，仅选出两张表中互相匹配的记录，外连接则会选出其他不匹配的记录。子查询经常使用 in、not in、=、!=、exists、not exists 等。表连接很多情况下用于优化子查询。合并记录用 union(含 distinct)或 union all。
7. MySQL 数字类型及字节数、日期类型及字节数、字符串类型及字节数。

8. MySQL 可以使用 REGEXP 或 RLIKE 来使用正则匹配。NULL 不能用于"=,<,>,<>等"，但可以使用<=>比较 NULL。比较时不区分大小写，数字作为浮点数比较。短路与和非 NULL 结果都为 NULL，只有 1 或 NULL 为 1。
9. MySQL 常用函数：
    a) 字符串 CONCAT、INSERT、LEFT、RIGHT、REPLACE、SUBSTRING、LPAD、RPAD、TRIM、REPEAT 等。
    b) 数值函数 ABS、CEIL、FLOOR、MOD、RAND、ROUND、TRUNCATE 等。x%y 任意为 NULL 结果都为 NULL。产生 0~100 随机数为 select ceil(100*rand()), ceil(100*rand())。
    c) 日期和时间函数 CURDATE、CURTIME、NOW、UNIX_TIMESTAMP、FROM_UNIXTIME、DATE_FORMAT、DATEDIFF、DATE_ADD 等，格式为'%Y-%m-%d %H-%i-%s'。
    d) 流程函数 IF、IFNULL、CASE WHEN THEN ELSE END、CASE (exp) WHEN THEN ELSE END。
    e) 其他函数 DATABASE、VERSION、USER、INET_ATON、INET_NTOA、PASSWORD、MD5。
10. MySQL 存储类型：MyISAM、InnoDB、BDB、MEMORY、CSV 等，只有 InnoDB 和 BDB 支持事务，其他都不支持。MySQL 默认存储引擎为 InnoDB。
11. **InnoDB 和 MyISAM 的区别？事务、外键、访问速度、占用空间、存储方式(MyISAM：静态表、动态表和压缩表。InnoDB：共享表空间、多表空间)等方面。**
12. MEMORY 表使用 HASH 索引，数据量大小由 max_heap_table_size 变量来决定，默认 16MB。
13. 如何选择合适的存储引擎？
    a) 以读操作和插入操作为主，少量更新和删除，对事务完整性、并发性要求不高则使用 MyISAM 引擎。
    b) InnoDB 则适合于并发要求高的场合，事务完整性要求较高，除了可以有效降低删除和更新导致的锁定，还可以确保事务的完整提交和回滚，适合财务管理系统。
    c) MEMORY 通常用于更新不太频繁的小表，可以快速得到结果。
    d) MERGE 组合了 MyISAM 表，突破了单个 MyISAM 表的限制，并可以分布在多个磁盘上，有效改善 MERGE 表的访问效率。

14. 选择合适的数据类型？
    a) CHAR 和 VARCHAR - MyISAM：建议固定长度的数据列 CHAR。InnoDB：建议使用 VARCHAR 类型。
    b) TEXT 和 BLOB - 尽量选择满足需求的最小的存储类型、大量删除后为提高性能应定期使用 OPTIMIZE TABLE 功能进行标的碎片整理。同时可以使用合成的索引(Synthetic)来提高大文本字段的查询性能，此种索引主要利用散列值，只能用于精确匹配的查询"="。不必要时尽量避免检索 BLOB 和 TEXT。单独将 BLOB 和 TEXT 分成一张表。
    c) 定点 DECIMAL 和浮点 FLOAT\DOUBLE - 定点更精确，字符串存储，而浮点数不精确，而且比较容易出错。
    d) 日期类型选择：尽量选择满足需求的最小存储的日期类型，需要记录年份久远，则 DATETIME。如果需要不同时区使用，则使用 TIMESTAMP。
15. 选择合适的字符集：
    a) 满足应用支持语言的需求、考虑和已有数据的兼容性、汉字居多可以考虑 GBK 2 字节，而 UTF8 3 字节，而英文则应该 UTF8 1 字节，其他 2 字节。
    b) 如果需要比较，排序等字符运算，则最好选择定长字符集。尽量避免字符集转换。
16. 字符集用来 MySQL 存储字符串的方式，校对规则用来定义比较字符串的方式。
    a) 查看方式：show character set; show collation like '%%';
    b) 校对规则_ci 大小写不敏感、_cs 大小写敏感、_bin 基于编码值的比较。
17. MyISAM 和 InnoDB 默认都是 BTREE 索引，目前不支持函数索引，但支持前缀索引。此外，还支持全文索引。默认情况下 MEMORY 使用 HASH 索引，也支持 BTREE 索引。
18. 索引操作：
    a) 创建索引 - CREATE [unique|fulltext|special] INDEX index_name [USING index_type] ON tbl_name (index_col_name, ...)
    b) 修改增加索引 - ALTER TABLE table_name ADD [unique|fulltext|special] INDEX index_name [USING index_type] index_col_name
       index_col_name = col_name[(length)] [ASC|DESC]
    c) 可以通过 explain select 语句查看是否使用了索引。key:xxx 为使用索引。
    d) 删除索引 - DROP INDEX index_name ON tbl_name

19. MyISAM 索引前缀长度 1000 字节，InnoDB 前缀索引长度最长 767 字节。
20. **设计索引的原则：**
    a) 索引列应选择 SELECT WHERE 语句后的列和表连接 ON 子句后的列。
    b) 尽量使用唯一索引，分布越分散效果越好，如性别就不适合索引，分布太密集。
    c) 尽量使用短索引，查询更快，IO 更少。
    d) 利用最左前缀。
    e) 不要过度索引，会降低使用效率。
    f) InnoDB 会根据主键、唯一索引或内部列的排序来存储，所以尽量自己指定主键，选择常用列作为主键。另外，InnoDB 表的普通索引都会保存主键的键值，所以主键要尽可能选择较短的数据类型，可以有效减少索引的磁盘占用，提高索引缓存效果。
21. **HASH 索引只能使用在 = 或<=>等式比较时。优化器不能使用 HASH 索引来加速 ORDER BY 操作。**
    BTREE 索引可以用在>，<，>=，<=，BETWEEN，!=，<>，LIKE "pattern"(pattern 不能以通配符开头)。简而言之，范围查询适用于 BTREE，不适用 HASH 索引。
22. MySQL 使用索引，在检索的时候不需要查找所有数据，能快速定位需要的数据。大多数索引存储在 BTREE 中，只有空间列类型的索引使用 RTREE，MEMORY 表支持 HASH 索引。
23. 为什么需要使用视图？
    a) 使用视图的用户不需要考虑对应表结构，关联条件和筛选条件。
    b) 使用视图可以控制用户访问到行列级别。
    c) 视图结构确定了，可以屏蔽表结构变化对用户的影响。
24. 视图操作：
    a) 创建视图 - CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] VIEW view_name [{column_list}] AS select_statement [WITH [CASCADED | LOCAL] CHECK OPTION]
    b) 修改视图 - ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] VIEW view_name [(column_list)] AS select_statement [WITH [CASCADED | LOCAL] CHECK OPTION]
    c) 删除视图 - DROP VIEW [IF EXISTS] view_name [, view_name] [RESTRICT | CASCADE]

d) 查看视图 - SHOW TABLES

e) 查看视图信息 - SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']

f) 查询视图定义 - SHOW CREATE VIEW view_name

25. MySQL 不允许在创建视图时使用子查询，可以使用查询视图语句来实现。一些视图不可更新：

a) 包含聚合函数(SUM、MIN、MAX、COUNT 等)、DISTINCT、GROUP BY、HAVING、UNION 或者 UNION ALL。

b) 常量视图

c) SELECT 包含子查询

d) JOIN

e) FROM 了一个不能更新的视图

f) WHERE 子句的子查询引用了 FROM 子句的表

26. WITH CASCADED|LOCAL CHECK OPTION 决定了是否允许更新数据使记录不再满足视图的条件，LOCAL 表示只要满足本视图的条件就可以更新，CASCADED 必须满足所有针对该视图的所有视图条件才可以更新。

27. 什么是存储过程？它有什么好处？它和函数有什么区别？

28. 存储过程操作：

a) 创建存储过程 - CREATE PROCEDURE sp_name ([proc_parameter]) [characteristic] routine_body

b) 创建函数 - CREATE FUNCTION sp_name ([func_parameter]) RETURNS type [characteristic] routine_body

proc_parameter = [IN | OUT | INOUT] param_name type

func_parameter = proc_parameter

type = MySQL 任意数据类型

characteristic = LANGUAGE_SQL | [NOT] DETERMINISTIC | {CONTAINS_SQL | NOSQL | READS SQL DATA | MODIFIES SQL DATA} | SQL SECURITY {DEFINER | INVOKER} | COMMENT 'string'

routine_body = 合法 SQL 语句

c) 修改函数或存储过程 - ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]

characteristic = {CONTAINS_SQL | NOSQL | READS SQL DATA | MODIFIES SQL DATA} | SQL SECURITY {DEFINER | INVOKER} | COMMENT 'string'

d) 调用存储过程 - CALL sp_name([parameter])

e) 删除存储过程或函数 - DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name

29. 首先调用 DELEMITER $$将结束符从;改成$$，使得过程和函数中的;不会被解释成语句结束而提示错误。然后从 DELEMITER ;将结束符改回来。

30. 存储过程变量的使用：

a) DECLEAR last_month_start DATE;

b) 变量赋值：SET var_name = expr; SELECT col_name INTO var_name FROM tbl_name....

31. 存储过程可以使用流程控制语句：

a) IF THEN ELSEIF THEN ELSE END IF ...

b) 也可以使用 CASE WHEN THEN WHEN THEN ELSE END CASE。

c) 循环使用：

i. LOOP 语句：begin_label: LOOP statement_list END LOOP end_label。

ii. LEAVE 语句：可以使用 LEAVE 跳出循环(LEAVE begin_label)或 BEGIN END。

iii. ITERATE 语句：同 CONTINUE(ITERATE begin_label)

iv. REPEAT 语句：begin_label: REPEAT statement_list UNTIL search_condition END REPEAT begin_label.

v. WHILE 语句：begin_label: WHILE search_condition DO statement_list END WHILE end_label;

32. 事件调度器：类似于时间触发器，定时执行任务。

a) 创建语法： CREATE EVENT myevent ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR

DO

UPDATE myschema.mytable SET mycol = mycol +1;

b) 事件调度器默认关闭。

c) 禁用或删除事件调度器：ALTER TABLE event_name DISABLE； DROP EVENT event_name;

33. 事件调度器的优势：

a) 避免数据库相关的定时任务部署在操作系统层，减少操作系统管理员产生误操作的风险，并方便迁移，迁移过程包括该数据库。

b) 适用于定期收集统计信息、定期清理历史数据、定期数据检查。

c) 复杂的处理适合程序实现。

34. 触发器是与表有关的数据库对象，会在满足一定条件定义时触发，并执行触发器定义的语句集合。

a) 创建触发器 - CREATE TRIGGER trigger_name trigger_time trigger_event ON tbl_name FOR EACH ROW trigger_stmt。

b) 同一张表相同时间相同事件的触发器，只可以定义一个。

c) 触发器时间可分为 BEFORE 和 AFTER，事件可分为 INSERT UPDATE DELETE 等。

d) 可以使用 INSERT INTO .... ON DEPLICATE KEY UPDATE XXX 来测试触发器执行顺序，如果已存在，则先 insert 在 update after update 。如果不存在，则 insert after insert。

e) 删除触发器 - DROP trigger ins_film;

f) 查看触发器 - show triggers ; information_schema.triggers 表

35. 什么时候使用触发器？

a) **不可以直接调用直接返回客户端数据的存储过程。**

b) **也不能使用开始或结束事务的语句(start transaction commit rollback 语句)。**

c) **不要将过多的逻辑写在触发器中，影响 CURD 效率。**

36. MySQL 支持 MyISAM 和 MEMORY 锁表，对 BDB 进行锁页，对 InnoDB 锁行。默认自动锁表和锁行，但有些时候用户会明确锁表或者进行事务的控制，以确保事务的完整性，这样就需要使用事务控制和锁定语句来完成。

37. LOCK TABLES 可以锁定用于当前线程的表，UNLOCK TABLES 会释放当前线程获得的任何锁定。

a) 锁定语句 - LOCK TABLES tbl_name [AS alias] {READ [LOCAL] | [LOW_PROORITY] WRITE} ...

b) 释放语句 - UNLOCK TABLES

38. MySQL 的事务控制：

a) SET AUTOCOMMIT - 设置是否自动提交，0 表示不自动提交

b) START TRANSACTION/BEGIN - 开始一项新事务

c) COMMIT - 提交事务

d) ROLLBACK - 回滚事务

39. 如果锁表期间如果执行 start transaction 命令，会有隐含 unlock table 执行。lock 方式锁表，不能使用 rollback 进行回滚。

40. **所有的 DDL 语句不能回滚**，并且部分 DDL 语句会造成隐式提交。可以定义 savepoint name 来实现不同阶段的回滚，rollback to savepoint name。

41. MySQL 的分布式事务：

a) 只支持 InnoDB 引擎。

b) 分布式事务还有问题：

i. 如果分布式事务 prepare 时数据库重新启动，重启以后还可以进行提交或回滚，但此时不会写 binlog，会造成主从数据库不一致。

ii. 如果客户端连接异常终止，那么数据库会回滚还未完成的分支事务，如果此时分支事务已经执行了 prepare 状态，其他分支可能已经成功提交， 那么事务会不完整。

iii. 处于 prepare 状态的事务不记录 binlog，如果数据库服务宕机，则会丢失数据。

42. 防止 SQL 注入的几种措施：

a) 使用 PrepareStatement 绑定变量，将输入的单引号转义输入，避免了 SQL 注入。

b) 使用应用程序提供的转换函数。

c) 自定义函数进行校验。(正则)

43. MySQL SQL Mode：REAL_AS_FLOAT、PPES_AS_CONCAT、ANSI_QUOTES、GNORE_SPACE、ANSI。STRICT_TRANS_TABLES 严格模式。

44. 严格模式不允许插入超过定义长度的数据。

45. SQL Mode 常见功能：

a) 校验日期数据的合法性，(ANSI 模式对非法日期会警告，而 TRADITIONAL 模式则直接提示日期非法，拒绝插入)

b) MOD(X, 0)时，TRADITIONAL 模式会抛出警告。

c) NO_BACKSLASH_ESCAPES 会将"\"变成普通字符。

d) 提供 PIPES_AS_CONCAT，使得"||"可以连接字符串。

46. 常用 SQL Mode：

a) ANSI - 更符合标准 SQL

b) STRICT_TRANS_TABLES - 严格模式，可以用在事务表和非事务表，不允许非法日期，不允许超出长度，不正确的值会报错

c) TRADITIONAL - 严格模式，可以应用在事务表和事务表，出现错误立刻回滚。

47. MySQL 分区：根据一定的规则，数据库把一个表分解成多个更小的更容易管理的部分。逻辑上一个表或者一个索引，实际上会有多个分区，每个分区都是独立的对象，可以独立处理，也可以作为表的一部分处理。

48. MySQL 分区的优点：

a) 和单个磁盘或者文件系统分区相比，可以存储更多的数据。
优化查询，where 查询可能只需要查询几个分区，聚合函数时容易并行处理。

b) 对于不需要的数据可以删除有关分区来达到快速删除的目的。

c) 分散数据查询，以获得更大的查询吞吐量。

49. 创建分区表：CREATE TABLE XXX ()ENGINE=INNODB PARTITION BY HASH(MONTH(birth_date)) PARTITIONS 6;

50. MySQL 的分区类型：RANGE 类型(一定范围)、List 类型(枚举出值来分区)、Hash 类型(根据分区个数分配)、key 类型(与 Hash 类似)

Hash 分区键必须是 INT 类型，而其他三种类型分区可以使用其他类型(不算 BLOB 和 TEXT)的列来作为分区键。

51. MySQL 创建分区键，要么表中没有主键和唯一键，否则分区键必须为主键或唯一键。

52. RANGE 分区：PARTITION BY RANGE(key) (

```
PARTITION p0 VALUES LESS THAN (10),
PARTITION p1 VALUES LESS THAN (20),
PARTITION p2 VALUES LESS THAN (30)
```

)

适合场景：方便删除。经常使用分区键查询。

53. LIST 分区：PARTITION BY LIST(key) (

```
PARTITION p0 VALUES IN (3,5),
PARTITION p1 VALUES IN (1, 10),
PARTITION p2 VALUES IN (4, 9)
```

)

更灵活。插入不在分区内的数据会插不进，报错。

54. Columns 分区：PARTITION BY RANGE COLUMNS(a, b) (

    PARTITION p01 VALUES LESS THAN (0, 10),

    PARTITION p02 VALUES LESS THAN (10, 10),

    PARTITION p02 VALUES LESS THAN (10, MAXVALUE),

    PARTITION p02 VALUES LESS THAN (MAXVALUE, MAXVALUE),

)

可以支持整数、日期时间和字符串三大数据类型，支持多列分区。

55. Hash 分区： PARTITION BY [LINEAR] HASH(store_id) PARTITION 4;

    a) **支持两种分区：常规 HASH 分区和线性 HASH 分区，常规 HASH 使用取模算法，线性 HASH 使用 2 的幂运算。**

       比如上述分区，MOD(234，4)=2 所以分到第二个分区中。

    b) **常规 HASH 分区的缺点：增加分区后，HASH 值可能需要重新计算，分区管理很复杂。**

    c) **线性 HASH 分区：当线性 HASH 分区个数是 2 的 N 次幂时，线性 HASH 的分区结果和常规 HASH 分区结果一致。优点是在分区维护时处理的迅速，缺点是线性分区数据分布不太平衡。**

56. KEY 分区：不允许使用自定义表达式，需要 MySQL 服务器提供的 HASH 函数，它可以支持非整数分区。创建分区表时可不指定分区键，默认主键，没有会选择唯一键。也可以创建常规和线性的分区。

57. 可以对已经分区的表在建立子分区。SUBPARTITIONS。

58. 分区表的 NULL 值处理：RANGE 中为最小值，LIST 必须出现在枚举列表中，否则不接受。HASH/KEY 中 NULL 为零值。

59. RANGE&LIST 分区管理：

    a) 删除分区：alter table tbl_name drop partition p2;

    b) 增加分区：alter table tb_name add partition (partition p5 values less than (xxx))

    c) 拆分分区：alter table tbl_name reorganize partition p3 into (xxx);

60. HASH&KEY 分区管理：

    a) 合并(删除)分区：ALTER TABLE tbl_name COALESCE PARTITION p_name;

b) 减少分区数量：先删除再重新定义。COALESCE 不能用作增加。

c) 增加分区：ALTER TABLE tb_name ADD PARTITION partitions 8;(不是增加到 8，而是增加 8 个)

61. SQL 优化的步骤：

a) **通过 show [session | global]status 命令了解各种 SQL 的执行频率**，可以了解当前应用是插入更新为主还是查询操作为主。对于事务型应用，可以通过回滚操作的次数来判断应用编写是否存在问题。

b) **定位执行效率较低的 SQL 语句**：

i. 通过慢查询日志来定位。

ii. 可以使用 show processlist 来查看当前 MySQL 正在执行的线程，包括线程状态，是否锁表等，可以实时查看 SQL 的执行情况。

c) **通过 Explain 分析低效 SQL 的执行计划**：

i. 在 SQL 之前加入 Explain 子句，可以打出 SQL 的执行计划，通过查看计划，可以知道当前的 SQL 是否使用了索引等。

ii. 常见访问类型：从左到右，性能最差到最好：

ALL(全表扫描) < index(索引全扫描) < range(索引范围扫描) < ref(使用非唯一索引扫描或唯一索引前缀扫描，返回一行) < eq_ref(使用唯一索引，返回一行) < const, system(单表中最多有一个匹配行) < NULL(不需要访问表或索引，直接得到结果)

iii. 通过 explain extended 命令加上 SQL 执行后的 show warnings 可以看到 SQL 真正被执行之前，优化器做了哪些 SQL 改写。

d) **如果使用 explain 不能很快定位 SQL 问题，可以选择 profile 联合分析**：

i. Select @@have_profiling 可以查看数据库是否支持 profile，默认关闭，可以 set profiling = 1 开启 profile。

ii. **通过执行 show profiles 可以查看当前运行的 Query，接着 show profile for query query_ID 可以看到执行过程中线程的每个状态和消耗的时间。**

iii. **仔细检查 show profile for query 输出，可以发现时间都消耗在 sending data 状态中。为了清晰看到排序结果，可以查询 information_schema.profiling 表，按 DESC 排序。**

iv. 还可以 show profile (cpu、all、block io、context switch、page faults) for query 4；来查看各个部分占用的时间。

v. 这里对比了 MyISAM 和 InnoDB，发现 InnoDB 多了 Sending data 这个步骤，速度比 MyISAM 慢很多。

e) MySQL5.6 提供了 trace 文件，可以分析优化器如何选择执行计划：

i. 首先打开 trace，设置格式 JSON，设置 trace 最大使用内存大小。

ii. 执行 SQL 语句。

iii. 检查 Information_schema.optimizer_trace 可以找到执行计划的日志。

62. MySQL 的索引专题：

a) 索引的几种实现：

i. B-Tree 索引：最常见的索引类型，大部分引擎都支持 B 树索引。(可以范围查询)

ii. HASH 索引：只有 Memory 引擎支持，使用场景简单。(查询速度快，但不适合范围查询)

iii. R-Tree 索引：空间索引是 MyISAM 的一个特殊索引类型，主要用于地理空间数据类型，通常使用较少。

iv. Full-text 索引：全文索引也是 MyISAM 的一个特殊索引类型，主要用于全文索引，InnoDB 从 MySQL5.6 版本开始提供对全文索引的支持。

b) B-Tree 索引的使用：

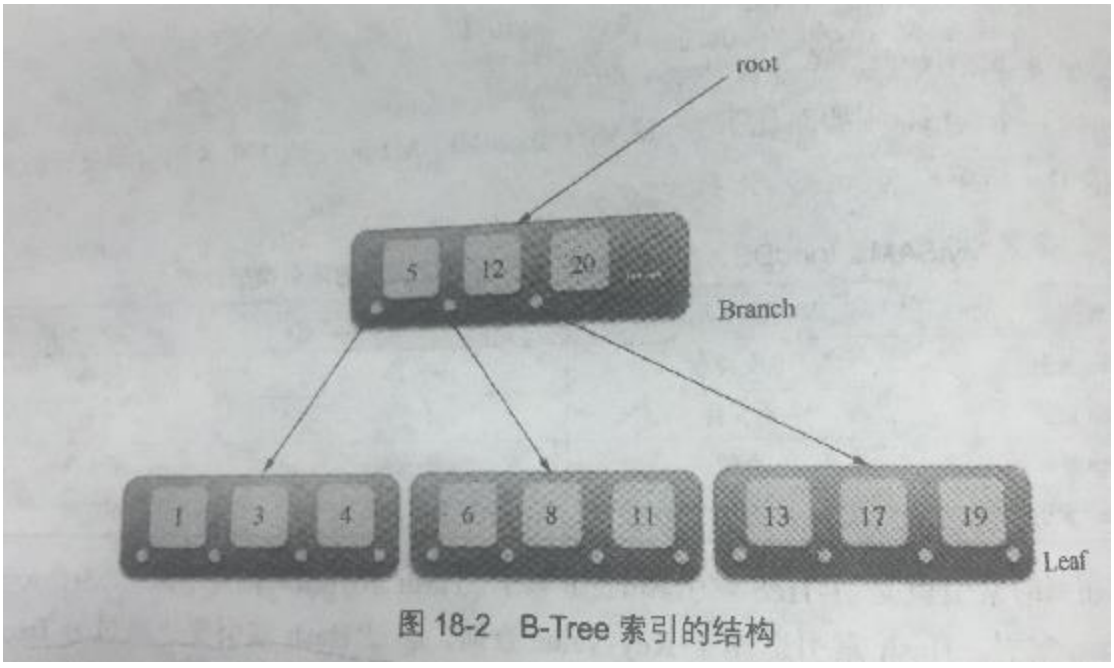

图 18-2 B-Tree 索引的结构

i. 全值匹配：type: const

ii. 匹配值的范围查询：type:range

iii. 匹配最左索引：type:ref。比如三个列形成的联合索引 i1,i2,i3，只有从左到右使用索引进行查询才算使用了索引，否则不使用索引，如 i1 和 i3。

最左匹配原则算是 MySQL 中 B-Tree 索引使用的首要原则。

iv. 仅仅对索引进行查询：type:ref，当查询的列都在索引字段中时，效率更高。(Using Index，仅仅需要访问索引，不需要根据索引访问表数据，它也称为覆盖索引扫描)

v. 匹配列前缀：type:range。使用了前缀索引，并且只查询以 XX 开头的信息，这时 Using where，需要索引回表查询。

vi. 索引匹配精确，其他条件是范围匹配：type:ref。按照索引的精确匹配来选择。

vii. 如果列名是索引，那么 column_name is null 会使用索引。

viii. 如果查询条件中包含索引条件，首先使用该条件进行过滤，接着回表使用普通条件进行过滤查询。这种情况在 MySQL5.6 中进行了提升，使用 ICP 优化降低了不必要的 IO 访问，在回表的过程中就已经排除了不符合条件的记录。

c) 不能使用索引的场合：

i. LIKE 以%开头，不能使用索引。解决办法：先访问索引表，select 索引字段，找到符合 like %xxx%条件的记录，然后再 select 回表查询，省去了全表扫描的 IO 请求。

ii. 数据类型出现隐式转换时不会使用索引。如字符串不加单引号，则不使用索引，使用全表扫描。

iii. 复合索引查询条件不包含最左边部分，不使用复合索引。

iv. 如果 MySQL 估计使用索引比全表扫描更慢，则不使用索引。(如全表查询输出的情况，使用索引浪费了查询索引的时间)

v. 用 or 分割开的条件，如果 or 前的条件中列有索引，而后面的列没有索引，那么涉及的索引都不会被用到。

d) 简单实用的优化方法：

i. 定期分析和检查表，使得 SQL 能够生成正确的执行计划，检查表的作用是检查一个或多个表是否有错误。

ii. 定期优化表：optimize table tb_name，可以清理由于删除或者更新造成的碎片浪费。只对 MyISAM、BDB、InnoDB 有效。

63. 常用的 SQL 优化：

a) 大批量导入数据：

i. 使用 load data infile 'address' into table tb_name，

ii. 在导入之前关闭索引，导入之后打开索引 ALTER TABLE tbl_name DISABLE/ENABLE KEYS

iii. 关闭唯一性校验，set unique_checks = 0;

iv. 关闭自动提交，导入结束后再打开

b) 优化 INSERT 语句：

i. 采用多值插入，values(),(),…

ii. 如果不同客户插入很多行，可以通过 insert delayed 语句得到更高的速度，使得 insert 的数据存放到内存队列中，并不真正写入磁盘。

iii. 将索引文件和数据文件分开存放。

iv. 如果进行批量插入，可以增加 bulk_insert_buffer_size 来提高速度，只对 MyISAM 有效。

v. 使用 load data infile 替换 insert，速度加快 20 倍。

c) 优化 ORDER BY 语句：

i. MySQL 有两种排序方式，第一种通过有序索引顺序扫描直接返回有序的数据，这种情况在 explain 的时候返回 Using index，不需要额外排序。另一种通过返回数据进行排序，通常称为 Filesort 排序，所有不是通过索引直接返回排序结构的排序都叫 Filesort 排序。

ii. Order By 的优化目标是**尽量减少额外的排序，通过索引直接返回有序数据。WHERE 条件和 ORDER BY 条件使用相同的索引，并且 ORDER BY 的顺序和索引顺序相同，并且 ORDER BY 的字段都是升序或降序的，**只有这样才能减少额外的排序。

iii. 对于不可避免的 Filesort 排序，有两种排序算法：

1. 两次扫描算法：根据条件取出排序字段和行指针信息，之后在排序区 sort buffer 中排序。优点是排序时内存开销较少，缺点是第二次读取的随机 IO 比较大。

2. 一次扫描算法：一次性取出满足条件的行的所有字段，然后在排序去 sort buffer 中排序后直接输出结果集。排序时内存开销比较大，但排序效率比两次扫描算法高。

d) 优化 Group By 语句：

i. Group By 默认根据后面的字段进行排序，如果不希望额外的排序，可以使用 ORDER BY NULL 来取消排序。

e) 优化嵌套查询：

i. 使用表连接 JOIN 替代嵌套查询，由于嵌套查询需在内存中创建临时表来完成这个逻辑上需要两个步骤的查询工作。

f) 优化 OR 条件：

i. MySQL 处理 OR 条件，实际上是对每个条件都查询一次，然后用 Union 合并这些查询，所以对每个 OR 条件都需要建索引

g) 优化分页查询：

i. 使用 limit m,n 时，MySQL 实际上会全表扫描，然后获得需要的记录，丢弃不需要的记录，这样查询和排序的代价很高：

ii. 一种方法是表关联，和 limit m,n 的索引进行内连接查询，可以使用索引加快查询，减少查询的行数。

iii. 另一种方法是记录上一次翻页的记录数，然后在查询时使用范围查询，并用 limit n 来查询，这样也不需要全表查询

h) 使用 SQL 提示：显式加入提示达到优化目的：

i. USE INDEX (XX)：添加 USE INDEX 来告诉 MySQL 参考哪个索引列表，可以让 MySQL 不再考虑其他可用的索引。

ii. IGNORE INDEX (XX)：指定需要忽略的索引

iii. FORCE INDEX (XX)：有些时候，MySQL 认为全表扫描会比索引查询更快，这时候可以使用 FORCE INDEX (XX)来强制使用索引查询。

64. 常用 SQL 技巧：

a) 使用正则表达式：

表 18-2 正则表达式中的模式

序 列	序 列 说 明
^	在字符串的开始处进行匹配
$	在字符串的末尾处进行匹配
.	匹配任意单个字符，包括换行符
[...]	匹配出括号内的任意字符
[^...]	匹配不出括号内的任意字符
a*	匹配零个或多个 a（包括空串）
a+	匹配 1 个或多个 a（不包括空串）
a?	匹配 1 个或零个 a
a1|a2	匹配 a1 或 a2
a(m)	匹配 m 个 a
a(m,)	匹配 m 个或更多个 a
a(m,n)	匹配 m 到 n 个 a
a(,n)	匹配 0 到 n 个 a
(...)	将模式元素组成单一元素

i. Select ‘abcde’REGEXP ‘^a’,’e$’，还可以使用”[^abc]”等等。可以避免使用过多的 like 来进行匹配，方便查询。

b) 使用 RAND()来提取随机行：

i. Select * from xx order by rand() limit 5;可以随机获取行数据。

c) 利用 GROUP BY 的 WITH ROLLUP 子句：

i. 可以简单地实现合计，但不能和 order by 一起使用，Limit 子句放在 with rollup 之后。

d) 使用 bit group functions 做统计：

i. Bit_or(字段) … group by 另外字段。可以实现符合聚组条件的条目的指定字段进行位或操作。

ii. Bit_and(字段) … group by xxx。同上，与操作。

iii. 该方法可以简洁的数据表示丰富的信息，节省存储空间。

e) 数据库名、表名的大小写会由不同的操作系统而不同，需要关注 lower_case_tables_name 属性来设置。

f) 使用 MyISAM 存储引擎创建表时，可以使用外键约束语句，只是外键不起作用，在 show create table tb_name 的时候也不显示出来，只是备忘和注释的作用。

65. 优化数据库对象：

a) 优化表的数据类型，尽量选择符合条件的占用空间最小的数据类型。

b) 通过拆分表来提高表的访问效率：

   i. 垂直拆分：一些列不常用时可以拆分，可以使数据行变小，一页可以放更多的信息，查询时可以减少 IO 次数。缺点是需要管理冗余列，查询时需要 JOIN。

   ii. 水平拆分的使用场景：

      1. 表很大，分割后可以降低查询时需要读取的数据和索引的页数，降低了索引层数，提高查询速度

      2. 表中的数据有独立性，比如日期独立，地点独立等。

      3. 需要把数据存放到多个介质上。

      4. 水平拆分会给应用增加复杂度，通常在查询时需要多个表名，查询时所有数据需要 UNION 操作。进行水平拆分时要考虑数据量的增长速度，根据实际情况考虑是否需要对表进行拆分。

c) 逆规范化：（以空间换时间）

   i. 增加冗余列：避免连接查询

   ii. 增加派生列：减少连接操作，避免使用集函数

   iii. 重新组表：经常表连接的表可以组成一个表来减少连接

   iv. 分割表：见上

d) 逆规范化的数据完整性保证：

   i. 批处理维护：修改积累一定时间后运行批处理来对冗余和派生列进行修改。实时性不高。

   ii. 应用逻辑：在应用操作时使用事务来保证一致性。不易于维护，容易遗漏。

   iii. 触发器：对数据库的任何修改立刻出发对复制列和派生列的相应修改，易于维护，实时性高，是一种推荐的方法。

e) 使用中间表提高统计查询速度：

   i. 中间表复制源表的部分数据，并且与源表相隔离，在中间表做查询不会对在线应用产生负面影响。

   ii. 可以灵活地添加索引或增加临时用的新字段，从而达到提高统计查询效率和辅助统计查询的作用。

66. 数据库的锁问题：

a) MySQL 的 3 种锁的特性：

i. 表级锁：开销小，加锁块，不会死锁；锁定粒度大，锁冲突概率最高，并发度最低。MyISAM 和 MEMORY 采用表级锁

ii. 行级锁：开销大，加锁慢，会出现死锁；锁定粒度小，发生锁冲突的概率最低，并发度高。InnoDB 采用行级锁，也支持表级锁，默认行级锁。

iii. 页面锁：开销和加锁时间介于表锁和行锁之间；会出现死锁，锁定粒度介于表锁和行锁之间，并发度一般。BDB 采用页面锁。

b) 表级锁适合以查询为主，只有少量按索引条件更新数据的应用。而行级锁适合有大量按索引条件并发更新少量不同数据，同时又有并发查询的应用，如 OLTP 系统。

c) MyISAM 的表锁：

i. 监控锁：可以使用 show status like ‘table%’来查看是否存在严重的表级锁争用的情况。

ii. 锁模式：MySQL 的表级锁有两种模式：表共享读锁和表独占写锁（同 JAVA 读写锁）

iii. **加锁规则：MyISAM 在执行 SELECT 语句前，会自动给涉及的表加读锁，在执行 UPDATE、DELETE、INSERT 前，会加写锁，不需要手动 LOCK TABLE 命令显式加锁。在自动枷锁的情况下，MyISAM 会获取 SQL 语句锁需要的全部锁，这也是它不会出现死锁的原因。**

iv. 注意：在 lock table 时，如果在查询使用了别名，那么 lock 的时候也必须使用别名 lock。

v. 并发插入：实质上，MyISAM 支持并发插入，即查询时插入到表尾，由 concurrent_insert 系统变量来控制。默认是如果没有空洞，则可以插入。所以，定期执行 optimize table 整理碎片，有利于加快 MyISAM 的查询和插入的并发。

vi. MyISAM 的锁调度：同时申请读锁和写锁，写锁优先获得，而且等待队列中读锁先于写锁，也是写锁优先获取。锁调度需要避免出现饥饿现象，如果写锁总是执行，可以降低写锁的优先级，让读锁能稍微运行。同理，也尽量减少读锁占用的时间，防止写饥饿。

d) InnoDB 锁问题：

i. 监控锁：
   1. 查看 InnoDB_row_lock 状态可以分析系统上行锁争夺情况。
   2. 通过查询 information_schema 表可以了解锁等待情况：select * from innodb_locks \G;
   3. 通过设置 InnoDB Monitors 观察锁冲突情况：Create table innodb_monitor(a int) engine=InnoDB; show engine innodb status \G;

ii. 锁模式：InnoDB 实现了两种类型的锁（共享锁 S 和排他锁 X），另外还实现了两种内部使用的意向锁(表锁)：意向共享锁 IS 和意向排他锁 IX。

iii. **加锁规则：意向锁由 InnoDB 自动加上，不需要用户干预。对于 Update、Delete 和 Insert 语句，InnoDB 会自动给涉及数据集加排他锁 X，对于普通 SELECT，InnoDB 不加锁。可以通过 SQL 中显式获取锁，如 select xxx lock in share mode，但如果 select 获取共享锁，容易导致死锁，推荐获取独占锁，for update。**

iv. **InnoDB 行锁的实现方式：通过给索引上的索引项加锁来实现，如果没有索引，则通过隐藏的聚簇索引对记录加锁。**行锁分三种情况：
   1. 对索引项加锁
   2. 对索引项之间的间隙加锁
   3. 对记录及其前面的间隙加锁

v. 注意：
   1. 因为上述的特性，所以如果 InnoDB 不使用索引来查询时，会导致行锁退化成表锁，降低性能。
   2. 由于针对索引加锁，如果使用相同的索引键查询，也会导致锁冲突。
   3. 当表有多个索引的时候，不同的事务可以使用不同索引锁定不同行，无论是主键索引、唯一索引或普通索引，都会使用行锁来对数据加锁。
   4. 即使再条件中使用索引字段，如果 MySQL 认为全表扫描效率更高，也会采用表锁。

vi. Next-Key 锁：当我们采用范围条件查询而不是相等条件查询时，InnoDB 会把这个条件对应的间隙也锁定，以防止幻读(读入了新插入的数据)，并满足恢复和复制的需要。由于这种方法会阻塞所有符合条件的记录，所以在使用

时，尽量使用相等条件来访问更新数据，避免使用范围条件。另外，如果 InnoDB 使用相等条件请求一个不存在的记录加锁，也会是用 Next-Key 锁。

vii. InnoDB 的恢复和复制的需要：MySQL 通过 BINLOG 记录执行成功的 INSERT、UPDATE、DELETE 等更新数据的 SQL 语句，并由此实现 MySQL 数据库的恢复和主从复制。MySQL5.6 新增了四种复制模式：基于 SQL 语句的复制 SBR，基于行数据的复制 RBR，混合复制，使用全局事务 ID 的复制。

viii. InnoDB 锁的使用：

1. 应用中尽可能降低事务隔离级别，减少锁冲突，可以动态更改事务隔离级别来实现要求高的需求。
2. 在必要的情况使用表锁代替行锁：需要大量更新数据，或事务涉及多个表，可能引起死锁。注意在锁表的解锁时必须先提交，否则 unlock tables 会隐含提交事务。

ix. 关于死锁，死锁的发生和死锁的避免：

1. 在应用中，如果不同程序并发存取多个表，应约定以相同顺序访问表，防止死锁。
2. 在程序以批处理方式处理数据时，若先对数据排序，保证每个线程按固定顺序来处理记录，也可以降低出现死锁的可能。
3. 更新记录尽量选用排他锁。
4. 两个事务隔离级别 Repeatable Read 的前提下同时查询不存在数据，都能加锁成功，造成死锁，可以降低隔离级别来解决。
5. 如果出现死锁，使用 show innodb status 命令来确定最后一个死锁产生的原因。

67. 事务的特性？

a) **原子性(Atomicity)：事务是一个原子操作单元，对数据的修改要么都执行，要么都不执行，不可拆分。**

b) **一致性(Consistent)：事务开始和完成，数据都必须保持一致状态。事务开始和结束时，所有内部数据结构都是正确的。**

c) **隔离性(Isolation)：数据库系统提供一定的隔离机制，保证事务在不受外部并发操作影响的独立环境执行。这意味着事务处理过程中的中间状态对外部不可见。**

d) 持久性(Durable)：事务完成后，对数据的修改时永久的，即使出现系统故障也能保持。

68. 并发事务处理带来的问题？

a) 更新丢失：最后的更新覆盖了由其他事务所做的更新。

b) 脏读：事务读到了其他事务未提交的数据。

c) 不可重复读：一个事务某时间先读了某数据，再次读取发现读出的数据已经发生改变。(读到其他事务已提交的数据)

d) 幻读：一个事务读到了其他事务插入的新数据。

69. 事务的隔离级别的实现方式？MySQL 的 4 种隔离级别？

a) 实现方式：

i. 在读取数据前，加锁，阻止其他事务对数据进行修改。

ii. 通过一定机制生成一个数据请求时间点的一致性数据快照，并用这个快债提供一定级别的一致性读取。这种技术叫数据多版本并发控制 MVCC，也常称为多版本数据库。

b) 隔离级别：

i. Read uncommitted 未提交读：最低级别，只保证不读取物理上损坏的数据。

ii. Read committed 已提交读：语句级，可以防止脏读。

iii. Repeatable read 可重复读：事务级，可以防止脏读和不可重复读。

iv. Serializable 可序列化：最高级别，事务级，可以防止脏读、不可重复读和幻读。

c) Oracle 只有 Read committed 和 Serializable 两种，还有自定义的 Read Only。

d) Sql Server 除了 4 种以外，还有快照隔离级别。

70. MySQL 的缓存机制？

a) InnoDB 缓存：

i. InnoDB 用一块内存区做 IO 缓存池，该缓存池不仅用来缓存 InnoDB 的索引块，而且也用来缓存 InnoDB 的数据块，这一点与 MyISAM 不同。

ii. 内部来看，InnoDB 缓存池逻辑上由 free list、flush list 和 LRU list 组成。分别为空闲缓存块列表、需要刷新到磁盘的缓存块列表、InnoDB 正在使用的缓存块，LRU list 是 InnoDB buffer pool 的核心。

iii. 由于 Innodb_buffer_pool_size 决定 InnoDB 存储引擎表数据和索引数据的最大缓存区大小，越大则缓冲命中率越高，一般设置 80%的物理内存。

b) InnoDB 的 doublewrite 机制：MySQL 的数据页 16KB，操作系统 IO 数据页 4KB。这里会导致断写或部分写。简单来说，就是写两份，一份到 IO 一份到 buffer，如果 redo 发现不一致的页，则用 Buffer 来替换。

71. MySQL 应用层面的优化：

a) 使用数据库连接池：大大减少了创建新连接锁耗费的资源，连接返回后，本次访问的连接还要还给连接池。

b) 避免对统一数据做重复检索：尽量减少 SQL 查询的次数，尽量在一句 SQL 完成所有功能。

c) 使用查询缓存：适用于对象更新不频繁的表，当表更改后，查询缓存的相关条目会被清空。

d) 增加 CACHE 层：在应用端增加 CACHE 层来减轻数据库负担的目的。（CACHE 如何刷新？多久刷新？）

e) 负载均衡：它的机制就是利用某种均衡算法，将固定的负载量分布到不同的服务器上，以此来减轻单台服务器的负载，达到优化的目的。数据库层常用做法：

i. 利用 MySQL 复制分流查询操作(主从复制)：主要问题是主数据库更新频繁或者网络出现问题的时候，主从之间的数据可能存在比较大的延迟更新，从而造成查询结果和主数据库上有差异。(一致性)

ii. 采用分布式数据库架构：适合大数据量、负载高的情况，具有良好的可扩展性和高可用性。局限是分布式事务只支持 InnoDB 引擎。