问题一：在聚簇索引上执行查询，尤其是范围查询时，它的执行效率会非常高。请问， 为什么聚簇索引的效率高?

- 聚簇索引：索引的顺序和数据存储的顺序完全一致的索引叫做聚簇索引。InnoDB中，表数据文件本身就是按B+Tree组织的一个索引结构，聚簇索引一般就是按照每张表的主键构造一颗B+树，同时叶子节点中存放的就是整张表的行记录数据，也将聚集索引的叶子节点称为数据页。
- 为什么快：索引记录的位置和存储的位置在顺序上是一样的，这样找起来就非常快。例如：某个节点有多个子节点，从第一个节点到最后一个节点排出来顺序是依次递增的时候，这样显然在查找一个范围的时候就非常快，可以按照顺序来读取磁盘。反之如果存储的地址顺序是乱的，在查找一个范围的时候就不能一次读取一片区域出来，效率就会变低，要读取多次。

问题二：如果要将你的图书封面以 Base64 的形式存储在 MySQL 中，那么你在数据表的设计中，是以 VARCHAR 类型来存储，还是以 LONGBLOB 形式存储？请详细解释你的理由。

使用LONGBLOB来存储图片。因为图片的Base64是一个很长的文本信息，如果使用VARCHAR是直接存在表里面的，但是LONGBLOB是存的一个指针，指向一块区域，然后在这个区域里面存文本。当我们要进行查询的时候，如果是用varchar存储图片，会导致每个Page读取到的条目数量大大减小，查询效率降低，反之如果使用了LONGBLOB，由于表里面存的是一个指针，所以一个Page里面可以包括更多的书的信息，查询效率更高。

问题五：当你的图书表上建立复合索引以加速涉及多个字段的查询时，请用 SQL 语句给 出你建立该复合索引的方式，并详细解释你为什么构建这样的索引，包括字段 顺序、升降序排序等因素是如何确定的。

这个需要视情况而定，一般来说，重复率最高的放在最前面，按照我们上课讲的例子，如果要对姓名建立索引，肯定姓在前面，名在后面，因为姓重复概率更大。

我们在图书表建索引的时候，需要考虑的列有书名、出版商、作者、还有价格（比如有人要搜索价格区间），如果这个书店是那种大型书店，卖各种书的，那出版商重复的概率是最大的，然后就是作者，然后就是书名，我们假设是这种情况。

由于前面那些用户搜索功能用的都是全文搜索之类实现的，不会用到这个索引，所以也就是说管理员在管理书籍的时候才会使用到我们这个建立的索引。

- 比如管理员要统计某个时间段发布的书籍，由于我们习惯把最新发布的书籍放在最上面，所以用降序排列更符合习惯，也会提高效率
- 比如管理员要统计某个出版商的书籍
- 比如管理员要统计某个出版商的，某个作家写的书籍
- 比如管理员要统计某个出版商的，某个作家写的在某个价格区间的书籍内容
- 由于一般查询价格的时候，都习惯于从低价往高价开始，所以在索引里面暂定设计为增加。
- 由于模糊匹配`%content%`是不会使用索引的，而我一些查询书名的是用的SQL的`like`，比如书籍的展示标题，这时候就没有必要建立索引了，而且这部分的搜索应该是在elasticSearch里面完成的。
- 所以综上来说，这个索引的建立比较适合区间查找（比如说我要查找某个价格范围的书，某个时间段的书，就很合适，但是显然是不适合模糊查找的）

```
1  CREATE TABLE `books` (
2    `ID` int unsigned NOT NULL AUTO_INCREMENT COMMENT '主键',
3    `ISBN` varchar(255) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAULT
   NULL COMMENT '书籍的ISBN, 978-7-000-00000-0',
4    `bookname` varchar(255) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAULT
   NULL COMMENT '书籍的名字（简写版本）例如《程序设计》',
```

```
  `displaytitle` varchar(255) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci NOT NULL COMMENT '书籍的名字（详细版本，用于关键词检索）',
  `inventory` int unsigned DEFAULT '0' COMMENT '书籍的库存数量',
  `departure` varchar(255) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAULT NULL COMMENT '书籍的库存地点',
  `author` varchar(100) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAULT NULL COMMENT '书籍的作者，33个汉字可以存储，稍微做了一定的限制',
  `price` int DEFAULT '0' COMMENT '书籍的价格',
  `sellnumber` int unsigned DEFAULT '0' COMMENT ' \r\n书籍的销量，默认最大值为：2147483647，已经够用了',
  `imgtitle` varchar(255) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAULT NULL COMMENT '书籍的图片，暂且先存文件的地址',
  `publisher` varchar(255) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAULT NULL COMMENT '书籍的出版商',
  `description` varchar(3000) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAULT NULL COMMENT '书本的描述字符',
  `bookid` int DEFAULT NULL,
  `create_itemtime` datetime DEFAULT NULL COMMENT '该书籍项目的创建时间',
  PRIMARY KEY (`ID`) USING BTREE,
  INDEX create_itemtime DEC,
  INDEX basicInfo(publisher, author, bookname, price ASC),
) ENGINE=InnoDB AUTO_INCREMENT=25 DEFAULT CHARSET=utf8mb3 ROW_FORMAT=DYNAMIC;
```

问题四：你认为你的订单表的主键使用自增主键还是 UUID 好？为什么

自增主键好。

1. 考虑唯一性的管理：如果不好管理，最好交给数据库自增管理，如果自己可以管理好唯一性，就可以使用UUID。自增主键不需要考虑
2. UUID很大，占用的空间需要16个字节，由于他的大小还有无序性，可能引起性能问题。
3. UUID之间比较大小相对数字慢不少， 影响查询速度。
4. 但是考虑到安全的时候，UUID往往更加安全，因为如果订单按照1、2、3的ID，如果没有鉴权机制，很容易导致订单的数据被泄露（例如有人恶意爬网站的订单数据），但是我的后端在针对订单查询的时候，有登录和鉴权的机制，所以相对来说比较安全，即使是知道订单的编号，没有相关的权限依然无法查询。

问题五：请你搜索参考文献，总结 InnoDB 和 MyISAM 两种存储引擎的主要差异。

MyISAM和InnoDB两者之间具体区别如下：

- 事务方面：MyISAM不支持事务，而InnoDB支持事务。
  - InnoDB的AUTOCOMMIT默认打开的，即每条SQL语句会默认被封装成一个事务自动提交，这样会影响速度。
  - MyISAM是非事务安全型的，而InnoDB是事务安全型的，默认开启自动提交，宜合并事务，一同提交，减小数据库多次提交导致的开销，大大提高性能。
- 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。
- 存储结构：
    - MyISAM：可被压缩，存储空间较小。支持三种不同的存储格式：静态表(默认，但是注意数据末尾不能有空格，会被去掉)、动态表、压缩表。
    - InnoDB：需要更多的内存和存储，它会在主内存中建立其专用的缓冲池用于高速缓冲数据和索引。
- 事务与恢复机制、容灾
    - 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。
- 全文搜索：
    - MyISAM：支持(FULLTEXT类型的)全文索引
    - InnoDB：不支持(FULLTEXT类型的)全文索引，但是innodb可以使用sphinx插件支持全文索引，并且效果更好。
- 外键：MyISAM：不支持，InnoDB：支持
- 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。InnoDB的主键范围更大，最大是MyISAM的2倍。