Hbase确确实实是一个实实在在的数据库，并且有多个机器组成一个集群的方式进行提供数据的服务，然后肯定还有很多人就是之前对这个hive和hbase分不清，这个可以提前大家先知道一下就是hive这个东西它就是一个mapreduce，你就这么简单的认为，那么hbase它是实实在在的一个数据库，那既然数据库它肯定要存数据的是吧？就像我们在传统的数据库中比较熟悉常见的就是mysql和oracle对吧，那这些数据库其实就可以存数据了，你可以认为在里面插入数据也可以删数据也可以去读对吧？

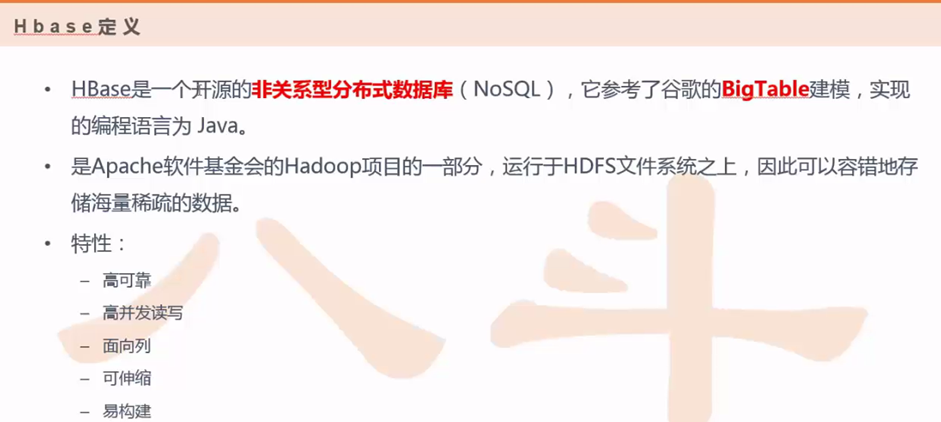
那么Hbase也是这样做的，那Hbase的数据是存在哪里的呢？Hbase大部分的数据都是存在HDFS上这是没问题的，但是还有少量的数据是存在内存的，这个大家要记住，它并不是把所有的数据都存在HDFS上。



然后整个的Hbase我们是第一部分是讲理论，第二部分是实操，实操部分我这里就不写了。

那么我们来看一下Hbase的定义（如下图1所示）

图1：



Hbase是一个开源的然后非关系分布式数据库，那么关系型数据库就是通常的mysql和oracle，那么非关系型数据库通常就叫做Nosql，然后Hbase你可以认为是参考了谷歌这个BigTable它的相关的论文然后相关的实现做了一个开源版本，然后并且是通过java去开发的。

然后属于Apache基金会一个很重要的一部分，运行在HDFS文件系统之上，为什么呢？因为它的数据也是存在HDFS上的，并且可以容错的存储海量稀疏数据，那什么是稀疏数据呢？

（如下图2所示）

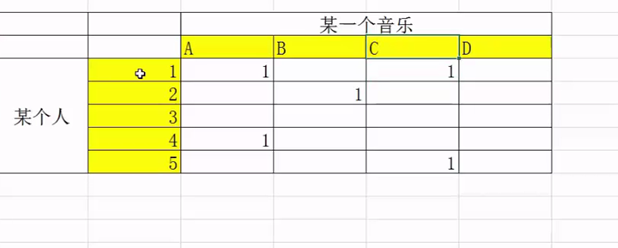
图2：



行是某一个音乐，列是某个人，然后如果是1这个人他比如是听过A这个音乐的话他打一分，B他没有听过就不管，C他听过打一分，然后某个人只要是听某个音乐就给他打一分

（如下图3所示）

图3：

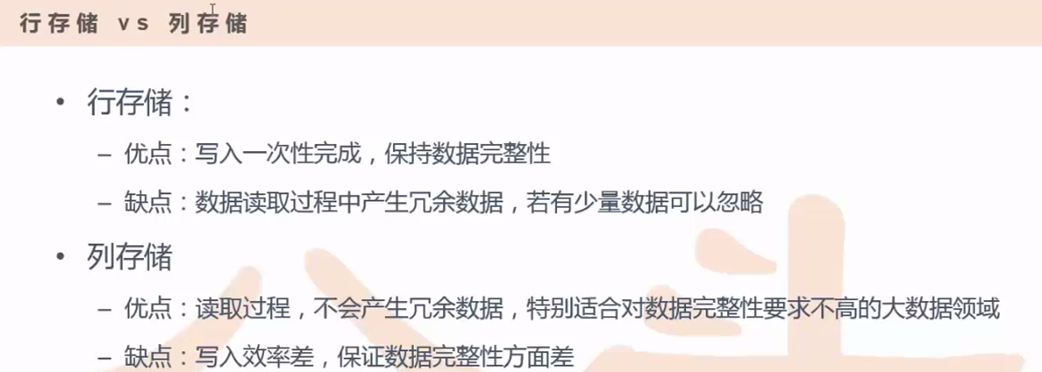
这样的话你会发现这么一个数据它永远不会填满了是吧？，因为不可能所有的人都会把所有的音乐都听完，这个应该大家有共识的对吧？所以这样的数据总会是一个就是有值的这个字段很少的，所以这就相当于一个稀疏的数据，就是说它允许有很多并且存在大量的字段是空的，所以这就是稀疏的。

所以为什么要稀疏呢？海量稀疏就是说你不能够保证你一个key它的列就一定有数据，像我们之前操作mysql或者其他的数据库，那你对某一条的记录它上面的所有列你都要给它赋一个值，即使当前这个列它没有值的话你也必须给它一个默认。

那么HBase像这种特性有很多，比如高可靠，高并发读写，面向列，可伸缩，易构建这些特性不仅仅HBase具有的，所以这特性都是比较通用的，但是唯独有一个面向列，面向列这个可能需要我们再去多说一下。

那对于数据库的存储来说它分两种，一个行行存储，一个是列存储（如下图4所示）

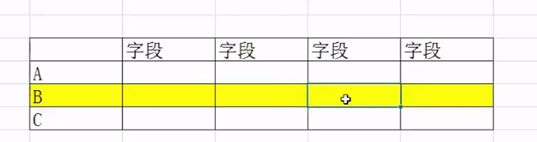
图4：



那什么是行存储呢？就相当于我们之前搞的那个mysql，像这些传统的关系型数据库它都是行存储的，像刚才提到的那个Nosql的一些数据库基本上都是这种列存储。

那为什么存储分行分列呢？，那么我们先看一下什么是行存储，行存储优点就是写入的时候一次性要完成，并且它是保证你数据的完整性，比如说我们这有一张表（如下图5所示）

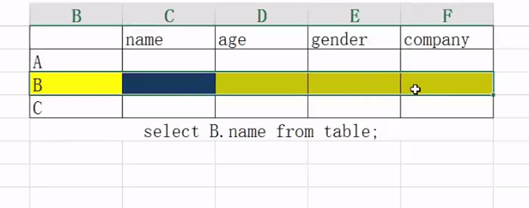
图5：



比如说上图5的每一行都是一条记录，你在去插这个记录的时候你是不是要求你每一个字段都不能是空的对吧？你写数据的时候要求你每一个字段都是完整的，要求你写入这个数据，一旦这个数据写到你的数据库里面来的话，你这个记录是包含所有字段的，你读任何一个字段都可以读到相应的字段的数据，然后因为有这么一个特点，那它是可以保证你数据的完整性的并且它要求的是你写的时候必须把所有的字段都一次性的完成，你不能说你第一步你先写这个字段第二步再写这个字段不能这么搞的，如果即使你写了这么一个字段的话，那相当于其他字段都会给你默认赋一个值明白这个意思把？

优点就是它的数据是完整性的，当然缺点就是因为它数据的完整所以你再去读取数据的时候你会产生很多这种冗余的数据，比如说你要求select B name From table 就这么一条语句，虽然你只把这个B这个name这个字段你取出来了，你只取了name这么一个字段，但是其他相同行的字段的数据虽然都没有读，其实它在内部读的时候它也是把整条记录读进来的（如下图6所示）

图6：

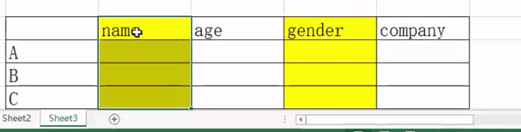


只不过是在最后给你呈现的时候，是因为考虑到你只读取到一个name字段，于是把这三个字段会给你过滤掉明白了把？但是你虽然看到了是name字段，但不代表它没有去读所有的字段的数据，大家有没有理解？所以它内部来说它是来产生这种冗余数据，所以你会发现一个特点就是这个行存储的优点就是写，那么缺点的话就很明显就是你读取的时候你会产生很多的一些你不需要读的数据它也会给你读出来，但只不过是没有展示出来而已。

那么我们再来看看列存储是什么意思？我们刚刚举得例子是按行来，这是我们经常所理解的那个数据库形式是这个样子，那么列存储是什么意思？

列存储就是说它这个字段是以列的形式进行存储的（如下图7所示）

图7：



那么列存储有什么好处呢？他的最大的好处就是行存储的缺点，比如说你同样在列存储里面输入命令select B name From table，那么这个时候直接把列存储的name直接取出来就好了，然后这三个字段你根本就不需要去查，明白这意思了把？

那它的优点就是读取过程，它这种是不会产生冗余数据的，特别适合对那些数据完整性要求不高的大数据领域，因为在大数据的领域读数据，就是一旦你写入数据，那剩下的工作就是读了，就是来做数据分析，所以在大数据领域来说通常面对的情况就是读。

缺点就是写入的效率差，相当于你这行存储的优点就变成了列存储的缺点，行存储的缺点变成列存储的优点，那它为什么写入的效率差呢？因为它不像行存储那样你写完一条记录之后，它可以保证你数据完整性，你在列存储的数据库里面它是没有办法保证你的数据的完整性的。

所以这个时候就有疑问了，就是说你读一列数据的时候，就相当于把一列数据读进来，其实不是这个样的，然后我们再搞mapreduce的时候，我们通常面对数据是key和value，就相当于我们之前做的word count，前面是word 的后面是count。就是我们之前在操作mapreduce的时候前面第一列通常是作为key后面是作为它的value对不对？

那其实假设说它分了四列（如下图8所示）

图8：



那相当于列存储它就是分了四个不同的文件，每一个文件它的存储是这样存的，比如说你列字段里面都有值了，我们只做A和B和C和name和zhangsan和list做例子，其中第一个文件应该是这么存的： A \tzhangsan，B \t lisi，因为c没有name，所以就不存了。

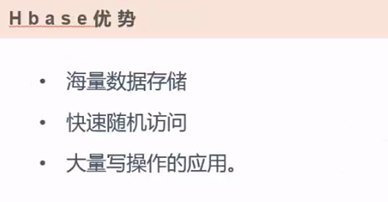
那么第二个文件存的是：A \t 20 和C \t 30，然后第三个文件和第四个文件我就不写了，所以这两个文件的具体内容可以看出一些特点，就是说当你select name 的时候，那它只去查第一个列的文件的内容就可以了，然后所以这剩下的三个文件你是不需要动的，所以这样的话这剩下的三个文件里面的一些字段你根本不需要读，因为你这个select name的时候你根本就不需要其他的字段是吧？所以它就不会产生冗余数据。

然后因为在Nosql里面它不是要求你所有的字段都是要有值的，因为是稀疏存储，稀疏存储的话是不要求你的数据的完整性的，那么像图8中的数据是缺失的，那一旦缺失的话我这边是允许的，那如果说允许缺少字段的话，那么在文件里面是根本就不会留给这个空间做存储明白了把？就不会给这个没有值的这些字段做预留的一些存储空间，大概是这样的一个样子。

所以你select B name的时候，相当于是把A \tzhangsan，B \t lisi这里面的值一起查出来就可以了，那你觉得是这样的方式，对文件直接把这个数据直接定位到某一个字段，它属于具体的哪一个文件，然后把A \tzhangsan，B \t lisi这里面的数据直接给它导出来是不是要比你这个行存储要去导某一列的数据要容易很多？这个大家理解了吗这个？

然后我们再看一下HBase还有很多的优势（如下图10所示）

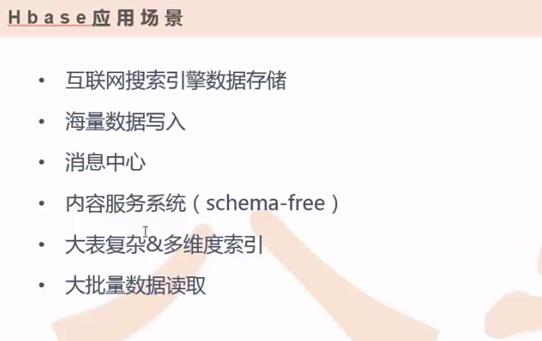
图10:



这个大家看一下了解一下就可以了，这个海量数据存储就不用多说了，因为它都存在HDFS上的，那么第二个也很重要，就我们开头的时候就已经说了它是可以支持随机的写入和访问的，因为像这个HDFS里面这个数据通常是顺序读的，原因就是吞吐能力，那所以HDFS是不适合你的随机访问的，但是你的Hbase它是可以随机访问的，所以大量的写，因为在HDFS里面你有大量的写的话你是可以通过追加的方式也可以满足这么大的吞吐的场景是吧？

然后我们看一下HBase应用场景，其实一个公司里面，HBase应用场景还是很广泛的（如下图11所示）

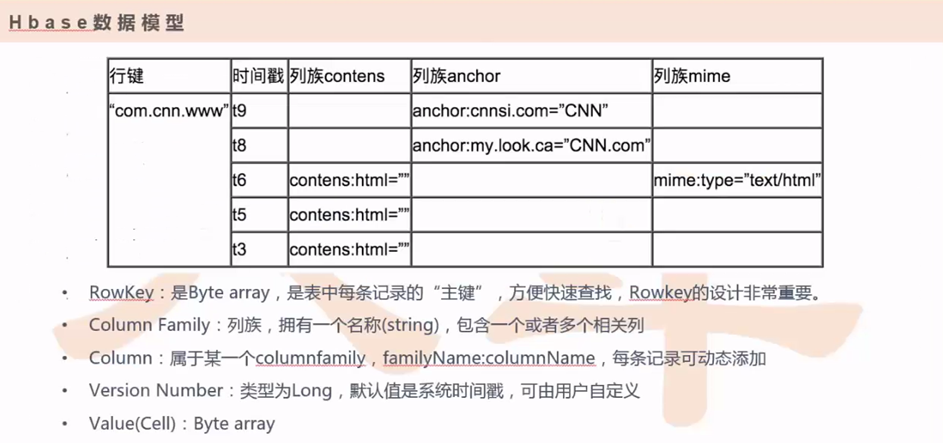
图11：



那通常呢就是把一些非结构化的数据然后通过一些转行然后变成一些结构化的一些信息，然后通过Hbase这么一个数据库然后进行结构化维护起来是这么一个情况。

那么我们先看一下这个数据模型(如下图12所示)

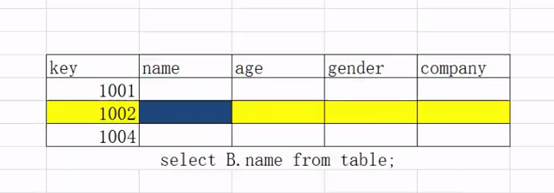
图12：



这一块的话是很重要的，搞一些传统数据表格的时候都是一个二维图，然后某一行然后每一个字段然后字段与字段之间彼此之间是一个平行关系，但是在Hbase里面就不一样了，Hbase里面它是允许有一个行，这个行里面的话它又划分出多个列，然后这列与列之间它也是有这种层级关系的，有这么一个行键叫做Rowkey，通过在这个HBase里面这个RowKey它都是这种字节形式去存储的，那就是一个主键。

像我们在传统数据库里面也是，你通过你的主键去select一些数据，通过一个key去主键一个数据，那在Hbase里面你想建一个数据的唯一的一个key就是它的一个RowKey就是它的主键，然后为了方便它的快速查找，所以你RowKey的设计也是非常重要的，这个RowKey因为它是这种字节的方式去存储的，传统的数据库里面的话，因为你每条记录它的前面的这个key都是不一样的，那在Hbase里面存的这个key，它这个key前后之间是按照顺序存的，就是一旦说这个字节的话它相当于是在一个表格里面每一个key它是按照字节去排好的，比如说前面是0001，后面的话你就可以1004，但是后面不允许出现1002，如果你要出现1002的话你必须把这个1002排到1004的前面（如下图13所示），所以它这个在Hbase里面它前面这个rowkey一定是按照这种字节去排序的。

图13：

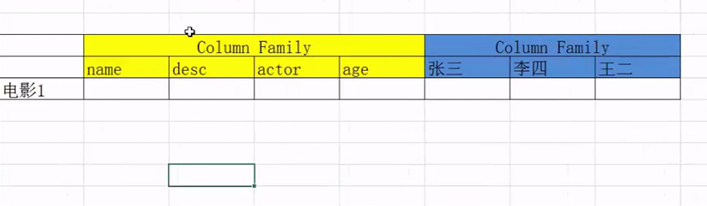


为什么要按照字节排序？这个二分法之前我们已经学过了，那用二分法通常的目的是干嘛？就是为了检索对吧？就是因为你这个key一旦排好序之后你可以快速的帮你定位到相应的记录，所以rowkey在整个的hbase表里面一定是按照你的字节去排序的，为了方便快速查询。

那接下来相对来说比较复杂一点的话就是Column Family，什么是Column Family呢？每一条记录里面或者是在你去建Hbase的Schema设计的时候要指定好Column Family，那什么是Column Family呢？你可以认为就是一个列族，但是它这个列你可以在列族里面又可以细分出多个列出来，就是Column Family下面可以细分出多个子列，那这些子列称为Column qualifiar。

然后一个Column Family里面包含多个Column qualifiar。那么这Column Family和Column qualifiar有什么关系呢？假设说我们举个例子（如下图14）

图14：

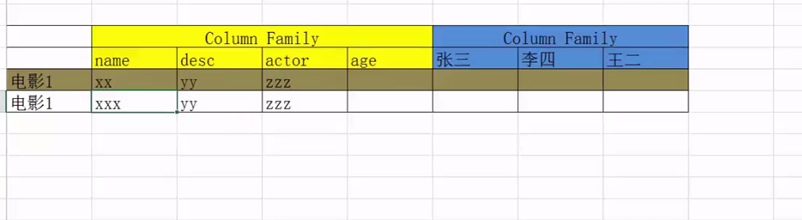


你会发现前面这四个属性和后面这三个属性完全不是一个类型对吧？，你可以把前前面三个叫做一个Column Family，后面也是Column Family，然后每一个Column Family代表它的内部的那个属性都是不一样的，然后因为这第一个Column Family下面的所有属性，你可以认为都属于当前这个电影1的一个原数据，但是后面的第二个Column Family他却面向另外一个类型了，就是说这个电影1被哪些人看过，跟这个电影1的名字描述完全都不是一个同类的一个属性，然后当然后面你可以再建多个Column Family，所以这么一个结构，所以这第一个大列可以认为是一个Column Family，这里面子列叫做Column qualifiar。

所以Column Family列族拥有一个名字，然后包含一个或者多个相关列。

假设这个电影1这个名字，你一开始这个数据库存的这个电影1的名字是一个错误的名字，那你想把它改掉，好了这个改名字这个时候就不应该在它原来的基础之上进行修改，它相当于在帮你去重新复制一份记录出来，然后再复制出来的记录上去修改名字（如下图15所示）

图15：



然后原来这个名字这个记录，它会以某一种版本的形式给你记录起来明白了把？就是通过一个时间戳给你记录起来，就是说如果这个名字被你改过的话，这个select 这个名字的时候就是读的是已经修改好的xxx名，那你想看一下这个名字之前的一个名字，就是还没被修改的名字叫什么？那么这个也可以读出来明白把？那当然这个任何一条记录的话你都可以像这种方式可以维护多个版本，就像这个下图16里面的

图16：

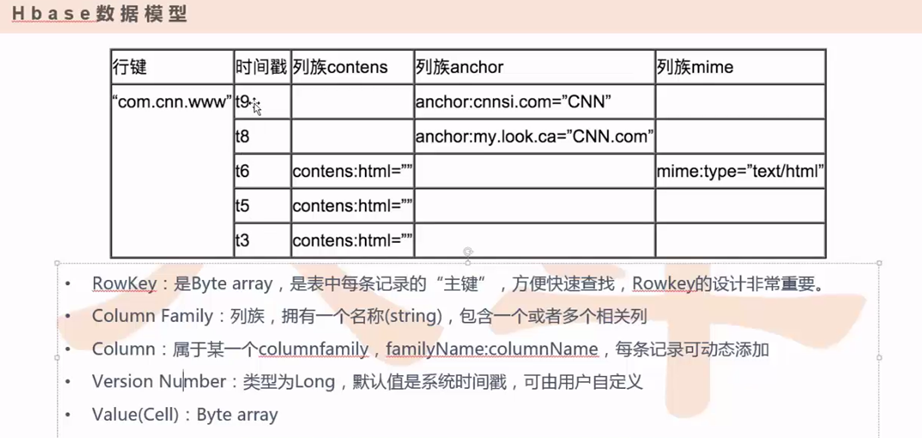
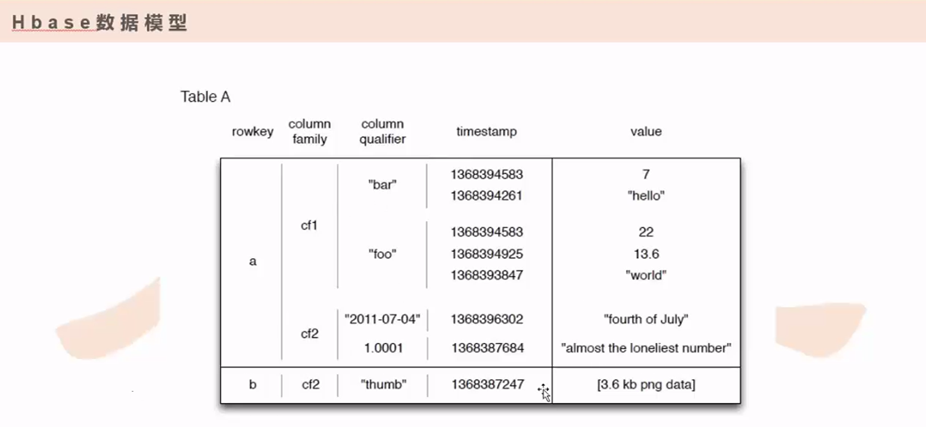


图16中这有时间戳，然后每一个时间戳的背后都是代表一个版本，那这个时间戳的话具体存了多少版本这是通过配置可以配的，这就是时间戳的概念。

然后另外还有一个这个Value（cell），cell就相当于图16这里面的列族contens具体的一个字段，像我们这个Column Family在这个图16中，Column Family叫做Contens是吧？那这个Contens这个列族，那这个列族里面包含了一些Column qualifiar就是这些子列，那你就可以在这个列族的后面加一个冒号：后面的名字就叫做一个子列的名字，然后这个子列的名字你可以随便去定义明白了把？

好了这是另外一个例子，你像这个下图17这个列族配有两个一个A和B，你看这个A它包含了很多的一些信息，B就虽然包含了一条信息（如下图17所示）

图17：



但是你真要去读的时候，相当于你只是对，就是你默认情况下就把默认的这种最新的时间戳它这个记录读出来就可以了，它不会把所有的信息都给你展现出来，这a有两个列族,CF1和CF2，但是B只有一个列族那就是cf2，然后a这个columnqualifier它这里面是包含了很多的一些列的名字，但是b里面就包含了一个，那这个columnqualifier这个设计就可以允许，就是hbase设计表的时候就非常的灵活。

那我们继续往下说，为了让整个Hbase你的检索能够更快速并且你的数据的维护更合理，那它这里面会体现了一个三维有序的这么一个特点（如下图18所示）

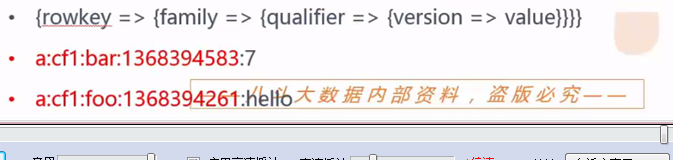
图18：



那什么叫做三维有序呢？就是三个纬度，那第一个纬度就是你的rowkey，这个刚才我们说过了，rowkey就是你的第一个端口是吧？你要查询某一条记录，你这个记录是key，这个key是它的首要的一个主键，你必须要知道你这条记录前面的这个key你才能查到相应里面的数据对不对？，所以这个rowkey是一个第一个很重要的信息。

第二个就是它的列，它的列包含了两个部分，一个是columnqualifier和一个是Column Family，那么这第一个和第二个加起来是一个列明白了吧？就是你要定位到某个hbase的某一列的时候你肯定先要指定他的Column Family然后再指定它的columnqualifier对吧？如果你只指定Column Family，因为它这个Column Family底下有很多的子列，你还要进一步的去指定这个cf1下面的具体的一个子列的一个名字，这才能够确定它具体是哪个列，那所以这两个信息要组成第二维，就是它的列里的维度，然后第三维就是他的version hello，version hello就是它的时间戳，就像下面举得这个例子一样，在这里面在这个a就是它的rowkey，Column Family就是它的ch1，这个bar是他的一个子列，后面是一个时间戳，然后最后一列就是它具体的一个value具体一个值是不是？，那这种存储的时候，是把除了value以后前面的三个部分去做一个排序。（如下图19所示）

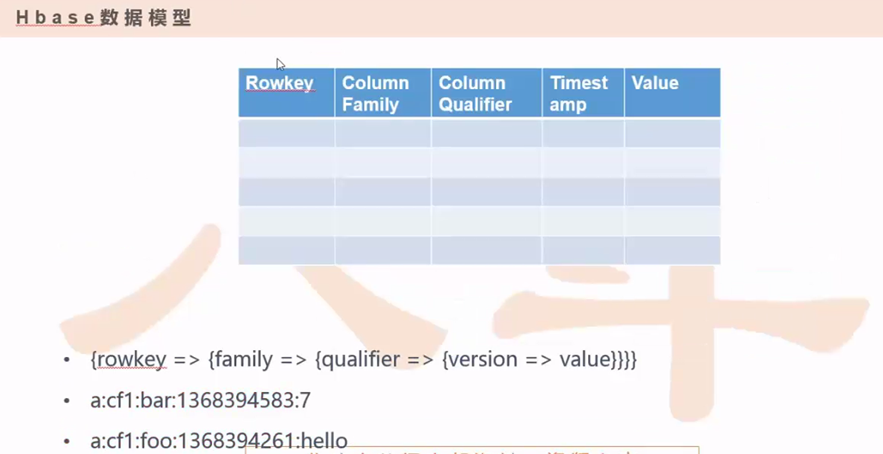
图19：



假设说我们这个如果你按照三维有序的概念的话大家可能会有疑问，有什么疑问呢？比如说你这个rowkey和你这个列都长的是一样的，那你这个时间戳不一样，那这个时间戳谁排前谁排后，时间戳这个数字大的排前面，因为最新的时间戳都是大的，那么最大的应该放在前面，应该最先检索对吧？所以就是这样前两列的是按照你的字节去的一个顺序排序的，然后时间戳这一列是按照你的倒序排序的。

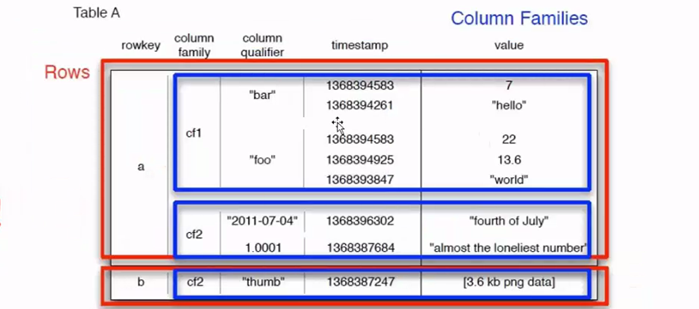
然后这个是一个如果是想把这个Hbase这样一个新型的一个数据的表达的方式呢，按照传统的方式去进行表达，因为我们之前学过mysql，然后去搞一些传统数据库这个，然后去搞这些传统数据库这些可能会比较多，你既然面对这样的一个二维表格（如下图20），大家接受起来就比较容易是吧？

图20：



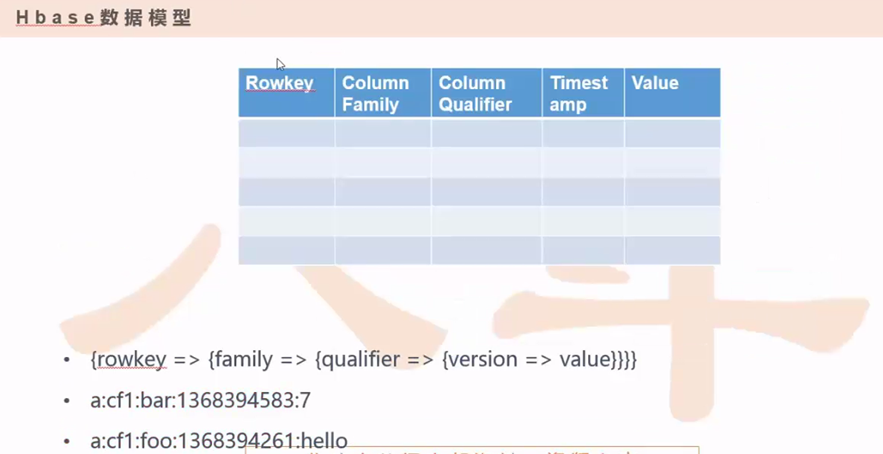
但是如果是把这些今天学的Hbase这种方式的话（如下图21）

图21：



你再去重新接受一遍的话可能有一些不太容易接受是吧？，那么我们还原成这种二维的方式（如下图22）

图22：

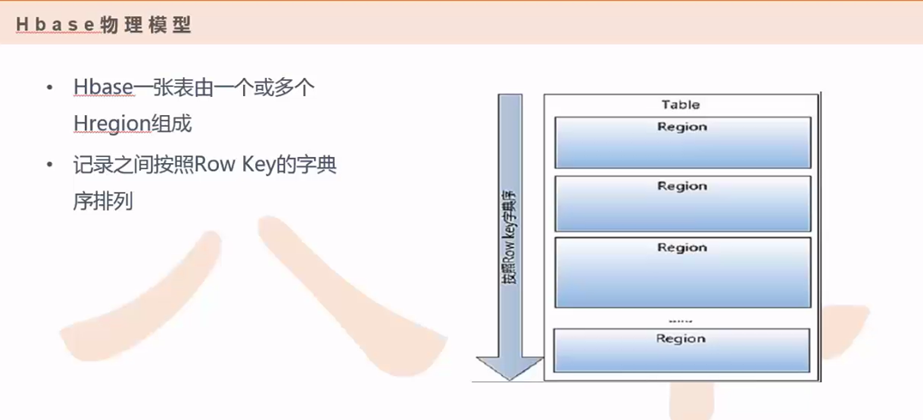


你就可以把这个数据往这个上面去填数据就可以了，只不过是你前面这几个数据都是上图22中前四项的一个key，只不过是最后一个value，所以这体现出了一个Hbase也是属于Nosql这么一个情况。

那刚才我们说的是HBase的数据模型，这个数据模型是怎么实现的？我们要进一步了解它的一个物理的模型，数据模型和物理模型这两个是有什么差异，这个大家应该是没有什么问题了，就是物理模型是称数据模型的实现，真正的细节在这个物理模型,只不过数据模型是我们应用的时候是容易接受的那种逻辑表达。

好了我们来看一下物理模型(如下图23)

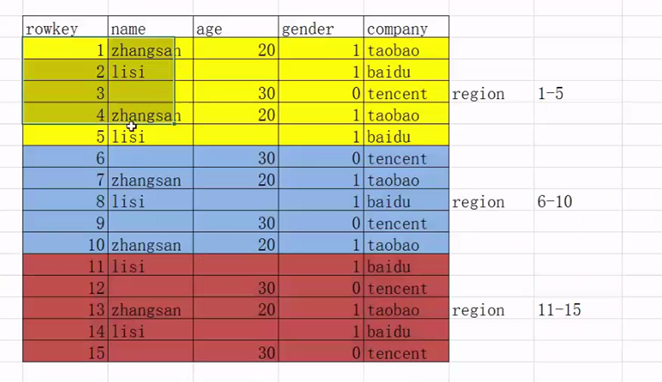
图23：



那对于一张Hbase的一张表其实它是由一个或多个Hreging来组成的，那什么叫做Hreging呢？那在Hreging大家可以暂时先认为它就是一个区域或者分区，所以认为一个Hbase一张表它是由一个或多个分区组成，然后并且这张表的数据是按照rowkey进行字典排序的，我们看上图23图中的图，相当于就是一张用户表，就类似于刚才那张一样（

如下图24所示）

图24：



这图24这张图就是你可以认为是一个我们正在用的一个数据表格，那这张数据表我可以对他进行多个区域进行划分，假设说我们按照每5行的方式去做一个划分，那么图24就有三个区域，那相当于是这每一个区域都叫做一个region，所以这个图就是想我们图23中的那种图的形式是一样的，那这个region是按照行去划分，然后有一个行的数目然后它有一个固定的一个每一个region在去划分的时候它是有一个范围的，比如这个region的范围是1到5行，比如第二个region是6到10，第三个就是11到15的范围，这里面画的是平均分，但实际中你这个是可以指定的，你也可以去平均都可以，所以图24的图能够对照图23中的那种图的，就是说一个用户的表格它可以分出多个区域多个region，然后每一个region包含了很多的行包含了很多的记录，然后并且这个记录与记录之间每一个region的内部他们这个rowkey按照顺序排序的，而且region与region之间也是按照这种全局排序的，这是一个整体的一个物理的逻辑图。

我们再继续往下说（如下图25）

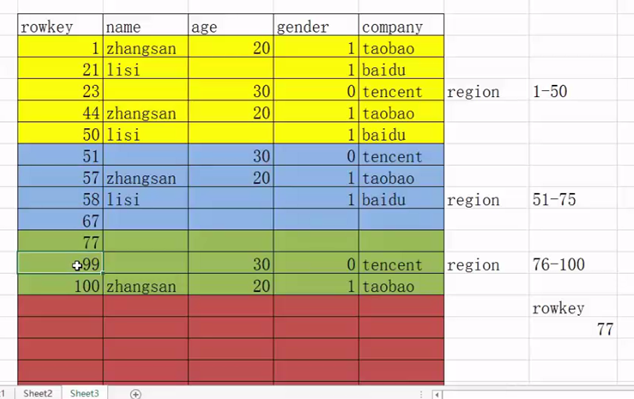
图25



那这个region呢因为是有大小概念的，有的region包含的记录多也有的region包含的记录少，那什么时候会region会变多有时候会变少？因为你这个数据是随着你的插入，使这个region是不断的增大。

假设说（如下图26所示）

图26：



那rowkey1到50，起初这里面记录很稀疏，因为你这个行是按顺序排序的，但是你不能够保证每一个数字都被占满了对吧？这可能rowkey是21，可能是23，可能是44，然后变成了50对不对？，比如说你rowkey你要新插入一个数据，这个数据的key是比如说是67，那么这个67这个记录是应该插入第二个region上面去是不是？，会在99上面开一个口子将67填充进去，如果你在插入一个77，那么也同样在会在99上面开一个口子将77填充进去，所以随着你的数据的不断插入，你这两个region之间的数据就不会那么的均匀了，那一旦有第二个region随着数据不断的插入，并且你这个51-100范围里面的这个数据它是存了那高品的部分，那第一个region1到50可能基本上就没有这样的一个key或者是低级的部分插入对吧？那随着你的这个数据的不断的插入，它可能大部分的数据都插入到了第二个51到100的region上面去，那也就是说第二个的region的成长速度要远远的超过了第一个region，那导致第二个region它占得空间就会越来越大，它一旦大到某一定程度的时候，这个region变得很大了，这个region就应该一分为二，变成了两个小的，怎么变小的呢？比如说在第二个region中间取个中位数75，然后下面这第三个region变成了一个76到100的范围的第三个region，从而变了一个新的第三个region，这个时候就由原来的一个region变成了两个不同的region了明白了把？

有点像西瓜分裂一样，分裂完之后，随着你的这个数据的不断插入，然后循环往复，然后不断的去分裂。

接下来到下课。请看Hbase笔记2.