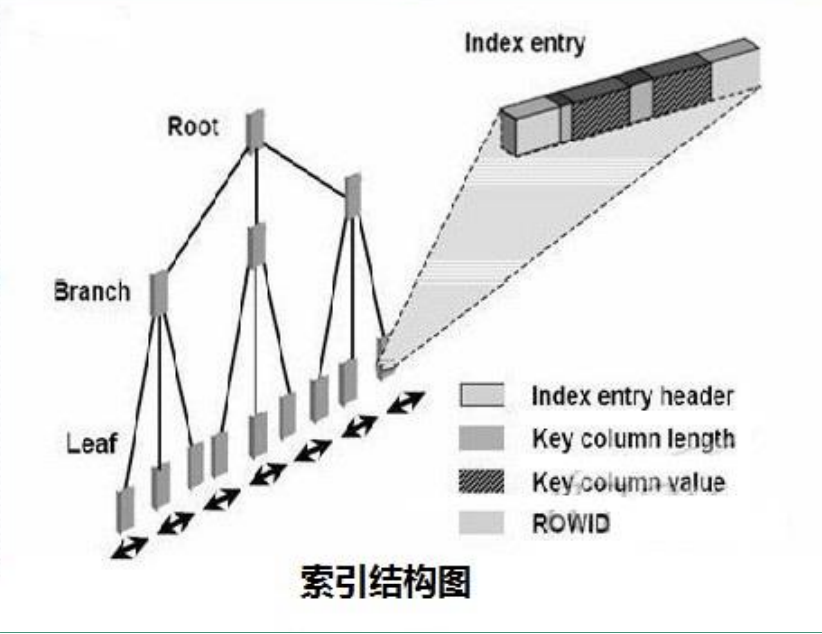
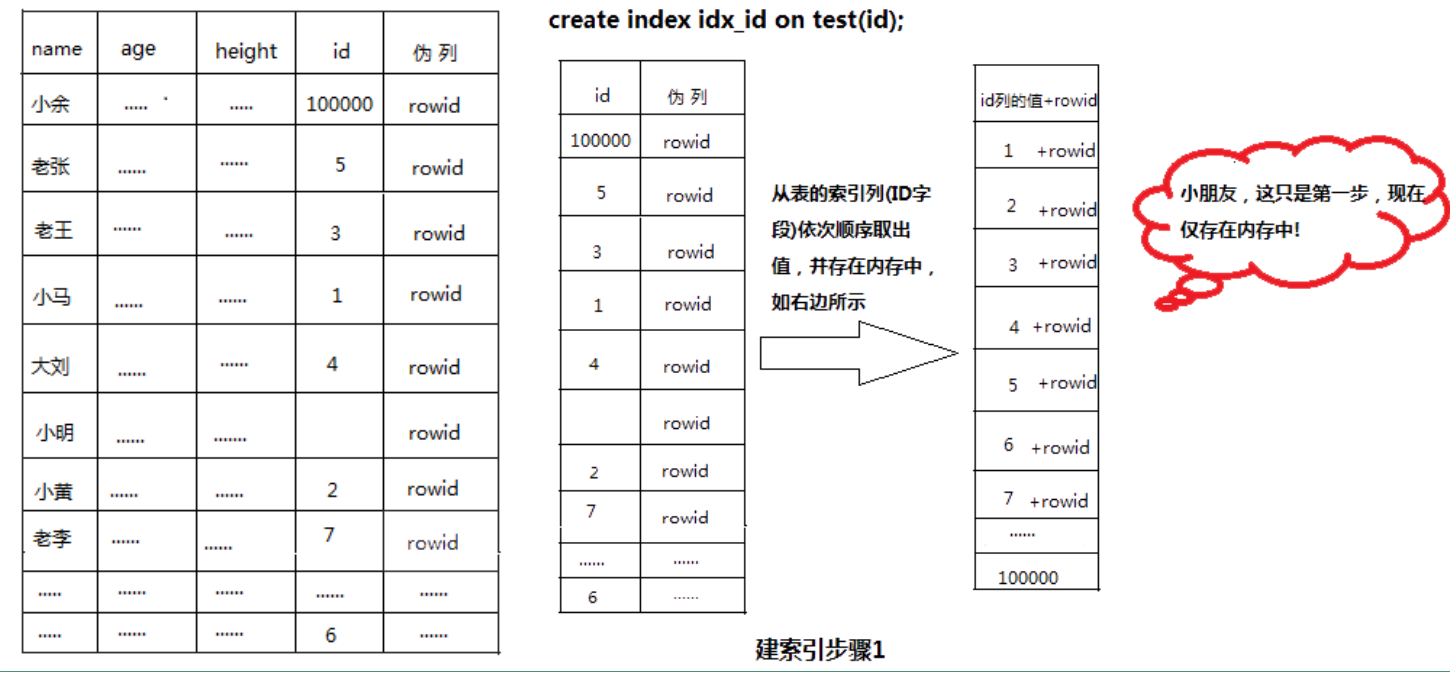
索引是什么

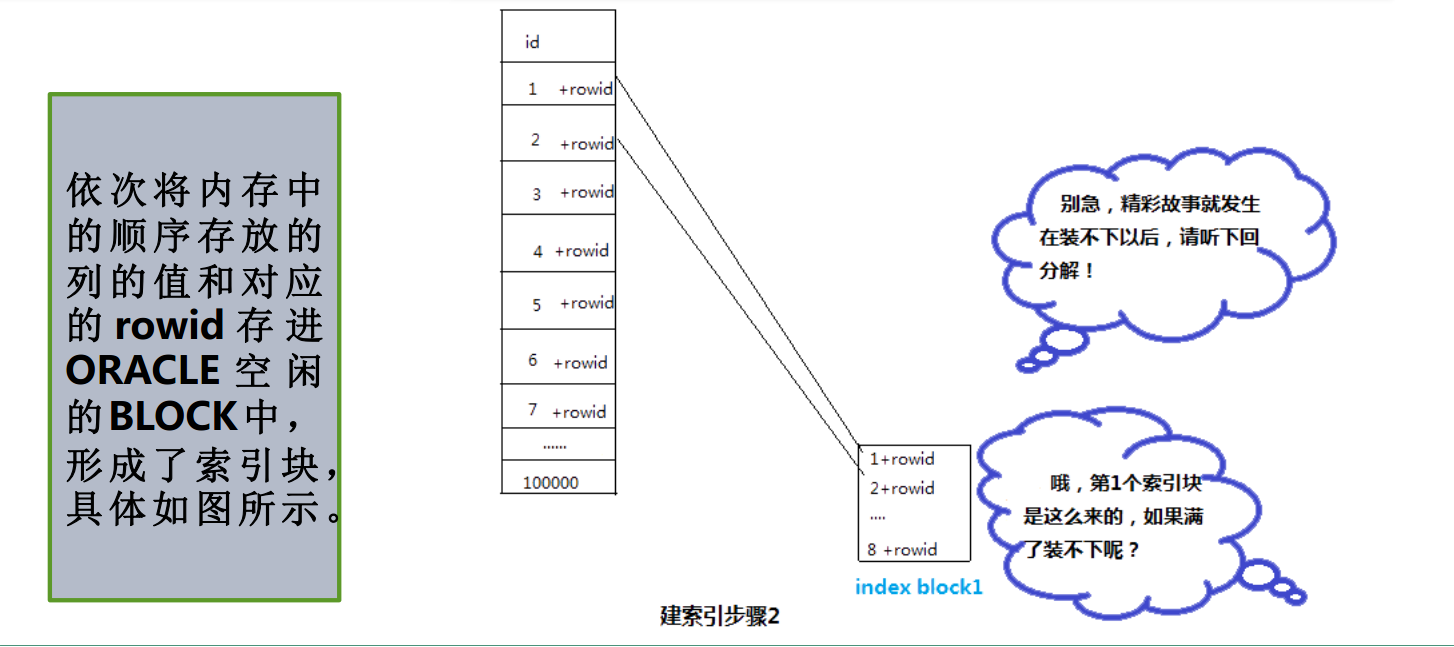
索引和表一样，都是逻辑体系结构中段的一种，比如建立一个T表，就产生一个T表的SEGMENT，当在T表某些列上建索引IDX\_T，就产生了一个INX\_T的索引SEGMENT，索引是健在表的具体列上，其存在的目的是让表的查询更快，效率更好、

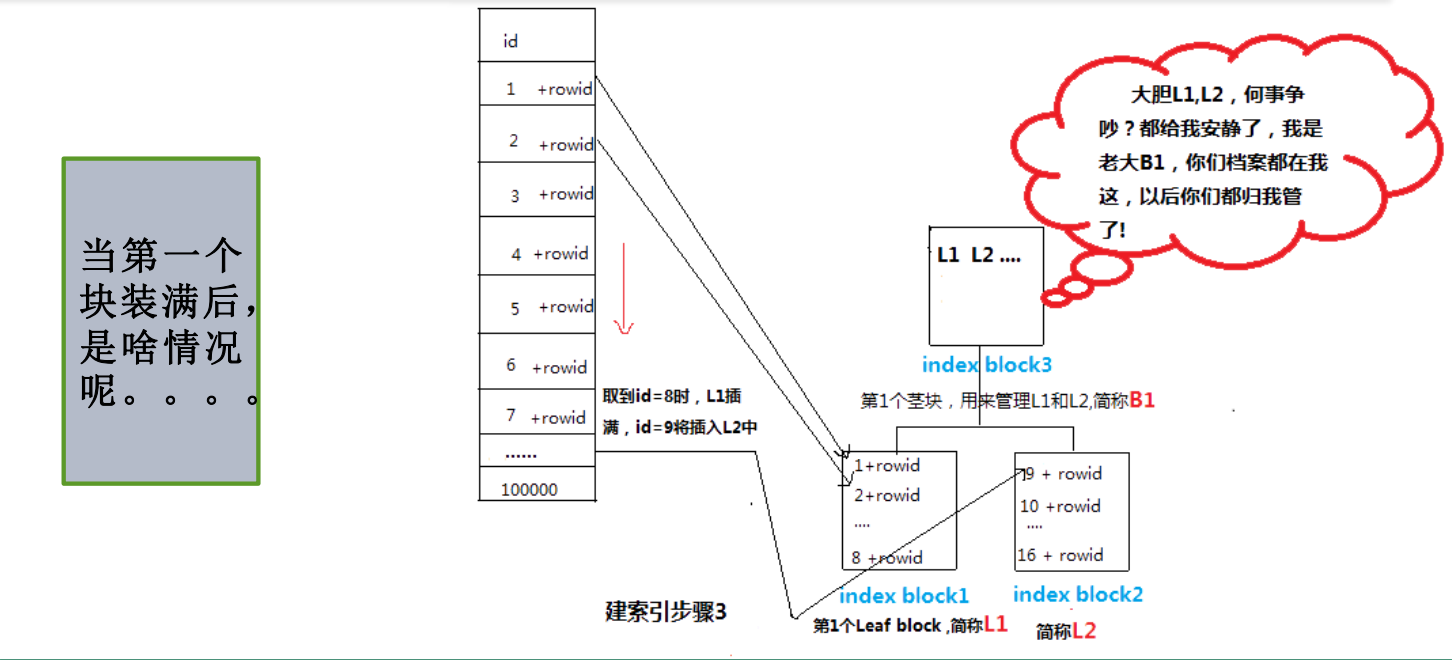
索引的体系结构

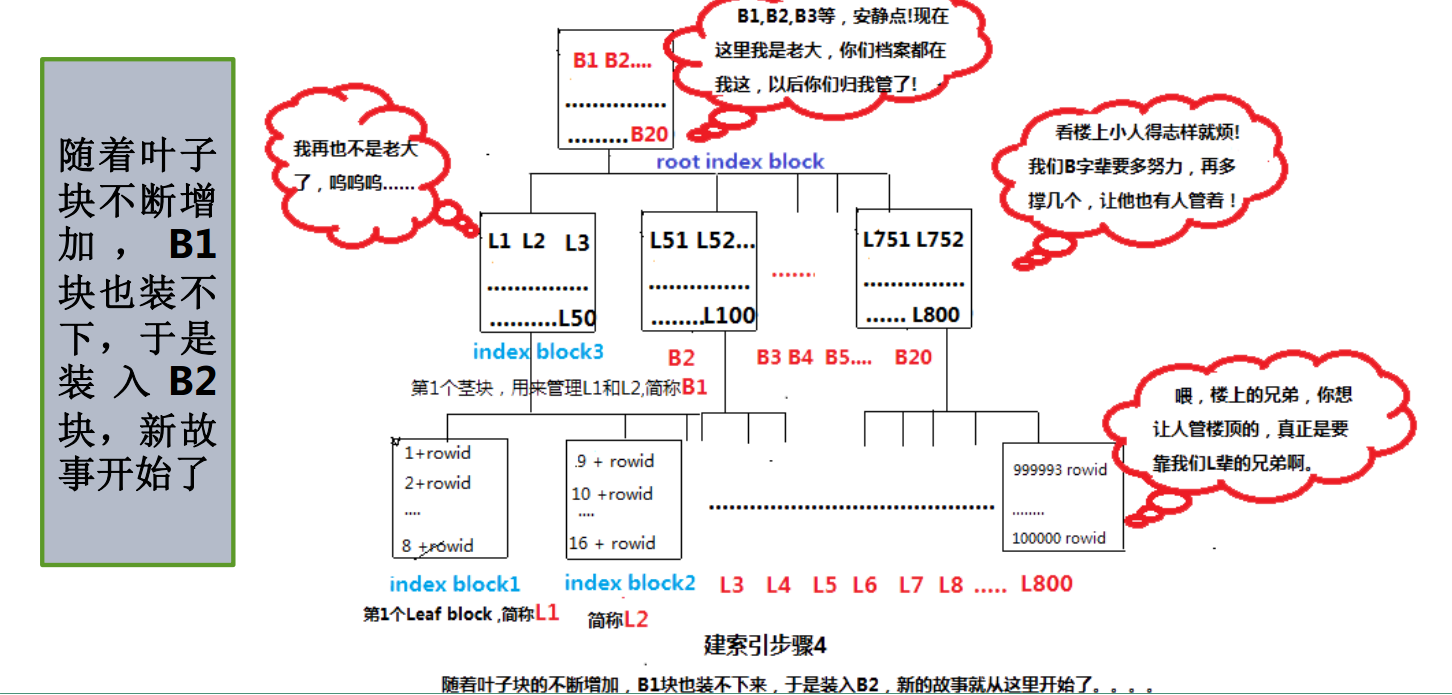


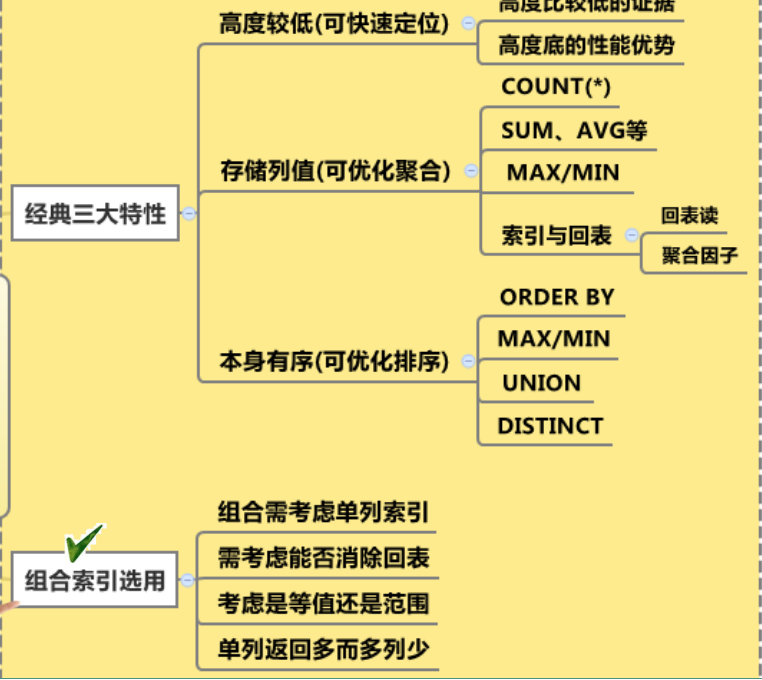
索引的建立过程：











索引高度：

从t1到t7(表记录依次增大10倍，从1到1000000),索引读的逻辑读是 2,3,3,4,4,4,5

从t1到t7(表记录依次增大10倍，从1到1000000)全表扫描的逻辑读是 3,5,19,148,1435,14298,142866



这种查询只是查询了一条数据

count（\*）会不会用到索引呢？不查空值，

我们知道索引的建立是ID+rowID，如果一张表仅有一个字段，这个索引比表还大，查询的效率低，那什么时候效果会好呢？表的字段很多，并且字段长度大多很长，其中一个非空且长度很短的列建立一个索引，这是索引的体积相对表来说特别小，那索引读的效率就会高很多、

在说明索引列非空的时候sum avg的情况与count情况类似，只要注意avg列不为0就ok



全表扫描可以扫描多个块，而索引的范围扫描只能扫描读取一个块

索引的长处就是返回少数数据，而不是全表的扫描

索引与排序：

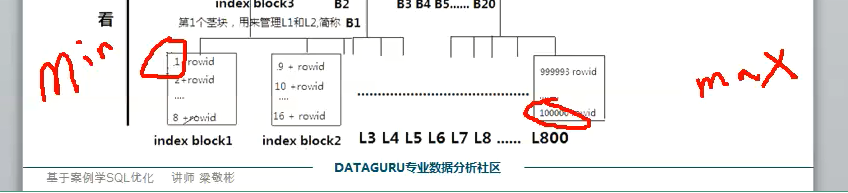
索引本身就是有序的

sorts产生的排序

tempspc



max/min







索引回表与优化：

索引回读表（table access by index rowid）

从索引中可以读到索引的信息，但是不可能到该列以外的其他列的信息，这里的查询select \* from t where object\_id<=5,这个字段表示所有的字段都需要返回，因此必然在扫描索引块中定位到具体object\_id<=5这部分索引块后，再根据这部分索引块的rowid定位到t表所在的数据块，然后从数据块中获取其他字段的记录，

我们在实际的业务情况中，有些字段必须出现，但有不多，如果只有一个字段上有索引，还是会出现回表的情况，这个时候可以考虑联合主键、



聚合因子决定了回表查询的速度

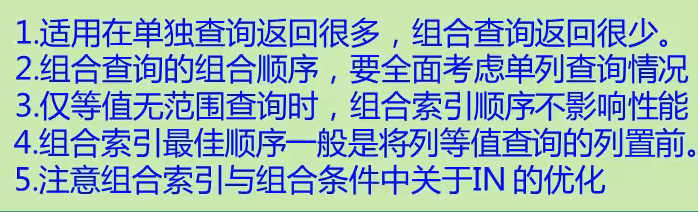
在回表不可避免的情况下，必须执行这个回表动作，是否回表查询方式也有高低之分呢？实际上，影响回表查询速度的就是聚合因子，表的插入顺序和索引列的顺序基本一致，从索引中回表查找数据块会更容易查找、



组合索引

并不只是避免回表的、





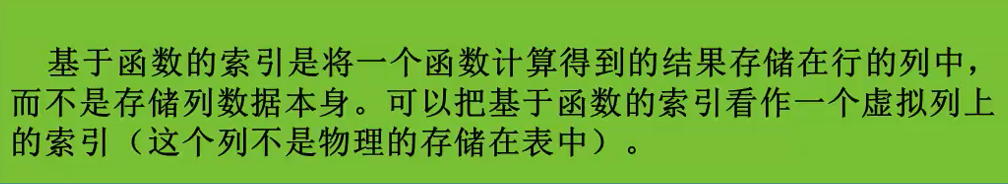
唯一索引：





位图

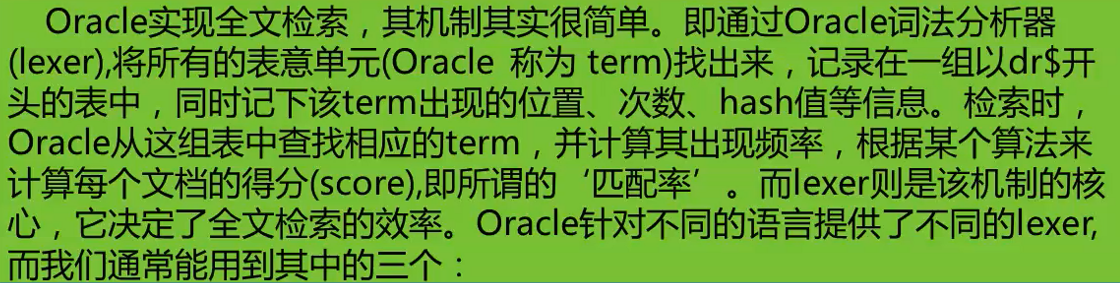
函数索引



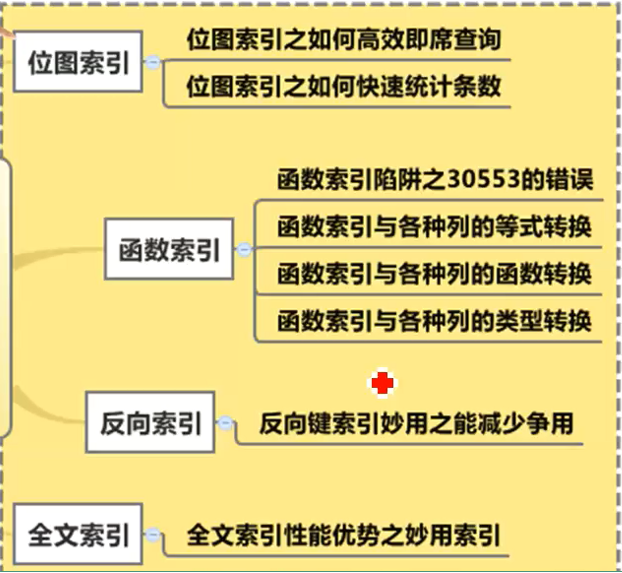
反向索引



全文索引



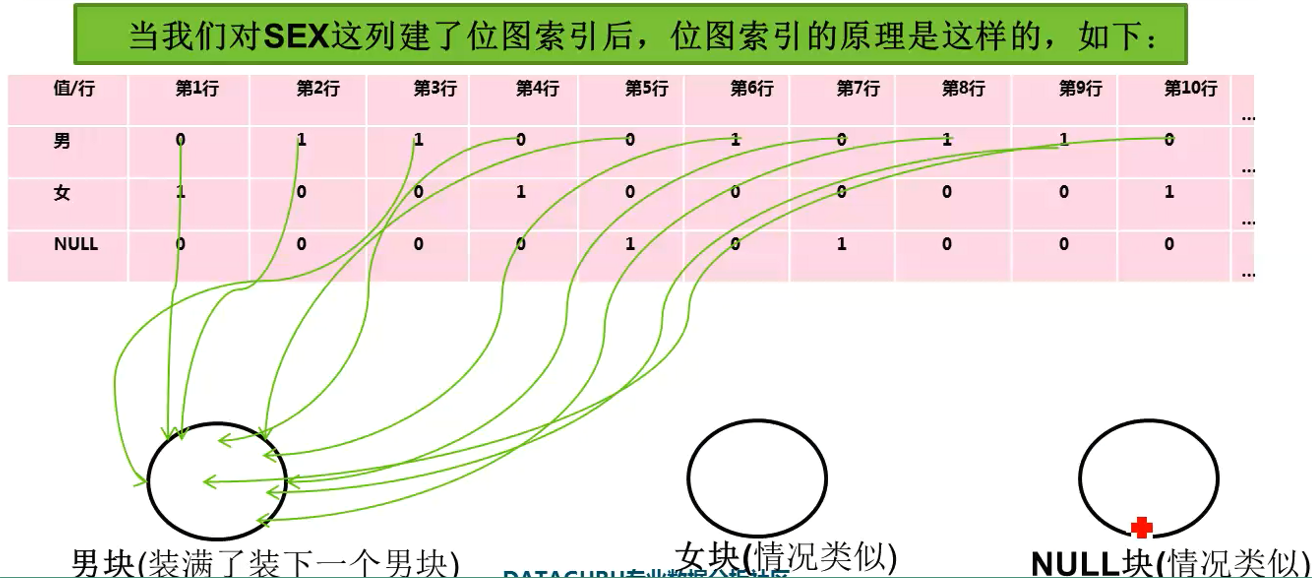
select \* from test where OBJECT\_NAME like '%高%';



位图索引

就是用位图表示的索引，对列的每个键值建立一个位图。









三个主要索引 http://blog.csdn.net/bzfys/article/details/45824549

1. B树索引(B-Tree index)  
                 适用范围：1）OLTP系统中

                                  2）主键、唯一性约束

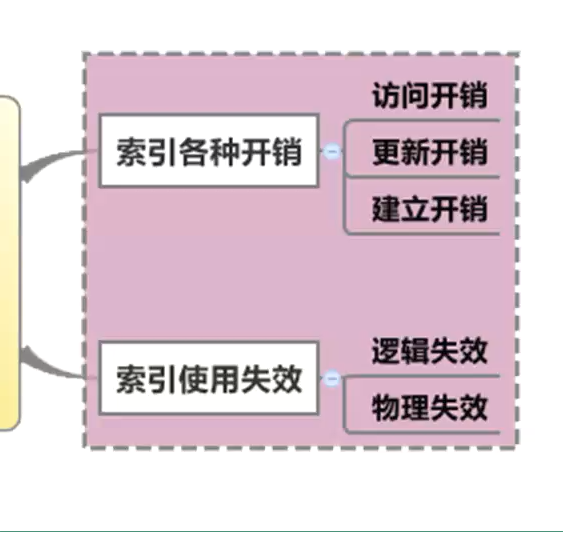
                                  3）重复度非常低的字段  
               不适用范围：重复度非常高的字段  
               特性：1）执行效率非常高  
                          2）查询时间基本不受数据量的影响  
                          3）如果重复度非常高，使用B-Tree索引还没有全盘扫描效率高

1. 位图索引(Bitmap index)  
                适用范围：1）OLAP系统中  
                                  2）主要是重复数据导致的查询性能低的问题，用来弥补B-Tree性能低的问题  
                                  3）使用OR\AND这类逻辑操作  
                不适用范围：大量DML操作、重复率较低的字段

              特性：1）适合在索引字段间进行OR\AND这类逻辑运算  
                         2）位图索引占用的空间比BTree索引用的少的多  
                         3）位图索引不适用于大量的DML操作（除select外），因为每个DML操作对应的位图索引键值对应多条数据，修改时键值会被上锁，如果此时其他用户需要更改数据，就会造成阻塞。

1. 全文索引(Text index)  
                适用范围：1）模糊查询  
                                  2）数据库空间种族，因为全文索引比较占用空间  
                                  3）语言文字范畴  
                不适用范围：全文索引占用大量的空间，不适合在硬盘资源紧张的库中（原数据的1.5倍）  
                                      BUG比较多，维护困难  
                        特性：1）适用于不变的数据，因为DML的新数据，不会在全文索引中包含  
                                   2）由于全文索引不是基于事物的，一旦有DML，就需要重新构建这个全文索引：alter index t\_idx rebuild parmeters('sync');

索引的危害



1.索引越多插入明显慢得多

2.无序插入索引影响更惊人

3.修改删除与插入略有差别

4.要增加IO和调整索引所计算的量。索引占用存储空间，随着table数据里增长，索引数据量也会增长，带来存储空间的消耗

b