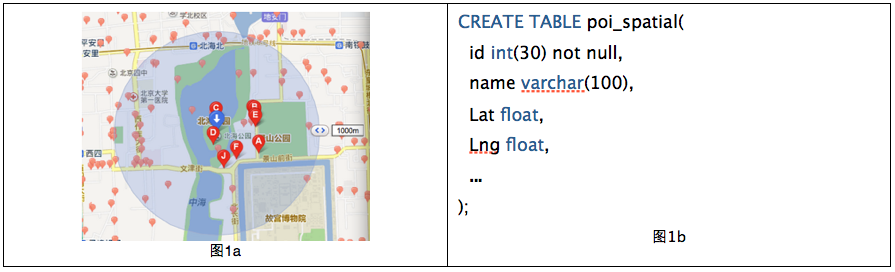
**一、问题**

      先思考个常见的问题：**如何根据自己所在位置查询来查询附近50米的POI**（point of interest，比如商家、景点等）呢（图1a）？

每个POI都有经纬度信息，我用图1b的SQL语句在mySQL中建立了POI\_spatial的表，其中lat和lng两个字段来代表纬度和经度。为后续分析方便起见，我人造了40万个POI数据。

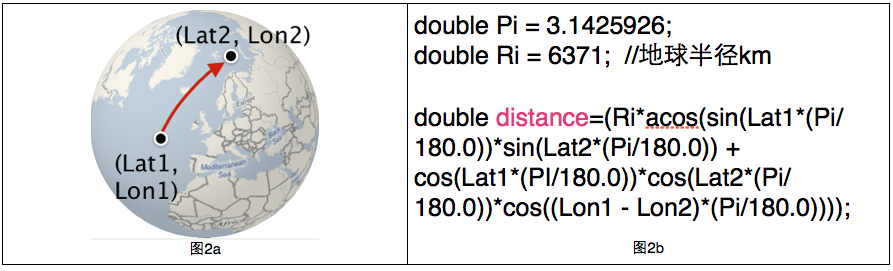


**二、传统的解决思路**

**方法一：暴力方法**

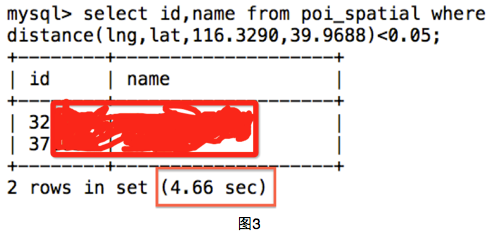
      该方法的思路很直接：计算位置与所有POI的距离，并保留距离小于50米的POI。

      插句题外话，计算经纬度之间的距离不能像求欧式距离那样平方开根号，因为地球是个不规整的球体（图2a），按最简单的完美球体假设，两点之间的距离函数应该如图2b所示。



       该方法的复杂度为：40万\*距离函数。我们将球体距离函数写为mysql存储过程distance，之后我们执行查询操作（图3），发现花费了4.66秒。

        该方法耗时的原因显而易见，执行了40万次复杂的距离计算函数。



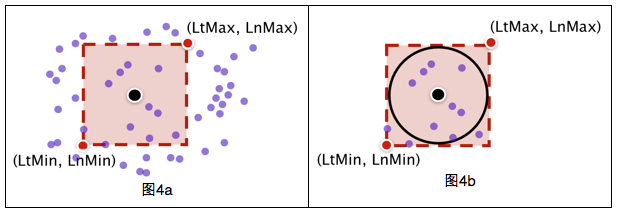
**方法二：矩形过滤方法**

      该方法分为两部：

       a）先用矩形框过滤（图4a），判断一个点在矩形框内很简单，只要进行两次判断（LtMin<lat<LtMax; LnMin<lng<LnMax），落在矩形框内的POI个数为n（n<<40万）；

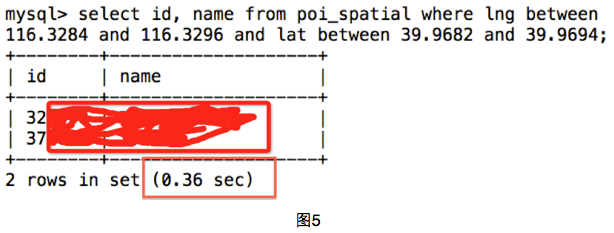
       b）用球面距离公式计算位置与矩形框内n个POI的距离（图4b），并保留距离小于50米的POI

矩形过滤方法的复杂度为：40万\*矩形过滤函数 + n\*距离函数（n<<40万）。



根据这个思路我们执行SQl查询（图5）（注： 经度或纬度每隔0.001度，距离相差约100米，由此推算出矩形左下角和右上角坐标），发现过滤后正好剩下两个POI。

此查询花费了0.36秒，相比于方法一查询时间大大降低，但是对于一次查询来说还是很长。时间长的原因在于遍历了40万次。



**方法三：B树对经度或纬度建立索引**

方法二耗时的原因在于执行了遍历操作，为了不进行遍历，我们自然想到了索引。我们对纬度进行了B树索引。

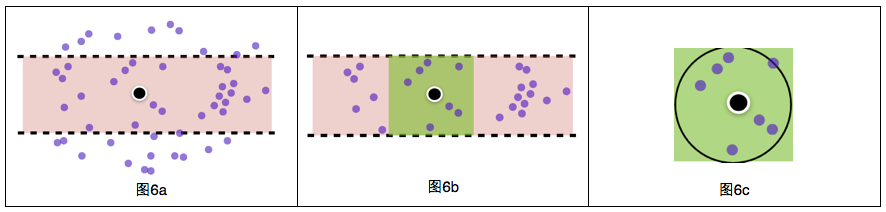
http://img.blog.csdn.net/20131027170902671

此方法包括三个步骤：

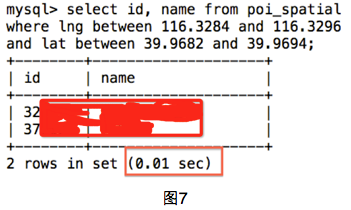
a）通过B树快速找到某纬度范围的POI（图6a），个数为m（m<40万），复杂度为Log(40万)\*过滤函数;

b）在步骤a过滤得到的m个POI中查找某经度范围的POI（图6b），个数为n(n<m)，复杂度为m\*过滤函数；

c） 用球面距离公式计算位置与步骤b得到的n个POI的距离（图6c），并保留距离小于50米的POI



执行SQL查询（图7），发现时间已经大大降低，从方法2的0.36秒下降到0.01秒。



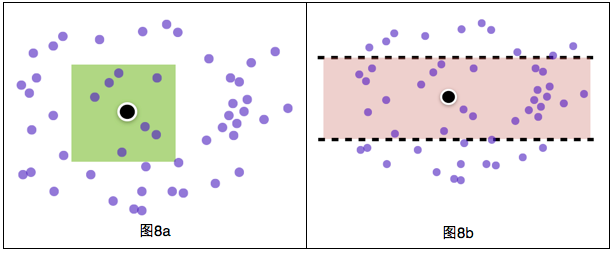
**三、B树能索引空间数据吗？**

这时候有人会说了：“方法三效果如此好，能够满足我们附近POI查询问题啊，看来B树用来索引空间数据也是可以的嘛！”

那么B树真的能够索引空间数据吗？

1）只能对经度或纬度索引（一维索引），与期望的不符

我 们期待的是快速找出落在某一空间范围的POI（如矩形）（图8a），而不是快速找出落在某纬度或经度范围的POI（图8b），想象一下，我要查询北京某区 的POI，但是B树索引不仅给我找出了北京的，还有与北京同一维度的天津、大同、甚至国外城市的POI，当数据量很大时，效率很低。



2）当数据是多维，比如三维（x，y，z），B树怎么索引？

比如z可能是高程值，也可能是时间。有人会说B树其实可以对多个字段进行索引，但这时需要指定优先级，形成一个组合字段，而空间数据在各个维度方向上不存在优先级，我们不能说纬度比经度更重要，也不能说纬度比高程更重要。

3）当空间数据不是点，而是线（道路、地铁、河流等），面（行政区边界、建筑物等），B树怎么索引？

对于面来说，它由一系列首尾相连的经纬度坐标点组成，一个面可能有成百上千个坐标，这时数据库怎么存储，B树怎么索引，这些都是问题。

既然传统的索引不能很好的索引空间数据，我们自然需要一种方法能对空间数据进行索引，即空间索引。

下节将对空间索引分类体系、原理、优缺点及数据库支持情况进行阐述。

来源： <<http://blog.csdn.net/zhanlijun/article/details/13171539>>