Oracle Spatial 的空间查询处理机制分析及优化

学　　院　计算机科学与技术

专　　业　计算机科学与技术

年　　级　2006 级

姓　　名　张连帅

指导教师　张坤龙

2010 年　6 月　21 日

摘　　要

随着信息技术的发展，空间数据的应用日益广泛，也越来越受到人们的重视。但是，由于空间数据自身的特点，查询空间数据要花费大量的时间，所以空间查询的效率问题一直是人们关注的重点。

众所周知，Oracle Spatial 是存储、管理、查询空间数据最好的工具之一，因此本文针对 Oracle Spatial，设计并实现了一个空间查询优化中间件，提高了空间查询的效率。在实现的过程中，首先分析空间查询的执行计划，并与普通一维查询的执行计划进行对比，通过对比可知 Oracle 仍采用通常的一维查询优化机制来处理空间查询，因此需要一个空间查询优化中间件来提高查询效率；然后依据将空间查询与非空间查询分开执行的原理，对 SQL 语句进行分解、重组，将生成的优化后的 SQL 语句提交给 Oracle 执行；最后，经过与原 SQL 语句的查询耗时对比后发现，优化后的 SQL 语句确实减少了执行时间。因此，可得出查询优化中间件确实提高了空间查询的效率的结论。

关键词：Oracle Spatial；OCI；空间数据；空间查询；优化

ABSTRACT

With the development of IT,spatial data is used more and more widely,and more and more attention has been paid.But because of the feature of spatial data,the spatial query costs lots of time,so people have always focused on the efficiency of the spatial query.

As we know,Oracle Spatial is one fo the best tools that is used to store,manage and query spatial data,so this article which based on Oracle Spaital ,designed and realized a plug-in that optimizes the spatial query,and improved the efficiency of the spatial query.Firstly,we analysed the explain plan of the spatial query and compared with the explain plan of the ordinary query,we achieved the conclusion that Oracle still use the ordinary optimize method to deal with the spatial query,so we need a plug-in that optimizes the spatial query to imporve the efficiency of the spatial query.Then based on the theory of separating the spatial query from the ordinary query,through dissecting and recombining the SQL statement,we submitted the SQL statement that has been optimized to Oracle.Finally ,we compared the time that optimized spatial query cost with the time that unoptimized spatial query cost and found that the time did cost less.So we can assure that the plug-in do improve the the efficiency of the spatial query.

key words: Oracle Spatial;OCI;Spatial data;Spatial query;Optimize

目　　录

第一章　　绪论 1
1.1　研究背景及意义 1
1.2　研究内容 1
1.3　论文结构 2
第二章　　相关理论与研究 3
2.1　Oracle Spatial 及相关概念 3
2.2　空间优化的理论和研究 10
第三章　　空间查询执行计划的分析 14
3.1　AUTOTRACE 14
3.2　执行计划的分析与对比 15
第四章　　空间查询的优化 19
4.1　空间查询优化的原理 19
4.2　空间查询优化的技术支持 19
4.3　空间查询优化程序的设计 24
4.4　空间查询优化程序的实现 26
第五章　　优化结果及结果分析 29
5.1　测试所用数据 29
5.2　程序运行说明 29
5.3　结果分析 30
第六章　　总结与展望 33

6.1 总结 33
6.2 展望 33
参考文献.....34
外文资料
中文译文
致谢

第一章　绪论

1.1　研究背景及意义

地理信息系统(简称 GIS)是一门多学科综合的边缘学科，是获取、存储、分析和管理地理空间数据的重要工具、技术和学科。随着信息技术的发展，它已关系到人们生产生活中的方方面面。地理信息系统主要处理与空间位置、空间关系有关的空间数据和带有非空间的属性信息的属性数据。

在以往的应用系统中，通常的做法是将两类数据分别存储，利用关系型数据库来存储属性数据；而由于空间数据的特殊性，则保持原有文件结构不变，以文件形式存储空间数据。通过在空间数据文件和关系型数据库中的属性数据之间建立关联为基础来构建应用系统。但是，随着地理信息系统向应用分布式的管理系统领域的转移，这种适用于单机的地理学领域的应用的两类数据分别存储的方式便不再适用，而且空间数据的文件管理模式在实现数据共享、网络通信、并发控制及数据的安全恢复机制等方面出现了难以解决的问题。[1]

当 Oracle 在其数据库中推出空间数据库组件 Oracle Spatial 时，这些问题便得到了解决。它是 Oracle 公司提供的空间数据管理模块，提供了一个完全开放的数据库管理机制。它是一个可以快速有效地存储、访问、分析空间数据的完整的功能程序集。在 Oracle Spatial 中，引入了抽象数据类型——SDO_GEOMETRY 来表示空间数据类型，因此空间数据就可以存储在一列中，而且 Oracle Spatial 发展了最新的空间数据和属性数据的全关系型数据库管理方式，利用关系型数据库来存储和处理空间数据，实现了空间数据与属性数据的一体化存储管理。

由于 Oracle 在数据库方面的优势，吸引了很多 GIS 公司的应用，空间数据库也渐渐的成为地理信息系统的基础。目前，Oracle Spatial 的应用日益广泛，但绝大多数情形都是仅仅使用它存储管理空间数据，对空间查询和分析功能涉及不多。而且由于空间数据自身的特点，查询空间数据要花费大量的时间，占有大量的空间。因此，如何优化空间查询性能成为一个重要的问题。但是，在 Oracle 公司推出的各个版本的数据库中，空间数据的查询和普通的一维数据查询一起执行。因此，底层的查询优化引擎可能会采用通常的一维查询优化机制来处理空间查询，这会导致空间查询效率的降低。因此，探究 Oracle Spatial 的底层查询优化机制对提高 Oracle Spatial 的性能有着至关重要的作用。

1.2　研究内容

本课题研究的内容是 Oracle 空间查询效率的问题。在 Oracle 中，空间查询与普通的一维数据查询一起执行，因此可能底层的查询优化引擎会采用通常的一维查询优化机制来处理空间查询，导致空间查询效率的降低。所以，要通过分析空间查询在 Oracle Spatial 中的实际执行计划来确定这一推断。经过对执行计划

的分析可以发现，相似的空间查询语句和一维数据查询语句的执行计划基本相同。因此可以得出，底层的查询优化引擎确实采用了一维查询优化机制处理空间查询。所以，需要编写一个优化查询中间件将空间查询 SQL 进行优化后输出给 Oracle 执行。这个优化机制的原理可以概括为：将空间查询的 SQL 语句转化为普通的一维数据查询和空间查询，然后先让 Oracle 执行非空间的查询，在此结果上执行空间查询。经过实验验证，经过这样的优化后，确实提高了空间查询的的效率。

1.3 论文结构

本文主要实现了一种提高空间查询效率的优化中间件。第一章，介绍了课题产生的背景和研究的内容。第二章主要介绍了 Oracle Spatial 的一些基本概念和在空间查询优化方面已取得的研究理论和成果。第三章使用 Oracle 提供的 autotrace 工具查看空间查询的计划，并通过其与非空间一维查询的对比，进行空间查询计划的分析。第四章主要介绍优化查询中间件所依据的原理和程序实现的一些细节。第五章对查询结果进行了对比、分析。第六章是全文的总结，并提出了一些不足和改进。

第二章　　相关理论与研究

2.1　Oracle Spatial 及相关概念

2.1.1　Oracle Spatial

Oracle Spatial 提供了一套 SQL 方案和函数，用来存储、检索、更新和查询数据库中的空间要素集合[2]。它包含下述组成部分：

- 一个方案（MDSYS），描述支持几何数据类型的存储、语法以及语义。
- 空间索引机制
- 一套操作和函数，用于执行兴趣区域查询、空间连接查询以及其他的空间分析操作。
- 管理工具。

2.1.2　对象关系模型和关系模型

Oracle Spatial 支持用对象关系模型和关系模型来表示几何体。对象关系模型使用 MDSYS.SDO_GEOMETRY 字段来表示一个几何体实例。它与 OpenGIS ODBC/SQL 地理空间要素的规范中的“带有空间几何体类型的 SQL”实现是相对应的。关系模型用多行记录和字段类型为 Number 的一张表来表示一个几何体实例，但是在 Oracle9i 的第二个发布版本例便不再支持，所以我们只讨论对象关系模型。

对象关系模型具有如下优点：

- 增加了几种几何类型的支持：弧、圆以及复杂多边形、复杂线串和优化矩形。
- 改进了易用性，具体体现在创建和维护索引、执行空间查询上。
- 索引直接由 Oracle 数据库服务器进行维护，减轻了开发人员的大量工作。
- 几何体使用单行单列进行模型化。
- 对很多性能进行了优化。

2.1.3　几何实体模型和数据模型

几何实体是指通过直线段或者弧线连接一系列顶点形成的实体。几何实体由它的类型来定义它的具体性质。Spatial 支持很多种原生类型，而几何实体正是有这些原生类型的几何组成，包含如下二维实体(见图 2-1)：

- 点及点簇
- 线串
- 多边形

- 弧状线串（所有弧都是圆弧）
- 弧状多边形
- 复合多边形
- 圆
- 矩形

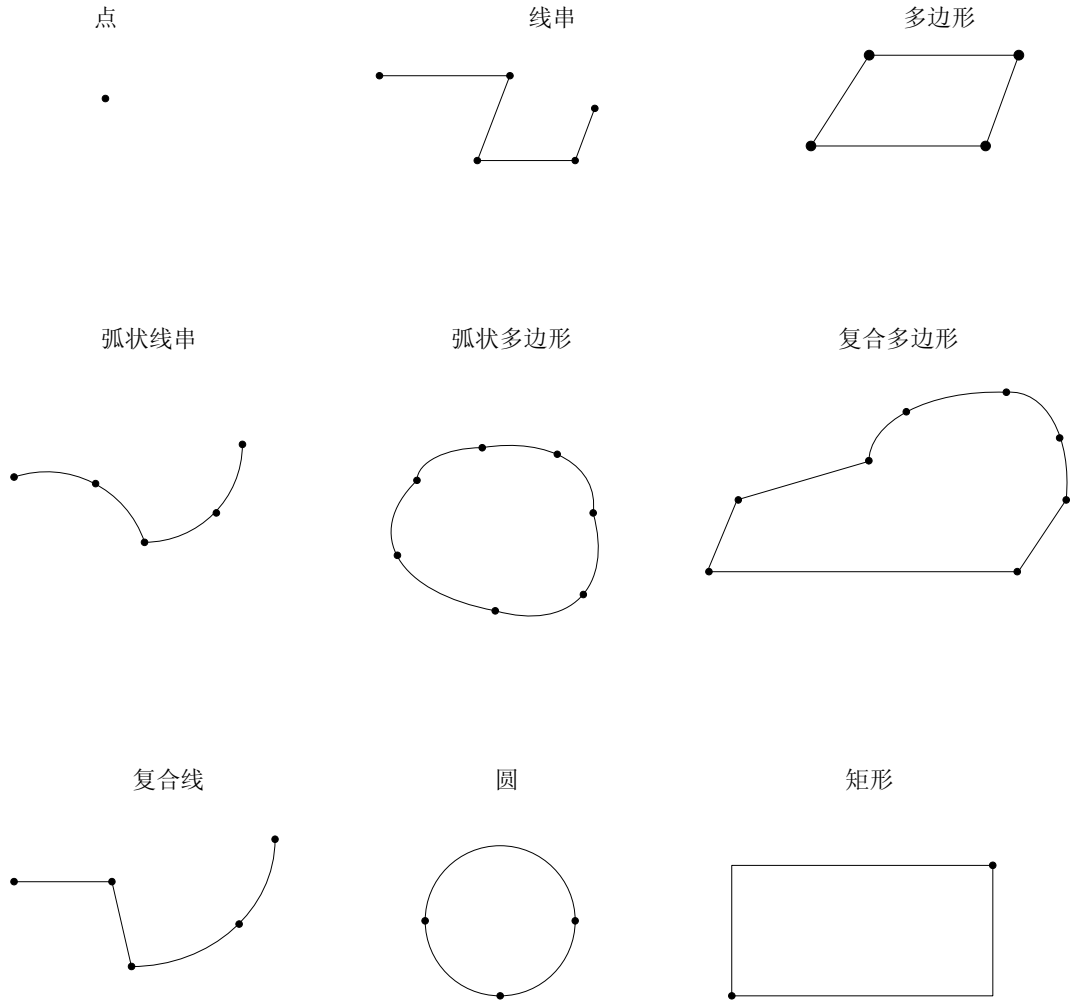


图 2-1　　Oracle 支持的几何实体类型[3]

数据模型是由元素、几何实体以及层所组成的层次结构，这与空间数据的表示完全相对应。层由几何实体组成，而几何实体由元素组成。[4]

元素是最基本的几何实体。Oracle Spatial 支持的空间元素类型有点、线串、多边形。例如，元素可以作为星座（点簇）、公路（线串）、县边界（多边形）的模型。元素的每个坐标都存成 X，Y 点对的形式。点数据由一个坐标构成，而线数据由代表一个线段元素的两个坐标构成，多边形数据则由坐标对值构成 ，多边形的每个线段都有一个顶点对。坐标的顺序是沿着多边形定义的，外环使用逆时针，内环采用顺时针顺序定义。

几何实体是对几何要素的表示，可以使用原生元素的有序几何来表达。一个几何实体可以由单个元素组成，它是 Oracle Spatial 支持的原生类型的一个实例，还可以由相同类型或者不同类型的元素的集合组成。例如，一个多边形，用来表示一个岛屿的集合，是由相同类型元素集合组成的。再例如，一个几何实体，里边有一个点和一个多边形，则它是由不同类型的元素的几何组成。

层是具有相同属性集的几何实体的集合。在 GIS 中，一个图层可能用来表示空间的拓扑关系，另外的一个图层可能用来表示人口的密度，第三个图层用来表示一个地区的道路与桥的网络。每一个图层的几何实体与其相应的空间索引都存储在 Oracle 的标准表格中，且一个空间图层与一个表格对应。

2.1.4 空间查询

空间查询又称空间检索或空间查找，指从数据库中找出满足条件的空间对象的过程，查找条件与空间位置有关。空间查询一般分为两大类，一类是点查询，给定一空间点，查找所有包含该点的空间对象；另一类查询是区域查询，包括相交查询和包含查询。

在 Oracle Spatial 中，使用双重查询模型执行空间查询，查询模型如图 2-2，双重查询分别为基本查询与再查询，基本查询对应于一次过滤，再查询对应于二次过滤。

一次过滤允许快速选择候选记录，以传给二次过滤操作。一次过滤操作比较几何实体的近似值，以减少计算的复杂性，通常被看作是低成本的过滤。因为一次过滤比较的是空间对象的近似值，返回精确结果的一个超集。

二次过滤操作对一次过滤得到的空间对象进行精确的计算，产生空间查询的准确结果。它的计算成本比较高，但它只是对一次过滤产生的数据集进行处理，而不是整个数据集。[5]

使用双重查询模型有两个优点：第一，空间对象一般都非常大，会消耗大量的主存储器空间，使用该模型只需消耗少得多的时间和空间就可以把一个对象的估计表示值载入内存。第二，空间对象上的计算一般都非常复杂昂贵，对象越复杂计算空间关系的处理要求就越多，但是计算估计对象速度会提高很多，而且对计算周期的要求会少得多。

2.1.5 空间索引

空间索引是 Oracle Spatial 的一个重要特点。空间索引是一种逻辑索引，空间索引的入口依赖于几何实体在坐标空间中的具体位置，但是索引值位于不同的域里。空间索引的好坏对空间查询的效率有极大的影响，一般来说，空间索引要做到如下两点：

- 找到索引数据空间中的对象，使其与给定点或者兴趣区域（查询窗口）

互相作用

- 找出两个索引数据空间中的对象，这些对象彼此之间相互作用

创建空间索引的原理基本相同，都采用分割原理，即把查询空间划分为若干区域，这些区域包含空间数据并可唯一标识。分割的方法有两种，一种是规则分

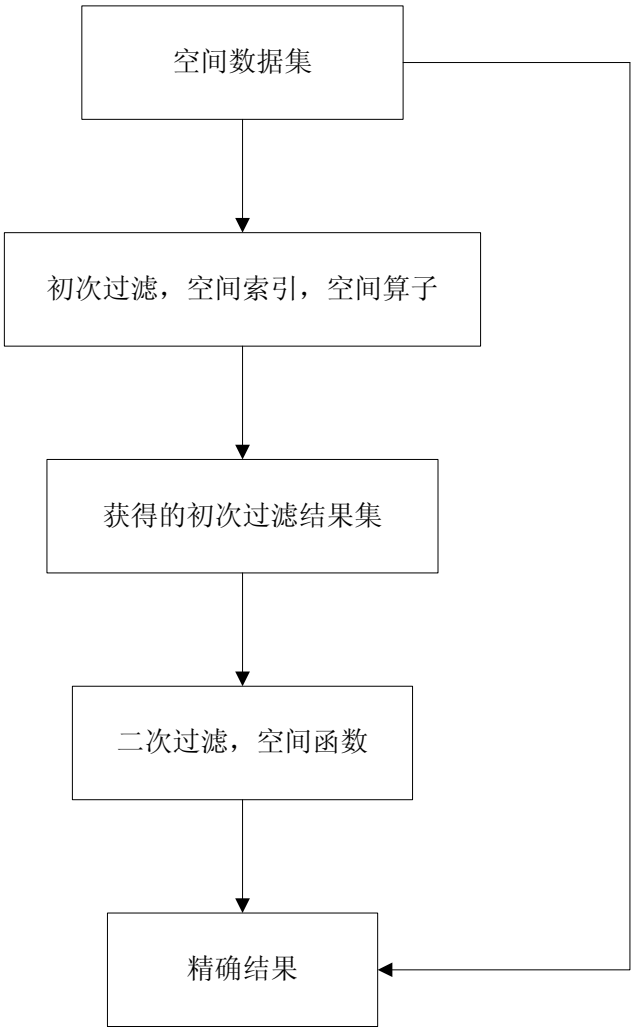


图 2-2　　优化查询模型[6]

割法，它是将地理空间按照规则或半规则方式分割，分割单元间接地域地理对象相关联，地理要素的几何部分可能被分割到几个相邻的单元中；另一种是基于对象的分割方法，这种方法的索引空间的分割直接由地理对象来确定，索引单元包括地理对象的最小外接矩形。基于这两种分割方法，产生两种索引机制，分别为 R 树索引和四叉树索引。[7]

R 树索引可以为三维和四维的空间数据进行索引，通过最小边界矩形（MBR——Minimum Bounding Rectangle）来对每个几何体近似索引。对线性参考系数据建立索引时，只能使用 R 树索引，而且 R 树索引是 Oracle Spatial 中默认的索引，如果在创建空间索引时没有明确指出，那么创建的就是 R 树索引。R

树索引存储在空间索引表 SDO_INDEX_TABLE 中，而该表有在试图 USER_SDO_INDEX_METADATA 中。R 树索引通过一个顺序数字发生器来确保当前用户对索引的实时更新。

在线性四叉树索引方案中，坐标空间要执行一个细化的过程，它定义了各个几何实体的覆盖小块。Oracle Spatial 可以使用固定大小片来覆盖几何实体，也可以使用可变大小片来覆盖几何实体。固定大小片通过片的精度来控制，如果精度是唯一的控制因素，那么当坐标空间分解到一定次数以后，细化过程将终止，这样，每个片形状和大小都固定。可变大小片要通过提供片的最大个数来控制，如果每个几何实体的片数 n 是唯一的控制因素，那么当 n 片都已经用来覆盖给定几何实体时，细化过程结束。Oracle Spatial 要用到 SDO_LEVEL 和 SDO_NUMTILES 值的组合确定使用的索引方式：

- 固定索引：SDO_LEVEL 值为非空非 0，而 SDO_NUMTILES 值为 0 或者空，产生固定大小片。
- 混合索引：SDO_LEVEL 值非 0 非空，而 SDO_NUMTILES 值也非 0 非空，对每个几何实体产生两个格网几何。一个集合包含着固定大小的片，另外一个集合包含着可变大小的片。

R 树索引和四叉树索引各有优劣，具体选择时可以参照表 2-1。

表 2-1　　R 树与四叉树索引对比

R 树索引	四叉树索引
不能调整对几何体的逼近精度（Spatial 使用最小边界矩形来进行微调）	通过设置格网级别以及格网数目调整对几何实体的逼近精度
索引的创建和调整容易	索引的调整比较复杂，设置合适的微调参数值对性能影响很大
对存储量需求较小	需要更多的存储量
对近邻查询速度较快	紧邻查询效率比较低
几何实体列的更新操作频繁时性能影响较大	大量更新不影响四叉树的性能
索引可以达到四维	只能索引二维数据
对于全球（whole-earch）索引需要使用 R 树对于大地数据上使用 SDO_WITHIN_DISTANCE 查询，推荐使用 R 树	

2.1.6　SDO_GEOMETRY 字段

SDO_GEOMETRY 类型是用来存储空间对象的信息的。它在 Oracle Spatial 中定义如下：

```
CREATE TYPE sdo_geometry AS OBJECT(
SDO_GTYPE NUMBER,
SDO_SRID NUMBER,
SDO_POINT SDO_POINT_TYPE,
SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY,
SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRAY);
```

SDO_GTYPE 表示几何实体的类型，采用 4 位数字表示，格式为 dltt，其中：

- d 表示维数（2、3 或者 4）
- l 表示三维线性参考系几何实体的线性参考度量，对于非三维线性参考系几何实体而言，必须接受 Oracle Spatial 默认的最后一维作为三维线性参考系几何实体的度量维，指定值为 0
- tt 表示几何实体的类型，00 至 07 是目前的有效值，08 和 09 保留，以备扩展之用

表 2-2　有效的 SDO_GTYPE 值

值	几何类型	说明
dl00	未知类型	用于存放自定义类型的几何对象，如样条曲线等
dl01	点	包含一个点
dl02	线或者曲线	包含一个线串，可以是直线或者圆弧段
dl03	多边形	几何实体含有一个带有洞或者不带洞的多边形
dl04	多种形状集合	点，线，多边形超集，可包含所有类型
dl05	复合点	几何实体含有一个或多个点，是点的超集
dl06	复合线	几何实体有一个或者多个线串
dl07	复合多边形	几何实体可以包含多个外环，多个不相交的多边形

SDO_SRID 也为 NUMBER 类型，用来标识与几何对象相关的空间坐标参考系，与几何实体关联。如果 SDO_SRID 值为空，则几何实体不与空间参考坐标系相关联。如果它非空，则必须为 MDSYS.CS_SRS 表中 SRID 字段的一个值,在创建含有几何对象的表时，这个值必须加入到描述空间数据表元数据的 USER_SDO_GEOM_METADATA 视图的 SRID 字段中。Oracle Spatial 规定，一个几何字段中的所有几何对象都必须为相同的 SDO_SRID 值。

SDO_POINT 是一个包含 X、Y、Z 数值信息的对象，用于表示几何类型为点的几何对象。如果 SDO_ELEM_INFO 和 SDO_ORDINATES 数组都为空,则 SDO_POINT 中的 X、Y、Z 为点对象的坐标值，否则，SDO_POINT 的值可以忽略(用 NULL 表示)。Oracle Spatial 建议用 SDO_POINT 存储空间实体为点类型的空间数据，这样可以极大地优化 Oracle Spatial 的存储性能，提高查询效率。

SDO_ELEM_INFO 使用变长 NUMBER 型数组来表示，用于解释存储在 SDO_ORDINATES 里边的坐标值。SDO_ELEM_INFO 中三个值的意义如下：

- SDO_STARTING_OFFSET 表示元素在 SDO_ORDINATES 数组中第一个坐标的位置。起始位置为 1，而不是 0。
- SDO_ETYPE 表示元素的几何类型。当它的值为 1、2、1003 和 2003 时，表明这个几何元素为简单元素；如果 SDO_ETYPE 为 1003，表明该多边形为外环(第一个数为 1 表示外环)，坐标值以逆时针存储；如果 SDO_ETYPE 为 2003，表明该多边形为内环(第一个数为 2 表示内环)，坐标值以顺时针存储；当 SDO_ETYPE 为 4、1005 和 2005 时，表明这个几何元素为复杂元素。它至少包含一个头三元组用以说明该复杂元素具有多少个简单几何元素。同样，当 1005 表示多边形为外环时，坐标值以逆时针存储；当 2005 表示多边形为内环时，坐标值以顺时针存储。
- SDO_INTERPRETATION 有两层含义，它的含义由 SDO_ETYPE 是否为复杂元素决定。如果 SDO_ETYPE 是复杂元素(4， 1005 和 2005)，则 SDO_INTERPRETATION 表示它后面有几个子三元组属于这个复杂元素；如果 SDO_ETYPE 是简单元素(1，2，1003 和 2003)，则 SDO_INTERPRETATION 表示该元素的坐标值在 SDO_ORDINATES 中是如何排列的。

表 2-3 列出了有效的 SDO_ETYPE 和 SDO_INTERPRETATION 值对。

表 2-3　有效的 SDO_ETYPE 和 SDO_INTERPRETATION 值对

SDO_ETYPE	SDO_INTERPRETATION	含义
0	任意数值类型	用于 Oracle Spatial 不支持的几何体类型
1	1	点类型
1	n>1	n 个点的点簇
2	1	由直线段相连的线串
2	2	由弧线段组成的线串，一个弧线段由起点、弧上任意一点以及终点组成。相邻两弧段的接点不需要重复存储。
1003 或 2003	1	由直线段组成的多边形,起点与终点必须相同
1003 或 2003	2	由弧线段组成的多边形，起点与终点必须相同。一个弧线段由起点、弧上任意一点以及终点组成。相邻两弧段的接点不需要重复存储
1003 或 2003	3	矩形，由左下角和右上角两点确定
1003 或 2003	4	圆，由圆周上的不同三点确定
4	n>1	由直线段和弧线段连成的复合线。

		SDO_INTERPRETATION 的值 n 表示组成线串的连续子元素的个数，SDO_ELEM_INFO 数组的下 n 个三元组将描述每个子元素。子元素的 SDO_ETYPE 值只能为 2，子元素的最后一点将是下一个子元素的第一个点，不重复存储。
1005 或 2005	n>1	由直线段和弧线段连成的复合多边形。SDO_INTERPRETATION 的值 n 表示组成多边形的连续子元素的个数，SDO_ELEM_INFO 数组的下 n 个三元组将描述每个子元素。子元素的 SDO_ETYPE 值只能为 2，子元素的最后一点将是下一个子元素的第一个点，不重复存储。多边形的起点与终点必须相同

SDO_ORDINATES 是一个变长的 NUMBER 类型数组，用于存储几何对象的真实坐标。SDO_ORDINATES 必须与 SDO_ELEM_INFO 变长数组联合使用，而且值都必须是有效的、非空的。SDO_ORDINATES 数组的顺序按维排序，如果几何对象为三维,则 SDO_ORDINATES 的坐标以{X1，Y1，Z1，X2，Y2，Z2，.....}的顺序排列；如果几何对象为二维，则 SDO_ORDINATES 的坐标以{X1，Y1，X2，Y2，.....}顺序排列。[8]

2.1.7 计算空间实体的几何体函数[9]

几何体函数用来计算空间实体的关系、面积等，常用的函数介绍如下，忽略其中的字段参数：

- SDO_GEOM.RELATE()——检查两个几何体对象，确定它们之间的空间关系
- SDO_GEOM.SDO_AREA()——计算两维几何多边形的面积
- SDO_GEOM.SDO_DISTANCE()——计算两个几何对象间的距离，距离为两个集合对象上距离最近的点或线段之间的距离。
- SDO_GEOM.SDO_LENGTH——计算一个几何体对象的长度或周长
- SDO_GEOM.WITH_DISTANCE——确定两个空间对象是否位于给定的欧氏距离以内
- SDO_GEOM.SDO_CENTROID()——确定几何实体的重心
- SDO_GEOM.SDO_DIFFERENCE()——计算两个几何体进行差运算的结果几何体
- SDO_GEOM.SDO_INTERSECTION——返回两个几何实体的交集结果

2.2 空间优化的理论和研究

由于空间查询在各个领域如地理信息系统(GIS)、计算机辅助设计(CAD)、多媒体系统、医学或卫星图象库等之中在的应用越来越广泛，对空间查询的研究也越来越受到关注。但由于空间数据的数据量大，数据结构复杂，操作代价昂贵等特点，对空间查询进行优化将是空间数据库研究的一个重点和难点，而且现有的关系数据库的优化方式并不完全适用于空间数据的查询，因此在空间查询优化领域，还没有一套完整的优化系统设计方案。

2.2.1 建立空间查询的代价模型对空间操作的代价进行估计

目前，提出的一种研究方法是建立空间查询的代价模型对空间操作的代价进行估计。操作代价主要包括执行代价和 I/O 代价。同时还要对空间谓词的选择性，即对满足查询条件对象的数目与源对象几何数目的比值进行估计。空间查询的优化过程大致分为以下 4 阶段：

（1）将查询转化为某种内部表示

（2）进行某些提高效率的表达式转换

（3）谓词代价计算

（4）选择代价最小的执行计划

对于阶段（1）来说，一般将查询条件用语法树的形式表示。对于阶段（2），可以先进行选择再连接，先进行投影再连接等的变换，而且变换规则通常是有正确性保证的。而且阶段（1）和阶段（2）与关系数据库的优化很类似，可以借鉴关系数据库的优化方法。

对于谓词代价的计算，一般谓词的执行代价包括 CPU 代价和 I/O 代价。在基于 R 树做代价分析时，CPU 和 I/O 代价都与节点访问次数密切相关，其中 I/O 代价分为读取操作对象的 I/O(记为 R.COST)和输出结果的 I/O(记为 W.COST)，而且可以近似地将 CPU 代价(记为 CPU.COST)和 W.COST 视为 R.COST 的线性函数，即

CPU.COST=w * R.COST

W.COST=S * R.COST

其中，w 是一个经验值，S 为谓词选择性因子，需要统计信息，优化器才会选择一个合适的值进行计算。这样，便得出了谓词的代价公式：

C=(1+w+S) * R.COST

在计算谓词代价时，分两步进行。第一步，用代价公式进行粗略估算，要考虑到操作本身及操作类别属性；第二步为加权细化，细化利用了操作对象的属性和统计数据以及操作本身区别于其所属类别的特性。[10]

2.2.2 改进几何算法，减少计算时间

改进几何算法，减少计算时间也可以提高空间查询效率。这可以通过提高几

何检测速度的办法实现。实际空间对象的形状为复杂的多边形，在求精步骤中需要对多边形做几何检测。改进多边形检测算法可提高查询效率。例如，Wael M.Badawy，Walid G.Aref 提出利用启发式信息(如对象边界的走向) 来判断两个空间数据集中的对象是否相交从而减少复杂的几何计算。

2.2.3 Oracle Spatial 中提高空间查询效率的方法

2.2.3.1 指明空间索引中的对象类型

由于 Oracle 各个版本中对 R 树索引的不断改进，用户往往会产生可以不假思索地按缺省方式使用的错觉，这使得用户没用将空间数据按照点、线、面分别存入不同的数据表中，并根据特定的空间对象类型，在建立 R 树空间索引时显式地指明特定的空间对象类型，所以错过了让 Oracle 根据特定的空间对象类型调用空间查询优化的机会。因此可以通过指明空间索引中的对象类型，建立带对象类型的空间索引来进行优化。

在建立空间索引时，可以用 layer_gtype=“空间对象类型”的方式命名空间索引中的对象类型，具体的语法为：

CREATE INDEX index_name ON table_name(row_name) INDEXTYPE IS MDSYS.SPATIAL_INDEX PARAMETERS(layer_gtype=POINT)

其中 index_name 为空间索引名，table_name 为表名，row_name 为列名，layer_gtype 的值除 POINT 外，还可以为 LINE，POLYGON 等。

2.2.3.2 利用表分区提高性能

当需要存储和检索的数据量庞大时，为了易于管理和提高空间数据检索的效率，可以在这个庞大的数据表中建立分区，然后在这些分区表上建立空间索引，这样不仅使表易于管理，减少每次查询的查询数量，也可利用上文提到的方法提高空间查询效率。

2.2.3.3 选择合理的空间操作符的参数次序

在执行包含空间操作符的 SQL 语句时，Oracle 会根据参数次序选择不同的执行计划。因此，需根据查询对象的特点选择合理的参数次序，以便使 Oracle Spatial 选择较好的执行计划，提高空间查询的效率。但是，并不是改变每个空间操作符的次序都可以提高空间查询的效率，仅改变那些对参数次序敏感的空间操作符如 SOD_WITHIN_DISTANCE 才可以显著地提高空间查询效率。

之所以合理的空间操作符的参数次序可以提高空间查询的效率，主要在于 Oracle Spatial 的空间查询机制。前面提到过，Oracle Spatial 对空间查询使用双重查询机制，进行主过滤环节和次过滤环节。调整空间操作符的参数次序可以使得主过滤阶段更充分的运用空间索引，从而提高空间查询的效率，缩短了空间查询

的时间。[11]

2.2.3.4 小结

以上的三种提高空间查询效率的方法，都已在实践中得到了验证，确实提高了空间查询的效率，但是，这只是针对具体的某个表来说，需要对每个不同的表进行不同的分析。因此提高空间查询效率的方法还需深入研究，以便其适用于每个表。

第三章　空间查询执行计划的分析

执行计划可以帮助我们了解 Oracle 优化器如何执行我们提交的 SQL 的，通过执行计划，我们了解 Oracle 计划用来满足数据请求的步骤，包括步骤目标，访问类型或步骤中涉及的数据交互，以及计算机资源中心中步骤的估计成本。当然我们也通过对执行计划的分析来确定 Oracle 是否对空间查询使用了不同于普通一维查询的优化机制。

3.1　AUTOTRACE

AUTOTRACE 是 Oracle 提供的一项功能，它可以自动跟踪 SQL 语句、生成该语句的执行计划并且提供与该语句的处理有关的统计[12]。在 SQL*Plus 或 iSQL*Plus 中，使用 AUTOTRACE 的命令如下：

- SET　AUTOTRACE　OFF——此为默认值，即关闭 AUTOTRACE
- SET　AUTOTRACE　ON——产生结果集和解释计划并列出统计
- SET　AUTOTRACE　ON　EXPLAIN——显示结果集和解释计划，但是不显示统计
- SET　AUTOTRACE　TRACEONLY——显示解释计划和统计，尽管执行该语句但是将不显示结果集
- SET　AUTOTRACE　TRACEONLY　STATISTICS——只显示统计

3.1.1　执行计划

对于 AUTOTRACE 提供的执行计划来说，其中包含了三种基本的数据交互类型，分别为数据访问、联合和其他种类操作。

数据访问用来描述 Oracle 如何实际检索数据，由于查询的本质是请求数据，所以每项执行计划都有数据访问，Oracle 通过直接访问表中数据块或使用索引来检索数据，常用的获取数据的方法如下：

- FULL TABLE SCAN——浏览整个表，这种方法虽然看似是最慢的访问数据的方法，但 Oracle 能很快的从表中直接读取数据。
- PARTITION——对查询中所需的每个分区进行一个或多个访问操作
- ROWID——是 Oracle 访问该行最快的方式
- INDEX——索引操作，但包含很多不同类型的索引访问操作，如 UNIQUE SCAN，RANGE SCAN，FULL SCAN，FAST FULL SCAN 和 SKIP SCAN
- BITMAP——通过使用位图索引进行访问

当查询需访问的表多于一个时，就需要用某种方式将两个表联合在一起。在联合的过程中，会从驱动器的表中的一行数据开始，而且 Oracle 优化器会尽量选择产生最小行数的驱动器表，以减少后续联合比较的负担。常见的联合方法如

下：

- NESTED LOOP——嵌套循环遍历一个数据集中的值，将其与另一数据的值进行比较。内部循环对第二个数据集进行比较。
- SORT-MERGE——两个数据集按照联合键进行分类，并联合在一起
- MERGE JOIN CARTESIAN——在表之间进行笛卡尔联合。
- UNION——将两个数据集联合在一起，并消除重复行。

除了数据访问和联合之外，还有许多其他的步骤，一些较常用的选项如下：

- SORT——按分类次序返回行。
- UNIQUE——使用分类操作消除重复行。
- JOIN——在合并联合操作前进行排序。
- FILTER——遍历行几何，基于条件消除一些行，通常作为某类注
- VIEW——执行定义视图并返回行的查询操作。

3.1.2 统计信息

统计信息就是执行 SQL 语句时，Oracle 如何使用资源来满足语句执行的信息，包括递归调用等统计，重要的统计内容和含义如下：

- recursive calls：递归调用——基于提交的 SQL 语句的形成的 SQL 调用。
- db block gets：db 块获取——磁盘中数据基块的检索，当需要进行数据当前版本的写操作时进行该操作。
- consistent gets：一致获取——检索缓冲区中的数据块，每行读操作要进行一致获取。
- physical reads：物理读取——从磁盘中获得数据块所需的 I/O 操作。
- redo size：重做大小——指出此 SQL 语句需要的重做缓冲区空间大小。重做缓冲区用来保护数据库失败情况下数据的完整性，重做只对写操作必需。
- bytes sent via SQL*NET to client、bytes received via SQL*NET from client、 SQL*NET roundtrips to/from client：SQL*NET 统计——收到并发送到客户端的字节数。
- sorts(memory，disk)：分类统计——内存中执行的分类操作数以及需要磁盘执行的分类操作数。

3.2 执行计划的分析与对比

3.2.1 执行计划的分析

上文已经给出了 Oracle 数据库中运行 SQL 语句的执行计划，需要指出的是上文提到的计划通常是执行一维普通查询时 Oracle 会采用的计划。现在让我们查看并研究 Oracle 执行空间查询时的执行计划，分析其与普通查询的执行计划

的异同。查询的语句及执行的计划如图 3-1：

输入 SQL, PL/SQL 和 SQL*Plus 语句。

```
select distinct a.id,a.name,sdo_geom.sdo_area(a.geom,0.005) myarea
from us_parks a,us_parks b where a.id!=b.id and a.name=b.name and
sdo_geom.sdo_area(a.geom,0.005)<5000;
```

执行 加载脚本 保存脚本 取消

已选择73行。

执行计划

```
Plan hash value: 3679502114

-------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name             | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                  |   834 |   817K|       |   431   (3)| 00:00:06 |
|   1 |  HASH UNIQUE             |                  |   834 |   817K|  1688K|   431   (3)| 00:00:06 |
|*  2 |   HASH JOIN              |                  |   834 |   817K|       |   251   (3)| 00:00:04 |
|   3 |    VIEW                  | index$_join$_002 |  6371 |   124K|       |    53   (4)| 00:00:01 |
|*  4 |     HASH JOIN            |                  |       |       |       |            |          |
|   5 |      INDEX FAST FULL SCAN| US_PARKS_PK      |  6371 |   124K|       |    18   (0)| 00:00:01 |
|   6 |      INDEX FAST FULL SCAN| US_PARKS_NAME    |  6371 |   124K|       |    34   (0)| 00:00:01 |
|*  7 |    TABLE ACCESS FULL     | US_PARKS         |   319 |   306K|       |   198   (3)| 00:00:03 |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("A"."NAME"="B"."NAME")
       filter("A"."ID"<>"B"."ID")
   4 - access(ROWID=ROWID)
   7 - filter("SDO_GEOM"."SDO_AREA"("A"."GEOM",0.005)<5000)
```

图 3-1　　空间查询的 SQL 语句

由图 3-1 可知，在执行空间查询时，Oracle 采用了如 HASH JOIN、HASH UNIQUE、FAST FULL SCAN 等优化方法。从空间查询的执行计划来看，Oracle 所用的优化措施基本与普通查询所采用的一样，似乎确实没有对空间查询做特别的优化措施以提高效率。

3.2.2　执行计划的对比

经过上面的分析，我们基本上已经得出了底层的查询优化引擎会采用通常的一维查询优化机制来处理空间查询的结论。为了更加确定这个结论，我们将一维普通查询和空间查询的执行计划进行对比。为了使对比更加明显，我们让普通一维查询语句与空间查询语句的结构基本相同，所使用的 SQL 语句、执行计划及统计信息如下图所示：

输入 SQL, PL/SQL 和 SQL*Plus 语句。 清除

```
select a.sal from emp a,emp b where a.job='CLERK' and
b.job='SALESMAN' and a.ename=b.ename
```

执行 加载脚本 保存脚本 取消

执行计划

```
Plan hash value: 3638257876

-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     3 |    96 |     7  (15)| 00:00:01 |
|*  1 |  HASH JOIN         |      |     3 |    96 |     7  (15)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| EMP  |     3 |    54 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| EMP  |     3 |    42 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("A"."ENAME"="B"."ENAME")
   2 - filter("A"."JOB"='CLERK')
   3 - filter("B"."JOB"='SALESMAN')
```

图 3-2　普通一维查询的执行计划

输入 SQL, PL/SQL 和 SQL*Plus 语句。 清除

```
SELECT SDO_GEOM.SDO_DISTANCE(c_b.shape,c_d.shape,0.005) FROM cola
c_b,cola c_d WHERE c_b.name='cola_b' AND c_d.name='cola_d' AND
c_b.owename=c_d.owename;
```

执行 加载脚本 保存脚本 取消

执行计划

```
Plan hash value: 3115108023

-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |  7718 |     7  (15)| 00:00:01 |
|*  1 |  HASH JOIN         |      |     1 |  7718 |     7  (15)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| COLA |     1 |  3859 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| COLA |     2 |  7718 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("C_B"."OWENAME"="C_D"."OWENAME")
   2 - filter("C_B"."NAME"='cola_b')
   3 - filter("C_D"."NAME"='cola_d')
```

图 3-3　空间查询的执行计划

统计信息

0	recursive calls
0	db block gets
15	consistent gets
0	physical reads
0	redo size
407	bytes sent via SQL*Net to client
400	bytes received via SQL*Net from client
2	SQL*Net roundtrips to/from client
0	sorts (memory)
0	sorts (disk)

1 rows processed

图 3-4　普通一维查询的统计信息

统计信息

0	recursive calls
0	db block gets
20	consistent gets
0	physical reads
0	redo size
459	bytes sent via SQL*Net to client
400	bytes received via SQL*Net from client
2	SQL*Net roundtrips to/from client
0	sorts (memory)
0	sorts (disk)

1 rows processed

图 3-5　　空间查询的统计信息

通过图 3-2 与图 3-3 的普通一维查询和空间查询的执行计划对比可知，当 SQL 语句结构相似时，执行计划基本相同；从图 3-4 与 3-5 的两种查询的统计信息来看，Oracle 对各种资源的调用也基本相同。

3.2.3　结论

通过对空间查询的执行计划的分析及其与普通一维查询的执行计划对比，可以看出 Oracle 并没有对空间查询采用特殊的优化机制，底层的查询优化引擎仍会采用通常的一维查询优化机制来处理空间查询，因此，这可能会导致空间查询效率的降低，对于空间查询的优化值得探究。

第四章　空间查询的优化

4.1　空间查询优化的原理

我们已经知道了 Oracle 在空间查询过程中会采用双重过滤的优化方法，而在实际应用的空间查询中，查询条件往往包含一些一维查询的条件的限制。当把查询直接提交给 Oracle 数据库进行执行时，Oracle 数据库并不会区分一维查询的条件和空间查询的条件，直接进行一次过滤与二次过滤，得到查询的数据，而在一次过滤时，往往要过滤的数据量就可能会很大，因此会十分耗时。如果当空间查询中包含普通一维查询条件时，或许可以利用一维查询条件使得执行一次过滤时的数据量减少，从而减少空间查询所需消耗的时间，提高空间查询的效率。

本文所采用的优化原理也正是基于上个段落中提到的想法,当空间查询包含普通一维查询时，将空间查询分两步进行：第一步，根据一维查询的条件执行普通一维查询，将查询得到的结果存入临时表中；第二步，根据空间查询的条件，对创建的临时表进行查询，得到原查询应得到的结果。如果，当根据一维查询的条件执行普通一维查询的所得到的数据较原查询一次过滤时需处理的数据量有明显减少时，便可起到减少时间、提高效率的优化作用。

4.2　空间查询优化的技术支持

4.2.1　OCI 及相关概念

4.2.1.1　OCI 基本概念

OCI 是 Oracle 提供的调用接口，英文全称为 Oracle Call Interface。它是 Oracle 数据库的最底层数据访问接口，在各类访问接口中，功能最全、效率最高，却也最复杂，但是 OCI 却提供了很好的灵活性和高效性，为用户的二次开发提供了很好的平台。它提供的功能如下：

- 通过高效使用系统内存和网络连接，改善系统的性能和可扩展性
- 在两层 C/S 或多层 C/S 环境中为动态会话和事务管理提供一致的接口
- N 层的授权
- 为应用程序使用 Oracle 对象进行开发提供综合支持
- 访问外部数据库
- 提供应用，以支持硬件条件不变的情况下适应用户数目的增加和请求数量的增加

而且相比于 Oracle 提供的其他访问接口相比，OCI 也具有很多优点：

- 可以对应用设计的各个方面进行粒度的控制
- 对执行程序的高度控制

- 支持动态的 SQL 语句
- 支持多个不同的平台
- 可以高效使用数据操纵语言，可以成批地进行数据插入、更新和删除

当使用 OCI 建立一个 OCI 应用程序时，开发过程的基本步骤如图 4-1 所示

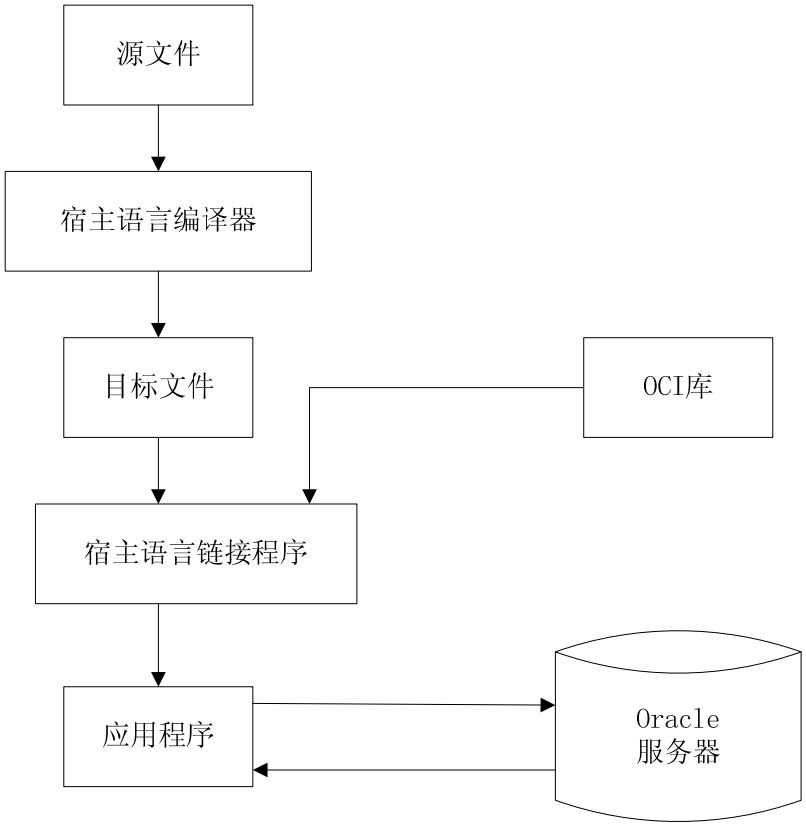


图 4-1　　OCI 应用程序的开发过程

4.2.1.2　OCI 程序结构

OCI 程序结构包括初始化环境、分配句柄、建立会话、执行 SQL 语句、终止会话和释放句柄等[13]，具体结构可参考图 4-2。

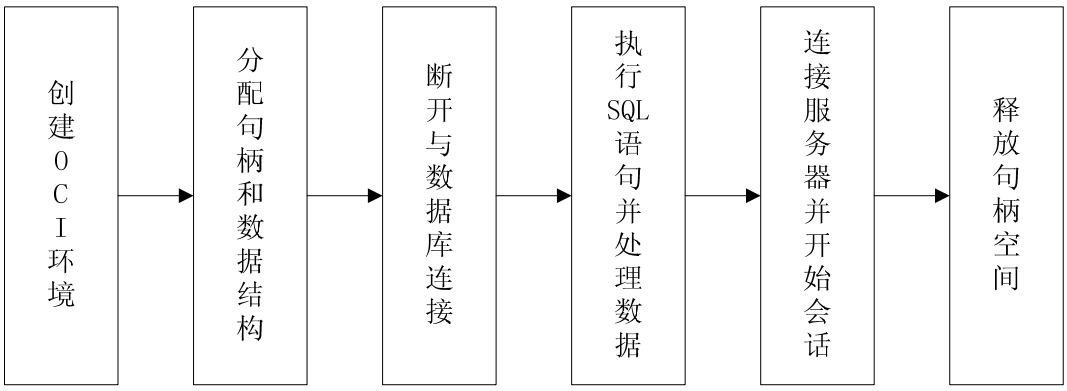


图 4-2　　OCI 程序结构图

4.2.1.3 OCI 数据结构

在 OCI 中经常用到的有三种不透明的数据结构，分别为句柄、描述符和定位符。它们的内部成员对用户不透明，用户也不需要知道它们的结构，可以通过对具体的分配句柄或描述符函数调用来初始化它们。

句柄在几乎在所有的 OCI 函数中都会用到，句柄是用来存储由 OCI 库分配的内存区域，可以存储如上下文或数据库连接等的信息。句柄的使用简化了编程，因为句柄是通过 OCI 库进行维护，而不是程序。常用的句柄及作用如下：

- OCI 环境句柄：定义了 OCI 函数的调用环境，是其他所有 OCI 句柄的父句柄
- OCI 错误句柄：作为其他 OCI 函数的参数，用来记录调用这些函数时产生的错误信息
- OCI 服务上下文句柄：定义 OCI 调用的服务器操作环境，包含服务器、用户会话和事务 3 种句柄
- OCI 语句句柄：标识 SQL 语句或 PL/SQL 程序块及其相关属性的环境
- OCI 绑定句柄：用于存储输入变量的信息
- OCI 定义句柄：为每个输出变量自动分配一个定义句柄
- OCI 描述句柄：用于 OCI 描述调用，应用程序可通过参数描述符获得存储在描述句柄中的模式对象信息
- OCI 线程句柄：用来描述 OCI 线程

此外，各句柄之间还有层次关系，关系如图 4-3 所示。

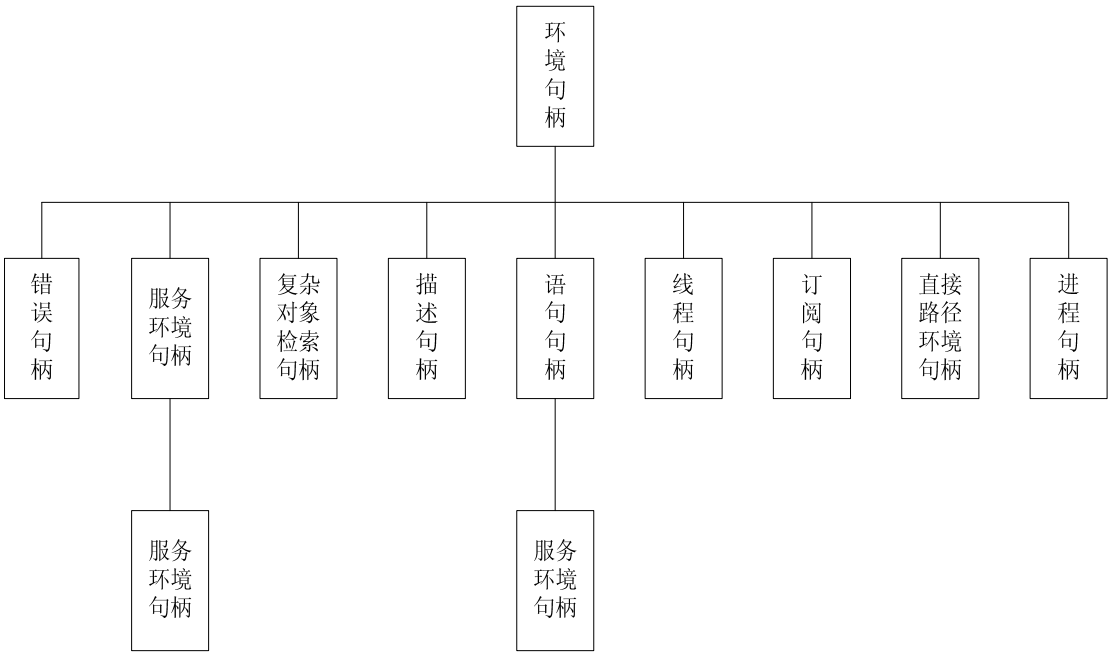


图 4-3　　OCI 句柄层次

描述符和定位符用来存储特定的信息，可用函数 OCIDescriptorAlloc()来分配一个描述符和定位符，用函数 OCIDescriptorFree（）来释放描述符和定位符，常用的描述符的作用如下：

- OCISnapshot：用于执行 SQL 语句操作
- OCILOBLocator：用于 BLOB、CLOB 或文件类型的读写调用
- OCIParam：用于描述调用
- OCIRowid：用于或定义行 ID 的值
- OCIDateTime 和 OCIInterval：用于时间数据类型
- OCIComplexObjectComp：用于复杂对象检索

4.2.2 优化程序中用到的 OCI 函数

在优化程序中使用了很多 OCI 提供的函数，包括创建环境句柄、分配句柄以及处理 SQL 语句时用到的函数。[14]

OCIEnvCreate()函数用来创建 OCI 环境，它的语法格式如下：

word OCIEnvCreate(OCIEnv **envhpp，ub4 mode，CONST dvoid *ctxp CONST dvoid *(*malocfp) (dvoid *ctxp，size_t size)，CONST dvoid *(*ralocfp) (dvoid *ctxp，dvoid *memptr，size_t newsize)，CONST void (*mfreefp) (dvoid *ctxp，dovid *memptr)，size_t xtramemsz，dovid **usrmempp)；

其中，envhpp 为输出参数，是一个指向环境句柄 OCIEnv 的双重指针，其编码设置由参数 mode 指定，语句句柄 OCIStmt 可以从 envhpp 继承这一设置。参数 mode，主要指定 OCI 环境初始化的方式，有效的方式有 OCI_DEFAULT、OCI_THREADED、OCI_OBJECT 等。参数 ctxp 指定内存回调函数的用户定义环境，malocfp 指定用户自定义情况下的内存分配函数。Ralocfp 是用户自定义情况下的内存重分配函数。Mfreefp 指定用户自定义情况下的内存释放函数，xtramemsz 指定在环境存活期内要分配的用户内存的大小。

OCIAttrGet()函数用来获取存储在句柄里的信息，它的语法格式如下：

sword OCIAttrGet(CONST dvoid *trgthndlp，ub4 trgthndlp，dvoid *attributep，ub4 *sizep，ub4 attrtype，OCIError *errhp)；

其中，sword 为 signed word，是带符号整型。trgthndlp 是要获取或设置属性的句柄指针，其为句柄类型。attribute 为输入型参数，是指向属性值的指针。sizep 为输出型参数，表示属性值的长度。size 为输入型参数，指出所设属性值的长度。attrtype 为输入型参数，说明属性设置或获取的类型。errhp 为输入/输出型参数，是一个错误句柄。

OCIStmtPrepare()函数用来为 SQL 语句的执行做一定的准备工作，如分配一定的内存空间等，其语法格式如下：

sword OCIStmtPrepare(OCIStmt *stmtp，OCIError *errhp，CONST text *stmt，

ub4 stmt_len，ub4 language，ub4 mode)；

其中，stmtp 为输入参数，是与 SQL 语句相关系的语句句柄。errhp 是错误句柄。stmt 为输入参数，是要执行的 SQL 语句文本串，必须是非空字符串。stmt_len 为输入参数，是 SQL 语句的长度。language 为输入参数，说明语句准备要用的语法格式，取值可为 OCI_7V_SYNTAX 和 OCI_NTV_SYNTAX。mode 为输入参数，指定环境的初始化方式，取值可为 OCI_DEFAULT 和 OCI_NO_SHARING。

OCIStmtExecute()用来执行 SQL 语句，基本语法如下：

sword OCIStmtExecute(OCISvcCtx *svchp，OCIStmt *stmtp，OCIError *errhp，ub4 iters，ubr rowoff，CONST OCISnapshot *snap_in，OCISnapshot *snap_out，ub4 mode)；

其中，svchp 为输入/输出参数，表示服务的上下文句柄。stmtp 为输入/输出参数，表示 SQL 语句的语句句柄，它定义了语句以及相关联的数据。errhp 为输入/输出参数，将它传递给 OCIErrorGet()，会得到错误诊断信息。iters 为输入参数，对于非 SELECT 语句，语句执行次数为 iters-rowoff；对于 SELECT 语句，如果不知道语句会取多少行记录，可以将 iters 值设为 0。rowoff 为输入参数，执行多行参数时，数组绑定里数据的起始序号。snap_in 为输入参数，且为可选参数。snap_out 为输出参数，同样为可选参数。mode 为输入参数，表示 OCIStmtExecute()函数的执行模式，常用的可选值可为：

- OCI_BATCH_ERRORS——批错误执行方式，用于执行数组方式的 DML 操作
- OCI_COMMIT_ON_SUCCESS——在语句执行成功的前提下，事物会在语句执行后立即提交
- OCI_DEFAULT——默认执行模式，在该模式下，会隐含返回选择列表的描述信息
- OCI_DESCRIBE_ONLY——不执行 SQL 语句，只返回选择列表的描述信息

OCIStmtFetch()用来获取查询结果。它的语法格式如下：

OCIStmtFetch(OCIStmt *stmtp，OCIError *errhp，ub4 nrows，ub2 orientation，ub4 mode)；

其中，stmtp 为输入参数，是查询语句的语句句柄。errhp 为输入参数，用于错信息的诊断。nrows 为输入参数，提出要从当前位置提取的行数。orientation 为输入参数，为默认值。mode 为输入参数，表明函数调用的模式，其值通常为 OCI_DEFAULT。

OCIDefineByPos()通过列的位置定义输出的结果。它的语法格式如下：

OCIDefineByPos(OCIStmt *stmtp，OCIDefine **defnpp，OCIError *errhp，

ub4 position，dvoid *valuep，sb4 value_sz，ub2 dty，dvoid *indp，ub2 *rlenp，ub2 *rcodep，ub4 mode)；

其中，stmtp 为输入/输出参数，是 SQL 查询语句对应的语句句柄。defnpp 为输入/输出参数，是双重指针，指向一个定义句柄。errhp 为输入/输出参数，错误句柄，用于错误信息诊断。position 为输入参数，是一个指向数值缓冲的指针，类型由 dty 参数决定。

4.2.3 运行的环境及配置

本文中应用的数据库为 Oracle 10.2.0.4.0，开发环境为 Microsoft Visual C++ 6.0，但是需要进行一些配置。

点击“工具→选项”菜单，选择“目录”页，在“目录”的组合框中选择 Include files 选项，添加%ORACLE_HOME%\OCI\include；再选择 Library files 选项，添加%ORACLE_HOME%\OCI\LIB 和%ORACLE_HOME%\OCI\LIB\MSVC\VC6 两项库文件。点击菜单“工程→设置”，选择“连接”，在对象/库模块下添加名为 oraocci10.lib 库文件；选择“C/C++”选项，在“预处理程序定义”中去掉_DEBUG 宏，添加 WIN32COMMON，_DLL，_MT 这三个宏，同时在“工程选项”中，将 MLd 改为 ML，去掉/GZ；点击“分类”组合框，选择“Code Generation”选项，在“User run-time library”中选择 Multithreaded DLL。点击“工程→添加到工程→文件”，将目录%ORACLE_HOME%\OCI\lib\MSVC 下的 oci.lib 和 ociw32.lib 两个库文件添加进来。[15]

经过上面的配置后，便可以用 VC 连接 Oracle 数据库了。

4.3 空间查询优化程序的设计

4.3.1 程序流程

空间查询优化程序接收输入的空间查询 SQL 语句，判定其是否符合优化程序的优化条件。如果不符合优化的条件，则直接执行 SQL 语句，进行空间查询；如果需要优化，则进入优化程序。首先，分析输入的 SQL 语句，提取语句中要查询的列、需要查询的表格、查询条件中关于空间查询的限制以及关于普通一维查询的限制；然后，根据已提取出的内容，生成新的 SQL 语句；最后提交给 Oracle 数据库执行，将结果显示出来并存入文件中。程序流程图见图 4-4。

4.3.2 可优化的条件

当查询的条件中包含两个或两个以上的约束，并且约束条件中至少有一个为空间查询的约束时，即可实行优化，否则的话，程序将不会进行任何优化，直接把 SQL 语句直接提交给 Oracle 数据库执行。

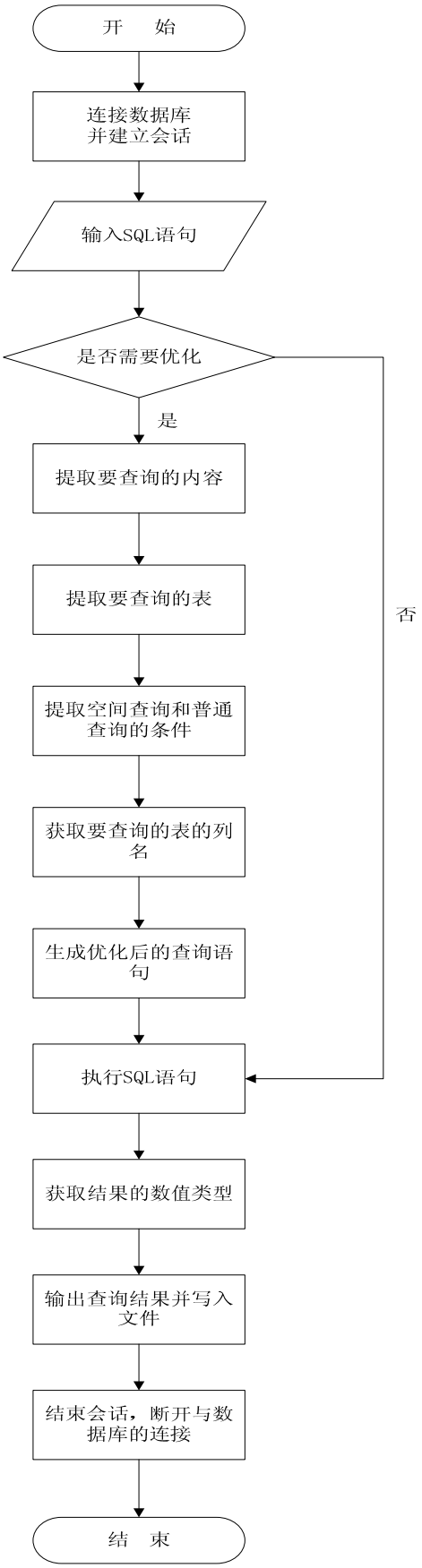

开　始
连接数据库
并建立会话
输入SQL语句
是否需要优化
是
否
提取要查询的内容
提取要查询的表
提取空间查询和普通查询的条件
获取要查询的表的列名
生成优化后的查询语句
执行SQL语句
获取结果的数值类型
输出查询结果并写入文件
结束会话，断开与数据库的连接
结　束

图 4-4　　程序流程图

4.4 空间查询优化程序的实现

4.4.1 SQL 语句的分析与优化

由于 SQL 语句良好的语法结构和空间查询函数的结构特点，因此很容易将空间查询的 SQL 语句进行分析，例如，可以很清楚的知道所要查询的内容在 select——from 字段之间，所查询的表在 from——where 之间，查询的约束条件在 where 字段之后；而约束条件的区分则依靠空间查询函数的特点——以“sdo_xxx”作为函数的开始。在分析过程中，由于要进行字符串的匹配，为了提高匹配的效率，在匹配过程中采用了 KMP 算法[16]。

首先，判定输入的 SQL 语句是否需要进行优化。若在 SQL 语句的 where 之后没有发现“and ”或者出现“and ”但没有找到“sdo_”字段，则该 SQL 语句不需优化，直接运行即可。之所以不需要优化是因为没有发现“and ”字段，说明查询条件只有一个；出现“and ”但没有找到“sdo_”字段说明约束条件中没有空间条件的约束，因此也不必进行优化。

然后，对查询的列名进行提取。具体的提取方法为先找到所要提取的字段的区间，即 select 字符串与 from 字符串之间的字段，这通过 KMP 字符串匹配算法实现。然后，通过“，”来标识一个列名的结束，提取全部列名直至整个区间结束。在提取的过程中需要注意的问题是，当列名为“a.name”的形式时，需要将“.”忽略，即将“a.name”提取成“aname”，但是保留小数点和诸如“sdo_geom.sdo_area”之类的空间函数中的“.”。

再次，对要查询的表名进行提取，即针对 from——where 字段进行分析。具体的方法也是先找到表名所在字段的区间——from 字符串与 where 字符串之间。然后通过“，”来标识一个完整的表名的结束，逐个提取表名存入字符数组中，直至所有表名都被存储。此外，如果查询语句对表进行了重命名，程序会区分出表名和对表的重命名，分别记入对应的字符数组中，如“Land area”，Land 为数据库中表的名字，area 为对 Land 的重命名，则在提取过程中将“Land”和“area”分别存放到结构体 Table 中的 table_name 和 table 中。

最后是对空间约束条件和非空间约束条件的提取，即针对“where”之后的字段的提取。由于空间函数都具备“sdo_”的结构特点，以及可作为分界符的“and”字段，因此可以很好的区分出空间约束条件与非空间约束条件。当约束条件为空间查询的约束条件时，存入存储空间查询的字符数组中，当约束条件为非空间查询约束条件，存入存储非空间查询的字符数组中。在这个过程中需要注意的问题是，在找到空间约束条件后，同样需要将除小数点和诸如“sdo_geom.sdo_area”之类的空间函数中的“.”之外的“.”忽略掉。

至此，空间查询的 SQL 语句就已经分析完，下一步工作需要生成优化的 SQL

语句。

4.4.2 优化语句的生成

优化语句的生成分两步执行。生成优化语句的第一步是根据普通一维查询的约束条件，将所要查询的表的符合约束条件的所有内容存入到临时表中。临时表的创建的语法可为：

- CREATE GLOBAL TEMPORARY <TABLE_NAME> (<column specification>) ON COMMIT PRESERVE ROWS
- CREATE GLOBAL TEMPORARY <TABLE_NAME> (<column specification>) ON COMMIT DELETE ROWS

两者的不同在于 ON COMMIT DELETE ROWS 说明临时表是事务指定，每次提交后 Oracle 将删除全部数据，而 ON COMMIT DELETE ROWS 说明临时表是会话指定，当中断会话时 Oracle 将删除全部数据。在优化程序中，采用了第一种创建临时表的方法，即：

CREATE GLOBAL TEMPORARY table_temp as select <column_name> from <table_name> where <ordinary_condition>

由于空间查询常常需要多个表进行联合，因此为了防止出现列名相同的情况，在生成创建临时表的 SQL 语句时，需要对列名进行重命名。重命名的规则为：该表的重命名+列名，如当表 Land 重命名为 area 时，列名 owename 将被重命名为 areaowename，当无重命名的表时，无需重命名。对列的重命名通过调用 OCI 提供的 OCIDescribeAny()函数实现，该函数可以获取表的各个字段的描述信息，因此用该函数获得每个表的各个字段的列名，并加上表的别名，即实现了对列的重命名。至此，生成优化语句的第一步完成。

生成优化语句的第二步是生成可以在创建的临时表中，根据空间查询的约束条件，查询出所要查询的内容的 SQL 语句。这只需将分析 SQL 语句时，提取出的查询内容和空间查询条件加入到 SQL 语句中，根据查询的内容和空间查询条件对临时表进行查询，找到所需内容即可完成。生成的语句的形式为：

SELECT <column_name> FROM table_temp WHERE <spatial_condition>

4.4.3 提交优化后的 SQL 语句

优化工作的最后一步为将优化后的 SQL 语句提交给数据库执行，显示查询结果并将结果写入到文件中。在输出的过程中，必须知道每一列的类型才可以输出结果。类型的明确通过调用 OCI 提供的 OCIAttrGet()函数实现，将每一列的类型记录到对应的标识列类型的数组中，在输出时，根据列的类型将相应的值写入用于输出的变量中，显示在屏幕上、存储在文件中。整个优化工作完成，结束会话，断开与数据库的连接，程序结束。

4.4.4 查询优化实例

例如，有一张统计个人土地数量及面积的表 Land(owner CHAR，landId NUMBER，landArea SDO.GEOMETRY)，现需要查询张三所拥有的面积小于 1000 平方米的土地，则查询语句应为：

select a.landId，sdo_geom.sdo_area(a.landArea,0.05) myarea from Land a where a.owner=’张三’ and sdo_geom.sdo_area(a.landArea,0.05)<1000

经过优化程序优化后，提交给数据库的 SQL 语句则为：

create global temporary table table_temp on commit preserve rows as select a.owner as aowner，a.landId as alandId，a.landArea as alandArea from Land a where a.owner=’张三’；

select alndId, sdo_geom.sdo_area(alandArea,0.005) myarea from table _temp where sdo_geom.sdo_area(alandArea,0.05)<1000

第五章　优化结果及结果分析

5.1　测试所用数据

本程序的测试需要大量的数据作为基础。如果数据量太少，查询消耗的时间不会太多，可能会造成 Oracle 提供的统计时间的工具的精确度不够，或者创建临时表的时间多于优化查询后节省的时间，从而影响优化的效果。只有数据达到一定的数量，才可以很好地判断出优化程序是否起到应有的作用。

本测试中用的数据为一张记录美国国家公园相关信息的名为 US_PARKS 的表[17]，其中了包含 6371 条数据，表的结构如下：

US_PARKS(ID NUMBER，NAME VARCHAR2(30)，FCC VARCHAR2(3)，GEOM MDSYS.SDO_GEOMETRY)

5.2　程序运行说明

程序经正确编译，可以运行后，首先连接数据库、建立会话，成功连接数据库、建立会话后，会提示输入 SQL 语句，并以“；”作为语句结束的标志，如图 5-1 所示：

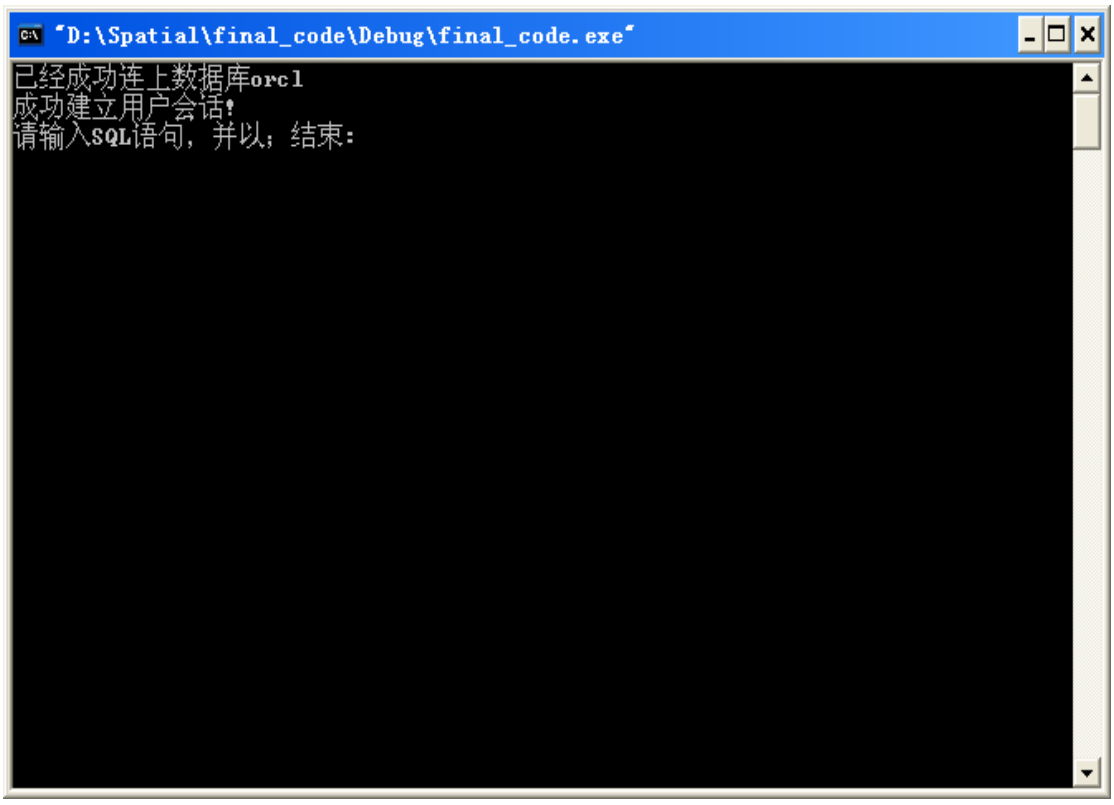


图 5-1　程序运行

接下来，输入 SQL 语句，如果不符合优化的条件，会提示“无需优化”，然后执行 SQL 语句，如图 5-2 所示：

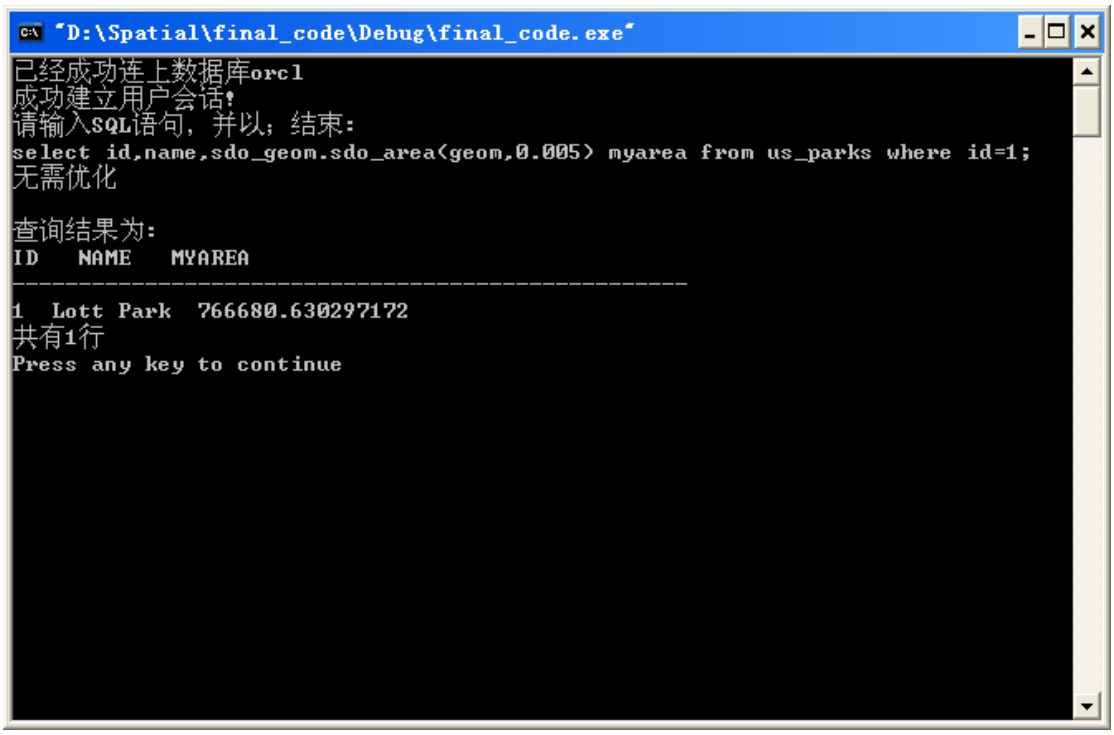


图 5-2　　无需优化的执行情况

如果符合优化的条件，则程序会如图 5-3 执行：

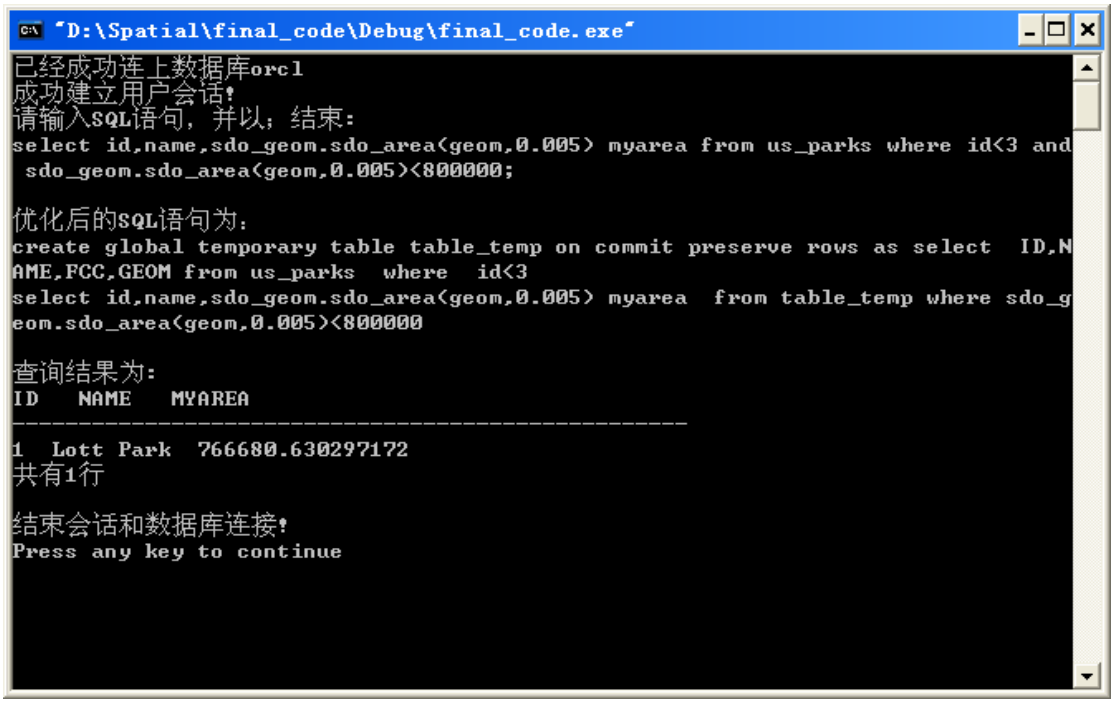


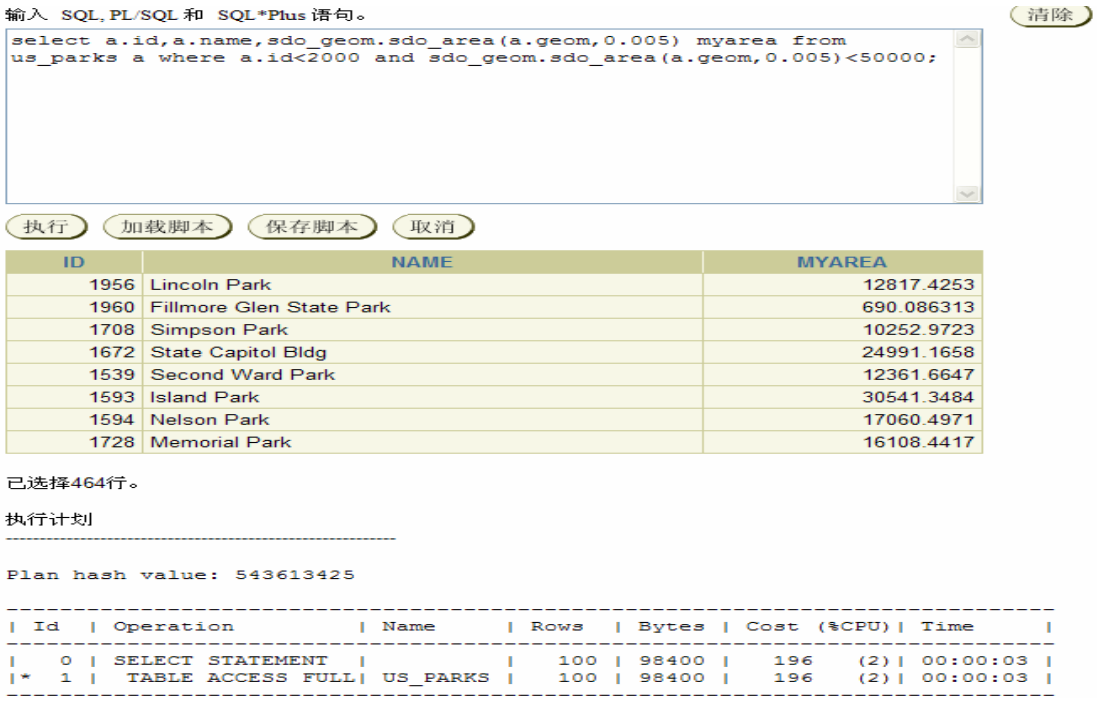
图 5-3　　优化时的执行情况

5.3　结果分析

由于 Oracle 提供了很好的工具——AUTOTRACE 作为查看包括执行时间在

内的执行计划及统计信息，所以，在对结果分析时，采用将原语句和优化后的语句交给 isqlplus 执行执行。如果优化后的语句耗时较少，则说明优化程序达到了提高查询效率、减少时间的作用。

原 SQL 语句以及通过 isqlplus 执行后的执行计划如图 5-4。



输入 SQL, PL/SQL 和 SQL*Plus 语句。 清除

```
select a.id,a.name,sdo_geom.sdo_area(a.geom,0.005) myarea from
us_parks a where a.id<2000 and sdo_geom.sdo_area(a.geom,0.005)<50000;
```

执行 加载脚本 保存脚本 取消

ID	NAME	MYAREA
1956	Lincoln Park	12817.4253
1960	Fillmore Glen State Park	690.086313
1708	Simpson Park	10252.9723
1672	State Capitol Bldg	24991.1658
1539	Second Ward Park	12361.6647
1593	Island Park	30541.3484
1594	Nelson Park	17060.4971
1728	Memorial Park	16108.4417

已选择464行。

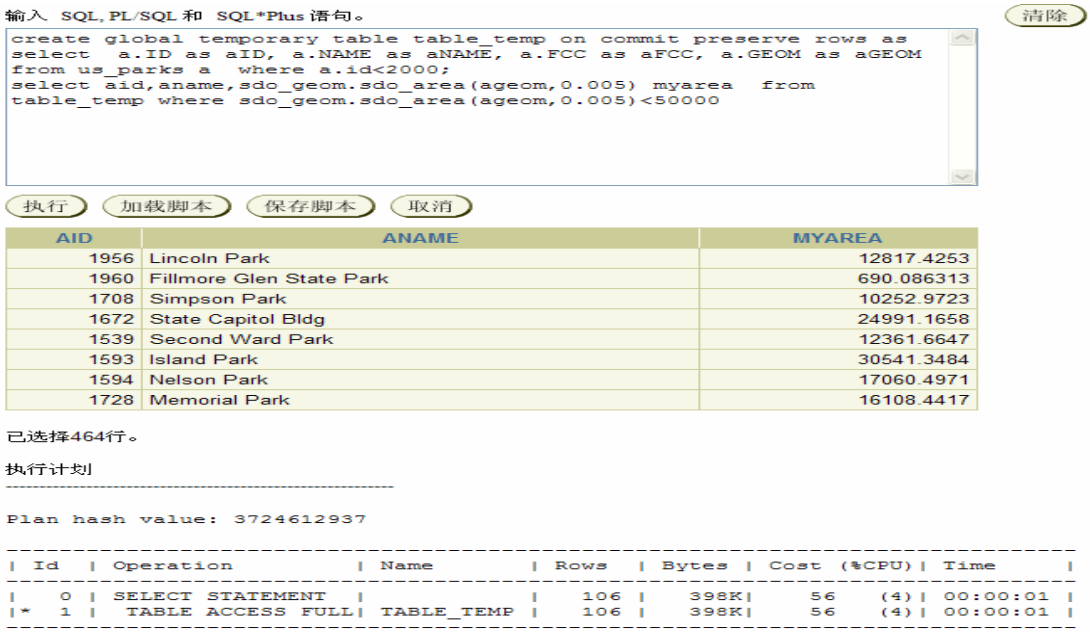
执行计划

```
Plan hash value: 543613425

--------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |   100 | 98400 |   196   (2)| 00:00:03 |
|*  1 |  TABLE ACCESS FULL| US_PARKS  |   100 | 98400 |   196   (2)| 00:00:03 |
--------------------------------------------------------------------------------
```

图 5-4　　未优化的 SQL 语句和执行计划

经过优化后的 SQL 语句和通过 isqlplus 执行后的执行计划如图 5-5



输入 SQL, PL/SQL 和 SQL*Plus 语句。 清除

```
create global temporary table table_temp on commit preserve rows as
select  a.ID as aID, a.NAME as aNAME, a.FCC as aFCC, a.GEOM as aGEOM
from us_parks a  where a.id<2000;
select aid,aname,sdo_geom.sdo_area(ageom,0.005) myarea  from
table_temp where sdo_geom.sdo_area(ageom,0.005)<50000
```

执行 加载脚本 保存脚本 取消

AID	ANAME	MYAREA
1956	Lincoln Park	12817.4253
1960	Fillmore Glen State Park	690.086313
1708	Simpson Park	10252.9723
1672	State Capitol Bldg	24991.1658
1539	Second Ward Park	12361.6647
1593	Island Park	30541.3484
1594	Nelson Park	17060.4971
1728	Memorial Park	16108.4417

已选择464行。

执行计划

```
Plan hash value: 3724612937

--------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            |   106 |  398K|    56   (4)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| TABLE_TEMP |   106 |  398K|    56   (4)| 00:00:01 |
--------------------------------------------------------------------------------
```

图 5-5　　优化后的 SQL 语句和执行计划

通过执行计划的对比，可以明显的发现，未经过优化的空间查询语句查找所需的内容所需时间为 6 秒，明显长于经过优化的空间查询语句所需的时间 2 秒，因此可以看出优化程序确实起到了提高查询效率、减少查询时间的作用。

第六章　总结与展望

6.1　总结

本文主要设计了一个用于提高空间查询效率、减少空间查询时间的 SQL 语句查询优化中间件。优化原理建立在 Oracle 对空间查询已有的优化机制——双重查询模型的基础上：双重查询模型执行两次过滤，第一次过滤进行粗略的计算，只得出近似值，减少计算时间；二次过滤进行精确计算，得出所需结果。所以，优化的主要针对目标为一次过滤。通过将空间查询中的非空间约束条件和空间约束条件分开，使得查询语句变为非空间查询和空间查询两部分，使用非空间查询减少一次过滤所需要的计算量——以达到减少查询时间的目的，最后使用空间查询得到需要的结果。正是基于上述的原理，编写了这个优化查询中间件，经过实验验证，确实起到了提高空间查询效率、减少空间查询时间的作用。

6.2　展望

虽然优化程序起到了提高空间查询效率、减少空间查询时间的作用，但是程序只是针对一般情况而言。在一些情况下并不一定能提高空间效率。例如，当要查询的表中的数据量并不是太大，那么执行非空间查询、创建临时表的时间就有可能超过通过减少一次过滤的数据量所节省下来的时间，这种情况下再执行优化程序就得不偿失了。又如，当查询涉及到两个表时，需要将两个表进行联合，如果联合后的数据量过大，反而会增加一次过滤时需要计算的数据量，这反而会降低空间查询的效率。总之，要根据实际的情况，如表的数据量的大小、两个表联合后的数据量的大小等，选择是否采用优化程序。

虽然空间查询优化技术目前并不完善，但随着空间数据日益广泛的应用、人们对空间查询效率的越来越高的要求以及对空间查询效率研究的日趋深入，一定会有更好的空间查询优化技术出现。

参考文献

[1]潘农菲. 基于Oracle Spatial的GIS空间数据处理及应用系统开发[J].计算机工程，2002，28（2）：278—280.

[2]王宝武，董慧君. 基于 Oracle Spatial 的 web 实现方案[J].信息技术与信息化，2007，05：123—125.

[3]梁鸿，丁仁伟，郑红霞. Oracle Spatial 空间数据库的设计及应用[J].测绘科学，2005，30（3）：91—94.

[4]雷英杰，王涛. 一种 Oracle 空间数据库的设计与实现[J].计算机工程与应用，2003，13：201—202.

[5]朱儒明，邓长春. 空间数据库中的优化空间查询算法研究[J].微计算机信息，2008，05:183—184.

[6]魏金占，邓青. Oracle Spatial 的空间查询分析功能浅析[J].信息技术与信息化，2006，02：99—101.

[7]李天琦，韦春桃，李天杰. 基于 Oracle 10g Spatial 空间数据库的索引与查询[J].桂林工学院学报，2008，28（1）：131—135.

[8]芮小平，张彦敏. Oracle Spatial 几何类型字段解析[J].物探化探计算技术，2004，26（4）：359—363.

[9]熊雷．基于 Oracle Spatial 的空间查询技术及其优化[J].软件导刊，2008，7（8）：7－9.

[10]方裕，楚放. 空间查询优化[J].中国图象图形学报，2001，6（4）：307—314.

[11]姚顺彬. 基于 Oracle Spatial 的空间按分析性能优化研究[J].林业资源管理，2007，01：87—90.

[12]孙杨，任鸿.Oracle 高级编程[M].北京：清华大学出版社，2007.590—597.

[13]张峻华. 采用 Oracle Spatial 基于 OCI 接口存取空间数据[J].电脑编程技巧与维护，2005，04：58—61.

[14]何雄等．Oracle Spatial 与 OCI 高级编程[M]．北京：中国铁道出版社，2006.293—317.

[15]何原荣，李全杰，傅文杰.Oracle Spatial 空间数据库开发应用指南[M].北京：测绘出版社,2008.237—252.

[16]严蔚敏，吴伟民.数据结构（C 语言版）[M].北京：清华大学出版社，2007.79—84.

[17]管会生，刘刚，安宁译．Oracle Spatial 空间信息管理——Oracle Database 11g[M]．北京：清华大学出版社，2009.

致　　谢

谨向所有对我的学习和写作过程中提供帮助和支持的人表示衷心的感谢。首先要衷心地感谢我的指导教师张坤老老师。在论文写作过程中，张老师对文章总体结构安排、分析方法、论文的字体格式等方面都提出了许多宝贵意见。虽然论文经过了多次修改，但每次张老师都不厌其烦地仔细阅读并提出修改建议。他的谆谆教诲和一丝不苟的科研精神将使我受用终身。

同时，要感谢天津大学多年的培养，感谢曾经教育和帮助过我的所有老师和同学。感谢百忙之中抽出时间参加论文评阅和评议的各位专家学者，感谢他们为审阅本文所付出的辛勤劳动。我向他们表示诚挚的谢意。

毕业设计为我的大学本科生活画上了一个完整的句号，最后谢谢所有帮助和支持我的老师、同学、朋友和亲人们。