Oracle9i 的查询优化

Oracle 白皮书

2002 年 2 月

ORACLE

Oracle9i 的查询优化

执行概要

本文描述了 Oracle 的查询优化程序，它是数据库的关键组件，能让 Oracle 的用户获得极佳的执行性能。Oracle 的查询优化技术在功能上无与伦比，本文详细讨论了查询优化的所有重要领域。

简介

什么是查询优化程序？

查询优化对于关系数据库的性能，特别是对于执行复杂 SQL 语句的性能而言至关重要。查询优化程序确定执行每一次查询的最佳策略。

例如，查询优化程序选择对于指定的查询是否使用索引，以及在联接多个表时采用哪一种联接技术。这类决策对 SQL 语句的执行性能有很大的影响，查询优化对于每一种应用程序都是关键技术，应用程序涉及的范围从操作系统到数据仓库，从分析系统到内容管理系统。查询优化程序对于应用程序和最终用户是完全透明的。

由于应用程序可能生成非常复杂的 SQL 语句, 查询优化程序必须精心构建、功能强大，以保障良好的执行性能。例如，查询优化程序可转换 SQL 语句，使复杂的语句转换成为等价的但执行性能更好的 SQL 语句。查询优化程序的典型特征是基于开销。在基于开销的优化策略中，对于给定查询生成多个执行计划，然后对每个计划估算开销。查询优化程序选用估算开销最低的计划。

Oracle 在查询优化方面提供了什么?

Oracle 的优化程序可称是业界最成功的优化程序。基于开销的优化程序自 1992 年随 Oracle7 推出后，通过 10 年的丰富的实际用户经验，不断得到提高和改进。好的查询优化程序不是基于纯粹的理论假设及谓词在实验室中开发出来的，而是通过适合实际用户需求开发和磨合出来的。

Oracle 的查询优化程序比任何其他查询优化程序在数据库应用程序的应用都要多，而且 Oracle 的优化程序一直由于实际应用的反馈而得到改进。

Oracle 的优化程序包含 4 大主要部分（本文将在以下章节详细讨论这些部分）：

- SQL 语句转换：在查询优化中 Oracle 使用一系列精深技术对 SQL 语句进行转换。查询优化的这一步骤的目的是将原有的 SQL 语句转换成为语义相同而处理效率更高的 SQL 语句。
- 执行计划选择：对于每个 SQL 语句，优化程序选择一个执行计划（可使用 Oracle 的EXPLAIN PLAN 工具或通过 Oracle 的 “v$sql_plan” 视图查看）。执行计划描述了执行 SQL 时的所有步骤，如访问表的顺序；如何将这些表联接在一起；以及是否通过索引来访问这些表。优化程序为每个 SQL 语句设计许多可能的执行计划，并选出最好的一个。
- 开销模型与统计：Oracle 的优化程序依赖于执行 SQL 语句的所有单个操作的开销估算。想要优化程序能选出最好的执行计划，需要最好的开销估算方法。开销估算需要详细了解某些知识，这些知识包括：明白每个查询所需的 I/O、CPU 和内存资源以及数据库对象相关的统计信息（表、索引和物化视图），还有有关硬件服务器平台的性能信息。收集这些统计和性能信息的过程应高效并且高度自动化。
- 动态运行时间优化：并不是 SQL 执行的每个方面都可以事先进行优化。Oracle 因此要根据当前数据库负载对查询处理策略进行动态调整。该动态优化的目标是获得优化的执行性能，即使每个查询可能不能够获得理想的 CPU 或内存资源。Oracle 另有一个原来的优化程序，即基于规则的优化程序。该优化程序仅向后兼容，在 Oracle 的下个版本将不再得到支持。绝大多数 Oracle 用户目前使用基于开销的优化程序。所有主要的应用程序供应商（如 Oracle 应用程序、SAP 和 Peoplesoft，仅列出这几家）以及大量近来开发的客户应用程序都使用基于开销的优化程序来获得优良的执行性能，故本文仅讲述基于开销的优化程序。

SQL 语句转换

使用 SQL 语句表示复杂查询可以有多种方式。提交到数据库的 SQL 语句类型通常是最终用户或应用程序可以最简单的方式生成的 SQL 类型。但是这些人工编写或机器生成的查询公式不一定是执行查询最高效的 SQL 语句。例如，由应用程序生成的查询通常含有一些无关紧要的条件，这些条件可以去掉。或者，有些从某查询谓词出的附加条件应当添加到该 SQL 语句中。SQL 转换语句的目的是将给定的 SQL 语句转换成语义相同（即返回相同结果的 SQL 语句）并且性能更好的 SQL 语句。

所有的这些转换对应用程序及最终用户完全透明。SQL 语句转换在查询优化过程中自动实现。

Oracle 实现了多种 SQL 语句转换。这些转换大概可分成两类：

试探查询转换：在可能的情况下对进来的 SQL 语句都会进行这种转换。这种转换能够提供相同或较好的查询性能，所以 Oracle 知道实施这种转换不会降低执行性能。基于开销的查询转换：Oracle 使用基于开销的方法进行几类查询转换。借助这种方法，转换后的查询会与原查询相比较，然后 Oracle 的优化程序从中选出最佳执行策略。

以下部分将讨论 Oracle 转换技术的几个示例。这些示例并非是权威的，仅用于帮助读者理解该关键转换技术及其益处。

试探查询转换

简单视图合并

可能最简单的查询转换是视图合并。对于包含视图的查询，通常可以通过把视图定义与查询“合并”来将视图从查询中去掉。例如，请看下面的非常简单的视图及查询。

```
CREATE VIEW TEST_VIEW AS
SELECT ENAME, DNAME, SAL FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO;

SELECT ENAME, DNAME FROM TEST_VIEW WHERE SAL > 10000;
```

如果不加任何转换，处理该查询的唯一方法是将 EMP 的所有行联接到 DEPT 表的所有行，然后筛选有适当的 SAL 的值的那些行。

如果使用视图合并，上述查询可以转换为：

```
SELECT ENAME, DNAME FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND E.SAL > 10000;
```

处理该转换后的查询时，可以在联接 EMP 和 DEPT 表前使用谓词 'SAL>10000'。这一转换由于减少了联接的数据量而大大提高了查询的执行性能。即便在这样一个非常简单的示例里，查询转换的益处和重要性也显而易见。

复杂视图合并

许多视图合并操作都是直截了当的，如以上示例。但是，较复杂的视图，如包含 GROUP BY 或 DISTINCT 操作符的视图合并起来就不那么容易了。Oracle 为合并这类复杂视图提供了一些高级技术。

请看以下带有 GROUP BY 语句的视图。在该示例中，视图计算每个部门的平均工资。

```
CREATE VIEW AVG_SAL_VIEW AS
SELECT DEPTNO, AVG(SAL) AVG_SAL_DEPT FROM EMP
GROUP BY DEPTNO
```

查询的目的是要找出 Oakland 每个部门的平均工资：

```
SELECT DEPT.NAME, AVG_SAL_DEPT
FROM DEPT, AVG_SAL_VIEW
WHERE DEPT.DEPTNO = AVG_SAL_VIEW.DEPTNO
AND DEPT.LOC = 'OAKLAND'
```

可以转换为：

```
SELECT DEPT.NAME, AVG(SAL)
FROM DEPT, EMP
WHERE DEPT.DEPTNO = EMP.DEPTNO
AND DEPT.LOC = 'OAKLAND'
GROUP BY DEPT.ROWID, DEPT.NAME
```

该特殊转换的执行性能优点立即显现：该转换把 EMP 数据在分组聚合前进行联接和筛选，而不是在联接前将 EMP 表的所有数据分组聚合。

子查询“展平”

Oracle 有一些转换能将不同类型的子查询转变为联接、半联接或反联接。作为该领域内的技术示例，我们来看下面这个查询，找出有工资超过 10000 的员工的那些部门：

```
SELECT D.DNAME FROM DEPT D WHERE D.DEPTNO IN
(SELECT E.DEPTNO FROM EMP E WHERE E.SAL > 10000)
```

存在一系列可以优化本查询的执行计划。Oracle 会考虑这些可能的不同转换，基于开销选出最佳计划。

如果不进行任何转换，这一查询的执行计划如下：

```
OPERATION            OBJECT_NAME   OPTIONS
SELECT STATEMENT
   FILTER
      TABLE ACCESS   DEPT          FULL
      TABLE ACCESS   EMP           FULL
```

按照该执行计划，将扫描 DEPT 表的每一行查找所有满足子查询条件的 EMP 记录。通常，这不是一种高效的执行策略。然而，查询转换可以实现效率更高的计划。

该查询的可能计划之一是将查询作为“ 半联接 ”来执行。 “ 半联接 ”是一种特殊类型的联接，它消除了联接中来自内表的冗余值（这实际上就是该子查询的原本的语义）。在该示例中，优化程序选择了一个散列半联接，尽管 Oracle 也支持排序-合并以及嵌套-循环半联接：

```
OPERATION            OBJECT_NAME   OPTIONS
SELECT STATEMENT
   HASH JOIN                       SEMI
      TABLE ACCESS   DEPT          FULL
      TABLE ACCESS   EMP           FULL
```

由于 SQL 没有用于半联接的直接语法，此转换过的查询不能使用标准的 SQL 来表示。但是，转换后的伪 SQL 将是：

```
SELECT DNAME FROM EMP E, DEPT D
WHERE D.DEPTNO <SEMIJOIN> E.DEPTNO
AND E.SAL > 10000;
```

另一个可能的计划是优化程序可以决定将 DEPT 表作为联接的内表。在这种情况下，查询作为通常的联接来执行，但对 EMP 表进行特别排序，以消除冗余的部门号：

```
OPERATION            OBJECT_NAME   OPTIONS
SELECT STATEMENT
   HASH JOIN
      SORT                         UNIQUE
      TABLE ACCESS   EMP           FULL
      TABLE ACCESS   DEPT          FULL
```

转换后的 SQL 语句为：

```
SELECT D.DNAME FROM (SELECT DISTINCT DEPTNO
FROM EMP) E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND E.SAL > 10000;
```

与视图合并一样，子查询展平也是获得良好查询执行性能的基本优化办法。

传递谓词生成

在某些查询中，由于表间的联接关系，一个表中的谓词可以转化为另一个表中的谓词。Oracle 会以这种方式演绎出新的谓词，这类谓词被称为传递谓词。例如，来看一个查询，找出定货当天运出的所有商品：

```
SELECT COUNT(DISTINCT O_ORDERKEY)
FROM ORDER, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE = L_SHIPDATE
AND O_ORDERDATE BETWEEN '1-JAN-2002'
AND '31-JAN-2002'
```

利用传递性，该 ORDER 表中的谓词也可以用于 LINEITEM 表：

```
SELECT COUNT(DISTINCT O_ORDERKEY)
FROM ORDER, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE = L_SHIPDATE
AND O_ORDERDATE BETWEEN '1-JAN-2002'
AND '31-JAN-2002'
AND L_SHIPDATE BETWEEN '1-JAN-2002'
AND '31-JAN-2002'
```

新谓词的存在可以减少要联接的数据量，并能够利用另外的索引。

消除通用子表达

如果同样的子表达或计算在一个查询中出现多次，Oracle 对每一行只评估一次该表达式。

请看以下查询，找出 Dallas 身为副总裁或工资超过 100000 的所有员工。

```
SELECT * FROM EMP, DEPT
WHERE
(EMP.DEPTNO = DEPT.DEPTNO AND LOC = 'DALLAS'
AND SAL > 100000)
OR
(EMP.DEPTNO = DEPT.DEPTNO AND LOC = 'DALLAS'
AND JOB_TITLE ='VICE PRESIDENT'}
```

优化程序明白如果转换成如下语句时，该查询将更为高效：

```
SELECT * FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO AND
LOC = ‘DALLAS’ AND
(SAL > 100000 OR JOB_TITLE = 'VICE PRESIDENT');
```

使用此转换后的查询，联接谓词和 LOC 的谓词对 DEPT 的每行仅评估一次，而不是两次。

谓词下推和上移

复杂查询往往包含多个视图与子查询，在这些视图和子查询中包含多个谓词。Oracle 可将谓词移入或移出视图，以产生新的高效查询。

可使用单一表视图来说明谓词下推：

```
CREATE VIEW EMP_AGG AS
  SELECT
    DEPTNO,
    AVG(SAL) AVG_SAL,
  FROM EMP
  GROUP BY DEPTNO;
```

假设进行下列查询：

```
SELECT DEPTNO, AVG_SAL FROM EMP_AGG
WHERE DEPTNO = 10;
```

Oracle 会将谓词 DEPTNO=10 推进该视图，将查询转换为下列 SQL：

```
SELECT DEPTNO, AVG(SAL)5 FROM EMP
WHERE DEPTNO = 10
GROUP BY DEPTNO;
```

此已转换的查询的好处是谓词 DEPTNO=10 在 GROUP-BY 操作前进行，从而大大减少了要处理的数据量。

Oracle 还有许多其他优良技术来将 WHERE 子句的条件下推到外来的查询块中，将条件从查询块中提出，并且将条件在联接的查询块间平移。一旦有 WHERE 子句条件出现，就有机会筛选出一些行并减少早期阶段要处理的数据量。这样，后续的操作，比如说联接或 GROUP-BY，就可以只处理小得多的数据集，从而提高了执行效率。还有，谓词下推和上移通过产生新的访问路径以提高执行性能，这在没有增加新的谓词前不可能做到。

用于“CUBE”查询的分组修整

SQL CUBE 表达式是 SQL group-by 运算符的扩展，使得可以用单个 SQL 语句就可以检索多个集合。对于包含带有 CUBE 表达式的视图的查询，有时可以减少评估该查询所需要的数据量。例如，请看以下查询：

```
SELECT MONTH, REGION, DEPARTMENT FROM
(SELECT MONTH, REGION, DEPARTMENT,
   SUM(SALES_AMOUNT) AS REVENUE FROM SALES
   GROUP BY CUBE (MONTH, REGION, DEPT))
WHERE MONTH = ‘JAN-2001’;
```

该查询可以转换为下列 SQL：

```
SELECT MONTH, REGION, DEPARTMENT FROM
(SELECT MONTH, REGION, DEPARTMENT,
   SUM(SALES_AMOUNT) AS REVENUE FROM SALES
   WHERE MONTH = ‘JAN-2001’
   GROUP BY MONTH, CUBE(REGION, DEPT))
WHERE MONTH = ‘JAN-2001’;
```

该转换后的 SQL 处理的数据集合大大减少，因为要计算的数据大大缩减了（只需计算 2001 年一月的数据），并且需合计的数量也减少了。对开发分析类应用程序的 SQL 程序员而言这是重要的一种转换，因为这些工具可能要对逻辑上的立方体进行查询，这些立方体在包含有 CUBE 运算符的视图中定义。

外联接到内联接的转换

在某些情况下，能够确定查询中的一个外联接能产生与内联接相同的结果。在这类情况下，优化程序会将外联接转变为内联接。这种转换让 Oracle 能够进一步合并视图或选用新的联接顺序，查询是一个外联接时就做不到这一点。

基于开销的查询转换

实体化视图重写

以实体化视图的形式预先处理和存储常用数据能够大大加速查询处理。Oracle 能对 SQL 查询进行转换，使查询中对一个或多个表的引用被对一个实体化视图的引用所取代。如果该实体化视图小于原来要引用的表，或比原来要引用的表有更好的访问路径，则该转换后的 SQL 语句比原查询的执行速度会快得多。

例如，请看以下实体化视图：

```
CREATE MATERIALIZED VIEW SALES_SUMMARY
AS SELECT SALES.CUST_ID, TIME.MONTH,
SUM(SALES_AMOUNT) AMT
FROM SALES, TIME
WHERE SALES.TIME_ID = TIME.TIME_ID
GROUP BY SALES.CUST_ID, TIME.MONTH;
```

该实体化视图能够用于优化以下查询：

```
SELECT CUSTOMER.CUST_NAME, TIME.MONTH,
SUM(SALES.SALES_AMOUNT)
FROM SALES, CUSTOMER, TIME
WHERE SALES.CUST_ID = CUST.CUST_ID
AND SALES.TIME_ID = TIME.TIME_ID
GROUP BY CUSTOMER.CUST_NAME, TIME.MONTH;
```

重写后的查询为：

```
SELECT CUSTOMER.CUST_NAME, SALES_SUMMARY.MONTH,
SALES_SUMMARY.AMT
FROM CUSTOMER, SALES_SUMMARY
WHERE CUSTOMER.CUST_ID = SALES_SUMMARY.CUST_ID;
```

在该示例中，转换后的查询速度可能快得多的原因是：销售汇总表可能比销售表小得多，同时转换后的查询少一次联接，而且无须计总。

Oracle 具有一系列强有力的实体化视图重写技术，允许每个实体化视图能被用在尽可能多的查询类型中。

Oracle 实体化视图的另一个显著特点是与 Oracle 数据库中的声明式维集成在一起。Oracle 可以允许生成维的元数据对象，描述维内的层次关系。该层次化元数据用来支持更为复杂的实体化视图查询重写。例如，只要存在描述月与季度之间的层次关系的时间维，一个包含月销售数据的实体化视图就能支持对季度销售数据的查询。

注意，使用实体化视图的转换后的查询并不总是比原查询效率高。因为，就算实体化视图比其基于的表小，但这些基础表可以有更好的索引，从而能够被更快地访问。选择优化执行计划的唯一途径是对使用和未使用实体化视图的执行计划进行计算并比较其开销。Oracle 正是这样做的，所以实体化视图重写是基于开销的查询转换范例。（有关实体化视图的更多信息，参见白皮书“ Oracle9*i* 实体化视图 ”）。

OR 扩展

该技术把在 WHERE 子句中带有 OR 的查询转换成一个包含多个不带 OR 的查询的 UNION ALL。当 OR 表示的是对不同表的限制条件时，这是大有好处的。请看以下查询，找出所有进出 Oakland 的运货：

```
SELECT * FROM SHIPMENT, PORT P1, PORT P2
WHERE SHIPMENT.SOURCE_PORT_ID = P1.PORT_ID
AND SHIPMENT.DESTINATION_PORT_ID = P2.PORT_ID
AND (P1.PORT_NAME = 'OAKLAND'
OR P2.PORT_NAME = 'OAKLAND')
```

该查询可以转换为：

```
SELECT * FROM SHIPMENT, PORT P1, PORT P2
WHERE SHIPMENT.SOURCE_PORT_ID = P1.PORT_ID
AND SHIPMENT.DESTINATION_PORT_ID = P2.PORT_ID
AND P1.PORT_NAME = 'OAKLAND'
UNION ALL
SELECT * FROM SHIPMENT, PORT P1, PORT P2
WHERE SHIPMENT.SOURCE_PORT_ID = P1.PORT_ID
AND SHIPMENT.DESTINATION_PORT_ID = P2.PORT_ID
AND P2.PORT_NAME = 'OAKLAND'
AND P1.PORT_NAME <> 'OAKLAND'
```

注意，每个 UNION ALL 分支可以有不同的优化的联接顺序。在第一个分支里，Oracle 可以利用对 P1 的限制，从该表开始联接。而在第二个分支里，Oracle 可以从 P2 开始。产生的计划可能比原来的查询快好几个数量级，这取决于表索引和其数据。这种查询转换必须是基于开销的，因为它并不保证每个查询的性能都会得到提高。

星型转换

星型方式是普遍用于数据中心或数据仓库的建模策略。星型模式一般包含一个或多个非常大的表（叫做事实表），用来存储交易数据，另外还包含大量较小的查找表（叫做维表），以来存放描述性数据。

Oracle 支持一种用于评估星型模式查询的技术（叫做 “ 星型转换 ”）。该技术通过进行转换（向原有的 SQL 中添加新的子查询）来提高星型查询的效率。

这些新的子查询允许利用位图索引更有效地访问事实表。

通过查看示例可以更好地理解这种星型转换。请看以下查询，它返回各州 2001 年三季度的饮料销售总量。事实表为销售。注意，此处的时间维是个“ 雪片 ”维，因为它包含 DAY 和 QUARTER 两个表。

```
SELECT STORE.STATE, SUM(SALES.AMOUNT)
FROM SALES, DAY, QUARTER, PRODUCT, STORE
WHERE SALES.DAY_ID = DAY.DAY_ID
AND DAY.QUARTER_ID =QUARTER.QUARTER_ID
AND SALES.PRODUCT_ID = PRODUCT.PRODUCT_ID
AND SALES.STORE_ID = STORE.STORE_ID
AND PRODUCT.PRODUCT_CATEGORY = 'BEVERAGES'
AND QUARTER.QUARTER_NAME = '2001Q3'
GROUP BY STORE.STATE
```

转换后的查询为：

```
SELECT STORE.STATE, SUM(SALES.AMOUNT)
FROM SALES, STORE
WHERE SALES.STORE_ID = STORE.STORE_ID
AND SALES.DAY_ID IN
(SELECT DAY.DAY_ID FROM DAY, QUARTER
WHERE DAY.QUARTER_ID = QUARTER.QUARTER_ID
AND QUARTER.QUARTER_NAME = '2001Q3')
AND SALES.PRODUCT_ID IN
(SELECT PRODUCT.PRODUCT_ID FROM PRODUCT
WHERE PRODUCT.PRODUCT_CATEGORY = 'BEVERAGES')
GROUP BY STORE.STATE
```

使用转换后的 SQL，该查询可以分为两个阶段有效处理。在第一阶段，借助位图索引从事实表中找出所有需要的行。在这种情况下，因为 DAY-ID 和 PRODUCT-ID 就是出现在子查询谓词中的那两列，所以按照它们产生的位图索引来访问事实表。

在查询的第二阶段（“ 联接回 ”阶段），维表被联接回到第一阶段的数据集。在该查询中，出现在选择列表中的唯一维表列是 store.state，因此 store 表是唯一一个需要联接的表。出现在查询第一阶段的包含 PRODUCT、DAY 和 QUARTER 的子查询消除了在第二阶段对这些表的联接的必要性，查询优化程序智能地去掉了这些联接。

星型转换是基于开销的查询转换，根据优化程序的开销估算决定是使用某个维的子查询开销较低还是查询重写比原有语句更好。

星型查询执行技术是 Oracle 独家拥有的专利技术。其他厂商也有类似的对星型查询的查询转换能力，但是没有一家厂商能将该技术与静态位图索引以及智能联接回功能结合在一起。

外联接视图的谓词下推

通常，当查询包含一个联接到其他表的视图时，该视图可以被合并，以实现更好的查询优化。但是，如果一个视图是用外联接方式联接的，则该视图就不能被合并。在这种情况下，Oracle 有一个特别的谓词下推操作，使联接谓词可以下推到该视图中，该转换使通过该视图中的某个表的索引执行外联接成为可能。这种转换是基于开销的，因为索引访问可能并不是最有效的。

选择访问路径

选择访问路径的目标是决定查询中联接表的次序、使用何种联接方法以及如何访问每个表中的数据。对于给定的查询，可以使用 Oracle 的 EXPLAIN PLAN 工具或 Oracle 的 v$sql_plan 视窗查看所有这些信息。

Oracle 的选择访问路径算法特别高深，因为 Oracle 提供了特别丰富的一套数据库结构和查询评估技术。Oracle 的选择访问路径和开销模型形成了对每一特性的全面理解，以能最佳地使用每个特性。

Oracle 的数据库结构包括：

表结构
表（缺省）
索引编排表
嵌套表
簇
散列簇

索引结构
B 树索引
位图索引
位图联接索引
反向关键字 B 树索引
基于功能的 B 树索引
基于功能的位图索引
域索引

分区技术
范围分区
散列分区
组合范围散列分区
列表分区
组合范围列表分区

Oracle 的访问技术包括：

索引访问技术
索引唯一关键字查找
索引最大/最小查找
索引范围扫描
索引降序范围扫描
索引全扫描
索引快速全扫描
索引跳扫
索引 and-equal 处理
索引联接
索引 B 树位图变换
位图索引 AND/OR 处理
位图索引范围处理
位图索引 MINUS (NOT) 处理
位图索引 COUNT 处理

联接方法
嵌套循环内联接、外联接、半联接以及反联接
排序合并内联接、外联接、半联接以及反联接
散列内联接、外联接、半联接以及反联接
智能化分区联接

本文不会就所有这些处理技术展开讨论，下面将讨论 Oracle 访问路径选择的一些关键属性。

联接排序

如果联接大量的表，所有可能的执行计划将占用很大的空间，优化程序要需要花费很长的时间来探究这些空间消耗。例如，对 5 个表的查询有 120 种可能的联接次序，基于各种索引、访问方法和联接技术的组合，每种联接次序又可能有几十种可能的执行计划。因此对 5 个表的查询的执行计划共有几千种，优化程序有可能选出最可能的执行计划。但是，对于有 10 个表的联接，存在逾 3 百万种的联接次序，通常就会有 1 亿多种可能的执行计划。因此，在寻找可能的执行计划时优化程序需要智取而非力夺。

适应式查找策略

Oracle 优化程序使用许多技术来智能地减少查找空间。值得关注的一种技术是 Oracle 采用了适应式查找策略。如果查询可在一秒钟内完成，用十秒种来进行查询优化就太浪费了。相反，如果一个查询多半要运行几分钟或是几小时，则为了找到一种更好的计划而在优化阶段花上几秒钟甚至几分钟也是值得的。Oracle 使用适应式优化算法确保在对复杂查询需要投入额外的优化时间时，优化时间仅是该查询预期执行时间的很少一部分（百分之几）。

有些数据库系统允许 DBA 定义一个“优化等级”，以控制用于查询优化的时间量。但是，相较于控制优化的系统级参数而言，适应式优化策略是更有效的一种技术。使用系统级参数时，DBA 不加区分地决定所有查询的优化等级，而 Oracle 的适应式优化策略为每个查询分别决定优化等级。

多重初始排序试探

Oracle 优化程序的另一个查找算法是创新性的“多重初始排序试探”。如果优化程序在查找过程的早期就找到最优的计划，优化程序会提前结束工作。这样，该试探借助高深的方法能即刻在查找空间中找到一些近于最优的、或至少是非常好的执行计划。优化程序从这些计划开始，而不是从随机生成的计划开始。该试探对于高效执行查询优化十分关键，因为它可大大减少查询优化所需要的时间。

位图索引

Oracle 创新性的位图索引（已获得专利）获得了广泛的应用，特别是在数据仓库应用方面。其他厂商提供的是“动态”位图索引，而 Oracle 支持真正的位图索引（除动态位图索引外）。真正的位图索引是一种索引结构，在数据库中按该结构存储压缩的用位图表示的索引，而动态位图索引是在查询处理过程中将数据库的 B 树索引转换成位图结构。真正的位图索引的好处显而易见，因为较之常规的 B 树结构，它们可以节约大量空间。对空间的节约又可以转化为性能上的好处，表现为磁盘 I/O 减少。真正的位图索引在处理许多查询时，速度能提高 10 倍，而使用的索引存储空间却为原来的十分之一。有关量化位图索引的好处的更多信息，请参见性能白皮书“Oracle9*i* 的关键数据仓库特性：比较分析”。

位图索引在评估结合 AND 和 OR 运算的多重谓词时特别有效。此外，位图索引使用 Oracle 的标准的一致性模型，以便在带有位图索引的表上与查询同步执行 DML 操作（插入、修改、删除）时，能完全保持数据的一致性。

Oracle 最丰富的位图索引功能为查询优化程序提供了许多新的执行策略。Oracle 的查询优化程序能够生成执行计划，这些计划包含位图运算复杂的树，这些位操作组合了对应于 WHERE 子句中的 AND、OR 和 NOT 条件的多种索引，不论是真正的位图索引，还是动态位图索引。这些位图的布尔运算非常快速，所以，只要查询能利用位图运算的巨大好处，通常情况下都能很好地执行。

此外，Oracle 支持对位图索引查询的“仅用索引”的访问方式。该“仅用索引”访问采用“AND（与）”位图的精确算法。这种精确性与那些采用动态位图索引并依赖对交叉行标识列表进行散列操作的数据库系统形成了鲜明的对比（参见 http://as400bks.rochester.ibm.com/cgi-bin/bookmgr/BOOKS/EZ30XB00/2.4.1）。即便使用动态位图索引，为保证正确的查询结果，这种系统也避免不了对表的访问。与之相反，Oracle 能在不访问表的情况下评估各种各样的查询。例如，假设有一家银行想要知道其在加利福尼亚州有多少已婚客户。

```
SELECT COUNT(*) FROM CUSTOMER
WHERE STATE = 'CA' AND MARITAL_STATUS = 'MARRIED'
```

假设有关于州和婚姻状态的位图索引，Oracle 就能简单地执行一个关于该位图的对‘CA’和‘MARRIED’的位级的‘与’运算，计算出结果位图中 1 的数目，这是一个非常快速高效的操作，因为整个查询的执行仅需访问两个高度压缩的索引结构。如果数据库系统不能仅通过索引得出该查询的结果，它就可能不得不去访问表中的上百万行，从而导至执行速度会慢许多。

Oracle 真正的位图索引的精确性（较之动态位图索引而言）对于 Oracle 的星型转换和位图联接索引的联接回减少特性而言，也是至关重要的。没有精确位图操作功能的数据库系统永远需要在联接回阶段联接所有的维表，而 Oracle 仅需要联接最少量的维表。

与其他类型的压缩索引结构相比，Oracle 的位图索引具有重要的优势。例如，在使用 Oracle 的位图索引访问一定范围的值时，查询优化程序将建立起始关键字和停止关键字，以保证只访问部分索引结构。这与有些索引形成了鲜明的对比，例如“编码向量索引”，在该方式下要想执行一个查询，会从头至尾扫描每个索引 一一种极浪费的做法。

Oracle 的查询优化程序能够将多种类型的索引组合起来访问同一个表。例如，一个位图索引可以与域索引和一个 B 树索引组合，访问一个给定的表。通过动态位图索引技术访问该域索引和 B 树索引用，以使它们能与位图索引组合在一起。

位图联接索引

“ 联接索引 ”是一种索引结构，能够跨越多个表，改善对这些表联接的执行性能。

联接索引是一种索引，其中索引列以及行标识/位图的表示涉及多个不同的表。因此，位图联接索引的一部分定义还将包括联接条件，指定如何匹配表中的行。通常，会给事实表产生一个联接索引，索引的列属于一个维表。例如，给定一个事实表，即销售表，一个维表，即产品表，于是就可以对销售表生成一个位图联接索引，但索引列是产品表中的产品类别，这是由于一个联接条件是产品标识。使用这样的索引，就可以直接从索引中找到每个产品目录里的销售行，而不需要进行联接。位图联接索引有两个主要优点：

1. 减少基数：多数情况下产品类别数会比产品标识少。所以，位图联接索引的基数会比任何基于事实表中的联接列的索引的基数少得多，对位图索引就变得更小。

2. 消除联接回：在许多情况下，采用位图联接索引可以消除查询中的联接。

可以通过组合多种索引的位图技术，在同一访问路径中一起使用位图联接索引与非联接索引。通常，由于星型转换与位图联接索引联合工作，因此会采用其他索引。

域索引及扩展性

Oracle 支持域索引，以高效地访问自定义的复杂数据类型（如文档、空间数据、图像和视频片段）。这类索引与内置的 Oracle 索引有所不同，但其属性可以在 Oracle 中登记。Oracle 优化程序是可扩展的，所以域索引和用户定义的功能可以有相应的统计数据和开销功能。它们和 Oracle 内置的索引功能有着同样的开销模型和查找空间，甚至能够通过 Oracle 位图技术，在同一访问路径中与常规的 Oracle 索引组合在一起。

快速全索引扫描

索引的快速全扫描是指能够将索引当作表来扫描，而不是像在索引中那样以树型顺序扫描。当索引中包含所有需要的表列时，就可以这样做，而不需要对表进行访问。在需要对大量数据进行存取时快速全扫描是很有用的，因为它们能够最大程度地利用多块磁盘 I/O 的优点，同时在并行化方面比范围扫描更好。因为索引通常比表小得多，扫描索引一般比扫描表本身要划算。

索引联接

在没有包含全部所需列的单一索引的情况下，索引联接能够用多个索引重新建立表列的一个子集。如果一组索引一起包含了全部所需的表列，优化程序就可能利用这些索引，通过标识扫描或全扫描避免可能的成本高昂的表访问。首先，对这些索引应用 WHERE 子句（如果有），返回索引项集合。来自不同索引的结果索引项按照其行标识的值大小进行联接。当要操作的表中存在许多列，但在查询中仅需要少数列时，这种访问方法十分有用。

索引跳扫

索引跳扫功能使优化程序可以利用多列索引，即便首列没有开始/结束关键字。如果索引中不是首列，而是尾随列包含选择的谓词，则优化程序仍然能够利用索引的优点，前提是首列的值的数量相对有限。对首列的每个值，使用选择的尾随列查找索引，以高效地查到相关的页块。因此，可以限定数量的跳扫遍历索引，其中每次跳跃都是高效的。有的查询中没有一个索引是全然匹配的，只得进行全表扫描或全索引扫描，该技术在这种情况下非常有用。

分区优化

分区是可管理性的一个重要特性。Oracle 支持表分区和索引分区。在 Oracle 所支持的各种分区方法中，范围分区 （或复合分区范围/散列或范围/列表）是最有用的分区方法。特别是在数据仓库应用中，会常常遇到数据的‘滚动窗口’。按照日期的范围分区对于在数据仓库中的装卸周期有很大的好处，既提高了效率又增加了可管理性。

分区对于查询处理也同样有用。Oracle 的优化程序充分利用了这一点。同样地，日期范围分区通常也是最重要的技术，因为决策支持查询常常用特定时间周期作为约束条件。来看这个例子，两年的销售数据以月份进行范围分区存储。假设我们想计算出最近三个月的销售总和。Oracle 优化程序会知道，访问数据的最有效的方法是扫描最近三个月的分区。用这种方式，只对相关的数据进行扫描。没有范围分区功能的系统只能扫描整个表，8 倍的数据量，或采用索引，当访问的数据量大的时候是效率非常低的访问方式。优化程序避免扫描无关数据的能力被称为*分区修整*。

智能化分区联接，分组聚合及排序

某些运算涉及了表的分区关键字时，这些操作可以以‘智能化分区’进行。例如，假设销售表是以日期作范围分区的。当查询按照时间顺序的销售记录时，Oracle 的优化程序知道每个分区可以单独排序（按照智能化分区），然后将结果简单联接在一起。分别对各分区进行排序比一次对整个表排序效率高得多。对联接和 GROUP-BY 操作可以做类似的优化。

消除排序

对大量的数据进行排序是查询处理过程中最消耗资源的操作之一。Oracle 消除不必要的排序操作。消除排序（为 DISTINCT, GROUP BY, ORDER BY 所作）有许多理由。例如，使用索引可以保证行的次序正确或唯一，唯一性约束可以保证只返回单独的一行，可能要确认对同一 ORDER BY 行来说所有的行有同样的值，而以前作过的排序操作，比如说一个排序合并联接可能已经保证了该正确次序。Oracle 的优化程序知道，关于譬如说哪种索引用在表上之类的一些局部决策能有广泛影响的副作用，当该表在查询时与所有其他的表联接时，可以用来决定 ORDER BY 排序是否可以被省掉。该决定还受所用的联接方法及联接顺序的影响，当选择表的索引时这些因素可能是未知的。所以除了根据局部优化选择、最佳索引、最便宜的联接方式等等生成执行计划之外，优化程序将试图生成一个明确针对全局排序的执行计划，而不选用正确的索引，联接方法与联接顺序。计算全局开销时，消除排序的计划通常比基于局部最小开销的计划要好。

OLAP 优化

Oracle 支持各种普遍用于 OLAP 的 SQL 构建。当使用 CUBE、ROLLUP，或分组集构建，或者 SQL 分析功能来处理查询时，可能需要大量的排序，因为不同分组的数目可能会很大。但是，如果能从一个分组看出数据已经排过序，则有可能对其他的分组减少排序需要，或改进开销，取决于该分组中列的匹配情况。因此，熟悉如何执行分组排序会导致性能大大改善。Oracle 的优化程序使用复杂的算法来决定是否排序并重新安排分组的列使得一个排序的结果可以加速或甚取消下一个排序。

并行执行

并行执行能够用多个进程（一般需要多个 CPU 和多个节点）来执行单个 SQL 语句。并行运行是查询和管理大型数据集的基本功能。

Oracle 的并行执行结构事实上允许任何 SQL 语句以任何等级的并行度执行。注意，该并行等级并不是基于要基本数据库对象的分区方式。这种灵活的并行结构有很大的优点，因为 Oracle 能根据一些因素，如要查询的表的大小，工作负载，发布该查询的用户的优先级等调整并行程度。

优化程序与并行执行全面配合工作，并在选择最佳执行方案时考虑并行执行影响。

提示

提示是加在 SQL 查询中的影响执行计划的指示。提示通常用于指明极少数情况下优化程序选择了半优化的计划，但提示在市场宣传中或销售演说中常常被误解或错用。有人宣称提示的存在表明优化程序有弱点，事实上，大多数数据库产品（Oracle、 Microsoft SQL Server、 IBM Informix、 Sybase）支持某种形式的优化程序指示。没有一个优化程序是完美的（参见 IBM Almaden 研究中心在 1999 年国际 DB2 用户组会议的一篇讲演 “ 为什么 DB2 的优化程序产生错误的计划？ ”），同时，指示（例如 Oracle 的提示）提供了最简单的工作区情况，即优化功能包选出了个半优化的计划。

提示不仅是补救偶然出现的半优化计划的有用工具，而且对于想对访问路径进行实验或者仅仅完全控制查询的执行的用户也是有用的工具。例如，用户能够比较在查询中半优化索引与优化索引的性能。如果优化索引仅用于该特定查询，但半优化索引用在许多不同的查询中，如果性能差别不大时用户可以考虑放弃优化索引。提示使进行这些类型的性能实验简单快捷（因为没有提示，用户将不得不放弃然后再重建优化索引，以完成该实验）。

提示设计用于只在不寻常的情况下指明执行性能，提示不应轻易使用，事实上，过多使用提示会降低执行性能，因为提示能阻碍优化程序在情况改变时调整执行计划(扩大表、添加索引,等等)，还有可能掩盖问题，比如使优化程序统计不准确。正确使用提示的示例是 Oracle 自己的应用程序：高度协调的 Oracle E-Business Suite 11i ERP 模块，在这些模块的 270,000 个 SQL 语句中有 3% 使用了提示。

开销模型和统计

基于开销的优化程序通过评估各种执行计划的开销然后挑选出开销评估最好的（开销最低的）计划。所以，开销模型是优化程序的关键组成部分，鉴于开销模型的准确性直接影响了优化程序识别与选择最佳执行计划的能力。

执行计划的‘开销’基于执行计划每个部分的细心的建模。开销模型与所有 Oracle 的访问方法及数据库结构的所有信息相结合，因此能够为所有类型的操作生成准确的开销。此外，开销模型依赖于‘优化程序统计’，这些统计描述了数据库中的对象和硬件平台的性能特点。下面将更详细地说明这些统计。为使开销模型更有效，开销模型必须有准确的统计。

Oracle 有许多功能有助于简化和自动化统计—收集。开销模型是查询优化程序中的一个非常复杂的组成部分。开销模型不仅要理解数据库提供的每一种访问方法，同时必须考虑一系列性能特征的影响，诸如缓存、I/O 优化、并行发运行及其他性能特征。此外，对开销没有单一的定义。对于某些应用程序，查询优化的目标是使返回第一行或 N 行的第一个集合的时间最短，而对于其他应用程序，其目标则是在最短的时间内返回全部结果集合。Oracle 的开销模型通过计算 DBA 首选的不同类型的开销来支持这两种目标。

优化程序统计

在优化查询时，优化程序根据开销模型来评估执行计划中的所有操作（联接，索引扫描，表扫描等）的开销。此开销模型则依赖于 SQL 查询涉及的数据库对象和基本硬件平台的有关信息。在 Oracle 这些信息（优化程序的统计）可以分成两类：对象级统计和系统统计。

对象级统计

对象级统计描述数据库中的对象。这些统计记录诸如每个表中块数和行数，以及每个 B 树索引的级数，等数值。还有一些统计描述每个表中的列。列的统计特别重要，因为它们用于估计与每个查询的 WHERE 子句中的条件相匹配的行数。Oracle 的列统计对于每列有最小值与最大值，以及不同的值的数目。还有，Oracle 支持直方图以更好地优化针对包含异常数据分布的列的查询。Oracle 支持高点平衡直方图和频率直方图并根据列的准确属性自动选择适当类型的直方图。

系统统计

系统统计描述硬件平台的性能特性。优化程序的开销模型把 CPU 开销与 I/O 开销区分开。但是在不同的系统里 CPU 的速度有很大的不同，而且 CPU 与 I/O 性能比也很不相同。所以，不是使用将 CPU 和 I/O 开销组合的固定公式，Oracle 而是提供了一个工具，来收集单个系统在典型工作负载下的特性信息，以决定每个系统的开销组合的最佳方式。收集的这些信息包括 CPU 速度和各种类型的 I/O 的性能（优化程序在收集 I/O 统计时能分辨单块，多块和直接硬盘的 I/O）。把每个硬件环境的系统统计统一起来，Oracle 的开销模型对任何硬件厂商的任何配置组合都非常准确。

用户定义的统计

Oracle 还支持用户定义功能和域索引的用户定义开销功能。用自己的功能模块及索引来扩展 Oracle 数据库系统能力的客户能够将自己的访问方法与 Oracle 的开销模型完全集成在一起。Oracle 的开销模型是模块化的，所以用户定义的统计被视为在与 Oracle 自己内置索引和功能一样的开销模型与查找空间内。

统计管理

数据库的性质会因数据的变化而随着时间改变，这些数据变化是由于事物处理活动或新的数据装入数据仓库而引起为保持对象级优化程序统计的准确性，当要统计的数据变化时，统计也需随之更新。收集统计问题给 DBA 带来若干挑战：

- 大型数据库统计的收集可能要花费大量资源。
- 确定哪一个表需要更新统计可以会很难。许多表可能改动不大，重新计算它们的统计是浪费资源。但是，在有上千个表的数据库中，DBA 很难人工记录每个表的变化程度，以及哪个表需要新的统计。
- 确定哪个列需要直方图可能很难。有些列需要，有些则不需要。给那些不需要的列生成直方图是浪费时间和空间。但是，不给那些需要直方图的列生成直方图，则可能导致不好的优化程序决策。

Oracle 统计收集程序对这些挑战一一处理。

并行取样

允许有效统计收集的基本功能是取样。与通过扫描（和排序） 整个表来收集统计不同，好的统计通过检查少量的样品行来收集数据。由于取样导致的速度提高是惊人的，因为取样不仅减少了扫描表使用的时间，而且因此也减少了处理数据的时间（因为要排序和分析的数据较少）。要想进一步加快在非常大的数据库中收集统计的速度，可以将取样与并行运行联合运用。Oracle 统计收集程序自动决定恰当的抽样百分比和适当程度的并行运行，根据要统计的表的数据特性确定。

监视

另一个简化统计管理的重要功能是监视。Oracle 记录表自上次统计收集以来有多少改变（插入、，更新和删除）。那些改动大量，需要新优化程序统计的表自动被监视进程做上标志。当 DBA 收集统计时，Oracle 将仅收集那些大量改动过的表的统计。

自动确定直方图

Oracle 的统计收集程序还在后台确定哪个列需要直方图。Oracle 借助检查两个特性作出决定：每列的数据分布和该列在 SQL 语句的 WHERE 子句中出现的频率。对于那些既高度异常又常常出现在 WHERE 子句中的列，Oracle 将生成直方图。

这些功能一起，真实地自动化了收集优化程序统计的过程。用一个语句，DBA 就能够收集整个视图的统计，并且 Oracle 在后台决定哪个表需要新的统计，哪个列需要直方图，以及适合于每个表的取样级别和并行程度。

动态取样

不幸的是，即便准确的统计也不能总满足优化查询执行的需要。根据定义，优化程序统计只是实际的数据库对象的大致描述。在某些情况下，这些静态的统计不完全。Oracle 通过用另外的统计（在查询优化过程中动态收集来的）补充这些静态对象级统计来解决这些情况。

大约有两种场合优化程序统计是不够的：

相关。经常查询有复杂的 WHERE 子句，在这些子句中对一个表有两个或多个条件。请看下面的非常简单的例子：

```
SELECT * FROM EMP
WHERE JOB_TITLE = 'VICE PRESIDENT'
AND SAL < 40000
```

通常的优化方式是假设这两列是互相独立的。即，如果 5% 的员工是‘副总裁’，40% 的员工工资少于 40,000，则简单方法是假设 .05 * .40 = .02 的员工同时符合该查询的两个条件。在这种情况下该简单的方式是不正确的。在职务和工资是相关的，因为具有副总裁职务的员工非常可能有较高的工资。尽管该简单方式表明该查询应应返回 2% 的行，实际上查询返回零行。

静态优化程序统计分别保存每列的信息，不能告诉优化程序哪些列可能是相关的。此外，试图储存统计来捕获相关信息是艰难的任务，因为可能的列组合数目是以指数级增加的大量。

临时数据。某些应用程序会生成中间结果集合，临时存放在一个表中。该结果集合用于未来操作的基础然后删除或简单地回滚。对于存放中间结果的临时表，要做准确的统计是非常困难的，因为这些数据仅存在很短的时间，甚至可能都还没有提交。DBA 不可能对这些临时对象来收集统计。

Oracle 的动态取样特性解决了这些问题。当查询被优化后，优化程序可能注意到一些列有可能是相关的，或有个表缺少统计。在这些情况下优化程序会对相应的表中少量的进行取样，并在执行过程中收集相应的统计。在相关的情形中， WHERE 子句中所有有关的条件同时作用于这些行来直接衡量相关的影响。这个动态取样技术对于临时数据也非常有效，因为取样与查询在同一个处理中发生，优化程序可以见到用户的临时数据，即便这些数据尚未提交。

优化成本类型

基于开销的优化程序评估各种执行计划的开销并选取具有最低开销估值的那一个。但是，‘ 最低开销 ’的概念可能是不同的，取决于给定应用程序的要求。在一个操作系统中（每次仅向最终用户显示几行数据）‘ 最低开销 ’执行计划可能是在最短的时间内返回第一行数据的执行计划。另一方面，在最终用户检查查询返回的整个数据集的系统中，‘ 最低成本 ’执行计划是在最短时间内，返回所有行的执行计划。

Oracle 提供两种类型优化程序：一个用于最小化返回所查询的第一个 N 行的时间，另一个用于最小化从查询中返回所有行的时间。数据库管理员可以另外指出 N 的值。以此方式，Oracle 的查询优化程序可以很容易地调整，以满足不同类型的应用程序的特殊需求。

动态运行时间优化

每个数据库的工作负载都是变化的，不同小时间、白天的负载与夜间的负载、工作日的负载与周末的负载以及从一般时刻的工作负载到季度末和年末的工作负载都有非常大的不同。没有一套静态的优化程序统计和固定的优化程序模型能够覆盖这些不断变化的系统的所有的动态的方面。对执行策略的动态调整对获得好的执行性能是必需的。

出于这一原因，Oracle 的查询优化不仅仅是选择访问路径。Oracle 有强有力的一套功能，可以调整每个查询的执行策略，不仅仅是根据 SQL 语句和数据库对象，而且还有整个系统在执行查询时的状态。

动态优化的关键考虑事项是适当的硬件资源管理，例如 CPU 和内存。动态优化的特点是对于每个查询的执行策略的动态调整，使硬件资源的使用能够使所有查询的吞吐量最大化。查询优化的其他多数目标致力优化单一 SQL 语句，而动态优化致力于在所有其他 SQL 语句正在执行的的情况下对每个 SQL 语句的优化。

动态并行程度

并行执行是在多处理器硬件环境中改善查询响应时间的很好的方式。但是查询的并行执行总体上可能会比串行执行多用一点资源。因此，在有资源争用现象的重负载系统中并行化一个查询或者用了太高的并行度可能会适得其反。相反，在轻负载的系统中，查询应该具有高并行度来充分利用可用资源。所以，死守固定的并行度是坏的想法，因为系统的工作负载随时间而变化。Oracle 自动调整查询的并行度，在工作负载增加时将其降低，以避免资源争用。工作负载降低时，再把并行度提高。

动态内存分配

有些操作（主要是排序和散列联接）如果能访问更多内存就会更快。这些操作通常将数据处理多次，放入内存的数据越多，在每次处理时需要存储到磁盘上的临时表空间的数据就越少。最理想的情况下，排序与散列联接可能全在内存中进行，临时磁盘空间完全用不上。

不幸的是，系统只有有限的物理内存，由同时执行的所有操作共享。如果内存过渡分配，就会发生交换而使性能急剧下降。相反，如果有可用内存能够用来加速排序或散列联接的执行，内存就应该分配，否则执行性能将是不理想的。为每个查询提供最佳数量的内存很困难：足够用来有效处理查询，又不过多，使其他查询也能得到它们应当享有的内存。

给每个 SQL 语句分配固定数量的内存（其他数据库厂商所用的方式）不是有效的解决方案。随着数据库工作负载的增加，需要越来越多的内存来应付数目不断增加的查询。最终，系统实有的内存将被分配殆尽，系统执行性能将急剧下降。

稍许聪明但仍效率不高的方式是给每个查询分配等量的内存，这样，如果有 100 个并行的查询，则每个查询得到 1% 的可用内存，如果有 1000 个并行查询，则每个查询得到 .1% 的可用内存。这个方案是好，因为每个查询处理的是不同大小的数据集合，而且每个查询可能有不同数量的排序及散列联接操作。如果每个查询分配固定数量的内存，则某些查询可能会内存太多而其他查询的内存不够。

Oracle 的动态内存管理解决了这些问题。DBA 给 Oracle 指定 SQL 操作的可用系统内存总数。Oracle 来管理这些内存，使每个查询在所有用于查询的内存范围内得到适当数量的内存，不超过 DBA 指定的范围。

对于每个查询，优化程序生成执行计划中每个操作的内存需求概要。这些概要不仅包含理想的内存级（即全部在内存中完成操作所需要的内存量）还有多次磁盘交换过程的内存需求量。在运行时间，Oracle 检查系统中的可用内存量和查询的需求概要，根据当前系统工作负载，给查询分配内存，以获得最佳的执行性能。

即便在查询运行时，Oracle 仍将对每个查询继续动态调整内存。如果一个指定查询用的内存少于预期值，多余内存会分配给其他查询。如果给指定查询增加较多内存时可被加速，则在有富余内存的情况下会给该查询增加内存。对每个单独的操作如何会受内存改变的影响的判断是基于内存配置文件。当动态调整内存分配时，Oracle 将挑选相对于整体执行性能来说最适合改变的那些操作。

这个独特的性能极大地改善了数据库的执行性能以及可管理性，同时将 DBA 从分配内存的工作里解脱出来，该工作用人工不可能解决得很好。

数据库资源管理程序

Oracle 的数据库资源管理程序提供了一个框架，使 DBA 能够仔细地控制如何在不同的用户间分配资源。数据库资源管理器控制查询执行的多个方面，包括分配给每个用户组的 CPU，各组用户的活动会话数量，和各组用户可以并行运行的程度范围。

数据库资源管理器有两大好处：

- 最大化整个系统吞吐量。数据库资源管理器保证充分利用硬件资源，同时管理工作负载，使系统不至过分使用。
- 把资源提供给适当的用户。有些数据仓库用户应该比其他用户拥有更高的优先级。数据库资源管理器能让 DBA 把更多的资源分配给重要用户。

数据库资源管理器对于查询优化很重要，原因有二：首先，资源管理器提供 SQL 执行时的动态调整。例如，复杂查询可能需要大型的，耗费 CPU 的排序。在运行时间数据库资源管理器可能会限制该查询使用的 CPU。数据库资源管理器以这种方式能够动态调整 SQL 执行，使典型查询不会弄垮系统和延长其他用户的响应时间。

数据库资源管理器与查询优化紧密结合的第二种方式是数据库资源管理器提供了活动前查询管制功能。对每个最终用户组，数据库管理员可以指定这些用户允许的最长 SQL 语句执行时间。例如，数据库管理员可指定一组给定的用户不允许提交任何执行时间超过 20 分钟的查询。如果有用户想提交长时间运行的 SQL 语句，该 SQL 语句会在执行前返回一个错误。该查询管制的目的是在活动前防止用户发出不可接受的运行时间长的查询。查询管制与查询优化器结合在一起在后台实施。查询优化器的开销估算用于估算每个查询的执行时间，这样数据库资源管理器就能判断哪个查询需要提前终止。

结论

查询优化是取得良好执行性能并简化管理的关键因素。Oracle 的查询优化程序提供了大量的功能。它不仅适用于各数据库厂商的五花八门的访问路径，Oracle 的优化程序还提供了特别创新的技术，可以在查询执行时进行动态的和适应性的调整。Oracle 将不断改进其优化程序，持续为其客户提供领先的执行性能和可管理性。

Oracle9*i* 的查询优化
2002 年 2 月
作者：George Lumpkin, Hakan Jakobsson
协作者：

Oracle Corporation
全球总部
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

全球咨询热线：
电话： +1.650.506.7000
传真： +1.650.506.7200
www.oracle.com