数据库实例是供应用程序和用户使用的逻辑实体，独立于数据的物理存储。类似地，实际表和列是物理数据库中的逻辑实体。用户在请求Oracle数据库中数据的时候，并不需要了解数据库实例和表空间，但必须知道数据的结构。表和列就是数据结构实施中的两个概念。为了充分发挥Oracle的功能，用户必须懂得Oracle数据库服务器是如何实现并使用这些逻辑数据结构的。

## 1、数据类型

数据类型是列或者存储过程中变量的一个属性。它描述并限制列中可以存储的信息类型，并且能够限制在该列执行的操作类型。

用户可以将Oracle支持的数据类型分为三个基本种类：

字符数据类型

数字数据类型

其他类别数据类型

下面的SQL语句可用于在表中创建任何类型的列：

CREATE SAMPLE\_TABLE(

char\_field CHAR(10),

varchar\_field VARCHAR2(10),

todays\_date DATE

)

用户也可以在PL/SQL存储过程中定义变量的时候使用数据类型。

### 1.1字符数据类型

字符数据类型可以保存任何串数据，包括数字组成的串。给一个字符数据类型赋一个长度大于其指定或允许长度的值将会导致运行时错误。用户可以使用串函数，例如，UPPER、LOWER、SUBSTR以及SOUNDEX等来操作标准的（非大型的）字符数值类型。

下面列出了几种不同的字符数据类型：

CHAR：

存储着具有固定长度的字符串。这种类型可容纳的字符长度为1~2000个字符。如果用户没有显式地指定CHAR的长度，那么它的默认长度为1。如果用户赋的值比CHAR类型指定的长度小，那么Oracle将会自动在后面填充空格。（CHAR(10)=”Rick ”）

VARCAHR2：

存储变长字符串。虽然用户必须给VARCAHR2类型指定长度，但是这只是一个最大长度，而不是要求的长度。赋给VARCHAR2类型的值并不会被允许填充空格。VARCAHR2最多可容纳4000个字符。VARCHAR2比CHAR类型更加节省空间。

（在Oracle8及其之后的版本中，VARCAHR类型等同于VARCHAR2类型。不过，考虑到未来会将VARCAHR和VARCHAR2向不同的方向发展，建议使用VARCAHR2。VARCHAR2(10)=”Rick”）

NCHAR和NVARCHAR2：

NCHAR和NVARCHAR2类型分别存储固定长度和变长字符数据；而这些字符使用了与数据库其他部分不同的字符集。在创建数据库的时候，用户需要指定对数据库中各种字符进行编码的字符集。用户还可以随意选择另一种字符集（这个字符集叫做国家语言集，即NLS）。NCHAR和NVARCHAR2类型的列将会使用这种字符集。例如，用户可能会使用一个描述列来存储日语字符，而数据库其他部分使用英语编码。在这种情况下，用户需要在创建护具库时将日语字符集指定为第二种字符集，并且将相关列设置为NCHAR或NVARCHAR2类型。

（从Oracle 9i开始用户可以以字符为单位指定NCHAR和NVARCAHR2列的长度，而不是以字节为单位。例如，用户可以指定这两种类型的列的长度为7个字符。如果第二字符集中的字符需要两个字节的存储空间，那么Oracle 9i数据库将自动把该列转化为14个字节的存储。）

（注意：Oracle 11g引入了全球化开发工具集GDK，帮助用户创建支持不同语言的互联网应用程序。GDK关键特性在于它为Java和PL/SQL开发人员提供了一个全球化最佳实践框架。

另外，Oracle 10g增加了对大小写不敏感的查询和排序的支持。如果期望在查询或排序操作中仅使用基本字母或者基本的字母和重音，那么可以使用这个特性。）

LONG：

LONG类型最多可容纳2GB的字符数据，它可以看作是从早期Oracle版本中遗留下来的数据类型。如果用户期望存储大量的字符数据，Oracle现在推荐使用CLOB和NCLOB数据类型。数据库表和SQL语句对LONG类型有很多限制，例如，用户不能够在WHERE，GROUP BY，ORDER BY，或者CONNECT BY子句中或者带DISTINCT修饰符的SQL语句中使用LONG类型。用户也不能在LONG类型的列上创建索引。

CLOB和NCLOB：

CLOB和NCLOB类型在Oracle 10g之前的版本中可以存储高达4GB的字符数据。Oracle 10g开始这个限制被扩展到128TB，不过这要依赖于数据块的大小。NCLOB类型用于存储NLS数据。Oracle 10g及之后的发布能够隐式地执行CLOB和NCLOB类型间的转换。

### 1.2数字数据类型

Oracle采用标准的变长内部格式来存储数字。它支持的数字精度高达38位十进制数字。

NUMBER就是Oracle的数字数据类型。将一列或者一个变量声明为NUMBER类型，它们将自动提供38位十进制数字的精度。该类型还可以接收如下两个修饰符：

column NUMBER (precision, scale)

数据类型的精度是指数字中重要位的总数。用户可以将数据的精度指定为1~38之间的任意数字。如果没有指明精确度，Oracle将默认使用精度38。小数点后位数，如果没有指定，Oracle默认小数点后的位数为0。

如果用户将一个负数赋值给小数点后位数，那么Oracle将会把这个类型的数据在小数点左边指定的位置进行四舍五入。

column\_round NUMBER(10,-2)

column\_round = 1234567，其中column\_round的值为1234600。

NUMBER是Oracle中唯一用来存储数字型数据的数据类型。ANSI标准类型如DECIMAL、NUMBER、INTEGER、INT、SMALLINT、FLOAT、DOUBLE、PRECISION以及REAL都是存储在NUMBER数据类型中。用户用来访问Oracle数据的语言或者产品可能支持这些数据类型，但是它们都存储在NUMBER类型的列中。

Oracle 10g增加了BINARY\_FLOAT和BINARY\_DOUBLE两种数字类型来支持IEEE 754-1985中定义的精度标准。Oracle 11g开始支持SIMPLE\_INTEGER数据类型。

### 1.3日期数据类型

就NUMERIC数据类型而言，Oracle将所有的日期和时间数据以标准内部格式保存起来。标准的Oracle日期格式采用DD-MON-YY HH:MI:SS形式。如果用户没有指定任何的时间值，那么在内部存储中默认的数据全为零。

用户可以通过修改数据库实例的NLS\_DATE\_FORMAT参数来改变日期的格式。用户也可以通过ALTER SESSION SQL语句来改变会话中的日期格式，或者在SQL语句中使用TO\_DATE表达式改变指定值的格式。

Oracle SQL支持日期的算数运算，整数部分表示日期，小数部分表示时、分、秒。例如：日期数值加.5表示12小时后的日期和时间。

12-DEC-07 + 10 = 22-DEC-07

Oracle 9i发布2还支持两种INTERVAL数据类型，即INTERVAL YEAR TO MONTH和INTERVAL DAY TO SECOND，它们用于存储指定的时间量。这种类型的数据也可用于日期算数运算。

### 1.4其他数据类型

RAW和LONG RAW：

通常，Oracle数据库不仅要存储数据，还要解释数据。当数据库被请求或者被导出数据库时，Oracle数据库有时候会对请求的数据进行处理。例如，当用户导出NUMBER列数据的时候，写入文件中的数值是数字格式，而不是内部存储格式。

RAW和LONG RAW类型的数据则不需要Oracle数据库做任何解释工作。当用户指定了这两种类型之一，Oracle将会把数据以提交给它的比特序列形式准确地存储起来。RAW类型通常会把对象以其自己的内部格式（如bitmap）存储起来。RAW类型可以存储2KB的数据，而LONG RAW类型可以存储2GB的数据。

ROWID：

一种特殊类型的列，又叫做ROWID伪列可以像正常列一样使用SQL SELECT语句访问。Oracle数据库中每一行都有一个ROWID伪列。ROWID表示特定行的特定地址。ROWID伪列可以用ROWID数据类型来定义。

ROWID与磁盘驱动器中的特定位置相关。因此，ROWID是获取某个行最快速的方法。然而，ROWID是获取某个行最快速的方法。然而，一个行的ROWID会随着数据库的卸载和重新加载而发生变化。基于这点考虑，我们并不推荐在多个事务代码中使用ROWID伪列的数值。例如，一旦当前应用程序使用完一行，就没有必要再存储对这个行的ROWID引用。

用户无法使用SQL语句来设置标准的ROWID伪列的数值。

ROWID伪列的格式在Oracle 8中有所变化。从Oracle 8开始，ROWID除了包含执行数据文件、数据块以及行的标识符外还包含指向数据库对象编号的标识符。用户可以通过解析ROWID伪列的数值来了解Oracle数据库中行的物理存储。

用户可以定义ROWID类型的列或者变量，但是Oracle并不保证存在这些列或者变量中的数值就是有效的ROWID。

ORA\_ROWSCN：

Oracle 10g及之后的发布都支持ORA\_ROWSCN伪列，它记录着最近对行进行修改的事务的系统变动编号。借助此列，用户可以很容易地检查出事务启动后指定行的变动。

LOB：

LOB，即大型对象数据类型，它可以存储高达4GB的信息。LOB有三种形式的变种：

* CLOB：仅用于存储字符数据。
* NCLOB：存储国家语言字符集中的字符数据。
* BLOB：存储二进制数据。

用户可以指定LOB类型存储Oracle数据库中的数据，也可以指定它指向包含数据的外部文件。

LOB类型可以参与事务。在Oracle中选择LOB类型数据将返回它的指针。用户必选使用DBMS\_LB PL/SQL内置包或者OCI接口来实际操作LOB类型数据。

为了简化LONG类型到LOB类型的转换，Oracle 9i在大多数支持LONG性的函数中在增加了对LOB型的支持。另外，Oracle 9i还在ALTER TABLE语句中增加了一个选项，从而允许LONG型自动迁移到LOB型。

BFILE：

存储指向Oracle数据之外的文件的指针。因而BFILE类型的列或者变量并不参与事务，存储在这些列中的数据只可以读。BFILE型数据的大小受底层操作系统规定的文件大小的限制。

XMLType：

作为支持XML的一部分，Oracle 9i引入XMLType数据类型。一个定义为这种类型的列将会把XML文档以字符的形式存储在LOB列中。Oracle提供内置的函数方便用户从XML文当中提取各个节点，用户甚至可以在XMLType文档中特定的节点上建立索引。

用户自定义数据：

Oracle 8及之后的版本允许用户定义他们自己的符合数据类型，即由前面讨论过的Oracle基本数据类型组合而成的类型。这些版本的Oracle甚至允许yoghurt创建由基本数据类型和用户自定义类型组合成的对象。

AnyType、AnyData、AnyDataSet：

Oracle 9i及更新的版本中，还有三种类型用于显示的定义现有数据类型之外的数据结构。在定义这些数据类型的时候，必须定义程序单元，从而让Oracle知道如何处理这些类型的指定实现。

## 2、类型转换

Oracle会自动根据数值出现的SQL语法将某些数据类型转换为其他类型。

当用户将一个字符数值赋给一个数字类型，Oracle会隐式地将字符串表示的ASCII码值转换为数字。例如，把一个字符值10赋给一个NUMBER类型的列，将会进行自动数据转换。

如果用户试图把字母数值赋给数字类型，将会得到一个无效数字。

用户还可以使用Oracle提供的的各种类型转换函数对数据执行显示的转换。由于Oracle对各种类型转换函数都做了清楚的定义，能够避免像隐式类型转换那样悄悄地进行类型转换，因而，如果转换是预期的，那么最好使用显示的数据转换。

### 2.1串联和比较

大多数平台上，Oracle SQL语言的串联操作符为两条竖线（||）。两个字符数据可以执行串联运算。Oracle的自动类型转换功能允许用户将两个数字数据串联起来。如果NUM1是一个值为1的数字数据类型，NUM2是一个值为2的数字数据类型，而NUM3是一个值为3的数字数据类型，那么：

NUM1 || NUM2 || NUM3 = “123”

NUM1 || NUM2 + NUM3 = “15”

NUM1 + NUM2 || NUM3 = “33”

上面这些表达式的结果为字符串，如需进一步计算，字符串数据可以被自动转换为数字型数据。

相同类型数据间的比较都能够向用户期望的那样进行。用户可以使用关系运算符来比较数字型数据或日期型数据。对于字符数据而言，单个字符的比较是基于字符的底层编码进行的。对于字符串而言，它们是逐个字符比较，直到出现第一个不同的字符为止。

当两个不同长度的字符串进行比较的时候，Oracle提供两种不同类型的比较：填充空格比较和无任何填充比较。对于填充空格比较来说，较短的字符串后面被填充了空格，比较操作安装前面描述的那样。对于无任何填充比较来说，如果两个字符串在其中较短的字符串长度范围内都相同，那么这个短的字符串被认为较小。举个例子，对字符串“A ”（A后面跟着一个空格）和字符串“A”（就一个字符A）进行填充空格比较，这两个字符串被认为是相等的，因为比较的时候会在第二个字符串的后面填充空格。如果是对这两个字符串进行无填充比较，那么第二字符串被认为较小，因为它的长度小于第一个字符串。无填充比较通常用于比较运算至少有一方数据是VARCAHR2或者NVARCHAR2类型，而填充空格比较通常用于参与比较的两个数据都不属于上面两种类型的情况。

Oracle 10g及之后的发布提供了一个名为表达式过滤器的特性。它使得用户能够将复合表达式作为行的一部分存储起来。用户可以使用EVALUATE函数来限制基于该表达式计算结果的查询。该表达式过滤器使用了正则表达式技术。

## 3、NULL

NULL这个值是关系型数据库的关键特性之一。实际上，NULL并不表示任何值——它仅表示无数据。如果用户需要为表创建一个必须有值的列，那么它可以将该列指定为NOT NULL，这意味着该列不能包含NULL值。当用户试图将一行数据写入数据表中，而又未提供NOT NULL列的值，那么Oracle将返回错误。

用户可以将NULL赋值给任何数据类型。NULL给SQL运算符引入了三状态逻辑。通常的比较只有两种状态：TRUE或者FALSE。而当比较的一方为NULL值的时候，就会有三种逻辑状态：TRUE、FALSE以及两个都不是。

如果列A中包含了NULL值，那么下面的条件表达式都为假：

A > 0

A < 0

A = 0

A != 0

这三种状态逻辑的存在往往会让用户感到迷惑。在实际应用中，经常会有列或者变量要求使用NULL值。

由于NULL值不等于零，也不等于其他任何值，因而用户必须使用关系运算符IS NULL来测试NULL值。NULL = NULL的计算结果也是FALSE，因为NULL值不等于任何值（包括它自己）。

你应该使用NULL吗？

三状态逻辑的概念有时候会让人有点混淆，特别是当你设想自己的终端用户会执行即兴查询并计算一个既非TRUE又非FALSE的值的时候。考虑这种应用场景，你可能会决定不再使用NULL值。

我们认为NULL有它适用的场合。例如，当一个列无需赋值的时候。取代使用NULL的方法可以是：使用一个数值（比如零）来标识特殊的意义，然后此案有某种方式来确定这个数值标识的就是其本身还是用来代替NULL。

如果你选择不使用NULL值，那么必须为每一行的每个列都赋一个值。这样做可能会给某些列赋予令人误解的值，在效果上也排除了一个列可以不被赋予一个值的可能性。这样会让终端用户感到误解，同时也会导致诸如AVG这样的总结性操作产生不准确的结果。

避免使用NULL值这种做法只是简单地将一个问题——培训用户或者为他们提供接口从而让他们慢慢理解NULL值——换成另外一组问题——保护数据的完整性。

## 4、基本数据结构

三种基本的Oracle数据结构：表、视图和索引。另外，还会讨论分区这个特性将会影响数据在表和索引中的存储方式。

### 4.1表

表是关系型数据库中基本的数据结构，它是行的集合。表中的每行包含一个或多个列。可以将一个表理解为一个文件或者一个非关系型数据库，而将表中的一行理解为非关系型数据库中的一个记录。

就Oracle 9i而言，用户可以定义外部表。外部表中的数据存储在数据库之外，通常是存储在flat文件中。外部表是只读的，因此yoghurt不能更新其中的数据。它适合用于在数据库和文件之间加载和卸载数据，以完成不同的功能。

Oracle 11g支持为表创建虚拟列。这些列由表达式定义，尽管表达式的结果没有被存储，应用程序仍然可以在运行时访问列。

### 4.2视图

视图是Oracle的一种数据结构，是通过SQL语句定义的。这个SQL语句存储在数据库中。当用户在查询中使用视图的时候，存储在数据库中的查询被执行，基表数据被返回给用户。视图本身并不包含数据，它仅表示以查询指定的方法来观察基表中的数据。

视图可用于以下几个目的：

* 简化对存储在多张表中的访问。
* 为表中的数据实施指定安全特性（例如，创建一个带有WHERE子句的视图能够限制用户通过视图可以访问到的数据）。从Oracle 9i开始，用户可以通过定义更细粒度的访问控制来达到这一目的。它使得用户能够根据行数据的值自动限制数据的访问。
* 将应用程序与底层表的特定结构相隔离。

视图通常是基于一组基表（实际的表或视图）构建而成。如果视图的任一基表被修改，不能再用于构建视图，那么这个视图本身也就不能再使用了。

通常而言，用户在单条SQL语句中只能写视图的一个底层基表中的列。对于INSERT、UPDATE和DELETE操作，还有其他限制；某些SQL子句阻止用户对视图中的任意数据进行更新。

用户可以使用INSTEAD OF触发器对不可更新的视图执行写操作，这点将会在后面讲解。

Oracle 8i引入了物化视图。根据本节的定义，物化视图并不是真正的视图，而是用于存储预总结数据的物理表，它很好地改善了数据仓库的性能。

### 4.3索引

索引这个数据结构用于加速对数据库中特定行的访问。它与特定的表相关联，包含来自这个表的一列或多个列的数据。

创建索引的基本SQL语法：

CREATE INDEX 索引名 ON 表名(column1,column2);

当表中对应列数据被修改时，Oracle数据库服务器将会自动修改索引中的数据。由于索引包含的数据少于表中的数据，而且索引是存储在适合快速访问的特殊结构中，因而获取索引中的数据只需要较小的I/O操作。基于索引选择行通常比基于行数据选择行要快。另外，大多数索引都是有序存储的（或者升序，或者降序，这取决于用户创建数据库的时候所做的声明）。正是因为有了这样的存储模式，当用户需要根据数据区间选择行或者需要返回有序行的时候，如果索引中包含了这个数据区间或者进行了这样的排序，那么这两种操作会变得很快。

除了索引中的数据之外，索引条目中还为索引相关的行存储了ROWID数据。ROWID是获取数据库中任何一行数据最快捷的方式，因而随后对数据库中行的获取都能以最优的方式进行。

索引值可以是唯一的（这意味着表或视图中的任何两行都具有不同的索引值），也可以是不唯一的。如果建立索引包含的值为NULL，那么该索引将不会包括这行数据。

Oracle中索引指的是数据库中的一种物理结构。键是一个表示逻辑实体的术语，通常就是指索引中存储的数据。除了外键约束之外，Oracle文档中的许多地方都会交换使用这两个术语。

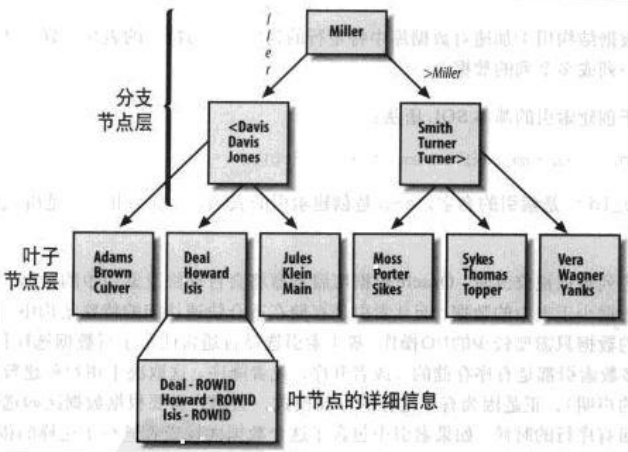
下面将描述Oracle中使用的四种不同类型的索引结构：

1. 标准B-tree索引
2. 反向键索引
3. 位图索引
4. 基于函数的索引

Oracle 11g支持使用不可见索引。Oracle还允许用户对表中的数据进行集群，从而提高性能。

1）B-tree索引

Oracle默认使用的索引类型。其结构类似一颗倒立的树，如图所示：



B-tree索引包含一个或多个分支节点层以及一个叶子节点层。一个分支节点层包含了下一层分支节点所覆盖的值范围。根节点与叶子节点之间的分支节点层的数目叫做索引深度。叶子节点包含实际的索引值以及相关行的ROWID。

由于B-tree索引结构不包括一些较高层次的分支节点，因而读取B-tree索引结构的I/O操作相对来说较少。另外，由于所有的叶子节点都在索引的同一深度，因而获取任意索引项都需要相同数量的I/O操作，这就使得索引的性能比较统一。

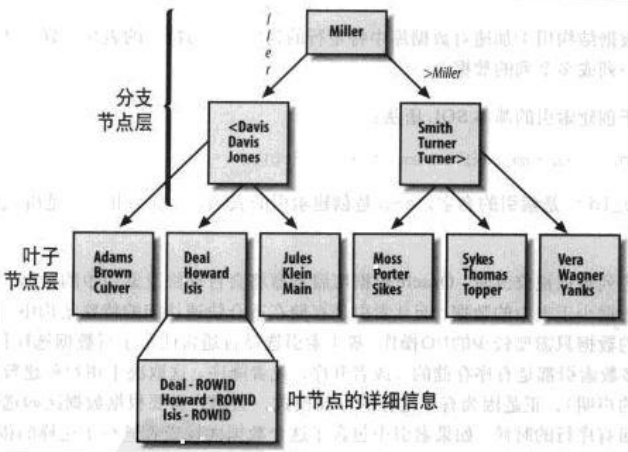
Oracle允许用户创建索引组织表（Index Organized Table，IOT），其中叶子节点存储了整个行的数据，而不仅仅是指向相应行的ROWID。由于不需要在叶子节点中存储ROWID，因而索引组织表减少了用于存储索引和表的空间总量。但是它并不能使用UNIQUE约束，也不能存储在集群中。另外，它也不支持分布、赋值和分区（其他章节讨论）。不过，在Oracle 10g之后的发布中，IOT可以和Oracle Streams一起用于捕捉和应用对数据库的变动。

Oracle 9i对索引组织表做了许多改进，具体包括：减少了把位图索引作为IOT辅助索引的限制，增加了在IOT中创建、重建或者链接辅助索引的功能。Oracle 10g继续对组织索引表做了一些改进，包括：允许索引组织表的复制，支持各种类型的分区以及其他的一些改进。

2）反向键索引

自动翻转存储在索引中的键值字节顺序。如果行的键值为“ABCD”，那么该行的反向索引则是“DCBA”。

为了理解反向键索引的必要性，用户必须回顾有关B-tree索引的一些基本知识。首先，B-tree的深度取决于叶子节点项的数目。B-tree的深度越深，分支节点的层次越多。那么定位并访问叶子节点的I/O操作数量就越多。



图中所示的索引是一个基于字母的索引。看起来，它是一个平衡二叉树，在叶节点范围内各项呈均匀分布，但是有些用于索引的值并没有设计好。递增的值（例如，升序数字或者逐渐往后的日期值）总是在索引的右边，而这正是存储较大值的位置。另外，当从索引中删除数据时，随着原有行的删除，索引树逐渐倾向左边。这些操作都会使得索引随着时间的迁移变成非平衡的B-tree，左边的索引比右边的叶子节点分布要稀疏得多。这种不平衡的增长会导致B-tree深度的增长，因为此时深度取决于索引右边项的数目。对于自动递减的值，也会出现上述的影响，除非左边不再接受更多的项。

可以通过删除并重新创建索引的方法来解决这个问题。但是，还可以通过反转索引值的方法来解决，它将对索引值的顺序进行反转。这种反转使叶子节点范围内的各项分布更加均匀。举个例子，不能将234、235和236加在索引节点最多的一边，而是将它们反转成432、532和632来存储，当获取数据时再将它们反转回来。这些值在叶子节点中的分布就会变得比较均匀。

反向索引的目的就是要修正由于对标准B-tree索引连续增加递增值而导致的不平衡。

3）位图索引（bitmap index）

在标准B-tree索引中，ROWID是存放在索引的叶子节点中。在位图索引中，索引的每一位都代表一个ROWID。如果某行包含某个值，那么在对应于这个值的位图中，该行的位就会被“打开”。由一个映射函数将位转换成相应的ROWID。和其他索引类型不同，位图索引可包含NULL值的项。

如果索引中的值不是很多的话，那么位图索引占用的空间要比标准B-tree索引少得多。下图说明了位图索引的存储方式：



位图索引的功能在数据仓库应用程序中显得尤为重要，因为数据仓库的每一维都包含了许多重复的值，而且查询操作一般都需要多维数据同时进行。

4）基于函数的索引

基于函数的索引是从Oracle 8i开始引入的。基于函数的索引是基于SQL函数的结果而构建，而标准B-tree索引或者位图索引都是基于列值而构建，除此之外，这三种类型索引的含义都相同。

在Oracle 8i之前的版本中，如果期望根据函数结果来选择行的话，Oracle首先会从数据库中获得所有的行，然后对每一行执行函数，进而决定接受或是拒绝该行。借助基于函数的索引，可以简单地利用索引来进行选择，没有必要逐行执行函数。

举个例子，在不使用基于函数索引的情况下，如果需要对数据执行不分区大小写的选择，那么必须在WHERE子句中使用UPPER函数，即先执行SQL语句获取每一行，然后对它们执行UPPER函数。在使用基于函数索引的情况下，就可以创建基于UPPER函数的索引，然后直接从索引中获取相应的行。

（注意：在Oracle 10g中，用户可以执行大小写不敏感或者重音不敏感的查询，这也胃上面的问题提供了另外一种解决方法。）

如果能够在Oracle数据库中创建自己的函数，这种索引的功能将会变得更有价值。用户可以先构造一个非常复杂的函数，然后创建一个基于该函数的索引，这样可以极大地改善利用函数进行查询的性能。

5）不可见索引

Oracle 11g引入了一个新地索引类型——不可见索引。在通常情况下，所有的索引都会被优化器使用。用户可以通过让索引下线或者删除索引的方式让优化器忽略索引。但是，如果采用了这两种方法，当日后用户需要将索引应用于数据库环境中的时候，必须采取一系列的操作将索引更新为最新状态。

如果用户只是希望让优化器在一段时间内忽略索引，那么又该怎么办呢？有了不可见这个选项，索引可以被相关的访问步骤所忽略，但是对底层数据进行的更新和删除操作仍然可以应用于索引。

### 4.4分区

对于Oracle 8企业版及其他版本而言，用户可以购买分区选项。这个选项用于帮助用户对表和索引进行分区存储。对数据结构进行分区存储意味着用户可以将数据结构中的信息分别存储在多个物理存储区中。数据结构中的信息通常根据表的列值进行划分。用户可以根据表中列值的范围划分表，也可以根据散列函数的值划分表（对表中一列或者多列值进行散列运算返回的值）。在Oracle 9i中，用户还可以使用一系列的值来定义分区，这在数据仓库的环境中使用特别有用。Oracle 11g增加了间隔分区特性，即可以自动地生成固定间隔的新分区，户这话当插入的数据不再现存分区的范围内，自动产生新范围的分区。

用户可以结合不同的分区方法拥有两层分区，即符合分区。在Oracle 11g之前的版本中，用户可以结合范围分区和散列分区这两种方法进行分区。Oracle 11g支持列表分区与列表、范围或者散列分区结合使用，也只会将范围分区与其他范围的分区结合使用。

Oracle非常智能，它能够通过以下两种方式充分利用分区特性来改善系统的性能：

* Oracle不愿费力查询哪些根本不包含任何满足查询的数据的分区。
* 如果分区中所有数据都满足查询WHERE子句的部分条件，Oracle会简单地选取分区中的所有行，而不会为每一行费力计算WHERE子句。

经过分区的表在数据仓库中特别有用，因为在数据仓库中所有的数据都可以基于其时间的跨度来分区。

同样重要的是，分区能够从根本上减小维护操作的范围，增加数据的可用性。用户可以在单个分区上执行所有的维护操作，包括备份、恢复和加载。这种灵活性使得处理差大型数据结构成为可能，因为即便是对超大型结构，这些维护操作也能够在合理的时间范围内完成。另外，如果用户出于某种原因必须恢复表的一个分区，该表的其他分区仍然可以在恢复操作期间保持在线。

如果用户使用的其他数据库并不提供这种类型的分区，那么可以通过下面的方法来实现类似的功能，即把表分成几个单独的表，然后使用UNION SQL命令来同时查看这几个表中的数据。对表进行分区存储具有上面做法的所有优点，而且使用起来没有任何复杂性。

为了让分区的优点最大化，有时候我们必要对表和索引进行相同的分区，让表分区和索引分区能够映射到相同的行。这种分区方法也叫对等分区。由于删除分区等标准操作能够透明地在索引分区和表分区上执行，因而LOCAL索引也简化了维护工作。

Oracle不断地增加分区的功能。从Oracle 10g发布开始，用户可以在线重组各个分区，分区的最多数目从64KB-1增加到128KB-1，借助分区修剪的查询优化也得到了改进。

Oracle 11g进一步改进了分区修剪的功能，使得应用程序可以控制分区，并增加了分区顾问来帮助用户理解哪些时候分区能够改善Oracle数据库的性能。

## 5、其他数据结构

### 5.1顺序

多用户数据库遇到的一个严重问题就是难以为键或者标识符提供一个唯一的序号。针对这种情况，Oracle允许用户创建一个顺序对象。顺序对象本身非常简单。当有人向它请求一个值的时候，它就返回一个值，并将增加其内部值，避免和请求应用发生任何的资源竞争和其他费时的交互。Oracle会在缓冲区为序号存储某个范围的数值，这样访问下一个数值就不用进行磁盘I/O操作。SGA中的数据区间可以满足应用程序的请求。

顺序号的定义包含一个名字、一个递增的数值以及其他的一些相关信息。它独立于任何特定的表面而存在，因而多个表可以同时使用一个顺序号。

设想一下，如果不使用Oracle顺序对象会发生什么问题。可以把最新的顺序号存储在表的某个列中。如果用户想得到下一个顺序号，就必须先读出上一个顺序号，对它加上某个固定值，然后再将新的顺序号写进列中。但是，如果同时有多个用户想获得一个顺序号，他们在写入新的顺序号之前都要读取所谓的“最后”

顺序号。当然我们可以锁定列中包含该顺序号的行，但是这样做会导致其他用户产生延迟。如何解决这个问题？答案就是创建一个顺序对象。

Oracle 11g运行在PL/SQL表达式中使用顺序号。

### 5.2同义词

所有的Oracle数据库的数据结构都存储在一个特定的模式（schema）中。模式与一个特定的用户名相关联，所有的对象都可以通过模式名加对象名进行引用。

例如，名为DEMO的模式中有一个名为EMP的表，这个表可以使用完整的名字DEMO.EMP进行引用。如果用户并没有提供特定的模式名，那么Oracle则会假设这个结构存在于当前用户名对应的模式中。

模式是一个很好的特性，因为对象名只需要在它自己的模式唯一，但是这种对象名容易产生混淆，特别是对终端用户来说。为了使名字更加简单并更加具有可读性，用户可以为所有的表、视图、快照或顺序创建同义词，也可以为PL/SQL过程、函数或软件包创建同义词。

同义词可以是共有的（public），即数据库的所有用户都可以使用它们；也可以是私有的（private），即只有该模式中包含这个同义词的用户才可以使用它。举个例子，一个名为DEMO的用户为其模式中的表EMP创建可一个共有的同义词EMP，其他用户都可以通过EMP来引用DEMO模式中的表EMP。如果DEMO这个用户没有创建公有的同义词，而SCOTT这个用户希望用名字EMP来引用DEMO模式中的表EMP，他就是要在自己的模式中创建一个私有的同义词。当然，SCOTT的最终目的是要访问DEMO模式中的表EMP。

同义词简化了用户对数据结构的访问。借助同义词还可以隐藏某个特定数据结构的位置，使得数据更容易传输；另外，隐藏模式所有者的名字还可以增加相关表的安全性。

在Oracle 10g之前的版本，如归用户改变了某个同义词所引用的位置，那么必须重新编译任何访问这个同义词的PL/SQL过程。

### 5.3集群

集群是一种能够改善数据获取性能的数据结构。集群和索引一样，不会影响表的逻辑视图。

集群提供了把磁盘上相关数据存储在一起的一种办法：Oracle每次只读一块数据，因此，如果把这些相关数据存储在一起，那么一个数据块只包含相关的行信息，这样可以减少I/O操作的数目。

集群由一个或多个表组成。它包括一个集群索引，用以存储所有集群键的值。集群索引的每一个值都有指向一个数据块，该数据块只包含和该集群键有相同值的行。

如果一个集群包含多个表，这些表需要连接在一起，那么集群索引应该包含构成连接基础的值。由于集群键值控制了与该键相关行的位置，如果改变该键值就会使Oracle改变与该键值相关的行位置。

集群不适用于需要定期进行完全扫描的表。对于这样的表，查询需要Oracle数据库对所有记录进行循环操作。由于需要通过集群索引来访问集群表，而它们指向一个数据块，因此对集群表执行全表扫描实际上要求更多的I/O操作，严重影响整体性能。

### 5.4散列集群

散列集群和集群虽然相似，但也存在一个显著的区别，从而使得散列集群的操作速度更快。每次对集群表数据的请求至少包含两次I/O操作，一次用于集群索引，一次用于数据。散列集群将相关数据行存储在一起，并根据集群键的散列值对行进行分组。散列值由散列函数计算得出，这意味着每次数据获取操作都要先计算散列值，然后直接转到包含相关行的数据块。

由于不需要通过集群索引，因而从散列集群表中获取数据比集群表更快。用户可以通过在创建集群的时候指定HASHKEYS参数来控制散列集群中可能散列值的数目。

因为散列集群直接指向表中行的位置，所以在创建集群的时候，必须为散列集群中所有可能的值分配空间。

当所有行在散列键值范围内均匀分布，此时散列集群效率最高。可能存在这样一种情况，散列键对应的列已经有了一个唯一的值，例如，唯一的ID值。这个时候，可以直接对散列键赋予这个唯一值作为散列函数的计算结果，这样做可以避免在数据获取过程中执行散列函数。另外，用户可以为散列集群定义自己的散列函数。

Oracle 10g引入了有序散列集群，即数据不仅根据散列值存储在集群中，而且还要根据插入的顺序进行排序。当应用程序需要根据插入顺序访问数据的时候，这种数据结构能够进一步提高性能。

## 6、数据的扩展逻辑

Oracle还增加了两种特性，这些特性并不是什么数据结构，而是用于指导用户如何使用数据库中的数据。它们是规则管理器和表达式过滤器。

### 6.1规则管理器

Oracle数据库总是在不断地扩展自身的功能，从简单的带有逻辑属性的数据存储到存储过程。Oracle 10g进一步扩展数据库的功能，引入了规则管理器。

规则管理器就是将规则存储在数据库中，供应用程序调用和计算。如果业务条件或者需求发生变化，规则也可以随着改变，而不需要改变应用程序的代码。规则可以在多个应用系统中共享，使它们标准化，减少维护这些应用程序的工作量。用户也可以创建细粒度的规则，用于不同的组合，从而实现各种环境条件。

规则可由事件调用。事件会导致规则被计算，也会导致规则行为立即或者一段时间之后被执行。

规则管理器采用事件条件行为结构，它可以帮助用户定义规则管理器应用程序所需要的五个元素：

* 定义事件结构，它是Oracle数据库中的一个对象。不同的事件，其事件对象

的属性值也不同。

* 创建规则，包括条件和随后的行为。
* 创建规则类别，使用类似的结构来存储规则，并对规则进行分组。
* 创建PL/SQL过程来实施规则。
* 定义结果视图，配置规则，用于PL/SQL行为不能被调用的外部场合，例如，一个运行在多个层次的应用程序，它的规则行为可以被应用服务器层调用。

用户可以定义冲突解决过程来处理多个规则匹配一个事件的情景。规则管理器还可以将多个不同的事件组织成一个复合事件，并维护它的状态直到所有子事件都已经收到为止。

规则是实现复杂逻辑的一个非常强大的工具，但它也会影响应用程序的设计。

### 6.2表达式过滤器

从Oracle 10g起，表达式过滤器就开始可用。它可以利用规则管理器来处理表达式。表达式是另外一种对象类型，它包含的属性可以由表达式过滤器来计算。用户给表增加了一个VARCHAR2类型的列。这个表存储着一个表达式的属性值。用户使用PL/SQL内置包将这个表达式加到这个列上，并使用标准SQL语句来给这个表达式赋值。为了比较数值和表达式，用户可以在SQL语句的WHERE子句中使用EVALUATE运算符。

由于表达式可以拥有许多的属性，因而表达式可以用于定义复杂的属性。用户甚至可以使用表达式来实现多对多的关系，而不需要使用表达式将两个表关联起来形成一个中间的表。

## 7、数据设计

在关系型数据库中，表和列表示了数据的逻辑视图。关系型数据库的灵活性使得在把原子数据（用列表示）分配到不同的表时有了更多的选择。为了更加有效地使用Oracle，用户必须理解并遵循某些数据库设计原则。

规范化数据设计背后的基本原理：

一个表只能包含与该表键值直接相关的信息。组织信息逻辑单元的这个过程称为数据库的规范化设计。

范式

实际上，有许多类型的规范化。规范化过程的每一步都结束于范式这个特定的结果。共有五种标准范式：第一范式（1NF）、第二范式（2NF）等等。这一节将简要描述第三范式（3NF），这也是规范化中最通用的类型。

规范化表设计的概念经过了不断地修改以适应目前关系型数据库的功能。因为一个查询可以将不同表中的数据关联起来，所有没有必要把某个特定对象相关的所有信息都保持在一条记录中。你应该将信息分解成相关的单元，当需要多个表信息的时候，只需简单地将适当的单元关联起来即可。



规范化数据有许多不同的办法，下面的例子介绍如何为应用定义需要的所有数据：

1. 为应用标识它所需要知道的对象（实体）。如图，实体的例子有employee，

location以及job。

1. 为这些实体标识数据的各个部分，数据建模人员将它们归为属性。如图，employee name和salary就是属性。一般来讲，实体对应表，属性则对应列。
2. 最后一步，根据业务关系来标识实体间的关系。在数据库模式中，这种关系都是通过外键结构来实现的。例如，DEPARTMENT NUMBER表中的主键可以是EMPLOYEE NAME表中的外键列，用于表示员工工作的部门编号DEPARTMENT NUMBER。外键也是一种约束类型。

规范化过程的优点是可以有效避免数据冗余。每个员工记录中都存储部门信息，不仅浪费空间，而且会导致数据库难以维护。如果部门名改变了，相应地就必须修改每个员工记录。如果将部分信息都归到一个表中，员工记录中只需简单地指向合适的部门记录，这样就可以避免数据重复等相关问题。

规范化还能减少表中每一行的数据总量。行中数据越少，获取数据所需的I/O操作就越少，这也有助于避免执行过程中的瓶颈问题。另外，行中数据越少，每个数据块就可以包含更多的行，这就增加了每次I/O操作能够获取更多符合条件数据行的可能性。而且行中的数据越少，Oracle系统缓冲区就可以存放更多的行，这样在需要某行时，能够直接命中的几率就越大，从而进一步减少磁盘I/O操作。

最后，规范化过程还包括外键关系和其他数据约束的创建。在数据库设计中，这些关系直接构建了一个层次的数据完整性。

不过，研究规范化数据库设计过程还有更加重要的原因。规范化的数据库设计过程将让数据库开发人员受益。通过认真思考目标应用试验数据的方式，可以更加清除的了解系统的设计需求。这样做才可以构建出一个更加专业的数据库和应用程序。

深入理解数据的应用方式也会对你的其他设计工作有帮助。例如，一旦完成了一个最佳的逻辑数据库设计，必须返回去思考应该添加什么样的索引来提高数据库的性能，是否应该将某个表指定为集群或者散列集群的一部分。

由于增加这些用于提高系统的数据结构并不影响数据库的逻辑表示，因而可以在掌握了应用程序测试阶段或者实用阶段使用数据库的方式之后再来做这一类的工作。

### 7.1约束

约束（constraint）用于实施数据库中某些数据完整性。当给某列增加一个约束，Oracle自动确保不满足此约束的数据是绝对不能被接受的。如果用户试图写一个不满足约束的数据，那么Oracle就会对这个非法的SQL语句产生一个错误。

约束可以在创建或者增加包含某列（通过关键字）的表时与该列进行关联，也可以在表创建后通过SQL命令ALTER TABLE来实现与该列的关联。在Oracle 8及之后的版本中，支持以下几种约束类型：

* NOT NULL

任何列都可以设为NOT NULL。如果在SQL操作中将一个NULL值赋给某个有NOT NULL约束的列，Oracle会为这个语句返回一个错误。

* 唯一性

如果将某个列设置为唯一，那么就不能在表中插入和这个列中已有值重复的行，也不能修改已有的列值使之与其他列值重复。

唯一性约束可以通过创建索引来实现，索引要求每行有一个唯一值。如果用多个列组成唯一键，就可以创建一个索引，它的唯一键包含所有的列。如果已经存在用于这样目的的索引，Oracle将自动使用已有的索引。

如果一个列具有唯一性但它又允许NULL值，那么任何行都可以含有NULL值，因为NULL值表示不含任何值。如果希望一个列的每一行都有一个真实的唯一值，那么应该将该列设置成唯一且是NOT NULL。

* 主键

每个表最多可以有一个主键约束。主键约束可以由表中的多个列组成。

主键约束保证每个主键都有一个唯一的值。因而可以说，主键约束既实施了唯一性约束，又实施了NOT NULL约束。如果指定的列事先不存在索引的话，主键约束将会创建唯一索引。

* 外键

外键约束是为了数据库中某个与其他表（称作父表）有关系的表（称作子表）而定义的。外键的值必须事先出现在某个特定表的唯一键或者主键中。例如，员工表中的部门ID列可以是部门表中部门ID主键的外键。

外键可以包含一列或者多列，但是其所参考的键也必须包含相同的列。外键可以和同一个表中的主键相关联，例如，经理的员工ID这个外键参考着同一个表中的ID列。

如果没有其他约束限制，外键可以包含NULL值。

由于要求外键的值必须存在于其他的表列中，因而外键约束在数据库中实施了参照完整性。外键不仅为关联相关的表提供了一种方式，同时也确保了两个表之间的关系具有要求的数据完整性。

通常来说，如果删除父表中的某行会导致子表中的行违背外键约束，那么就不能删除父表中的这一行。不过，也可以在外键约束中指定级联删除，意思就是说，删除父表中被参照的行将会导致子表中所有参照父表中被删除行主键值的行自动被删除。

* 校验

校验约束是一种更加通用的约束。校验约束就是一个布尔表达式，它的计算值要么等于TRUE，要么等于FALSE。如果校验约束的计算值为FALSE，那么导致这一结果的SQL语句则会返回错误。例如，有这样一个校验约束，它要求一个银行账户中的最小余额应大于100美元。如果一个用户试图更新该账户的数据并使得该账户的数据并使得该账户的余额小于这个要求的数量，那么这个校验约束将返回错误。

有些约束需要创建索引来辅助自己。例如，唯一性约束会创建一个隐式的索引来保证唯一性。你也可以在定义约束的时候指定一个特定的索引来实施这个约束。

约束的效果可以是立即实施，也可以是延迟实施。立即约束只要有写操作就会对表中被约束的列产生效果。延迟约束仅当导致被约束的列发生变化的SQL语句执行完成之后才会产生效果。由于单个SQL语句可能会影响几行，因而选择延迟约束还是选择立即约束将会在很大程度上影响该约束所实施的完整性。可以为单个约束指定是立即实施，还是延迟实施，也可在单个事务中为所有约束的实施设置时间限制。

### 7.2触发器

在用户试图写或者修改表中的某一行时可以使用约束自动实施数据完整性规则。有时候，你需要在自己的应用逻辑中使用类似的定时功能。因此Oracle引入了触发器。

注意：虽然可以编写触发器来执行约束检查工作，但是由于Oracle已经对约束操作进行了优化，因而如果可能的话你最好使用约束来代替触发器。

触发器本身是一段代码，每当特定的数据库事件发生在一个表上时，它就会被执行。有三种类型的事件会导致触发器被执行：

* 数据库UPDATE操作。
* 数据库INSERT操作。
* 数据库DELETE操作。

例如，可以定义一个触发器，每当用户修改行的时候自动写一条定制的审查记录。

触发器是定义在行这个级别。可以指定触发器根据行来触发，或者根据产生触发事件的SQL语句来触发。单条SQL语句会影响多个行，因而为触发器指定不同的属性将会极大地影响触发器的操作和数据库的性能。

触发器的触发有三个时机：

* 在触发事件执行之前。
* 在触发事件执行之后。
* 非触发事件。

将前两种触发时机选项和触发器所触发的行和语句相结合，则有四种可能的触发器实现：

* 在语句之前
* 在行之前
* 在语句之后
* 在行之后

Oracle 11g引入了复合触发器的概念，做了这样的改进之后，单个触发器可以包含各种时机对应的实现。由于这样的复合触发-\*

-器对于多个执行时机只需加载一次，因而有助于性能的提高。

Oracle 8引入了INSTEAD OF触发器。这种触发器有一个特定的目的，即：在通常被禁止的情况下，对视图中的数据进行处理，例如，对引用了多个基表的列的视图进行修改。在使用INSTEAD OF触发器时必须小心翼翼，因为修改视图基表中的数据会产生许多隐含的问题。使用INSTEAD OF触发器有许多限制（参考文档）。

任何触发器都可以有一个触发器限制。触发器限制是一个布尔表达式，如果它的值为FALSE，那么就会阻止触发器的触发。

触发器的定义和存储独立于使用触发器的表。由于触发器包含业务逻辑，因而必须使用功能强于SQL的某种数据访问语言来编写。从版本6开始，Oracle就包含了过程化语言PL/SQL。Oracle 8及之后的版本允许用户使用PL/SQL来编写触发器。Oracle 8i及之后的版本甚至还支持用+户使用过程化语言Java来编写触发器。

可以直接使用PL/SQL或者Java来编写触发器，也可以在触发器中调用现有的使用这两种语言编写的存储过程。

触发器的触发是那些会影响特定表中行的SQL语句执行的结果。触发器的行为可能会修改表中的数据，也可能会修改其他的表，而这些表激发它们自己的触发器。这样做最终的结果可能是：数据被修改了，但是Oracle认为这是不符合逻辑的。在这种情况下，Oracle会返回运行时错误，并指出被其它触发器修改的异常表，或者指出被其他约束修改的约束表。Oracle 8i删除了一些由于为触发器激活约束而导致的错误。系统时间触发器（有时候叫做数据库级别事件触发器）和用户事件触发器（有时候叫做模式级别事件触发器）。例如，可以将触发器放置在诸如数据库启动和关闭这样的系统事件上，也可以将触发器放置在诸如登入和退出这样的用户事件上。

## 8、查询优化

到目前为止本章讨论的所有数据结构都是服务器实体。而用户必须通过数据库查询向Oracle服务器请求数据。因而Oracle查询优化器必须能够针对每个查询确定最佳的数据访问方法。

不需要事先定义访问数据的路径是关系型数据库的优点之一。当用户向Oracle数据库提交一个SQL查询的时候，服务器必须确定如何访问数据。由于Oracle需要找到一种最佳的方式来检索数据，因而这一决策的过程又叫做查询优化。这种检索叫做执行路径。查询优化背后的技巧就是如何在众多可用的方式中选择一种最有效的方式来获得数据。

例如，即使是对于只涉及一个表的查询，Oracle也会采用下列方式之一：

* 使用索引来查找请求行的ROWID，然后直接从表中检索这些行。
* 对表进行扫描并检索每一行，这叫做完全表扫描。

虽然在通常情况下，使用索引来检索数据要快得多，但是根据索引获取数据会涉及附加的I/O操作。而查询优化可能与确定查询中是否包含应用于索引数据的选择条件一样简单。利用索引值来选择符合要求的行所需的I/O操作较少，这样做比从表中获取所有数据然后再对数据执行选择条件的效率高很多。

确定最佳查询执行计划的另一个因素在于查询中是否包含ORDER BY条件，预排序索引可自动实现这一功能。相反的，如果表很小，优化器可能会选择读取数据库中所有记录，而不使用索引，因为经过估算，索引的I/O操作加上表的I/O操作比起只有表的I/O操作而言，开销要大一些。

查询优化器必须做出某些关键性决策，即便是对于仅有单个表的查询。当用户提交的查询比较复杂，例如，涉及许多表的查询，其中这些表必须被有效地连接在一起，或者是设计复杂的选择标准和多重排序的查询，查询优化器必须完成更为复杂的任务。

在Oracle 10g之前，用户有两种不同的Oracle查询优化器可以选择：基于规则的优化器和基于成本的优化器；Oracle 10g不再支持基于规则的优化器。

### 8.1基于规则的优化

Oracle一直都提供查询优化器，但是在Oracle 7之前所提供的优化器仅是基于规则的。基于规则的优化器，就是使用预先定义的规则集合作为查询优化的主要决策依据。

基于规则的优化在某些情况下，它的性能比Oracle早期版本提供的基于成本的优化性能要高。但是它自身也有缺点，其中之一就是它只能提供简单化的规则集。Oracle基于规则的优化器提供大约20个规则，每个规则都被赋予了一个权重值。在复杂的数据库中，一个查询往往会涉及几个表，每个表又有几个索引，同时又具有复杂的选择条件和复杂的排序。这种复杂性意味着许多选项，而基于规则的优化器提供的规则结合比较简单，无法很好地判断每种选项进而做出最佳的选择。

基于规则的优化器为每种可能的执行路径都赋予了一个优化分数，然后根据最好的优化分数来选择最佳路径。基于规则的优化器的另一个缺点是当分数相同时无法确定优化执行语法来做出判断。最终被选定的执行路径往往由各个表的SQL语句中出现的顺序来决定。

通过思考下面的例子，你就会明白当分数相同时这种类型的处理方法可能产生的潜在影响。如图，在一个简单的环境中，SMALLTAB是一个只有10行数据的小型表，它需要和一个有10000行数据名为LARGETAB的大型表关联起来。如果优化器选择首先读取SMALLTAB中的数据，那么Oracle数据库将会读入其中全部的10行数据，然后开始读取LARGETAB并查找与SMALLTAB中数据相匹配的行。如果优化器决定首先读取LARGETAB，数据库会从LARGETAB中读入10000行数据，然后会读取SAMLLTAB10000次来查找匹配的行。当然，SAMLLTAB中的行可能会被缓存，这样可以减少每次查询的开销，但是可以看到这两种优化选择在性能上仍然存在巨大的差异。



由于基于规则的优化器会根据查询中的表名顺序来做出优化决策，因而上述的这种性能差距就可能出现。在前面的例子中基于规则的优化器返回的结果虽然相同，但是为获取到这些结果而消耗的资源则是大不相同。

### 8.2基于成本的优化

为了改进SQL语句的优化性能，Oracle 7开始引入了基于成本的优化器。基于成本的优化器比基于规则的优化器复杂许多，它所做的不仅仅是简单地查询优化规则的集合，它会根据逻辑I/O操作最少的原则来选择执行路径。这种方法可以避免我们在前一节讨论的错误。毕竟，基于成本的优化器能够知道哪一个表更大一些，它会选择合适的表开始查询，而不用考虑SQL语句的语法。

Oracle 8及之后默认使用基于成本的优化器来确定最佳执行计划。而且，从Oracle 10g开始，基于成本的优化器是Oracle唯一支持的优化器。为了正确衡量所有执行计划的成本，基于成本的优化器会对相关的数据结构的组成进行统计。Oracle 10g及之后的版本默认自动将这些统计信息收集到自动工作负荷仓库（AWR）。AWR收集的统计信息具体包括：数据库段访问及段使用统计、事件模型统计、系统和会话统计、负荷量最大的SQL语句统计以及活跃会话历史（ASH）统计。

1）如何使用统计

基于成本的优化器也是通过为每种可能的执行计划赋予一个优化分数来确定最佳的执行计划；不过这里的打分依据是其自身的内部规则和逻辑，以及能够反映数据库内部结构的状态的统计。这些统计与执行计划所包含的表、列以及索引相关。下表列出了每种类型数据结构的统计。

|  |  |
| --- | --- |
| 数据结构 | 统计类型 |
| 表 | 行的数目  数据块的数目  未使用的数据块数目  每个数据块可用的空闲空间  链接在一起的行的数目  行的平均长度 |
| 列 | 每个列不同值的数目  第二小的列值  第二大的列值  列的密度因子 |
| 索引 | B-tree索引结构的深度  叶子块的数目  不同值的数目  每个键叶子块的平均数目  每个键数据块的平均数目  集群因子 |

Oracle 10g之后还会收集整个系统的统计，包括I/O和CPU的性能和利用率。这些统计都存储在数据字典中，本章最后一节“数据字典表”将会对此进行描述。

你将会明白这些统计可以单独使用，也可以结合起来用于确定某个执行计划所需要的整体I/O开销。这些统计既可以反映表的大小，也可以反映数据块中未使用空间的数量，这个空间反过来能影响获取行数据所需要的I/O操作的数量。索引统计不仅能够反映索引树的深度和广度，还能够 反映树中每个值的唯一性，从而使得该索引可以简化数据的选择。

注意：

基于成本的优化器它的准确性取决于它所使用的统计的准确性，因而必须经常更新统计。之前，用户必须使用SQL语句ANALYZE来计算或者评估这些统计。在管理旧版本的Oracle数据库的时候，许多数据库管理员还会使用DBMS\_STATS这个内置的PL/SQL包，该工具包包含许多可用于自动收集统计的程序。

陈旧的统计信息会导致数据库的性能问题，因而Oracle数据库会自动收集最新的统计信息。这种统计收集工作粒度非常细。例如，在Oracle 10g中，用户可以根据一个表是否陈旧（这意味着表中百分之九十以上的对象发生了变化）或者是否为空表来开启针对这个表的自动化统计收集功能。

借助于统计信息，基于成本的优化器通常能够做出深思熟虑的决策，选择最佳的执行计划。例如，这种优化器可以在两个索引之间确定索引值的执行计划。而基于规则的优化器可能会给这两个索引打同样高的分，然后根据他们在WHERE子句中出现的顺序来选择执行计划。然而，对于基于成本的优化器而言，它知道一个索引有1000个项，而另一个索引有10000个项，它甚至了解包括1000个值的索引只包含20个不同的值，包含10000个值的索引包含5000个不同的值。这样选择大一点的索引效果会好很多，因而该索引被赋予一个比较高的优化分数，并被用于查询。

测试新统计信息的效果：

虽然某些时候不需要对统计进行更新，比如当数据库的数据分布已经达到一个稳定状态，或者当查询已经以最佳方式被执行（或者至少能够提供足够的一致性）。Oracle在保留返回到旧的统计信息这个选项的同时也会为用户提供一种验证新的统计信息是否会有帮助的方法：将旧的统计信息存储在单独的表中，然后收集新的统计信息。如果在应用程序中对这些新的统计测试之后，你还是喜欢旧的统计信息的工作方式，那么可以简单地重新加载存储起来的统计信息。

在Oracle 9i中，对于基于成本的优化器而言，可以将CPU的速度作为决定最佳执行计划的一个因素，并且有一个初始化参数用于启动或关闭此项特性。

即便是利用了所有的可用信息，基于成本的优化器还是有其自身的缺点。除了偶尔会发生问题之外（同其他的软件一样），基于成本的优化器由于使用的是统计信息，因而不能提供对数据结构的完整描述。在前面的例子中，统计信息告诉优化器的唯一信息就是索引中不同值的数量。它们不能显示那些索引值的分布情况。例如，大一点的所有中包含了5000个不同的值，这些值可能平均分布在所有的行中，或者一个索引值对应了50001个行，而其他索引值分别只对应一行，这两种情况下，选择索引的效果是大不相同的，这还要取决于SQL语句中所使用的选择值。Oracle 7.3为了解决这个问题，引入了对索引的柱状统计图的支持。在Oracle 11g之前的版本中，用户可以自己收集统计信息，利用ANALYZE INDEX命令来创建柱状图。

2）影响基于成本的优化器的因素

用户可以通过两种途径来影响成本的优化器选择执行计划的方法。第一条途径就是设置OPTIMIZER\_MODE初始化参数。ALL\_ROWS是该参数额的默认值，用于打开以最佳吞吐量为目的的优化。FIRST\_ROWS这个选项用于优化根据SQL语句返回第一批行数据。可以使用这个参数指定行的数目。优化器的模式会对评估优化分数产生轻微影响，不过，有时候即便是很小的影响也会导致最终确定不同的执行计划。

Oracle同时还提供了另一种方式来影响优化器的决策，其中用到了一种叫做提示的技术。提示相当于SQL语句中具有某种特定格式的注释。分为以下几类：

* 优化器SQL提示，用于改变查询优化器的目标。
* 全表扫描提示。
* 索引唯一性扫描提示。
* 索引范围扫描降序提示。
* 快速全索引扫描提示。
* 关联提示，包括索引关联、嵌入式循环关联、散列关联、排序合并关联、笛卡尔关联以及关联顺序。
* 其他优化器提示，包括访问路径、查询转换以及并行执行。

提示也有其自身的问题。在下面这个简单的SQL语句中，提示看起来和注释一样。它强制要求优化器在对EMP表的操作中使用EMP\_IDX索引：

SELECT /\*+ INDEX(EMP\_IDX) \*/ LASTNAME, FIRSTNAME, PHONE FROM EMP

如果一个提示没有放在SQL语句中的适当位置，或者提示关键字拼写错误，也有可能是数据结构被修改，那么这个提示就不能指向一个现有的结构，从而被忽略，就好像注释一样。因为提示是包含在SQL语句中的，所以在提示不能正常工作的情况下，修改它既耗时也容易失败。另外，如果在SQL语句中增加提示来解决由于基于成本的优化器中出现的故障而导致的问题，而随后这一故障得到修复，但SQL语句仍然不会使用正确的（以及可能被改进的）优化器。

然而，提示非常适合某些场合，比如当开发人员拥有自定义数据类型，从而要完成特定类型的访问时。优化器不能预期用户自定义数据类型的效果，但是提示可以恰当地修改检索路径。

### 8.3指定优化器模式

前面提到两种优化器模式：ALL\_ROWS和FIRST\_ROWS。Oracle 10g之前的版本还提供另外两种优化器：

RULE——强制使用基于规则的优化器。

CHOOSE——允许Oracle自己选择使用哪种优化器。

CHOOSE优化器模式过去是Oracle的默认设置。选择这种模式，如果SQL语句中所有表都有相关的统计信息，那么Oracle会使用基于成本的优化器。基于成本的优化器还会对那些没有统计信息的表进行统计性的估计。如果正在运行使用郭泽的旧版本Oracle数据库，你可能想知道迁移到只能使用基于成本的优化器的新版本Oracle数据库是否是个好主意。

### 8.4较新的数据库发布和基于成本的优化器

基于成本的优化器在做出决策之前需要掌握有关数据库中数据结构的各种信息。尽管基于成本的优化器在制定决策时并不是没有毛病可挑，但是它做出的决策时基于各种信息的，而且自从Oracle 7将其引入以来就不断地得到各种改进，日趋走向成熟，具有较高的准确性。

基于成本的优化器每次发布都会考虑到Oracle数据库自身的改进和新增的特性。例如，基于成本的优化器懂得对表进行分区存储对选择执行计划的影响，而基于规则的优化器却不能。基于成本的优化器能够优化星模式查询的执行计划，数据仓库会大量使用这类星模式查询，而基于规则的优化器则无法通过自身的不断改进来有效处理这类查询，也不能推动其他许多商业智能查询特性。

接受优化器的决策：

如果你正在运行旧版本的Oracle数据库，特别是10g之前那些需要执行脚本进行调整的Oracle数据库，你可能会质疑Oracle查询优化的效果。可能你也讲过这样一些案例，查询优化器选择错误的执行路径而导致性能下降。你也可能会觉得你比查询优化器更了解数据库的结果和用法。出于上述原因，你甚至可能会使用提示来强制数据库使用你觉得正确的执行路径。

建议对所有的查询都使用查询优化器而不要使用提示。

查询优化器相对于人为判断而言总是有三个优点：

* 优化器可以看到整个数据库的结构。许多Oracle数据库支持各种应用程序和用户，而且一个系统很可能会与其他系统共享数据，这就会使得数据的整个结构和组成在一定程度上超出你的控制范围。另外，你可能是在有限的环境中设计并测试自己的系统，因而你对最佳执行路径的看大未必适合实际的产品环境。
* 优化器掌握着一个有关数据库及其数据的动态视图。在基于成本的优化器中使用的统计信息能够随着每次统计信息收集的完成而得到更新。除恶变化的统计条件之外，优化器内部偶尔也会发生改变以适应Oracle数据库运行方式的变化。从Oracle 9i开始，基于成本的优化器开始考虑CPU的速度；从10g开始，Oracle数据库会充分利用I/O相关的统计。如果你借助提示来强制使用某个特定的查询计划，那么你将无法受益于Oracle的不断改进。
* 优化器有时候也会做出错误的选择，不过这个时候可能预示着数据库存在某些问题。对于大多数情况而言，查询优化器总是能选择最佳的执行路径。查询优化器所发现的错误实际上往往源于对数据库及设计的误解或源于实施过程中出现的问题。错误通常是一个学习的机会，可以把握任何机会来增强自己对Oracle及其优化器工作方式的全面理解。

建议只有当你经过深入研究了可能会导致优化问题的各种因素之后觉得非常必要的情况下，才考虑使用提示。提示语法包含在Oracle语法中，它仅作为处理异常情况的一种方法，而不能用于取代查询优化器。如果发现性能出现异常，并通过进一步调查发现查询优化器选择了一个错误的执行路径，当且仅当这个时候可以使用提示来查询。

即使在这种情况下，仍然建议在产品环境下密切注视提示查询，以保证强制执行的路径仍然工作在最佳状态。

### 8.5保存优化

当一个SQL语句被提交时，有时候你可能会希望优化器不要去计算新的执行计划。例如，如果你觉得当前SQL语句正运行在最佳状态，无论优化器和数据库未来会发生什么样的变化，都不希望再改变执行计划，就可能需要这样做。

从Oracle 8i开始，用户可以创建一个存储概要，用于保存优化器创建执行计划所需的属性。一旦有了这个存储概要，优化器就可以简单地使用存储属性来创建一个执行计划。在Oracle 9i中，用户还可以编辑存储概要中的提示。

在Oracle 11g中，Oracle建议用户将自己的存储概要移到SQL计划的基线。现在，除了可以手动加载计划之外，Oracle还能够自动捕捉计划的历史信息，并将它们存储在SQL计划基线中。Oracle收集的历史信息包括SQL文本、存储概要、绑定变量以及编译环境。当对SQL语句进行编译时，Oracle将会首先使用基于成本的优化器来生成执行计划，计算匹配SQL计划基线的相对成本，选择成本最低的执行计划。

### 8.6比较各种类型的优化

Oracle的每个版本都会对优化器进行改进，从而使之能够做出更高质量的决策。经过改进的优化器能够为任何可能导致性能降低的SQL语句创建执行计划。

SQL语言分析工具（SQL Analyzer）的设计目的就是为了让用户能够识别优化器升级而可能导致的问题。这个工具会对应用程序中SQL语句的执行计划进行比较，并标记出执行计划不同的语句。一旦识别了这些语句，SQL语言分析工具将会在各自的环境中执行SQL语句，并反馈每条语句的执行效率以及它的资源利用率。虽然SQL语言分析工具不能避免由于优化器升级而导致的潜在问题，但是这个工具的确能够简化复杂的测试流程。

Oracle 11g还提供一个数据库重放（Database Replay）的特性。它能够从产品系统中捕获工作负荷并允许它们被运行在测试系统中。有了这个功能，用户可以使用新的配置或新版本的数据库来测试实际的产品环境，数据库重放特性还可以用来监控不同平台可能会出现的性能问题。

### 8.6性能和优化

优化器的目的就是要为查询选择最佳的执行计划。但是优化数据库整体性能可做的工作有很多。

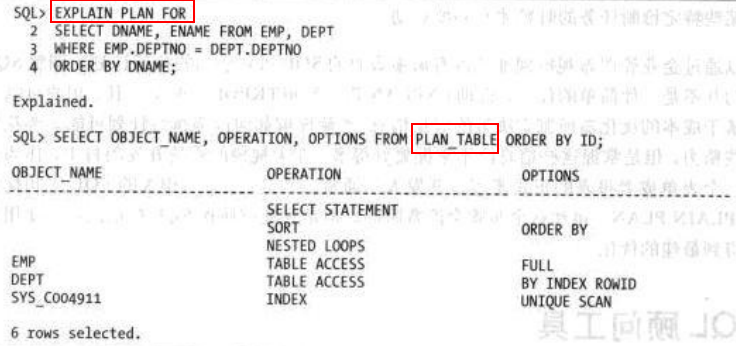
## 9、理解执行计划

Oracle的查询优化器自动为每个提交的查询选择一个执行计划。大体上来说，两种类型的优化器都能为执行计划的选择做出有益的工作，但是也有这样的情况，即数据库的当前性能就能反映出当前使用的是一个未被优化的执行计划。

了解优化器选择路径的方法是查看执行计划的设计。可以使用两种Oracle字符模式的工具来检查Oracle优化器选择的执行计划。利用这些工具可以查看Oracle收集，选择以及向用户返回数据的连续步骤。

1）第一个工具

是SQL EXPLAIN PLAN语句。使用EXPLAIN PLAN语句时，后面要跟上关键字FOR以及所期望观察期执行计划的sql语句，基于成本的优化器会返回一个有关SQL语句执行计划的描述，并回将此描述插入到数据库表中。随后，可以在那个表上执行查询来获得执行计划。



执行计划在表中以一系列的行表示，每行表示在执行SQL语句过程中Oracle采取的一个步骤。优化器还包含了某些与决策相关的信息，如每一步中的全部开销以及用于决策的一些统计信息。

优化器将所有这些信息写入到数据库的某个表中。在默认情况下，优化器使用PLAN\_TABLE这个表：在使用之前应确保它的存在。（Oracle数据库中国的utlxplan.sql脚本用于创建默认的PLAN\_TABLE表。）可以在语句中指定EXPLAIN PLAN 使用PLAN\_TABLE之外的其他表。

2）第二个工具

还有些时候，你可能需要跟踪正在检查的语句，并需要使用另外一种工具——TKPROF。这个工具可以将跟踪的结果信息以更加可读的格式存放在一个独立文件中供你使用。另外有些时候，当你在负责调整的应用程序中使用TKPROF的时候，可能会使用Oracle的SQL跟踪工具来生成一个记录之前产生的SQL操作的文件。

当你在启动TKPROF的时候必须使用EXPLAIN PLAN语句。你还可以指定TKPROF产生的结果如何排序。例如，可以将SQL语句按照它们使用的物理I/O数目来排序，或者按照解析、执行以及获取行数据的时间来排序，或者按照所涉及到行的总数来排序。

TKPROF工具将跟踪文件作为它的原始材料。不同的会话会创建自己的跟踪文件。可以通过运行带有开关的目标应用程序来启动收集跟踪文件，也可以通过在用3GL编写的应用程序中使用EXEC SQL调用或者ALTER SESSION SQL语句显示地将其打开。正如你可能设想的那样，跟踪过程会对应用程序的性能产生很大的影响，因而只有当需要执行某些特定诊断任务的时候才有必要启动。

可以通过企业管理器观察到那些占有最多资源的SQL语句它们的执行计划。调整SQL语句并不是一件简单的任务，借助EXPLAIN PLAN和TKPROF这两个工具，用户可以了解基于成本的优化器所指定决策的底层信息。准确理解如何读懂执行计划可能需要花费一些精力，但是掌握这些信息比不掌握要好得多。在大规模的系统开发项目中，作为完成一个表单或者报表的正确步骤，开发人员通常会为其提交给DBA的SQL语句提交EXPLAIN PLAN。虽然这个步骤会浪费时间，但是它能够确保SQL在正是投入使用之前得到最佳的优化。

## 10、SQL顾问工具

Oracle 10g增加了一个名为SQL调整顾问的工具。这个工具能够根据自动化工作负荷仓库收集到的工作负荷或者自己指定的工作负荷，对选择的SQL语句做高级优化分析。优化一旦完成，SQL调整顾问工具将会作出各种建议，包括：更新统计、添加索引或者创建SQL轮廓信息。这种轮廓信息存储在数据库中，可用于SQL语句未来之星的优化计划。它允许你“修改”错误的SQL计划，而不需要改动底层的SQL语句。

由于这个工具用于为物化视图及索引提供建议，因而它常常和SQL访问顾问工具一起使用。Oracle 11g引入的SQL顾问工具结合了SQL调整顾问和SQL访问顾问这两种工具的功能（现在还包括新的分区顾问工具）。为了改进SQL性能，分区顾问组件会对表格、物化视图以及索引进行分区方法提供建议。

## 11、数据字典表

Oracle数据字典的主要目的就是用来存储用于描述Oracle数据库中对象结构的数据。因此，Oracle数据字典中包含许多视图，它们提供了数据库中数据结构相关的属性和组成信息。

这一节列出的视图可分为三类，名字分别以不同的前缀开头：

DBA\_

包含数据库中所有的对象。用户若要使用这个视图必须具有DBA权限。

USER\_

仅包含用户自己的数据库模式中对象。

ALL\_

包含特定用户能够访问的数据库中的所有对象。如果一个用户被授予了访问其他用户数据库模式中对象的权限，那么这些对象将会出现在这个视图中。

这意味着，例如，有三个与表相关的视图：DBA\_TABLES、USER\_TABLES以及ALL\_TABLES

数据结构相关的数据字典视图：

|  |  |
| --- | --- |
| 数据字典视图 | 信息类型 |
| ALL\_TABLES | 有关对象和关系表的信息 |
| TABLES | 有关关系表的信息 |
| TAB\_COMMENTS | 有关表结构的注释 |
| TAB\_HISTOGRAMS | 有关表的使用情况的统计 |
| TAB\_PARTITIONS | 有关分区表的分区情况的信息 |
| TAB\_PRIVS\* | 详细描述一个表的所有权限的不同视图，这里的权限包括用户授予的全向和授予用户的权限 |
| TAB\_COLUMNS | 有关表和视图中列的信息 |
| COL\_COMMENTS | 有关各个列的注释 |
| COL\_PRIVS\* | 详细描述一个列的所有权限的不同视图，这里的权限包括用户授予的权限和授予用户的权限 |
| LOBS | 有关大型对象（LOB）类型的列的信息 |
| VIEWS | 有关视图的信息 |
| INDEXES | 有关表中索引的信息 |
| IND\_COLUMNS | 有关每个索引中列的信息 |
| IND\_PARTITIONS | 有关分区索引的分区情况的信息 |
| PART\_\* | 详细描述分区表和分区索引的组成和使用模式的不同视图 |
| CONS\_COLUMNS | 有关每个约束中的列的信息 |
| CONSTRAINTS | 有关表中各个约束的信息 |
| SEQUENCES | 有关顺序对象的信息 |
| SYNONYMS | 有关同义词的信息 |
| TAB\_COL\_STATISTICS | 可供基于成本的分析器使用的统计信息 |
| TRIGGERS | 有关表中触发器的信息 |
| TRIGGER\_COLS | 有关触发器中各个列的信息。 |