3

索引和索引-组织表

本章讨论索引，是一种能加速访问表中行的模式对象，还有索引组织表，是用索引结构存储的表。本章包含以下几个部分：

- 索引概述
- 索引组织表概述

索引概述

索引（index）是一个可选结构，与表或者**表簇（table cluster）**相关联，可以在某些时候加快数据访问速度。通过创建**索引（index）**在表的一个或多个列上，在某些情况下，你能增强从表中获取一小部分随机分布行的检索能力。索引是一种更多意味着减少磁盘 I/O 的手段。

如果堆组织表没有索引，那么数据库必须执行**全表扫描（full table scan）**来查找一个值。例如，不使用索引，在 hr.departments 表中定位 2700 的**查询（query）**需要数据库在每一个表块中查找这个值。这种方法不能随着数据量增长而扩展。

打个比方，假设一位人力资源经理有一架子纸箱。箱子中文件夹包含被随机插入的员工信息文件夹。员工 Whalen (ID 200)的文件夹是从 1 号箱子底部向上的 10 个文件夹，而 King (ID 100)的文件夹在 3 号箱子的底部。要定位一个文件夹，经理翻阅了 1 号箱子中的从底到上的每一个文件夹，然后从箱子到另一箱子挨个走，直到文件夹被找到。为了加快访问速度，经理可以创建一个按顺序列出每一个员工的 ID 和文件夹的位置索引：

ID 100: 箱子 3, 位置 1（底）

ID 101: 箱子 7, 位置 8

ID 200: 箱子 1, 位置 10 ...

同样地，经理可以为员工的姓氏、部门 ID 等创建单独的索引。

一般情况下，在下列任一一种情况中可以考虑在列上创建索引：

- 被索引的列是经常查询并且返回在表中总行数的一小部分。
- 引用的**完整性约束（integrity constraint）**存在于索引的一列或多列。

索引是一种用来避免全表**锁（lock）**的手段，否则如果你更新父表的**主键（primary key）**，合并到父表，或者从父表中删除时将被请求全表锁。

- 唯一键约束被置于表中并且你想手动指定索引和所有索引选项。

参见：第 5 章，“数据完整性”

索引特性

索引是和其关联的对象数据在逻辑上及物理上都相互独立的模式对象，所以，索引的删除和创建不会对其关联的表产生物理上的影响。

注意：如果你删除索引，而后应用仍然能工作。然而，访问先前索引的数据可能要慢一些。

索引存在或不存在都无需改变任何 SQL 语句的写法。索引是到单行数据的快速**访问路径（access path）**。它只影响执行的速度。给定一个已被索引的数据值，索引直接指向包含那个值的行的位置。

数据库自动地维护并使用索引在创建它们之后。数据库也自动的反映数据的变化，如增加、更新和删除行，不带有用户需要的额外动作在所有相关的索引中。被索引数据的检索性能几乎保持不变，即使有插入行。然而，有众多索引存在的表上会降低 DML 的性能，因为数据库必须同时更新索引。

索引有以下属性：

- 可用性

  索引有可用的（默认）或不可用的。**不可用索引（unusable index）**不被 DML 操作并且被**优化器（optimizer）**忽略。不可用索引可以提高批量加载的性能。代替删除索引并重新创建(re-creating)它，你可以让索引不可用，然后重建(rebuild)它。不可用索引和索引分区都不消耗空间。当你让一个可用索引变为不可用时，数据库删除它的索引**段（segment）**。

- 可见性

  索引有可见的（默认）或不可见的。**不可见索引（invisible index）**被 DML 操作并且默认不被优化器使用。使索引不可见是一种替代使它不可用或者删除它的方法。不可见索引对于测试在删除索引前移除它或者临时使用索引而不影响整个应用程序是特别有用的。

  参见：

  - "优化器概述" 在 7-10 页
  - *Oracle 数据库 2 天 DBA* 和 *Oracle 数据库管理员指南* 来了解如何管理索引
  - *Oracle 数据库性能调优指南* 来了解如何调整索引

键和列

键（key）是你可以建立索引的一组列或者**表达式（expressions）**。虽然这些术语经常互换使用，但是索引和键是不同的。**索引（Indexes）**是在数据库中用户使用 SQL 语句管理的存储结构。键在严格意义上是一个逻辑概念。

下面的语句在样表 oe.orders 的 customer_id 列上创建了一个索引：

```
CREATE INDEX ord_customer_ix ON orders (customer_id);
```

在前面的语句中，customer_id 列是索引键。索引自身被命名为 ord_customer_ix。

注意：主键和唯一键自动带有索引，但你可能需要在**外键（foreign key）**上创建一个索引。

参见：*Oracle 数据库 SQL 语言参考手册* CREATE INDEX 语法和语义

复合索引

复合索引(composite index)，也称为**级联索引(concatenated index)**，是在表中多列上的一种索引。在复合索引中列需要按顺序出现，使查询在表中检索到最有意义的数据时，并不需要相邻的数据。

复合索引可以加速对 SELECT 语句中 WHERE 子句引用的复合索引里所有的数据或者列开头部分数据的检索速度。因此，在定义时使用的列的顺序是非常关键的。通常，最常访问的列要放在第一个位置。

例如，假设一个应用频繁在 employees 表中查询 last_name、job_id 和 salary 列。再假设 last_name 具有高**基数（cardinality）**，这表示不同值的数量相对于表中行的数量很大。你可以创建带有下列顺序的索引：

```
CREATE INDEX employees_ix
ON employees (last_name, job_id, salary);
```

访问所有三个列、只有 last_name 列或者只有 last_ name 和 job_id 列的查询使用这个索引。在这个例子中，查询不访问 last_name 列也就不使用索引。

注意：在某些情况下，例如当起始列的基数非常低的时候，数据库可能会使用跳跃式扫描该索引（见“索引跳跃扫描”在 3-8 页）。

如果每个索引的列排列不同，则多个索引可以存在于同一个表中。如果你指定明显不同排列的列，那么你可以使用相同的列创建多个索引。例如，下面的 SQL 语句指定有效的排列：

```
CREATE INDEX employee_idx1 ON employees (last_name, job_id);
CREATE INDEX employee_idx2 ON employees (job_id, last_name);
```

参见：*Oracle 数据库性能调优指南* 来了解关于使用压缩索引的更多信息

唯一与非唯一索引

索引可以是**唯一（unique）**或者非唯一的。唯一索引保证在键列或列中，一张表没有两行重复值。例如，没有两个员工可以有相同的员工 ID。因此，在一个唯一索引中，**rowid（行的唯一标识）**存在于每一个数据值中。在叶子块中的数据仅按键的顺序排列。

在索引列中，非唯一索引允许重复值。例如，employees 表的 first_name 列可能包含多个 Mike 值。对于非唯一索引，rowid 按排序顺序被包含在键中，所以非唯一索引是按照索引键和 rowid（升序）排序的。

Oracle 数据库没有所有键列是**空（null）**的索引表行， 除位图索引或者当簇键列值为空以外。

索引类型

Oracle 数据库提供了多种索引方案，它提供了功能上互补的性能方案。索引可以被分为以下几类：

- B-树索引

  这些索引是标准索引类型。它们对于主键和高选择性索引是非常有效的。作为级联索引，B-树能按照索引列的排序顺序检索数据。B-树索引有以下的子类型：

  - 索引-组织表

    索引-组织表不同于堆-组织表，因为数据本身就是索引。见“索引-组织表概述”在 3-20 页。

  - 反向键索引

    在这种索引类型中，索引键的字节是反向的，例如，103 被存储为 301。反转字节插入到索引中伸展到许多块上。见“反向键索引”在 3-11 页。

  - 降序索引

    这种类型的索引用降序顺寻在特定列上存储数据。见“升序和降序索引”在 3-11 页。

  - B-树簇索引

    这种类型的索引用于索引表的簇键。键指向与包含簇键相关行的块，而不是指向单行。见“索引簇概述”在 2-23 页。

- 位图和位图连接索引

  在位图索引中，索引条目使用位图指向多行。与此相反，B-树索引条目指向单行。位图连接索引是用来连接两个或多个表的索引。见“位图索引”在 3-13 页。

- 基于函数索引

  这种类型的索引包含由函数转化的列，如 UPPER 函数，或者包含在表达式中。B-树或位图索引可以是基于函数的。见“基于函数索引”在 3-17 页。

- 应用程序域索引

这种类型的索引由在特定应用程序域中的用户为数据创建。物理索引不必使用传统的索引结构并且可以被存储在 Oracle 数据库的表或外部文件。见“应用程序域索引”在 3-19 页。

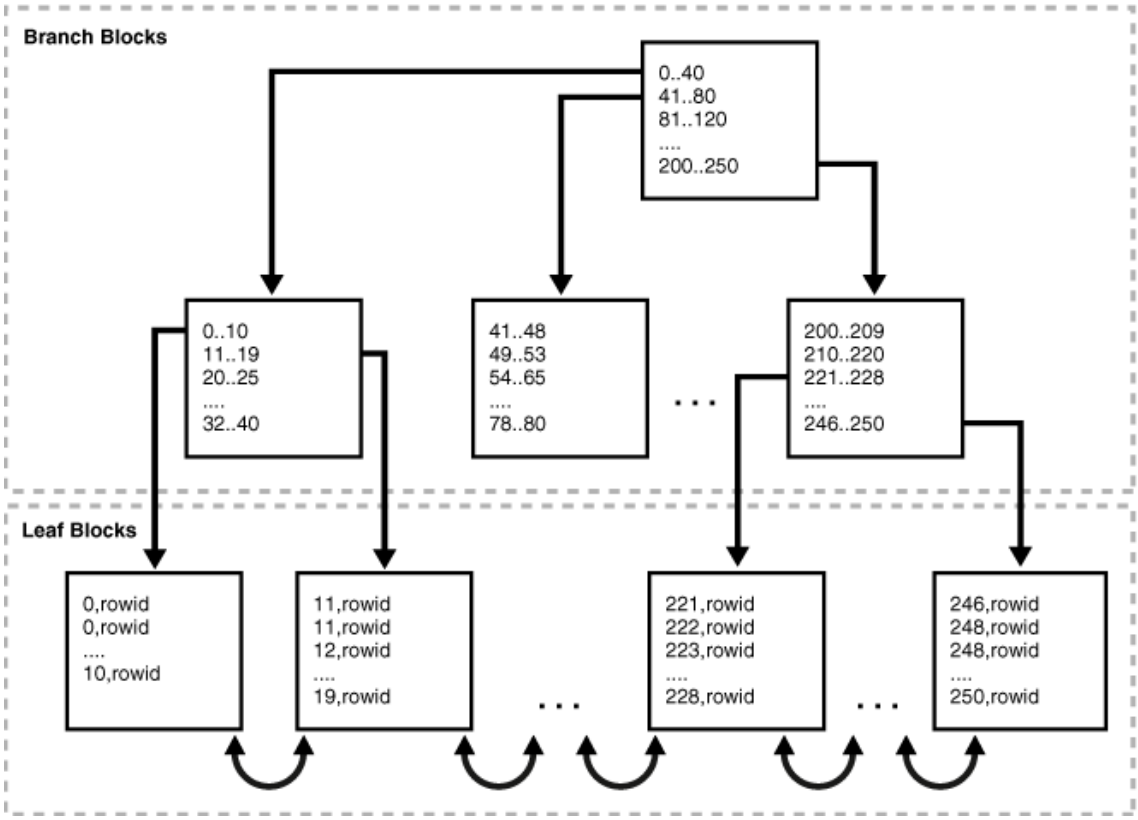
参见： *Oracle 数据库性能调优指南* 来了解关于不同索引类型

B-树索引

B-树，短于**平衡树（balanced trees）**，是数据库索引的最常见类型。B-树索引是划分值域的有序列表。通过关联带有一行或行范围中的键，B-树索引为大范围查询，包括精确匹配和范围搜索提供优良的检索性能。

图 3–1 示出 B-树索引的结构。这个例子显示了在 department_id 列的索引，是在 employees 表中的外键列。

图 3–1B-树索引内部结构



分支块与叶块

B-树索引有两种类型的块：**分支快（branch blocks）**用于搜索和**叶块（leaf blocks）**用于存储值。B-树索引的上级分支块包含的索引数据指向下级索引块。在图 3–1 中，根分支块有一个条目 0–40，指向在下面分支级别中最左边的块。这个分支块包含条目如 0–10 和 11–19。这些条目的每一个都指向包含落在范围中键值的叶块。

B-树索引是平衡的，因为所有的叶块自动的被放置在相同的深度。因此，从任意位置检索任何记录需要花费大约相同的时间。索引的**高度（height）**是需要从根块走到叶块的块数量。**分支级别（branch level）**是高度减 1。在图 3–1 中，索引有 3 的高度和 2 的分支级别。

分支块存储需要的最小键前缀使分支决策在两键之间。这种技术使得数据库适应尽可能多的数据在每一个分支块上。分支块包含的指针指向包含键的子块。键的数量和指针通过块大小限制。

叶块包含每一个索引的数据值和用于定位实际行对应的 rowid。每个条目按照（键、rowid）进行分类。叶块内，键和 rowid 链接到它左边和右边的兄弟条目上。叶块本身也是双向链接的。在图 3–1 中最左边的叶块(0–10)是链接到第二个叶块(11–19)。

注意：在列中带有字符数据的索引是基于在数据库字符集中字符的二进制值。

索引扫描

在**索引扫描（index scan）**中，数据库通过遍历索引来检索一行，使用语句中指定的索引列的值。如果数据库为一个值扫描索引，那么它将在 *n* 次 I/O 中找到这个值，其中 *n* 是 B-树索引的高度。这是 Oracle 数据库索引背后的基本原理。

如果一个 SQL 语句仅访问索引列，那么数据库直接从索引而不是表中读取值。如果语句访问除索引列外的列，那么数据库使用 rowid 来查找表中的行。通常情况下，数据库通过交替的读索引块和表块来检索表中数据。

参见：*Oracle 数据库性能调优指南* 来了解关于索引扫描的详细信息

完全索引扫描 在**完全索引扫描（full index scan）**中，数据库按顺序读取整个索引。如果**谓词（predicate）**（WHERE 子句）在 SQL 语句中引用了索引中的列，并在某些情况下当没有指定谓词时，完全索引扫描是可用的。完全扫描可以消除排序因为数据是按照索引键的顺序。

假设一个应用程序运行下面的查询：

```
SELECT department_id, last_name, salary
FROM employees
WHERE salary > 5000
ORDER BY department_id, last_name;
```

再假定 department_id，last_name 和 salary 是索引中的组合键。Oracle 数据库进行索引的完全扫描，按照排序顺序读取（通过部门 ID 和姓氏顺序）并在工资属性上过滤。在这种方式中，数据库将扫描比雇员表更小的数据集，表中包含列要比查询中包含的列多，并且避免了对数据进行排序。

例如，完全扫描可以读取索引条目如下：

```
50, Atkinson, 2800, rowid
60, Austin, 4800, rowid
70, Baer, 10000, rowed
```

80,Abel,11000,rowid

80,Ande,6400,rowid

110,Austin,7200,rowid ...

快速完全索引扫描 快速完全索引扫描（fast full index scan）是一种完全索引扫描，其中数据库在索引本身中访问数据，而不用访问表，并且数据库读取索引块没有固定的顺序。

当以下两个条件都满足的情况下，快速完全索引扫描是**全表扫描**（full table scan）的一种替代：

- 该索引必须包含查询所需的所有列。
- A 包含所有空值的行不能出现在查询结果集中。为了确保这一结果，索引中的至少一列必须具有下列条件之一：
  - NOT NULL 约束
  - A predicate applied to it that prevents nulls from being considered in the query result set

例如，一个应用程序发出下面的查询，其中不包括 ORDER BY 子句：

SELECT last_name, salary

FROM employees;

last_name 列有一个非空约束。如果姓氏和工资都是索引中的复合键，那么快速索引扫描可以读取索引中的条目来获取所需的信息：

Baida,2900,rowid

Zlotkey,10500,rowid

Austin,7200,rowid

Baer,10000,rowid

Atkinson,2800,rowid

Austin,4800,rowid ...

索引范围扫描 索引范围扫描（index range scan）是具有以下特征的索引有序扫描：

- 在条件中指定索引中的一个或多个导列。**条件**（condition）指定一个或多个表达式及逻辑（布尔）**运算符**（operators）的组合并返回一个 TRUE、FALSE 或 UNKNOWN 值。
- 对于索引键 0、1 或更多值是可能的。

数据库通常使用索引范围扫描来访问选择的数据。**选择性**（selectivity）是在表中查询选择行的百分比，并且 0 表示没有行而 1 表示所有行。选择性是绑定一个查询的**谓词**（predicate），像 WHERE last_name LIKE 'A%'或者谓词的组合。谓词变的更有选择性如值接近 0，较少的选择性（或更多非选择性）作为值接近 1。

例如，一个用户查询哪位员工的姓氏起始是 A。假设 last_name 列被索引，具有以下条目：

Abel,rowid

Ande,rowid

Atkinson,rowid

Austin,rowid

Austin,rowid

Baer,rowid

.

.

.

数据库可以使用范围扫描，因为 last_name 列在谓词中是指定的并且对每个索引键可能是成倍的 rowid。例如，两个员工的名字叫 Austin，所以两个 rowid 是用 Austin 键关联的。

索引范围扫描可以被划定两侧的界限，如在一个带有部门 ID 在 10 和 40 之间的查询，或者只在一边有界限，如在查询 ID 超过 40。为了扫描索引，数据库向前或者向后移动来穿过叶块。例如，一个对 ID 在 10 和 40 之间的扫描定位到包含最小键值是 10 或更大的第一个索引的叶块。扫描然后水平地前进通过叶子节点的链表直到定位到一个大于 40 的值。

索引唯一扫描 相比于索引范围扫描，**索引唯一扫描**（index unique scan）必须具有 0 或 1 的 rowid 与索引键关联。当谓词参考所有列在索引 UNIQUE 键中使用相等运算符时数据库执行唯一扫描。索引唯一扫描在它找到第一条记录时会迅速停止，因为没有第二条可能的记录。

作为一个例子，假设一个用户运行下面的查询：

```
SELECT *
FROM employees
WHERE employee_id = 5;
```

假设 employee_id 列是主键，并用下列条目作为索引：

1,rowid

2,rowid

4,rowid

5,rowid

6,rowid

.

.

.

在这种情况下，该数据库可以使用索引唯一扫描来定位员工 ID 为 5 的 rowid。

索引跳跃扫描 **索引跳跃扫描**（index skip scan）使用复合索引的逻辑子索引。如果数据库正在查找分开的索引，那么它会“跳跃”通过单一索引。跳跃扫描是有好处的，如果有几个不同值在复合索引的前列并且许多的不同值在非前列键的索引中。

数据库可能选择索引跳跃扫描，当复合索引的前列在查询谓词中不是指定的。例如，假设你对客户在 sh.customers 表中运行下列查询：

```
SELECT * FROM sh.customers WHERE cust_email = 'Abbey@company.com';
```

customers 表中有一 cust_gender 列，其中值是 M 或 F。假设复合索引存在于（cust_gender，cust_email）列。例 3–1 显示了索引条目的一部分。

例 3–1 复合索引条目

```
F,Wolf@company.com,rowid
F,Wolsey@company.com,rowid
F,Wood@company.com,rowid
F,Woodman@company.com,rowid
F,Yang@company.com,rowid
F,Zimmerman@company.com,rowid
M,Abbassi@company.com,rowid
M,Abbey@company.com,rowid
```

数据库可以使用这个索引的跳跃扫描，即使在 WHERE 子句中不指定 cust_gender。在跳跃扫描中，逻辑子索引的数量由在首列中不同值的数量决定。在例 3–1 中，首列有两种可能值。数据库逻辑的分割索引到带有键 F 的一个子索引和带有键 M 的第二个子索引。

当为客户的电子邮件是 Abbey@company.com 的记录查找时，数据库先查找带有值 F 的子索引，然后查找带有值 M 的子索引。从概念上讲，数据库处理该查询如下：

```
SELECT * FROM sh.customers WHERE cust_gender = 'F'
AND cust_email = 'Abbey@company.com'
UNION ALL
SELECT * FROM sh.customers WHERE cust_gender = 'M'
AND cust_email = 'Abbey@company.com';
```

See Also: *Oracle Database Performance Tuning Guide* to learn more about skip scans

索引簇因子 索引簇因子（index clustering factor）衡量行相对于索引值的顺序如员工的姓氏。这个值越按照顺序存在在行存储中，则它的簇因子越低。

簇因子作为通过索引读取整个表来粗略估算 I/O 需求数量是非常有用的：

- 如果簇因子高，那么 Oracle 数据库在一个大索引范围扫描期间执行一个比较高数量的 I/O。索引条目指向随机表块，所以数据库可能需要读并重读相同的数据块，一遍又一遍的检索由索引指向的数据。
- 如果簇因子低，那么 Oracle 数据库在一个大索引范围扫描期间执行一个相对低数量的 I/O。在一个范围中的索引键趋向于指向相同的数据块，所以数据库不需要一遍又一遍的读并重读相同的数据块。

簇因子是和索引扫描有关的，因为它能表明：

- 该数据库是否将会为大范围扫描使用索引
- 表组织级别相对于索引键
- 如果行必须按照索引键排序，你是否应该考虑使用索引组织表、分区表或表簇。

例如，假设 employees 表装载到两个数据块中。表 3–1 描绘出在两个数据块中的行（用省略号表示未示出的数据）。

表 3–1 在 Employees 表中的连个数据块的内容

Data Block 1					Data Block 2				
100	Steven	**King**	SKING	...					
156	Janette	**King**	JKING	...					
115	Alexander	**Khoo**	AKHOO	...					
.									
.					149	Eleni	**Zlotkey**	EZLOTKEY	...
.					200	Jennifer	**Whalen**	JWHALEN	...
116	Shelli	**Baida**	SBAIDA	...	.				
204	Hermann	**Baer**	HBAER	...	.				
105	David	**Austin**	DAUSTIN	...	.				
130	Mozhe	**Atkinson**	MATKINSO	...	137	Renske	**Ladwig**	RLADWIG	...
166	Sundar	**Ande**	SANDE	...	173	Sundita	**Kumar**	SKUMAR	...
174	Ellen	**Abel**	EABEL	...	101	Neena	**Kochar**	NKOCHHAR	...

按照姓氏顺序存储在块中的行（以粗体显示）。例如，在数据块 1 中底部的行描述了 Abel，向上的下一行描述了 Ande，按字母顺序以此类推，直到在块 1 中顶端的行 Steven King。在块 2 中底部的行描述了 Kochar，向上的下一行描述了 Kumar，按字母顺序以此类推，直到在块中的最后一行 Zlotkey。

假设一个索引存在于姓氏列。每个名字条目对应于一个 rowid。从概念上讲，索引条目将如下所示：

```
Abel,block1row1
Ande,block1row2
Atkinson,block1row3
Austin,block1row4
Baer,block1row5
.
.
.
```

假设一个单独的索引存在于员工的 ID 列。从概念上讲，带有员工 ID 分布的几乎随机的位置贯穿这两个块，索引条目可能如下所示：

```
100,block1row50
101,block2row1
102,block1row9
103,block2row19
104,block2row39
105,block1row4
.
.
.
```

例 3–2 查询 ALL_INDEXES 视图来查找这两个索引的簇因子。EMP_NAME_IX 的簇因子低，这意味着在单一叶块中的相邻索引条目倾向于指向在相同块中的行。EMP_EMP_ID_PK 的簇因子高，这意味着在相同叶块中的相邻索引条目不太可能指向在相同数据块中的行。

例 3–2 簇因子

```
SQL> SELECT INDEX_NAME, CLUSTERING_FACTOR
2 FROM ALL_INDEXES
```

```
3 WHERE INDEX_NAME IN ('EMP_NAME_IX','EMP_EMP_ID_PK');
INDEX_NAME CLUSTERING_FACTOR
-------------------- -----------------
EMP_EMP_ID_PK 19
EMP_NAME_IX 2
```

参见：*Oracle 数据库参考手册* 来了解关于 ALL_INDEXES

反向键索引

反向键索引（reverse key index）是 B-树的一种，在保持列顺序时，它物理地反转每一个索引键的字节。例如，如果索引键是 20，并且如果在一个标准 B-树索引中，这个键的两个字节按照十六进制存储是 C1，15，那么反向键索引存储的字节就是 15，C1。

反向键解决在 B-树索引右侧叶块的争议问题。这个问题尤其在 Oracle 真实应用集群（Oracle RAC）数据库中多个实例同时修改同一块时是很严重的。例如，在 orders 表中订单的主键是连续的。在集群中的一个实例增加顺序 20，这时另一个增加 21，每个实例都将它的键写入到索引右侧的相同叶块。

在反向键索引中，反向字节交叉有序的分散插入到索引中的所有叶块中。例如，像键 20 和 21 在标准索引中必然是相邻的，而现在存储到相距很远的单独块中。因此，I/O 对连续键的插入会分布的比较均匀。

因为在索引中的数据在它存储时是不按照列键排序的，反向键的部署消除了在一些情况下运行索引范围扫描查询的能力。例如，如果用户发出一个订单 ID 大于 20 的查询，那么数据库不能开始于包含这个 ID 的块并且不能通过叶块水平地处理。

参见：*Oracle 数据库性能调优指南* 来了解关于设计考虑反向键索引

升序和降序索引

在**升序索引（ascending index）**中，Oracle 数据库用升序方式存储数据。默认情况下，字符数据是由包含在每个字节的二进制值来排序，数字数据是从小到大的数值，而日期是从最早到最新的值。举一个升序索引的例子，考虑下面的 SQL 语句：

```
CREATE INDEX emp_deptid_ix ON hr.employees(department_id);
```

Oracle 数据库在 hr.employees 表的 department_id 列上排序。它加载 department_id 的升序索引和按升序排列相对应的 rowid 值，起始为 0。当它使用索引时，Oracle 数据库查找排序的 department_id 值并使用关联的 rowid 定位需要 department_id 值的行。

通过在 CREATE INDEX 语句中指定 DESC 关键字，你可以创建**降序索引（descending index）**。在这种情况下，索引按照降序储存数据在指定的一列或多列。如果图 3–1 中在 employees.department_id 列上的索引是降序，那么包含 250 的叶块将在树的左侧并且带有 0 的块在右侧。默认通过降序索引查询是从最高到最低值。

当一个查询用升序和其他降序排序一些列时，降序索引是有用的。举个例子，假设你创建一个复合索引在 last_ name 列和 department_id 列如下：

```
CREATE INDEX emp_name_dpt_ix ON hr.employees(last_name ASC, department_id DESC);
```

如果用户用升序（A 到 Z）查询 hr.employees 的姓氏并且部门 ID 按照降序顺序（高到低），那么数据库能使用这个索引检索数据并且避免多于的排序步骤。

参见：

- *Oracle 数据库性能调优指南* 来了解更多关于升序和降序索引查找
- *Oracle 数据库 SQL 语言参考手册* 来查询 CREATE INDEX 的 ASC 和 DESC 选项的描述

键压缩

Oracle 数据库能使用**键压缩（key compression）**来压缩在 B-树索引或者索引-组织表中主键列值的一部分。

在一般情况下，索引键有两个片，一是**分组片（grouping piece）**另一是**唯一片（unique piece）**。键压缩拆分索引键到**前缀条目（prefix entry）**，这是分组片，和**后缀条目（suffix entry）**，这是唯一或者几乎唯一片。数据库通过共享后在索引块的缀条目中的前缀条目实现压缩。

注意：如果键不定义唯一片，那么数据库提供一个附加的 rowid 到分组片。

默认情况下，唯一索引前缀包含所有键列而不包括最后一个，而非唯一索引前缀包含所有键列。例如，假设你在 oe.orders 表上创建一个复合索引如下：

```
CREATE INDEX orders_mod_stat_ix ON orders ( order_mode, order_status );
```

许多重复值出现在 order_mode 和 order_status 列中。一个索引块可以包含在例 3–3 中所示的条目。

例 3–3 在 Orders 表中的索引条目

```
online,0,AAAPvCAAFAAAAFaAAa
online,0,AAAPvCAAFAAAAFaAAg
online,0,AAAPvCAAFAAAAFaAAl
online,2,AAAPvCAAFAAAAFaAAm
online,3,AAAPvCAAFAAAAFaAAq
online,3,AAAPvCAAFAAAAFaAAt
```

在例 3–3 中，键前缀将包含 order_mode 和 order_status 的串联值。如果创建这个索引并带有默认键压缩，那么重复键前缀如 online,0 和 online,2 将被压缩。从概念上讲，数据库如下面的例子所示来实现压缩：

```
online,0
AAAPvCAAFAAAAFaAAa
AAAPvCAAFAAAAFaAAg
AAAPvCAAFAAAAFaAAl
```

online,2

AAAPvCAAFAAAAFaAAm

online,3

AAAPvCAAFAAAAFaAAq

AAAPvCAAFAAAAFaAAt

后缀条目构成索引行的压缩版本。每一个后缀条目都参考前缀条目，这是存储在和后缀条目相同的索引块中。

另外，当创建压缩索引时，你可以指定前缀长度。例如，如果你指定前缀长度为 1，那么前缀将是 order_mode 而后缀将是 order_status,rowid。对于在例 3–3 中的值，索引将分离出重复出现的 online 如下：

online

0,AAAPvCAAFAAAAFaAAa

0,AAAPvCAAFAAAAFaAAg

0,AAAPvCAAFAAAAFaAAl

2,AAAPvCAAFAAAAFaAAm

3,AAAPvCAAFAAAAFaAAq

3,AAAPvCAAFAAAAFaAAt

索引每次存储一个指定的前缀在最多一个叶块中。只有在 B-树索引的叶块中的键是压缩的。在分支块中的键后缀可以被截断，但键不是压缩的。

参见：

- *Oracle 数据库管理员指南* 来了解如何使用压缩索引
- *Oracle 数据库 VLDB 和分区指南* 来了解分区索引如何使用键压缩
- *Oracle 数据库 SQL 语言参考手册* 来查看 CREATE INDEX 的 *key_compression* 子句描述

位图索引

在**位图索引（bitmap index）**中，数据库将每个索引键存储在一个位图中。在常规的 B-树索引中，一个索引条目指向单一行。在一个位图索引中，每个索引键存储指向多个行。

位图索引主要为了数据仓库或者在一个特设方式中查询参考许多列的情况下设计的。可能需要位图索引的情况包括：

- 索引列含有低**基数（cardinality）**，也就是说，不同数量的值和表中行数相比是很小的。
- T 被索引表可以是只读的或者没有受到 DML 语句明显的修改。

举一个数据仓库的例子，sh.customer 表有一个 cust_gender 列仅有两种可能的值：M 和 F。假设查询特定性别顾客的数量是常见的。在这种情况下，customer.cust_gender 列将是位图索引的一个候选。

在位图中的每一个位都对应着一个可能的 rowid。如果位被设置，那么带有对应 rowid 的行包含这个键值。映射函数转换这个位的位置成为一个实际的 rowid，所以位图索引提供与 B-树索引相同的功能，尽管它使用了不同的内部表示。

如果在单行中的索引列被更新，那么数据库**锁定（locks）**索引键条目（例如，M 或 F）并且不止是映射到更新行的个别位。因为一个键指向许多行，在索引数据上 DML 通常要锁定所有行。因此，位图索引是不适用于大多数 OLTP 的应用。

参见：

- *Oracle 数据库性能调优指南* 来了解对性能如何使用位图索引
- *Oracle 数据库数据仓库指南* 来了解在数据仓库中如何使用位图索引

在单表上的位图索引

例 3–4 显示了 sh.customers 表的一个查询。在这张表中的一些列是位图索引的候选。

例 3–4 customers 表的查询

```
SQL> SELECT cust_id, cust_last_name, cust_marital_status, cust_gender
2 FROM sh.customers
3 WHERE ROWNUM < 8 ORDER BY cust_id;
CUST_ID CUST_LAST_ CUST_MAR C
---------- ---------- -------- -
1 Kessel M
2 Koch F
3 Emmerson M
4 Hardy M
5 Gowen M
6 Charles single F
7 Ingram single F
7 rows selected.
```

cust_marital_status 和 cust_gender 列具有低基数，而 cust_id 和 cust_last_name 没有。因此，位图索引可能适合于 cust_marital_status 和 cust_gender。位图索引或许对其他列没有作用。相反，在这些列上的唯一 B-树索引有可能提供最高效的表达和检索。

表 3–2 展示了位图索引在例 3–4 中对 cust_gender 列输出显示。它由两个单独的位图组成，一个对应一个性别。

Table 3–2 Sample Bitmap

Value	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1

映射函数转换在位图索引中的每一个位成为 customers 表的 rowid。每个位值都依赖于在表中关联行的值。例如，对于 M 值的位图包含一个 1 作为它的第一个位，因为在 customers 表的第一行性别是 M。位图 cust_gender='M'有一个 0 在行 2、6、7 的位，因为这些行它们的值不包含 M。

注意：位图索引能包含完全由空值组成的键，不像 B-树索引。索引了空值可以对一些 SQL 语句非常有用，如带有聚合函数 COUNT 的查询。

分析师调查客户的人口发展趋势可能问,“我们的女顾客中有多少是单身或者离异的？”这个问题相当于以下 SQL 查询：

```
SELECT COUNT(*)
FROM customers
WHERE cust_gender = 'F'
AND cust_marital_status IN ('single', 'divorced');
```

位图索引可以有效地通过计算 1 的数量在生成的位图中处理这个查询，如在表 3–3 中展示的。为了确定哪位顾客符合标准，Oracle 数据库可以使用生成的位图来访问表。

表 3–3 示例位图

Value	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1
single	0	0	0	0	0	1	1
divorced	0	0	0	0	0	0	0
single or divorced, and F	0	0	0	0	0	1	1

位图索引有效的合并对应的几个条件在 WHERE 子句中的索引。在表本身被访问之前，条件将过滤掉一些，而不是所有的行。这种技术提高了响应时间，通常会很显著。

位图连接索引

A **位图连接索引（bitmap join index）**是对**连接（join）**两个或多个表的位图索引。在表列中的每个值，索引存储在被索引表中对应行的 rowid。相比之下，一个标准的位图索引是被创建在单表上的。

位图连接索引是减少数据量的有效途径,在性能限制的前提下数据必须被连接。例如当使用位图连接索引，假设用户经常查询特定作业类型的员工数量。一个典型的查询可能看起来如下：

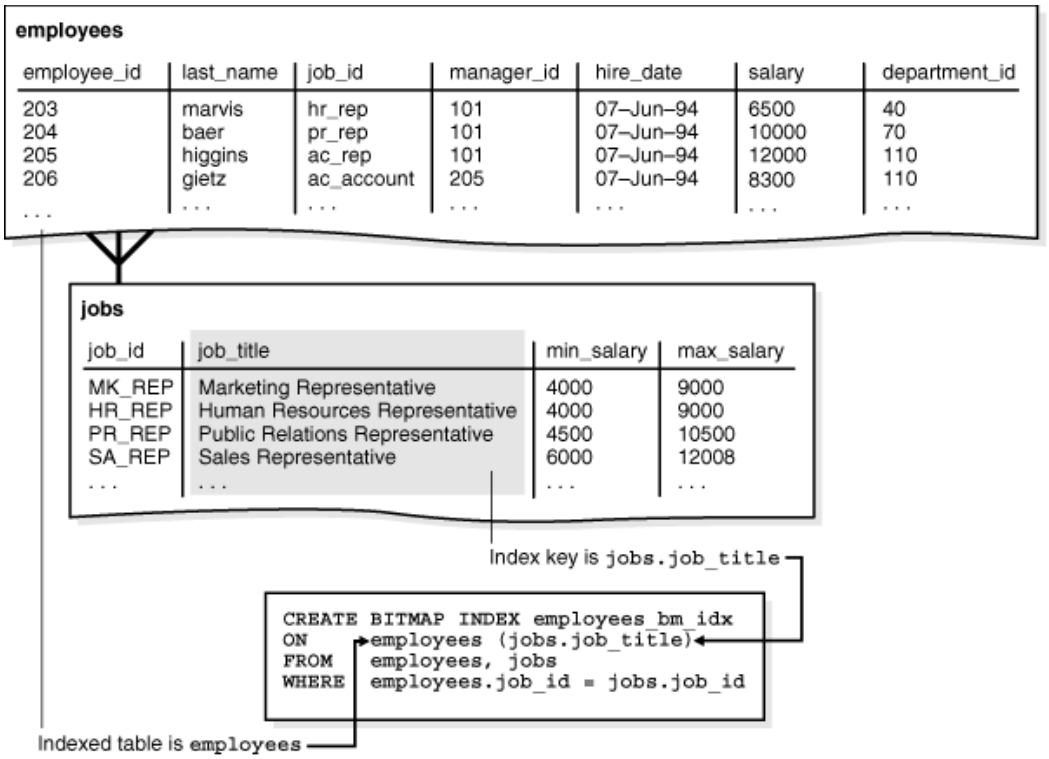
```
SELECT COUNT(*)
FROM employees, jobs
WHERE employees.job_id = jobs.job_id
AND jobs.job_title = 'Accountant';
```

上面的查询通常在 jobs.job_title 上使用索引来检索 Accountant 的行然后作业 ID 和在 employees.job_id 上的索引来查找匹配的行。为了从索引自身而不是从表的扫描中检索数据，你可以创建位图连接索引如下：

```
CREATE BITMAP INDEX employees_bm_idx
ON employees (jobs.job_title)
FROM employees, jobs
WHERE employees.job_id = jobs.job_id;
```

如图所示在图 3–2 中，索引键是 jobs.job_title 而索引表是 employees。

图 3–2 位图连接索引



从概念上讲，在例 3–5 中（包括示例输出）如 SQL 查询所示 employees_bm_idx 是 jobs.title 列的索引。在索引中的 job_title 键指向在 employees 表中的行。会计数量的查询可以使用索引来避免访问 employees 表和 jobs 表，因为索引本身包含所需的信息。

例 3–5 employees 表和 jobs 表的连接

```
SELECT jobs.job_title AS "jobs.job_title", employees.rowid AS "employees.rowid"
FROM employees, jobs
WHERE employees.job_id = jobs.job_id
ORDER BY job_title;
jobs.job_title employees.rowid
---------------------------------- ------------------
Accountant AAAQNKAAFAAAABSAAL
Accountant AAAQNKAAFAAAABSAAN
Accountant AAAQNKAAFAAAABSAAM
Accountant AAAQNKAAFAAAABSAAJ
Accountant AAAQNKAAFAAAABSAAK
Accounting Manager AAAQNKAAFAAAABTAAH
Administration Assistant AAAQNKAAFAAAABTAAC
Administration Vice President AAAQNKAAFAAAABSAAC
Administration Vice President AAAQNKAAFAAAABSAAB
.
.
```

.

在数据仓库中，**连接条件（join condition）**是在维度表的主键列和事实表中的外键列之间的**等值连接（equijoin）**（它使用相等运算符）。位图连接索引有时比物化连接视图在存储方面更有效，前提是替代物化视图连接。

位图存储结构

Oracle 数据库使用 B-树索引结构来为每个索引键存储位图。例如，如果 jobs.job_title 是位图索引的键列，那么索引数据被存储在一个 B-树中。各个位图被存储在叶块中。

假设 jobs.job_title 列具有唯一值 Shipping Clerk、和 Stock Clerk 其他几个。对于这个索引的位图索引条目包含以下组件：

- 职位标题作为索引键
- 一个低 rowid 和一个高 rowid 作为 rowid 的范围
- 位图给在范围内制定的 rowid

从概念上讲，在这个索引中的索引叶块可以包含条目如下：

Shipping Clerk, AAAPzRAAFAAAABSABQ, AAAPzRAAFAAAABSABZ, 0010000100

Shipping Clerk, AAAPzRAAFAAAABSABa, AAAPzRAAFAAAABSABh, 010010

Stock Clerk, AAAPzRAAFAAAABSAAa, AAAPzRAAFAAAABSAAc, 1001001100

Stock Clerk, AAAPzRAAFAAAABSAAd, AAAPzRAAFAAAABSAAt, 0101001001

Stock Clerk, AAAPzRAAFAAAABSAAu, AAAPzRAAFAAAABSABz, 100001

.

.

.

相同的工作标题出现在多个条目中，因为 rowid 的范围不同。

假设一个会话更新员工的作业 ID 从 Shipping Clerk 到 Stock Clerk。在这种情况下，会话需要独占访问索引键条目对旧值（Shipping Clerk）和新值（Stock Clerk）。Oracle 数据库通过这两个条目锁定行指向——而非由 Accountant 或其他任意键的行指向——直到 UPDATE 提交。

位图索引的数据在一个**段（segment）**中存储。Oracle 数据库在一个或多个片中存储每个位图。每一片都占用单个**数据块（data block）**的一部分。

参见：“用户段”在 12-21 页

基于函数索引

你可以在函数和表达式上创建索引，包括一个或多个列在被索引的表中。**基于函数索引（function-based index）**计算一个包括一个或多个列的函数或表达式的值并在索引中存储它。基于函数索引可以是一个 B-树或者位图索引。用于建立索引的函数可以是一个算术表达式或者一个包含 SQL 函数、用户定义的 PL/SQL 函数，包函数或者 C 调用的表达式。例如，一个函数可以在两列中增加值。

■ *Oracle 数据库管理员指南* 来了解如何创建基于函数索引
■ *Oracle 数据库性能调优指南* 来了解更多关于使用基于函数索引
■ *Oracle 数据库 SQL 语言参考手册* 来了解对基于函数索引的限制和使用说明

基于函数索引的使用

基于函数索引对评估在 WHERE 子句中包含函数的语句是有效的。当在一个查询中包含这个函数时，数据库仅使用基于函数索引。当数据库处理 INSERT 和 UPDATE 语句时，然而，它仍然必须评估函数来处理该语句。

例如，假设你创建以下基于函数索引：

```
CREATE INDEX emp_total_sal_idx
ON employees (12 * salary * commission_pct, salary, commission_pct);
```

在处理查询时，数据库可以使用前面的索引，如例 3–6（包含部分示例输出）。

例 3–6 查询包含一个算术表达式

```
SELECT employee_id, last_name, first_name,
12*salary*commission_pct AS "ANNUAL SAL"
FROM employees
WHERE (12 * salary * commission_pct) < 30000
ORDER BY "ANNUAL SAL" DESC;
EMPLOYEE_ID LAST_NAME FIRST_NAME ANNUAL SAL
----------- ------------------------- -------------------- ----------
159 Smith Lindsey 28800
151 Bernstein David 28500
152 Hall Peter 27000
160 Doran Louise 27000
175 Hutton Alyssa 26400
149 Zlotkey Eleni 25200
169 Bloom Harrison 24000
```

基于函数索引定义于 SQL 函数 UPPER(*column_name*)或 LOWER(*column_name*)上以便于区分大小写的搜索。例如，假设在 employees 中的 first_name 列包含混合大小写字母。你在 hr.employees 表上创建下面的基于函数索引：

```
CREATE INDEX emp_fname_uppercase_idx
ON employees ( UPPER(first_name) );
```

emp_fname_uppercase_idx 索引可以加速如下查询：

```
SELECT *
FROM employees
WHERE UPPER(first_name) = 'AUDREY';
```

基于函数索引对于在表中仅索引特定行也是非常有用的。例如，在 sh.customers 表中的 cust_valid 列有任意 I 或 A 作为值。为了仅索引 A 的行，你可以写一个对除 A 行以外任意行来返回一个空值的函数。你可以创建如下索引：

```
CREATE INDEX cust_valid_idx
ON customers ( CASE cust_valid WHEN 'A' THEN 'A' END );
```

参见：

- *Oracle 数据库全球化支持指南* 来了解关于语言索引的信息
- *Oracle 数据库 SQL 语言参考手册* 来了解更多关于 SQL 函数

使用基于函数索引优化

优化器（optimizer）可以在基于函数索引上使用索引范围扫描对在 WHERE 子句中带有表达式的查询。当谓词(WHERE 子句)有低**选择性(selectivity)**时，范围扫描**访问路径（access path）**是非常有帮助的。在例 3–6 中的优化器可以使用索引范围扫描，如果索引建立在表达式 12*salary*commission_pct 上。

虚拟列（virtual column）对于加速访问从表达式的数据导出是非常有用的。例如，你可以定义虚拟列 annual_sal 作为 12*salary*commission_pct 并创建基于函数索引在 annual_sal 上。

优化器通过解析在 SQL 语句中的表达式执行表达式匹配，然后比较语句的表达式树和基于函数索引。这种比较是不区分大小写的并且忽略空格。

参见：

- “优化器概述”在 7-10 页
- *Oracle 数据库性能调优指南* 来了解更多信息关于收集统计信息
- *Oracle 数据库管理员指南* 来了解如何添加虚拟列到表中

应用程序域索引

应用程序域索引（application domain index）特定于应用程序的自定义索引。Oracle 数据库提供**可扩展索引（extensible indexing）**来执行以下操作：

- 容纳在定制、复杂的数据类型上的索引，如文档、空间数据、图像和视频剪辑（见“非结构数据”在 19-11 页）
- 确保特定索引技术的使用

你可以封装应用程序特定的索引管理常规作为一个**索引类型（indextype）**模式对象并且定义一个与索引在表列上或者对象类型的属性上。可扩展的索引可以有效地处理应用程序特定的**操作符（operators）**。

一个应用程序软件，叫做**盒式磁带(cartridge)**，控制结构和域索引的内容。数据库和应用程序交互来构建、维护和搜索域索引。索引结构本身可以被存储在数据库中作为一个索引-组织表或者外部的一个文件。

参见：*Oracle 数据库数据磁带开发者指南* 来了解关于 Oracle 数据库可扩展体系结构中使用数据磁带的信息

索引存储

Oracle 数据库在**索引段（index segment）**中存储索引数据。在**数据块（data block）**中可用于索引数据的空间是数据块大小减块的开销、条目开销、rowid 和每个被索引值的一字节长度。

索引段的**表空间（tablespace）**是所有者的默认表空间或者在 CREATE INDEX 语句中指定的名称的表空间。为了便于管理，你可以在单独的表空间中存储来自它表的索引。例如，你可能选择不备份只包含索引的表空间，它可以重建，因此减少了备份需要的时间和存储。

参见：第 12 章，“逻辑存储结构”

索引-组织表概述

索引-组织表（index-organized table）是存储在一个异构 B-树索引结构中的表。在**堆-组织表（heap-organized table）**中，行被插入到适合它们的地方。在索引-组织表中，表的行被存储在一个由主键定义的索引中。在 B-树中的每个索引条目也存储非-键列值。因此，索引是数据，并且数据也是索引。应用程序操纵索引-组织表就像堆-组织表一样，使用 SQL 语句。

举个对索引-组织表的比喻，假设一个人力资源经理有一书柜纸箱。每一个箱子都用数字—1、2、3、4 等标注—但是箱子不是按照序列顺序放在架子上的。相反，每个箱子包含指向到下一个箱子在序列中的架子位置。

包含员工记录的文件夹存储在每个盒子中。文件夹按照员工 ID 存储。员工 King 有 ID 号 100，这是最小的 ID，因此他的文件夹在箱子 1 的底部。员工 101 的文件夹在 100 的上面，102 是在 101 的上面，以此类推直到箱子 1 已满。序列中的下一个文件夹是在箱子 2 的底部。

在这个比喻中，按照员工 ID 排序的文件夹使得它可以有效的搜索文件夹而不需要维护一个单独的索引。假设用户需要员工 107、120 和 122 的记录。而不是在一个步骤中搜索索引并在一个单独步骤中检索文件夹，经理可以按顺序搜索文件夹并检索每个被找到的文件夹。

索引-组织表按照主键或者键的有效前缀提供表行的快速访问。在叶块中行的非键列的存在避免了额外的**数据块（data block）**I/O 。例如，员工 100 的薪水被存储在它自身的索引行中。同时，由于行按照主键顺序存储，通过主键或前缀访问范围涉及最小块的 I/O。另一个好处是避免单独主键索引的空间开销。

当相关的数据必须存储在一起或者数据必须按照指定顺序物理的存储在一起时，索引-组织表是有用的。这种类型的表通常使用于信息检索、空间（参见“Oracle 空间概述”在 19-14 页）和 OLAP 应用(参见“OLAP”在 17-19 页)。

参见：

- *Oracle 数据库管理员指南* 来了解如何管理索引-组织表
- *Oracle 数据库性能调优指南* 来了解如何使用索引-组织表来提升性能
- *Oracle 数据库 SQL 语言参考手册* 关于 CREATE TABLE ... ORGANIZATION INDEX 的语法和语义

索引-组织表的特征

数据库系统通过操纵 B-树索引结构来在索引-组织表上执行全部操作。表 3–4 总结了索引-组织表和堆-组织表之间的差异。

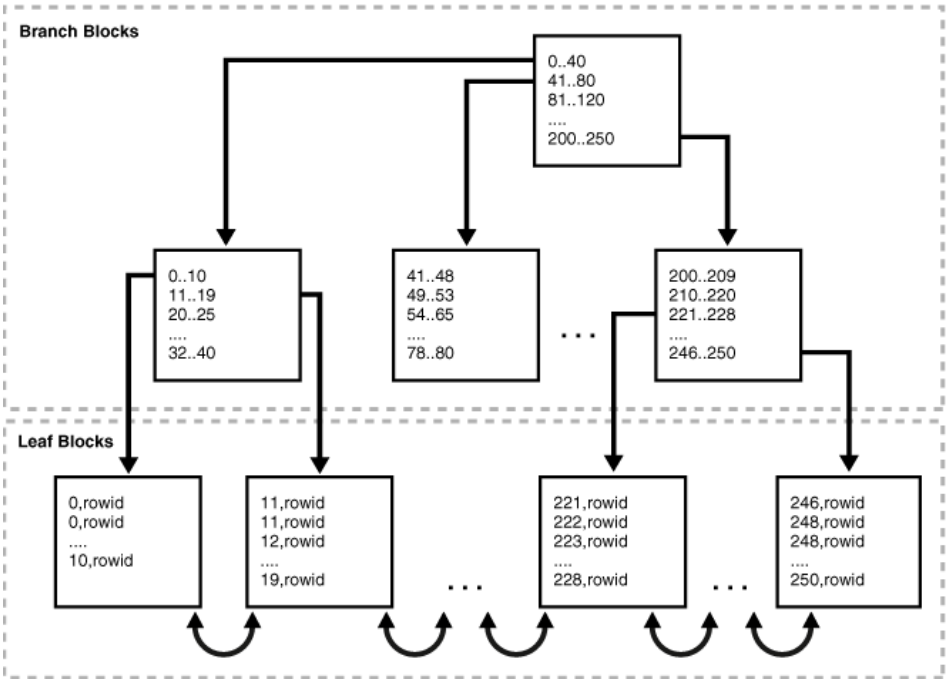
表 3–4 堆-组织表和索引-组织表的比较

堆-组织表	索引-组织表
rowid 唯一标识一行。主键约束可以有选择的定义。	主键唯一标识一行。主键约束必须定义。
ROWID 伪列（pseudocolumn）中的物理 rowid 指向建立的二级索引。	在 ROWID 伪列中的逻辑 rowid 指向建立的二级索引。
各行可以直接通过 rowid 访问。	访问各行可根据主键间接实现。
顺序**全表扫描**（full table scan）按照某些顺序返回所有行。	**完全索引扫描**（full index scan）或者快速完全索引扫描按照某些顺序返回所有行。
可以与其他表存储在**表簇**（table cluster）中。	不能被存储在一个表簇中。
可以包含一列 LONG 数据类型和多列 LOB 数据类型。	可以包含 LOB 列，但不能包含 LONG 列。
可以包含**虚拟列**（virtual columns）（仅支持关系型堆表）。	不能包含虚拟列。

图 3–3 显示了一个索引-组织表 departments 的结构。叶块包含表中的行，按主键顺序排列。例如，在第一个叶块中的第一个值显示了一个部门 ID20，部门名称是 Marketing，经理 ID 是 201，并且位置 ID 是 1800。

索引-组织表概述

图 3–3 索引-组织表



索引-组织表在相同的结构中存储所有数据并且不需要存储 rowid。如图 3–3 中所示，在索引-组织表中的叶块 1 可能包含如下条目，按照主键顺序：

20, Marketing, 201, 1800

30, Purchasing, 114, 1700

在索引-组织表中的叶块 2 可能包含如下条目：

50, Shipping, 121, 1500

60, IT, 103, 1400

按主键顺序的索引-组织表的扫描读取块按照以下顺序：

1. Block 1

2. Block 2

对比堆-组织表和索引-组织表的数据访问，假设堆-组织 departments 表段块 1 包含如下行：

50, Shipping, 121, 1500

20, Marketing, 201, 1800

相同表中块 2 包含行如下：

30, Purchasing, 114, 1700

60, IT, 103, 1400

对于这个堆-组织表的 B-树索引叶块包含如下条目，第一个值是主键而第二个值是 rowid：

20, AAAPeXAAFAAAAAyAAD

30, AAAPeXAAFAAAAAyAAA

50, AAAPeXAAFAAAAAyAAC

60, AAAPeXAAFAAAAAyAAB

表行的扫描按照主键顺序读取表段块在如下顺序中：

1. Block 1
2. Block 2
3. Block 1
4. Block 2

因此，在这个例子中的块 I/O 的数量是在索引-组织表例子中的双倍。

参见：

- “表组织”在 2-18 页
- “逻辑存储结构导论”在 12-1 页

带有行溢出区的索引-组织表

当创建一张索引-组织表时，你可以指定独立段作为**行溢出区（row overflow area）**。在索引-组织表中，B-树索引条目可以很大，因为它们包含一行条目，因此独立段来包含该条目是非常有用的。相比之下，B-树条目通常很小，因为它们包含键和 rowid。

如果一个行溢出区是指定的，那么数据库可以在索引-组织表中分割一行到如下的部分中：

- 索引条目

这部分包含所有主键的列值、一个指向溢出区行的物理 rowid 和一些可选的非键列。这个部分被存储在索引段中。

- 溢出部分

这部分包含剩余的非键列的列值。这部分被存储在溢出存储区域段中。

参见：

- *Oracle 数据库管理员指南* 来了解如何使用 CREATE TABLE 的 OVERFLOW 子句来设置一个行溢出区。
- *Oracle 数据库 SQL 语言参考手册* 关于 CREATE TABLE ... OVERFLOW 语法和语义

在索引-组织表上的二级索引

二级索引（secondary index）是一个在索引-组织表中的索引。在某种意义上说，它是在索引上的索引。二级索引是一个独立模式对象并且在索引-组织表外单独存储。

如在 2-13 页“rowid 数据类型”的解释，Oracle 数据库在索引-组织表中使用称为**逻**

辑 rowid（logical rowids）的行标识符。逻辑 rowid 是一个 base64 编码表示的表主键。逻辑 rowid 长度依赖于主键长度。

由于插入，在索引叶块中的行可以在内部或者块之间移动。在索引-组织表中的行不像堆-组织表的行那样迁移（参见“行链接和迁移”在 12-16 页）。因为在索引-组织表中的行没有永久物理地址，数据库使用基于主键的逻辑 rowid。

例如，假设 departments 表是索引-组织的。location_id 列存储每个部门的 ID。表存储如下的末值为位置 ID 的行：

10,Administration,200,1700

20,Marketing,201,1800

30,Purchasing,114,1700

40,Human Resources,203,2400

在 location_id 列上的二级索引可能有如下索引条目，其中逗号后面的值是逻辑 rowid：

1700,*BAFAJqoCwR/+

1700,*BAFAJqoCwQv+

1800,*BAFAJqoCwRX+

2400,*BAFAJqoCwSn+

二级索引提供快速且有效的访问使用既不是主键也不是主键前缀的索引-组织表。例如，一个部门 ID 大于 1700 的名称查询可以使用二级索引来加速数据访问。

参见：

- *Oracle 数据库管理员指南* 来了解如何在索引-组织表上创建二级索引
- *Oracle 数据库 VLDB 和分区指南* 来了解关于在索引-组织表分区上创建二级索引

逻辑 rowid 和物理预测

二级索引使用逻辑 rowid 定位表行。逻辑 rowid 包含**物理预测（physical guess）**，这是当它第一次生成时索引条目的物理 rowid。Oracle 数据库可以使用物理预测来直接探测到索引-组织表的叶块中，绕过主键搜索。当一行的物理位置改变，逻辑 rowid 仍然有效，及时它包含一个陈旧的物理预测。

对于堆-组织表，通过二级索引访问包括二级索引扫描和一个额外的 I/O 来获得包含该行的**数据块（data block）**。对于索引-组织表，通过二级索引访问有所不同，取决于使用和物理预测的准确度：

- 没有物理预测，访问包括两个索引扫描：二级索引扫描，其次是主键索引扫描。
- 带有物理预测，访问取决于它们的精确度：

– 带有准确的物理预测，访问包括二级索引扫描和额外的 I/O 来获得包含行的数据块。

– 带有不准确的物理预测，访问包含二级索引扫描和一个 I/O 来获取错误的数

据块（由预测所示），其次是索引组织表按照主键值的索引唯一扫描。

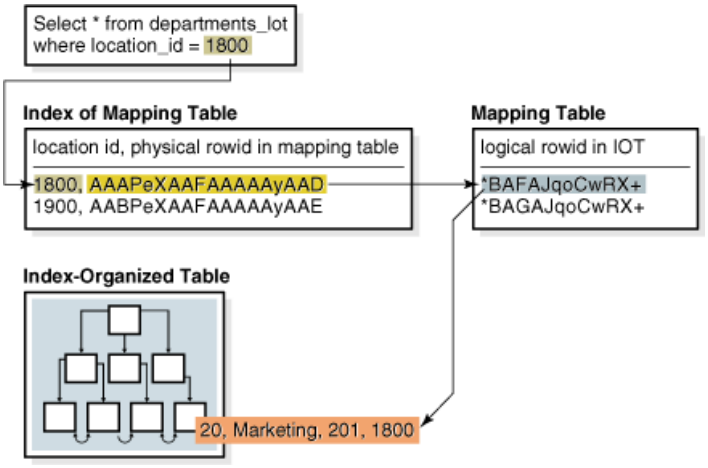
在索引-组织表上的位图索引

在索引-组织表上的二级索引可以是一个**位图索引（bitmap index）**。如在 3-13 页的“位图索引”中解释的，一个位图索引为每个索引键存储一个位图。

当位图索引存在于索引-组织表时，所有位图索引使用一个堆-组织的**映射表（mapping table）**。映射表存储索引-组织表的逻辑 rowid。每个映射表行为对应的索引-组织表行存储一个逻辑 rowid。

数据库使用搜索键访问位图索引。如果数据库查找这个键，那么位图条目被转换为物理 rowid。对于堆-组织表，数据库使用物理 rowid 来访问基表。对于索引-组织表，数据库使用物理 rowid 来访问映射表，从而得出数据库用来访问索引-组织表的逻辑 rowid。图 3–4 展示了对 `departments_iot` 表查询的索引访问。

图 3–4 在索引-组织表上的位图索引 **注解：**



注意：在索引-组织表中的行迁移不会留下建立在索引-组织表上不能使用的位图索引。

参见：“行片的 rowid”在 2-19 页