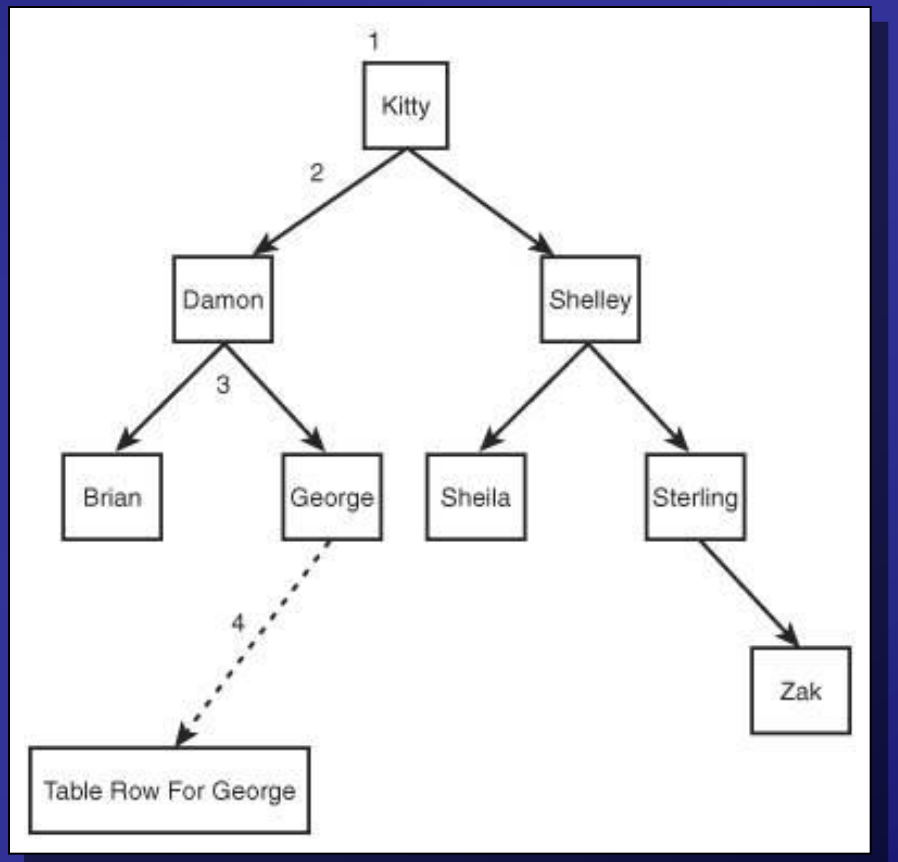
索引（Index）

单世民

索引的基本概念

- 索引是一种数据结构，可以通过该结构迅速地访问表中的数据



索引的基本概念

- 索引的分类

逻辑设计分类	物理实现分类
单列索引 （复合索引）	分区索引和非分区索引
唯一性索引 （非唯一性索引）	B^+树索引
基于函数的索引	位图索引
…	正向索引和反向索引
	位图联接索引

索引

- 创建索引的一般语法

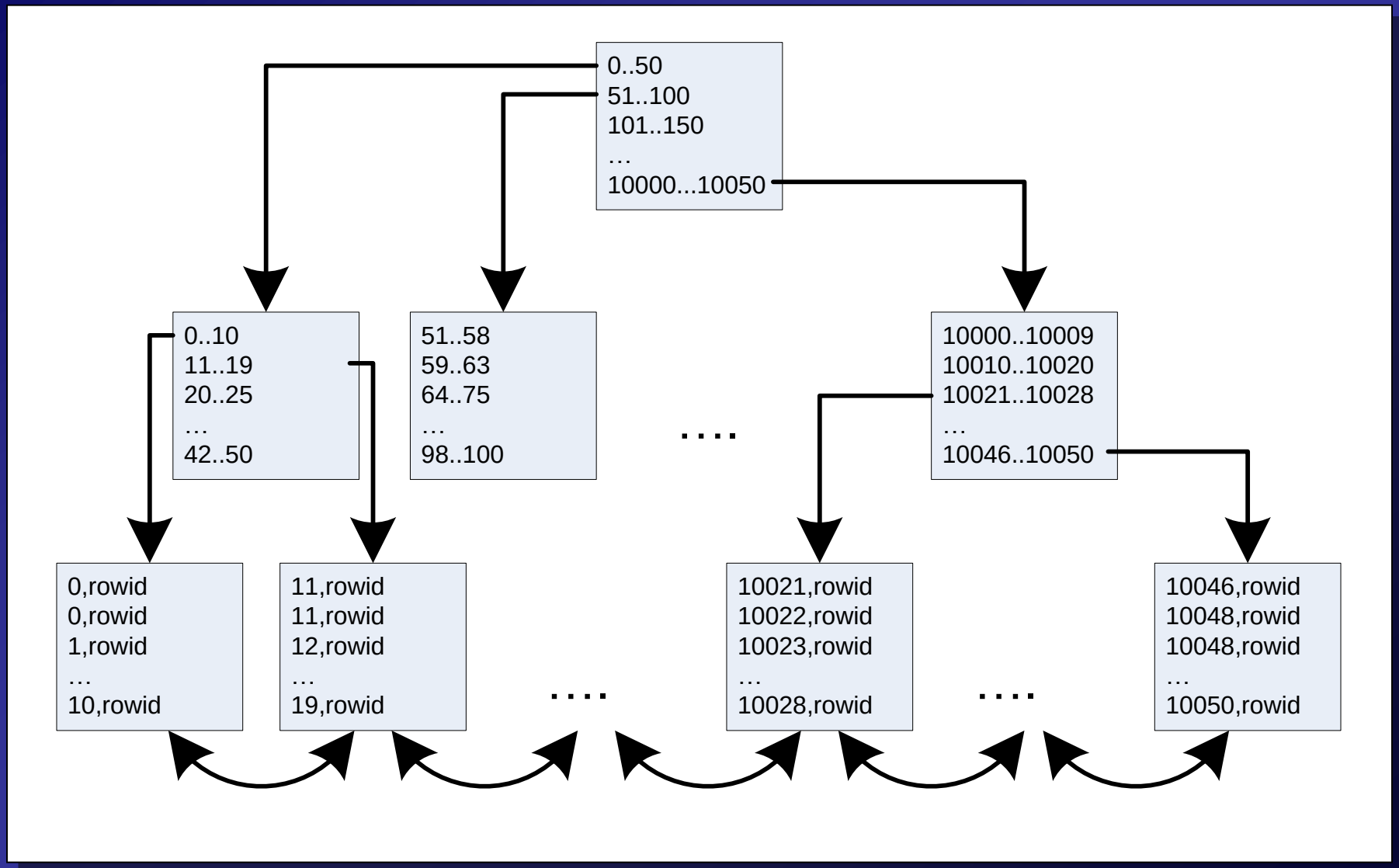
```
CREATE INDEX index_name ON table_name(column_name)
```

```
CREATE UNIQUE INDEX index_name ON table_name(column_name)
```

B^+树索引

- B^+树索引就是通常使用的“传统”索引，是数据库中最常使用的一类索引结构。其实现与二叉查找树很相似。其目标是尽可能减少Oracle查找数据所花费的时间

B+树索引


0..50
51..100
101..150
…
10000...10050
0..10
11..19
20..25
…
42..50
51..58
59..63
64..75
…
98..100
10000..10009
10010..10020
10021..10028
…
10046..10050
0,rowid
0,rowid
1,rowid
…
10,rowid
11,rowid
11,rowid
12,rowid
…
19,rowid
10021,rowid
10022,rowid
10023,rowid
…
10028,rowid
10046,rowid
10048,rowid
10048,rowid
…
10050,rowid

B+树索引

- B+树最底层的块称为叶子节点（leaf node）或叶子块（leaf block），包含各个索引键以及一个rowid（指向索引的行）。叶子节点之上的内部块称为分支块（branch block），用于在结构中实现导航
- B+树的特点之一是：所有叶子块都应该在树的同一层上。这一层称为索引的高度（height）换句话说，索引是高度平衡的（height balanced）

B+树索引中不存在非惟一性条目。在一个非惟一性索引中，Oracle会把rowid作为一个额外的列追加到键上，使得键惟一

反转键索引（反向键索引）

- 反转键索引是一种特殊的B^+树索引。实现反转键索引的目的是为了减少“右侧”索引中对索引叶子块的竞争，缓解忙索引右侧的缓冲区忙等待

- 反转键索引比非反转键索引更大、更空，需要更多的I/O才能获取指定的索引项。因此，在选择进行反转之前，用户需要确定争用问题已经非常严重，以至于采用反转键索引的优点大于其缺点。

- 如果用户选择使用反转键索引，那么只需在通常的索引语句末尾添加一个关键字reverse即可。

位图索引

- 位图索引（bitmap index）是从Oracle 7.3版本开始引入的。目前Oracle企业版和个人版都支持位图索引，但是标准版不支持。位图索引是为数据仓库/即席查询环境设计的。它特别不适用于OLTP系统，如果系统中的数据会由多个并发会话频繁地更新，这种系统也不适用于位图索引。

位图索引

- 对于一个位图索引job_idx，

```
CREATE BITMAP INDEX job_idx ON Emp(job);
```

其存储的结构可能是

值\行	1	2	3	4	5	6	7	8	9	10	11	12
CLERK	1	0	0	0	0	0	0	0	0	1	0	1
SALESMAN	0	1	1	0	1	0	0	0	1	0	0	0
MANAGER	0	0	0	1	0	1	1	0	0	0	0	0
PRESIDENT	0	0	0	0	0	0	0	1	0	0	0	0
ANALYST	0	0	0	0	0	0	0	0	0	0	1	0

位图索引

- 位图索引对于相异基数（distinct cardinality）低的数据最为适合。

- 如果有大量即席查询（ad hoc query），特别是查询以一种即席方式引用了多列或者会生成诸如COUNT之类的聚合，在这种环境中，位图索引就特别有用

基于函数的索引

- 基于函数的索引（function-based index）是Oracle 8.1.5中增加的一种索引，现在已经是标准版的一个特性。

- 利用基于函数的索引，我们能够对计算得出的列建立索引，并在查询中使用这些索引。比如，执行大小写无关的搜索或排序等等。

基于函数的索引

- 示例：
  如果想更有效的处理类似下面的查询，我们可以建立基于函数的索引

```
SELECT VName FROM Video WHERE UPPER(VEngName)= 'LEON';
```

```
CREATE INDEX video_engname_upper_idx
ON Video(UPPER(VEngName))
```

获取有关索引的信息

- 查看用户建立了哪些索引

```
SELECT * FROM user_indexes WHERE table_name='VIDEO';
```

- 查看用户所建索引的细节信息

```
SELECT * FROM user_ind_columns WHERE table_name='VIDEO';
```

对索引的操作

- 修改索引

```
ALTER INDEX video_engname_upper_idx RENAME TO vupper_idx;
```

- 删除索引

```
DROP INDEX index_name;
```

索引的优缺点

- 优点
  - 提高数据检索的效率
- 缺点
  - 过多的索引会影响DML操作的性能

索引的开销

- 使用索引可以提高检索效率，索引的高选择性使其总是比全表搜索能更有效地获取数据。但是，索引的出现会对插入、更新等DML操作带来负面影响

索引的联接、压缩和跳跃

- 在利用复合索引时，会涉及到索引的联接、压缩和跳跃处理问题。

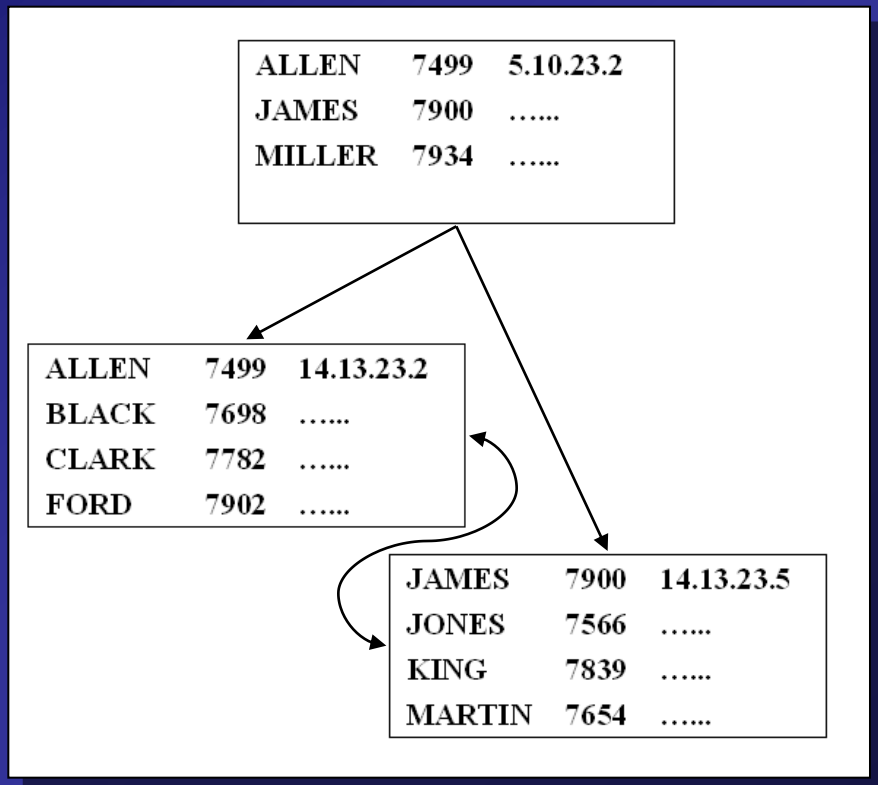
```
CREATE INDEX idx_ename_empno
ON emp(ename,empno)
```

```
CREATE INDEX idx_empno_ename
ON emp(empno,ename)
```

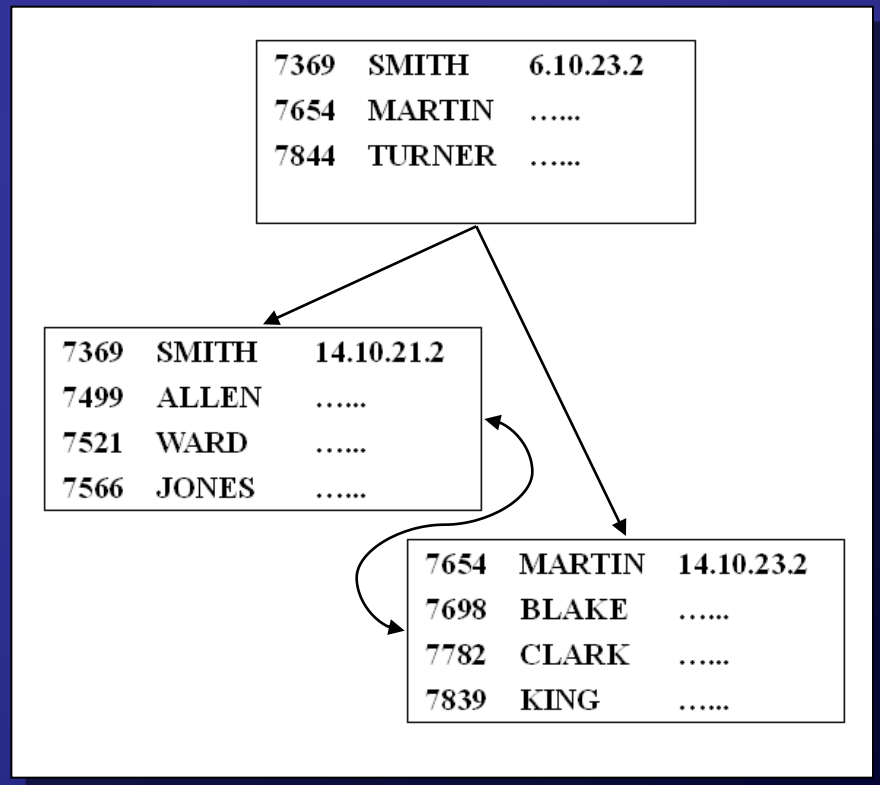
索引的联接、压缩和跳跃

- 索引的联接

```
CREATE INDEX idx_ename_empno
ON emp(ename,empno)
```



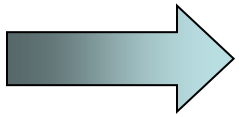

```
CREATE INDEX idx_empno_ename
ON emp(empno,ename)
```



索引的联接、压缩和跳跃

- B+树索引的键压缩是从Oracle 8i引入的特性。键压缩技术的核心是将复合索引中公共的索引项前缀放置到节点的特殊“前缀”区域，将其余的索引项值作为主叶子节点索引项保存，从而避免相同索引项值的冗余存储。

SUMPAL	RB1	PRUNE
SUMPAL	RB1	MULCH
SUMPAL	RB1	SPRAY
SUMPAL	SB1	MULCH
SUMPAL	SB1	WEED
SUMPAL	SB1	HOE



Prefix 0:	SUMPAL,RB1	3
Prefix 1:	SUMPAL,SB1	3
0	PRUNE	
0	MULCH	
0	SPRAY	
1	MULCH	
1	WEED	
1	HOE	

索引的联接、压缩和跳跃

- 索引的跳跃搜索是从Oracle 9i开始引入的功能。它的出现意味着用户不需要再像以前那样担心联接索引中的字段次序，既使用户查询正在选取的字段不是索引的起始列，优化器也可以智能地搜索索引。
- 优化器能够考虑分支节点所暗示的内容，对叶子节点的内容进行推演。使用跳跃搜索不需要特殊的匹配和设置。然而，它可以让联接索引更加有效，甚至它们的起始键没有作为查询谓词提供也是如此。

索引和约束

- 一些特定的约束会隐式地创建索引
  - 唯一性约束
  - 主键约束

索引和约束

- 创建约束时需要考虑的索引相关问题
  - 如果使用唯一性约束，则在禁用约束时，会在没有任何警告的情况下立即删除索引；而在重新启用约束时，导致以前被删除的索引重新创建
  - 在声明主键或唯一性约束时，需要考虑是否有机会将它们与其它列上创建的索引相结合。

小结

- 基本概念，分类
- B^+树索引
- 反转键索引
- 位图索引
- 基于函数的索引
- 关于索引的操作
- 索引开销
- 索引的联接、压缩和跳跃
- 索引和约束

The End