

Recyclebin zu umgehen. Ab Oracle 10g werden Tabellen nicht wirklich gelöscht, solange der Initialisierungsparameter RECYCLEBIN auf ON steht und sie in einem lokal verwalteten Tablespace liegen. Sie werden inklusive der abhängigen Indizes umbenannt bzw. in den Recyclebin verschoben.

```
SQL> SHOW PARAMETER RECYCLEBIN
```

NAME	TYPE	VALUE
recyclebin	string	on

Löschen einer Tabelle und Anzeigen des Recyclebins:

```
SQL> DROP TABLE EMPLOYEES;
```

Tabelle wurde gelöscht.

```
SQL> SHOW RECYCLEBIN
ORIGINAL NAME RECYCLEBIN NAME          OBJECT TYPE DROP TIME
-----
```

ORIGINAL NAME	RECYCLEBIN NAME	OBJECT	TYPE	DROP	TIME
EMPLOYEES	BIN\$T72/pHekSVKXkYcKCskp5g==\\$0	TABLE		2011-01-

Durch Angabe der Option PURGE wird die Tabelle wirklich gelöscht und kann nicht mehr aus dem Recyclebin zurückgeholt werden.

```
SQL> DROP TABLE EMPLOYEES PURGE;
```

Tabelle wurde gelöscht.

```
SQL> SHOW RECYCLEBIN
SQL>
```

Tabellen, die sich im Recyclebin befinden, können mit der Flashback-Technologie wiederhergestellt werden. Indizes, die auf diese Tabellen erzeugt wurden, werden ebenfalls aus dem Recyclebin wiederhergestellt. Sie behalten allerdings den Namen aus dem Recyclebin und müssen deshalb nach der Wiederherstellung umbenannt werden.

```
FLASHBACK TABLE [Tabellenname] TO BEFORE DROP;
```

```
SQL> DROP TABLE EMPLOYEES;
```

Tabelle wurde gelöscht.

```
SQL> SHOW RECYCLEBIN
ORIGINAL NAME RECYCLEBIN NAME          OBJECT TYPE DROP TIME
-----
```

ORIGINAL NAME	RECYCLEBIN NAME	OBJECT	TYPE	DROP	TIME
EMPLOYEES	BIN\$HT6YbMPDQ+G/0W7yPhYZ/A==\\$0	TABLE		2011-01-

```
SQL> FLASHBACK TABLE EMPLOYEES TO BEFORE DROP;
```

Flashback abgeschlossen.

```
SQL> SHOW RECYCLEBIN
```

Fremdschlüssel sowie Tabellen, die auf Dictionary verwalteten Tablespace und im SYSTEM-Tablespace liegen, können nicht wiederhergestellt werden.

7.24. Überblick über Indizes

Indizes sind Objekte, die einen beschleunigten Zugriff auf Daten innerhalb von Tabellen ermöglichen. Hierbei unterscheidet Oracle zwischen B*-Baum-Indizes, Bitmap-Indizes, partitionierten Indizes und Index-organisierten Tabellen, deren physikalische Strukturen unterschiedlich sind und je nach Anwendung und

Datenstruktur innerhalb einer Tabelle zum Einsatz kommen können. B*-Baum Indizes können als normale, zusammengesetzte, funktionsbasierte und Reverse-Key-Indizes erzeugt werden. Diese Arten und Aufgaben der Indizes werden im Folgenden behandelt.

7.25. B*-Baum Indizes

B*-Baum Indizes sind die Standardindizes innerhalb der Oracle Datenbank und werden für die meisten Aufgaben verwendet. Ein B*-Baum Index besitzt einen Aufbau ähnlich einem Index eines Buches. Das Charakteristische an einem Index eines Buches ist, dass alle Schlagwörter in alphabetischer Reihenfolge sortiert sind, während hinter dem Schlagwort sich die entsprechende Seitenzahl befindet, auf deren Seite dieses Wort behandelt wird. Wenn man also ein Buch als Nachschlagewerk verwendet, dann kann man große Bereiche innerhalb des Index überspringen, bis man zu dem Anfangsbuchstaben des Wortes gelangt. Innerhalb dieses Bereichs wird alphabetisch das entsprechende Wort gesucht und mit der Seitenzahl die entsprechende Seite geöffnet.

Analog dazu funktioniert der Index einer Tabelle. Auch bei einem B*-Baum Index einer Tabelle sind die Werte der indizierten Spalte oder Spalten in alphabetischer oder numerischer Reihenfolge sortiert, hinter denen zusätzlich die ROWID abgelegt ist, welche ein eindeutiger Zeiger auf die Zeile innerhalb der Tabelle ist.

Das nächste Bild veranschaulicht den Aufbau eines normalen B*-Baum Index, welcher den Namen IDX_LAST_NAME besitzt und auf die Spalte LAST_NAME der Tabelle EMPLOYEES gelegt wurde. Ein B*-Baum Index besteht aus drei Bereichen, der Wurzel oder Root, die der Einstiegspunkt des Index darstellt, die Nicht-Blatteinheit oder Non-Leaf-Level und wiederum darunter die Blatteinheit oder Leaf-Level, die alle Werte der indizierten Spalte in sortierter Reihenfolge beinhalten. Hinter jedem Wert der Blatteinheit befindet sich die ROWID der Tabellenzeile.

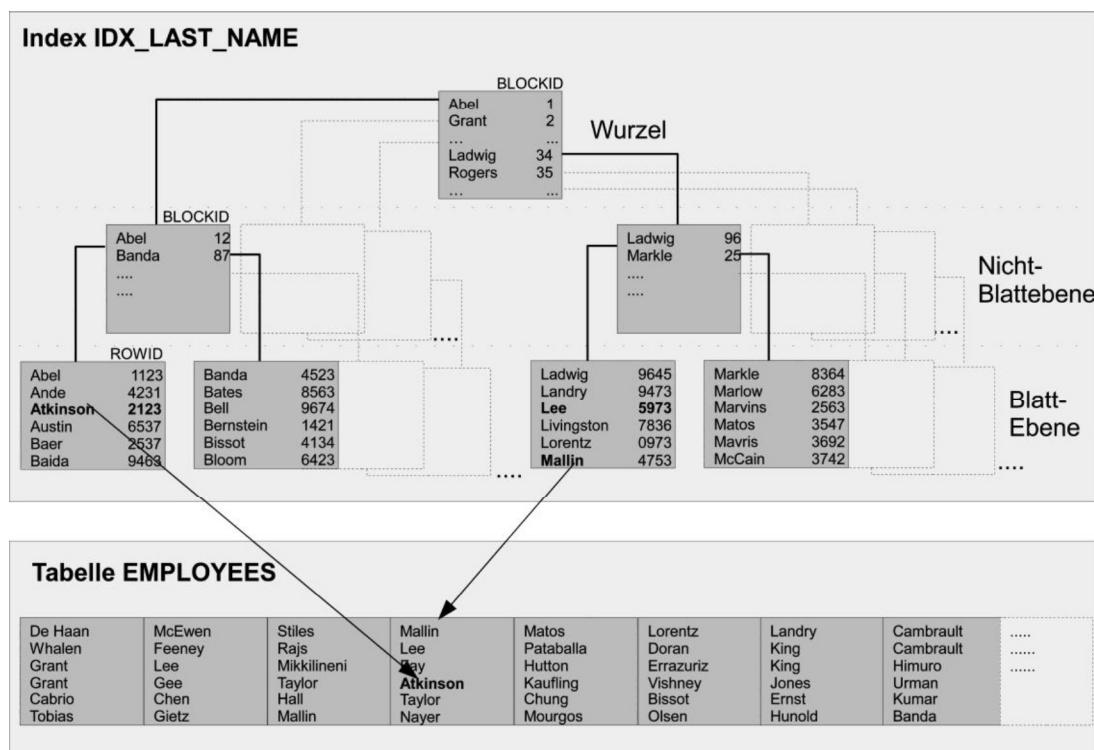


Abbildung 67. Aufbau eines B*-Baum Index

Im Folgenden wird in ungefährer Weise dargestellt, wie ein Index funktioniert. Stellen wir uns vor, dass die Blöcke der Nicht-Blattebene immer nur den ersten Wert der Blöcke der Blattebene und die Wurzel immer nur den ersten Wert der Blöcke der Nicht-Blattebene aufnehmen. Auf die Tabelle EMPLOYEES wird die SQL-Anweisung

```
SELECT *
FROM EMPLOYEES
WHERE LAST_NAME = 'Atkinson';
```

ausgeführt, indem alle Spalten der Tabelle EMPLOYEES für die Zeile ausgelesen werden, bei der die indizierte Spalte LAST_NAME den Wert „Atkinson“ beinhaltet. Das System entscheidet sich bei dieser Anweisung, den Index IDX_LAST_NAME zu verwenden, lokaliert den Block der Wurzel des Index und greift sich den ersten Wert „Abel“. Es wird die Frage gestellt: „Atkinson >= Abel?“. Dies ist der Fall, worauf der nächste Wert „Grant“ aus der Wurzel gegriffen wird. Auch hier wird wieder die Frage gestellt: „Atkinson >= Grant?“, was zu verneinen ist, weil alphabetisch gesehen „Grant“ nach „Atkinson“ kommt. Also weiß Oracle, dass sich der gesuchte Wert „Atkinson“ zwischen „Abel“ und „Grant“ befinden muss. Da sich hinter jedem Wert die Blockadresse des darunter liegenden Blockes der nächsten Ebene befindet und aufgrund der Sortierung der gesuchte Wert sich in der nächsten Ebene befinden muss, springt das System in den Block, auf den „Abel“ zeigt. In dieser Ebene beginnt die Fragestellung erneut: „Atkinson >= Abel?“, was zu bejahen ist, „Atkinson > Banda?“, was zu verneinen ist. Also befindet sich aufgrund der Sortierung der gesuchte Wert zwischen „Abel“ und „Banda“ und das System verwendet die Blockadresse für den Block der nächsten Ebene, auf den der Wert „Abel“ zeigt. Da dieser Block nun ein Block der Blattebene ist, muss sich der gesuchte Wert in ihm befinden, worauf er durchsucht und die ROWID des Wertes „Atkinson“ verwendet wird, um direkt aus der Tabelle die gesamte Zeile zu lesen.

7.25.1. Wann sind Indizes sinnvoll

Die Erstellung von B*-Indizes macht hauptsächlich Sinn, wenn über die zu indizierende Spalte gesucht wird und aus der Tabelle oft einzelne Datensätze gelesen werden beziehungsweise die Ergebnismengen der SQL-Anweisungen im Verhältnis zur Tabelle klein sind.

Zu lesende Blöcke

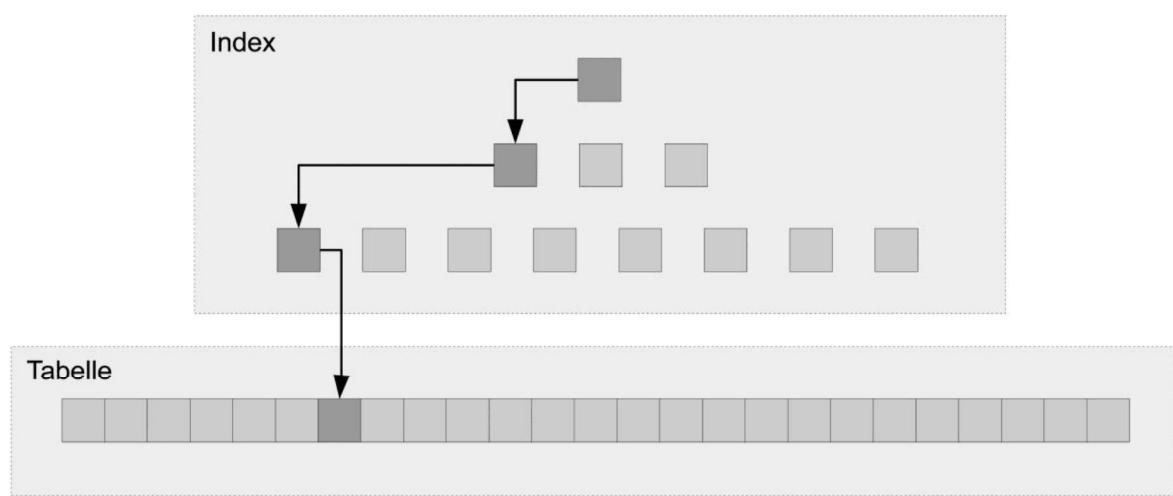


Abbildung 68. Lesen einer geringen Ergebnismenge über einen Index

So ist es nicht immer sinnvoll, Indizes auf Spalten zu erstellen, die zwar in SQL-Anweisungen für Suchoperationen verwendet werden, aber eine hohe Anzahl an Datensätzen im Verhältnis zur Tabelle zurückliefern (außer bei Abdeckung einer SQL-Anweisung über einen Index). Hierbei wird sich Oracle eher für ein Full-Table-Scan, also ein gesamtes Auslesen der Tabelle entscheiden, weil die Anzahl der zu lesenden Blöcke aus der Tabelle bei einer großen Ergebnismenge plus die Blöcke des Index höher sein können, als alle Blöcke der Tabelle zusammen. Ebenso machen (B*-Baum) Indizes auf Spalten keinen Sinn, bei denen das Unterscheidungsmerkmal der Werte sehr gering ist, beispielsweise auf eine Spalte „Anrede“, die die Werte „Herr“ und „Frau“ beinhaltet. Da die Wahrscheinlichkeit, dass die Werte „Herr“ und „Frau“ gleichmäßig über die Tabellendaten verteilt sind, sehr hoch ist, würde Oracle sich für einen Full-Table-Scan entscheiden und den Index ignorieren.

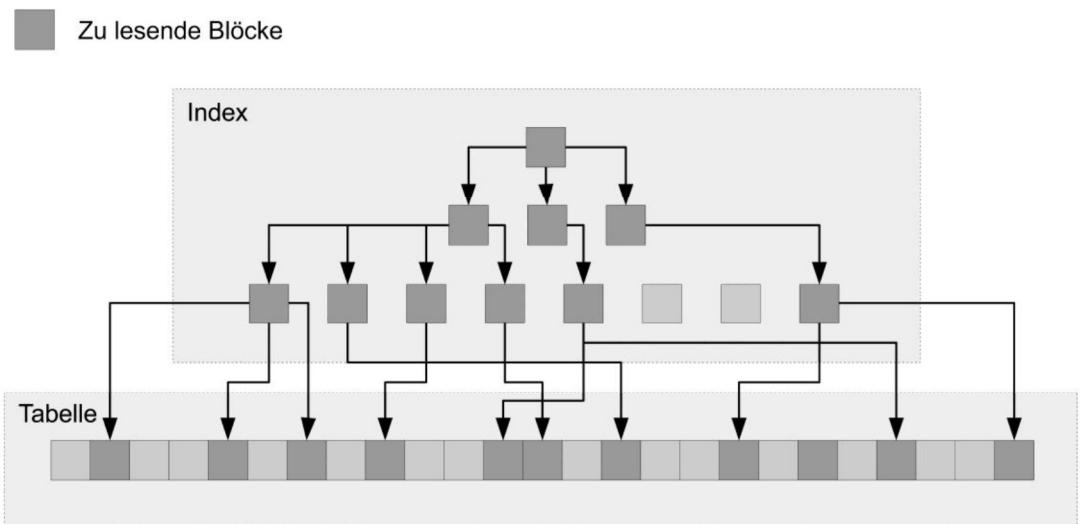


Abbildung 69. Lesen einer großen Ergebnismenge über einen Index

Zusätzlich fasst Oracle bei einem Full-Table-Scan mehrere Blöcke zu einer Leseeinheit(I/O) zusammen, was einen beschleunigten Lesevorgang zur Folge hat, als blockweise über den Index die Zeilen aus der Tabelle zu extrahieren, also für jeden Block ein I/O auszulösen.

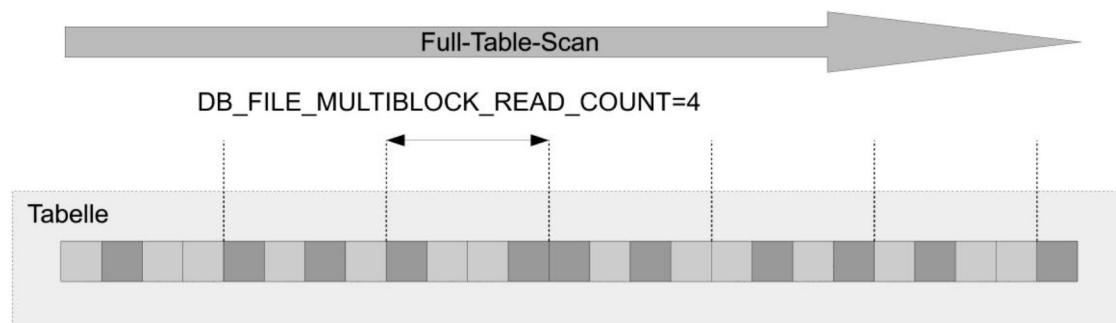


Abbildung 70. Lesen einer großen Ergebnismenge mit einem Full-Table-Scan

Die Anzahl der Blöcke, die bei einem Full-Table-Scan zu einer Leseeinheit zusammengefasst werden sollen, wird über den Parameter `DB_FILE_MULTIBLOCK_READ_COUNT` konfiguriert. Ab Oracle 10g Release 2 besteht die Möglichkeit, Oracle automatisch bestimmen zu lassen, wie viele Blöcke zu einer Leseeinheit zusammengefasst werden, um einen Full-Table-Scan zu beschleunigen.

Da ein Index bei Aktualisierung von Tabellen ebenfalls mit aktualisiert werden muss, haben sie einen Performanceeinfluss auf die Verarbeitung der DML-Anweisungen INSERT, UPDATE, DELETE und MERGE. Aus diesem Grund ist es nicht sinnvoll, Indizes auf Tabellen zu erstellen, in denen eine hohe Anzahl von Änderungen ausgeführt wird.

7.25.2. Anzeigen von Indexinformationen

Um Informationen über Indizes aus dem Data-Dictionary zu erhalten, kann die View DBA_INDEXES verwendet werden. Alle Einstellungen bezüglich eines Index werden in dieser View angezeigt. In dem unteren Beispiel werden die Einstellungen aller Indizes der Tabelle EMPLOYEES des Besitzers HR aus dem Data-Dictionary ausgelesen. Hierbei liefert die Spalte INDEX_TYPE die Art des Index zurück.

SQL> DESCRIBE DBA_INDEXES	Name	Null?	Typ
	OWNER	NOT NULL	VARCHAR2(30)
	INDEX_NAME	NOT NULL	VARCHAR2(30)
	INDEX_TYPE		VARCHAR2(27)
	TABLE_OWNER	NOT NULL	VARCHAR2(30)
	TABLE_NAME	NOT NULL	VARCHAR2(30)
	TABLE_TYPE		VARCHAR2(11)
	UNIQUENESS		VARCHAR2(9)
	COMPRESSION		VARCHAR2(8)
	PREFIX_LENGTH		NUMBER
	TABLESPACE_NAME		VARCHAR2(30)
	INI_TRANS		NUMBER
	MAX_TRANS		NUMBER
	INITIAL_EXTENT		NUMBER
	NEXT_EXTENT		NUMBER
	MIN_EXTENTS		NUMBER
	MAX_EXTENTS		NUMBER
	PCT_INCREASE		NUMBER
	PCT_THRESHOLD		NUMBER
	INCLUDE_COLUMN		NUMBER
	FREELISTS		NUMBER
	FREELIST_GROUPS		NUMBER
	PCT_FREE		NUMBER
	LOGGING		VARCHAR2(3)
	BLEVEL		NUMBER
	LEAF_BLOCKS		NUMBER
	DISTINCT_KEYS		NUMBER
	AVG_LEAF_BLOCKS_PER_KEY		NUMBER
	AVG_DATA_BLOCKS_PER_KEY		NUMBER
	CLUSTERING_FACTOR		NUMBER
	STATUS		VARCHAR2(8)
	NUM_ROWS		NUMBER
	SAMPLE_SIZE		NUMBER
	LAST_ANALYZED		DATE
	DEGREE		VARCHAR2(40)
	INSTANCES		VARCHAR2(40)
	PARTITIONED		VARCHAR2(3)
	TEMPORARY		VARCHAR2(1)
	GENERATED		VARCHAR2(1)
	SECONDARY		VARCHAR2(1)
	BUFFER_POOL		VARCHAR2(7)
	FLASH_CACHE		VARCHAR2(7)
	CELL_FLASH_CACHE		VARCHAR2(7)
	USER_STATS		VARCHAR2(3)
	DURATION		VARCHAR2(15)
	PCT_DIRECT_ACCESS		NUMBER
	ITYP_OWNER		VARCHAR2(30)
	ITYP_NAME		VARCHAR2(30)
	PARAMETERS		VARCHAR2(1000)
	GLOBAL_STATS		VARCHAR2(3)

DOMIDX_STATUS	VARCHAR2(12)
DOMIDX_OPSTATUS	VARCHAR2(6)
FUNCIDX_STATUS	VARCHAR2(8)
JOIN_INDEX	VARCHAR2(3)
IOT_REDUNDANT_PKEY_ELIM	VARCHAR2(3)
DROPPED	VARCHAR2(3)
VISIBILITY	VARCHAR2(9)
DOMIDX_MANAGEMENT	VARCHAR2(14)
SEGMENT_CREATED	VARCHAR2(3)

```
SQL> SELECT INDEX_TYPE, INDEX_NAME FROM DBA_INDEXES
  2 WHERE OWNER='HR' AND TABLE_NAME='EMPLOYEES';
```

INDEX_TYPE	INDEX_NAME
NORMAL	EMP_EMAIL_UK
NORMAL	EMP_EMP_ID_PK
NORMAL	EMP_DEPARTMENT_IX
NORMAL	EMP_JOB_IX
NORMAL	EMP_MANAGER_IX
NORMAL	IDX_LAST_NAME
NORMAL	IDX_FIRSTNAME
NORMAL	IDXSAL

8 Zeilen ausgewählt.

Die View DBA_IND_COLUMNS liefert Informationen, welcher Index einer Tabelle auf welche Spalten gelegt wurde.

```
SQL> DESCRIBE DBA_IND_COLUMNS
Name          Null?    Typ
-----        -----   -----
INDEX_OWNER      NOT NULL VARCHAR2(30)
INDEX_NAME       NOT NULL VARCHAR2(30)
TABLE_OWNER      NOT NULL VARCHAR2(30)
TABLE_NAME       NOT NULL VARCHAR2(30)
COLUMN_NAME      VARCHAR2(4000)
COLUMN_POSITION  NOT NULL NUMBER
COLUMN_LENGTH    NOT NULL NUMBER
CHAR_LENGTH      NUMBER
DESCEND          VARCHAR2(4)
```

```
SQL> SELECT COLUMN_POSITION, COLUMN_NAME
  2  FROM DBA_IND_COLUMNS
  3  WHERE INDEX_NAME='IDX_LAST_NAME' AND TABLE_NAME='EMPLOYEES' AND
  4      TABLE_OWNER='HR';
```

COLUMN_POSITION	COLUMN_NAME
2	FIRST_NAME
1	LAST_NAME

7.25.3. Erstellen von B*-Baum Indizes

Für die Erstellung eines B*-Baum Index wird die angegebene Syntax verwendet.

Syntax für die Erstellung eines B*-Baum Index:

```
CREATE [UNIQUE] INDEX [Indexname]
ON [Tabellenname] ([Spalte1, Spalte2, ..., SpalteN])
[TABLESPACE [Tablespace-Name]]
  [PCTFREE Wert]
  [MAXTRANS Wert]
  [INITRANS Wert]
  [LOGGING|NOLOGGING]
[STORAGE
  (
    [INITIAL Wert K|M|G]
```

```

[NEXT Wert K|M|G]
[PCTINCREASE Wert]
[MINEXTENTS Wert]
[MAXEXTENTS Wert]
[FREELISTS Wert]
[FREELIST GROUPS Wert]
)
[ONLINE COMPUTE STATISTICS]
[REVERSE]

```

Erstellung eines Index auf die Spalte LAST_NAME der Tabelle EMPLOYEES:

```

SQL> CREATE INDEX IDX_LAST_NAME
  2  ON EMPLOYEES (LAST_NAME)
  3  TABLESPACE EXAMPLE
  4  PCTFREE 20
  5  INITTRANS 2
  6  MAXTRANS 10
  7  STORAGE ( INITIAL 1M NEXT 512K)
  8 /

```

Index wurde erstellt.

Klauseln für die Erstellung von Indizes:

UNIQUE	Erzeugt einen eindeutigen Index.
PCTFREE	Prozentsatz freien Speichers, der in einem vollen Block verbleibt für weitere Datensätze in der Blattebene.
PCTUSED	Wird bei Indizes nicht unterstützt.
MAXTRANS	Maximale Anzahl an offenen Transaktionen in einem Block. Der Maximal- und Standardwert ist 255. Ab Oracle 10.2 wird dieser Parameter bei Angabe ignoriert.
INITTRANS	Anzahl von Initialtransaktionen in einem Block. Jede Transaktion muss sich in den Blockheader eintragen. Dies hat zur Folge, dass sich der Blockheader vergrößert. Um im Vorhinein Platz für mögliche Transaktionen zu besitzen, kann dieser Wert angepasst werden, um Wartezeiten zu reduzieren.
LOGGING	Aktiviert die Protokollierung von Redo- und Undo-Daten bei entsprechenden Ladevorgängen.
NOLOGGING	Deaktiviert die Protokollierung von Redo- und Undo-Daten bei entsprechenden Ladevorgängen.
INITIAL	Größe des ersten zugewiesenen Extents des Segmentes. Wird ein Index initial auf einer gefüllten Tabelle erzeugt, so kann das erste Extent die Größe für die zu haltenden Daten bekommen, um für die Initialdaten zusammenhängenden Speicher zu erhalten.
NEXT	Größe des zweiten zugewiesenen Extents des Segmentes
PCTINCREASE	Prozentualer Größenwachstum der folgenden Extents auf NEXT
MINEXTNTS	Minimale Anzahl von Extents in diesem Segment bei Dictionary verwalteten Tablespaces. Bei lokal verwalteten Tablespaces wird dieser Parameter für die Berechnung des Initial-Extents verwendet (INITIAL*MINEXTNTS).
MAXEXTENTS	Maximale Anzahl von Extents in diesem Segment. Diese Klausel ist nur in Dictionary verwalteten Tablespaces verwendbar und wird bei ASSM-Tablespaces ignoriert.
FREELISTS	Anzahl der Freispeicherlisten. Die Verwendung dieses Parameters bei ASSM-Tablespaces liefert einen Fehler.
FREELIST GROUPS	Anzahl der Gruppen der Freispeicherlisten. Die Verwendung dieses Parameters bei ASSM-Tablespaces liefert einen Fehler.
TABLESPACE	Zu speichernder Tablespace des Segments.
ONLINE COMPUTE STATISTICS	Erzeugt bei der Erstellung Index-Statistiken.
REVERSE	Erzeugt einen Reverse-Key Index.

Die Erstellung eines Index im Database Control startet durch Klicken des Buttons „Erstellen“ auf der Index Seite. Hier muss der Besitzer des Index, die Tabelle auf die der Index erstellt werden soll mit den zu indizierenden Spalten und die Reihenfolge der Spalten im Index angegeben werden. Ebenfalls kann eine Auswahl getroffen werden, auf welchen Tablespace der zu erstellende Index gelegt werden soll. Eine

zusätzliche Auswahl bestimmt, welche Art des Index zu erstellen ist (Standard B*-, Baum- oder Bitmap-Index).

7.25.4. Zusammengesetzte Indizes

Indizes können auch über mehr als eine Spalte erstellt werden. Dies ist sinnvoll, wenn entweder in der Kombination der angegebenen Spalten gesucht wird, oder wenn über die ersten Spalten im Index gesucht und alle oder ein Teil der indizierten Spalten in der SELECT-Klausel der SQL-Anweisung angegeben werden.

Der zweite Fall ist interessant, da zur Abdeckung der SQL-Anweisung nur der Index verwendet wird, ohne Zugriffe auf die Tabelle durchzuführen, weil die benötigten Spaltenwerte bereits im Index vorhanden sind.

Hierzu ein Beispiel:

Auf die Spalte LAST_NAME der Tabelle EMPLOYEES wird ein Index mit dem Namen IDX_LAST_NAME erstellt. Die SQL-Anweisung liest die Spalte FIRST_NAME für alle Mitarbeiter der Tabelle EMPLOYEES, deren Nachnamen in der Spalte LAST_NAME „King“ lauten. Da der Index nur auf die Spalte LAST_NAME erstellt wurde, werden die betreffenden Zeilen über den Index gesucht und der Vorname der Spalte FIRST_NAME aus der Tabelle EMPLOYEES herausgegeben. Der Ausführungsplan hierzu zeigt, dass erst der Index durchsucht wird und dann mit den betreffenden ROWIDs die Zeilen aus der Tabelle extrahiert werden.

Erstellung eines Index nur auf die Spalte LAST_NAME:

```
SQL> CREATE INDEX IDX_LAST_NAME
  2  ON EMPLOYEES (LAST_NAME)
  3  TABLESPACE EXAMPLE
  4  PCTFREE 20
  5  INITTRANS 2
  6  MAXTRANS 10
  7  STORAGE ( INITIAL 1M NEXT 512K)
  8  /
```

Index wurde erstellt.

```
SQL> SET AUTOTRACE TRACEONLY EXPLAIN
SQL> SELECT FIRST_NAME
  2  FROM EMPLOYEES
  3  WHERE LAST_NAME='King';
```

Ausführungsplan

Plan hash value: 1857044302

Id	Operation	Name	Rows
0	SELECT STATEMENT		2
1	TABLE ACCESS BY GLOBAL INDEX ROWID	EMPLOYEES	2
* 2	INDEX RANGE SCAN	IDX_LAST_NAME	2

Als Nächstes wird der vorherige Fall wiederholt, allerdings wird der Index auf die Spaltenkombination LAST_NAME, FIRST_NAME erstellt. In diesem Fall nimmt der Index die Werte beider Spalten auf und sortiert sie erst nach der ersten Spalte LAST_NAME und innerhalb dieser Spalte nach FIRST_NAME. Gedanklich könnte man sich den Inhalt der Blöcke der Blattebene folgendermaßen vorstellen:

Daten der Blattebene eines zusammengesetzten Index aus den Spalten LAST_NAME und FIRST_NAME:

LAST_NAME	FIRST_NAME	ROWID
Abel	Ellen	AAAzeaAAJAAAACLAAN
Ande	Sundar	AAAzeccAAJAAAACbAAS
Atkinson	Mozhe	AAAzebAAJAAAACtAAQ
Austin	David	AAAzebAAJAAAACtAAAD
Baer	Hermann	AAAzeZAAJAAAACDAAB
Baida	Shelli	AAAzebAAJAAAACtAAI
Banda	Amit	AAAzeccAAJAAAACbAAT
Bates	Elizabeth	AAAzeccAAJAAAACbAAV
Bell	Sarah	AAAzeaAAJAAAACLAAO
Bernstein	David	AAAzebAAJAAAACtAAc
Bissot	Laura	AAAzebAAJAAAACtAAP
Bloom	Harrison	AAAzebAAJAAAACtAA1
Bull	Alexis	AAAzebAAJAAAACtAAy
Cabrio	Anthony	AAAzeccAAJAAAACbAAd
Cambrault	Gerald	AAAzeccAAJAAAACbAAM
Cambrault	Nanette	AAAzebAAJAAAACtAAf
Chen	John	AAAzebAAJAAAACtAAf
Chung	Kelly	AAAzebAAJAAAACtAA0
.....		

Um die entsprechenden Zeilen zu finden, wird genauso wie im vorherigen Fall eine Suche über den Index durchgeführt. Wenn man aber den Ausführungsplan betrachtet, entfällt der Zugriff auf die Tabelle, weil die Werte der Spalte FIRST_NAME ebenfalls im Index vorhanden sind und aus ihm direkt zurückgegeben werden können.

Abdeckung der Abfrage über einen Index:

```
SQL> CREATE INDEX IDX_LAST_NAME
  2  ON EMPLOYEES (LAST_NAME, FIRST_NAME)
  3  TABLESPACE EXAMPLE
  4  PCTFREE 20
  5  INITTRANS 2
  6  MAXTRANS 10
  7  STORAGE ( INITIAL 1M NEXT 512K)
  8  /
```

Index wurde erstellt.

```
SQL> SET AUTOTRACE TRACEONLY EXPLAIN
SQL> SELECT FIRST_NAME
  2  FROM EMPLOYEES
  3  WHERE LAST_NAME='King';
```

Ausführungsplan

```
Plan hash value: 2450145581

-----| Id   | Operation          | Name           | Rows  |
-----|   0  | SELECT STATEMENT   |                |      2 |
| *  1  |  INDEX RANGE SCAN  | IDX_LAST_NAME |      2 |
```

Unter den genannten Umständen kann ein Index auch dann Vorteile bieten, wenn eine größere Menge an Datensätzen im Verhältnis zur Tabelle gelesen werden soll. Besitzt beispielsweise eine Tabelle eine größere Anzahl an Spalten und dadurch eine große Zeilenbreite, allerdings werden durch SQL-Anweisungen nur ein paar Spalten der Tabelle abgefragt, so kann es sinnvoll sein, über diese Spalten einen Index zu legen. Aufgrund der größeren Zeilenbreite der Tabelle können weniger Zeilen pro Block abgelegt werden, als wenn nur die benötigten Spalten in einem

Index abgelegt werden. Hat die Tabelle zum Beispiel 60 Spalten, eine SQL-Anweisung benötigt allerdings oft nur zwei dieser Spalten, so könnte die Erstellung eines Index über diese beiden Spalten die SQL-Anweisung stark beschleunigen, da weniger Blöcke gelesen werden müssen. Selbst ohne Angabe einer WHERE-Klausel zum Filtern von Daten würde unter diesen Umständen ein Full Table Scan entfallen und stattdessen würde ein Fast-FullIndex-Scan durchgeführt, welcher die gesamte Blattebene des Index ausliest. Auch der Parameter DB_FILE_MULTIBLOCK_READ_COUNT kommt bei diesem Zugriffspfad zum Tragen.

Wichtig:

Wichtige Voraussetzung für ein Fast-Full-Index-Scan ist aber, dass die führende Spalte des Index eine NOT NULL Spalte ist, da Null-Werte in Indizes nicht mit aufgenommen werden und somit nicht alle Zeilen der führenden Spalten im Index vorhanden sind, wenn sie NULL ist. Ist die führende Spalte im Index eine NULL-Spalte, so wird ein Full-Table-Scan durchgeführt.

Full Table Scan und Fast Full Table Scan:

```
SQL> CREATE INDEX IDX_LAST_NAME
  2  ON EMPLOYEES (LAST_NAME, FIRST_NAME)
  3  TABLESPACE EXAMPLE
  4  PCTFREE 20
  5  INITTRANS 2
  6  MAXTRANS 10
  7  STORAGE ( INITIAL 1M NEXT 512K)
  8  /
```

Index wurde erstellt.

```
SQL> SET AUTOTRACE TRACEONLY EXPLAIN
```

```
SQL> SELECT LAST_NAME, FIRST_NAME FROM EMPLOYEES;
```

Ausführungsplan

```
-----  
Plan hash value: 2513133951
```

```
-----  
| Id  | Operation          | Name      |  
-----  
|   0 | SELECT STATEMENT   |           |  
|   1 |  TABLE ACCESS FULL | EMPLOYEES |
```

```
SQL> ALTER TABLE EMPLOYEES MODIFY LAST_NAME VARCHAR2(30) NOT NULL;
```

Tabelle wurde geändert.

```
SQL> SELECT LAST_NAME, FIRST_NAME FROM EMPLOYEES;
```

Ausführungsplan

```
-----  
Plan hash value: 1897246266
```

```
-----  
| Id  | Operation          | Name      |  
-----  
|   0 | SELECT STATEMENT   |           |  
|   1 |  INDEX FAST FULL SCAN| IDX_LAST_NAME |
```

7.25.5. Reverse-Key Indizes

Reverse-Key Indizes sind eine spezielle Art von B*-Indizes, die ein Problem mit stetig ansteigenden Werten von indizierten Spalten minimieren sollen. Soll ein Index beispielsweise auf eine Spalte gelegt werden die Rechnungsnummern beinhaltet, so haben Rechnungsnummern die Charakteristik, dass sie fortlaufend sein müssen. Dies bedeutet, dass aufgrund der Sortierung des Index neue Werte in den letzten Block der Blattebene abgelegt werden müssen. Werden viele Zeilen hinzugefügt, entsteht ein sogenannter heißer Block und es entstehen Wartezustände in diesem Index.

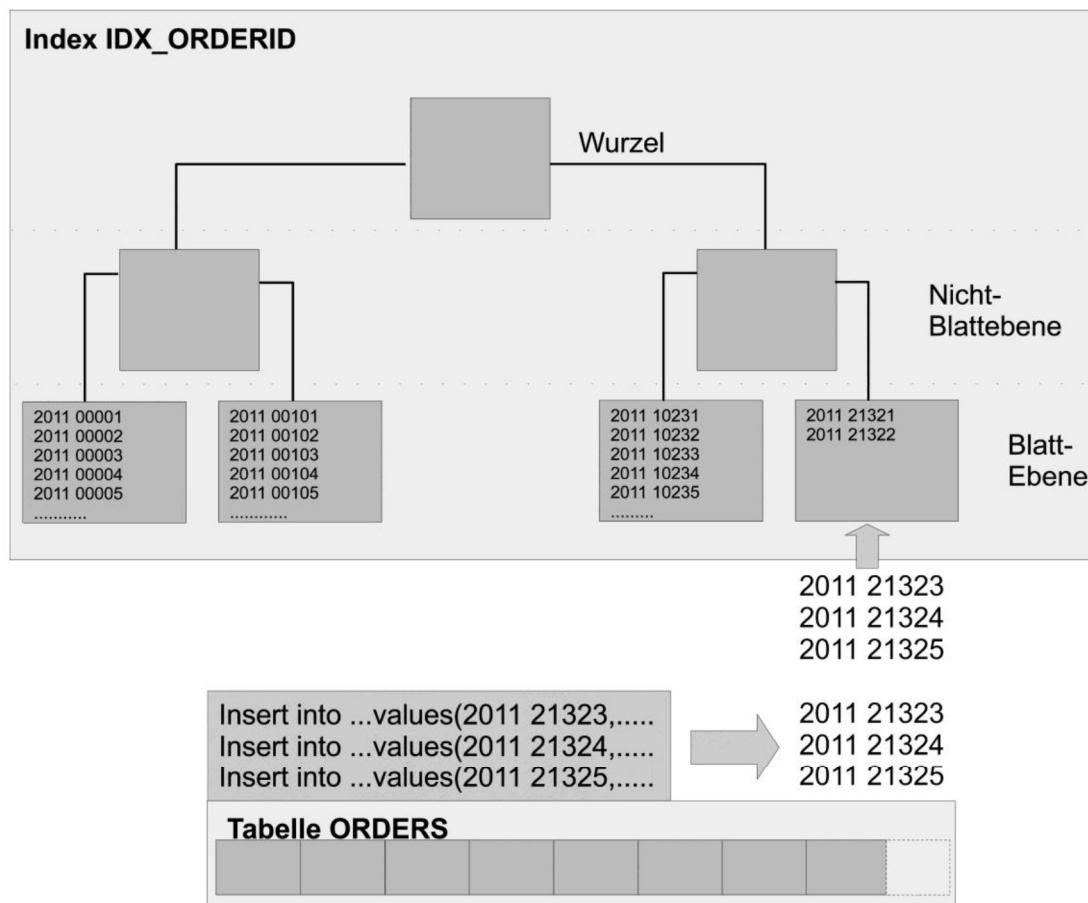


Abbildung 71. Einfügen von stetig ansteigenden Werten in den letzten Block der Blattebene

Um dieses Problem zu lösen, bietet Oracle einen Reverse-Key Index an, bei dem die Werte der indizierten Spalte umgekehrt in den Index eingetragen werden, wodurch wieder eine gleichmäßige Verteilung der Werte über den Index erfolgt und heiße Blöcke vermieden werden.

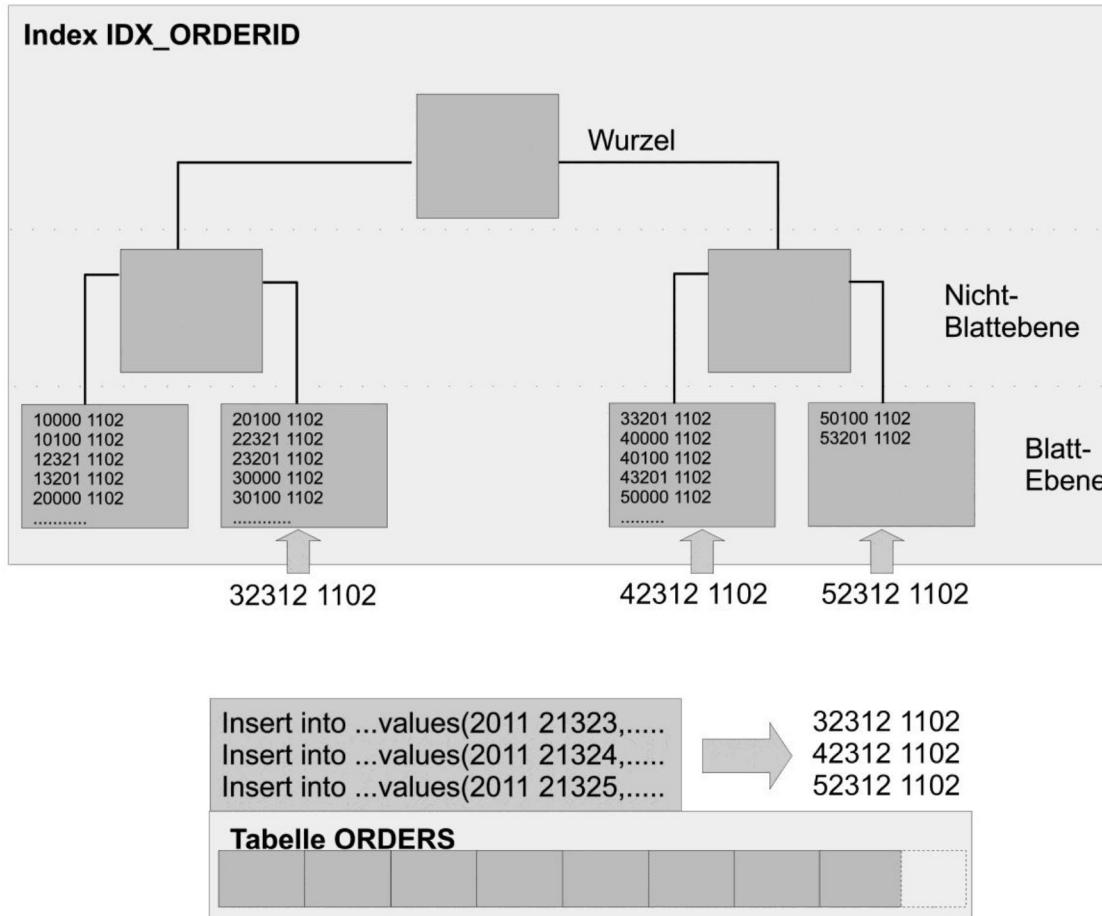


Abbildung 72. Verwendung des Reverse-Key Index

Die Erstellung eines Reverse-Key Index wird mit der Klausel REVERSE erreicht.

Hinweis:

Bei dieser Art von Indizes werden keine Bereichsabfragen unterstützt.

Erstellung eines Revers-Key Index:

```
SQL> CREATE INDEX IDX_ORDERID
  2  ON ORDERS(ORDERID) TABLESPACE USERS
  3  REVERSE;
```

Index wurde erstellt.

```
SQL> SELECT * FROM ORDERS WHERE ORDERID=201121322;
```

Ausführungsplan

```
Plan hash value: 3208761503
```

Id	Operation	Name	Rows
0	SELECT STATEMENT		1
1	TABLE ACCESS BY INDEX ROWID ORDERS	1	
* 2	INDEX RANGE SCAN	IDX_ORDERID	1

Predicate Information (identified by operation id):

```
2 - access ("ORDERID"=201121322)
```

Bei einer Bereichsabfrage über die indizierte Spalte wird der Reverse-Key Index nicht verwendet:

```
SQL> CREATE INDEX IDX_ORDERID
```

```

2 ON ORDERS(ORDERID) TABLESPACE USERS
3 REVERSE;
Index wurde erstellt.

SQL> SELECT * FROM ORDERS
  2 WHERE ORDERID BETWEEN 201121322 AND 201121323;

```

Ausführungsplan

Plan hash value: 1275100350

Id	Operation	Name	Rows
0	SELECT STATEMENT		3
* 1	TABLE ACCESS FULL	ORDERS	3

Predicate Information (identified by operation id):

1 - filter("ORDERID"><=201121323 AND "ORDERID">>=201121322)

```
SQL> DROP INDEX IDX_ORDERID;
```

Index wurde gelöscht.

```
SQL> CREATE INDEX IDX_ORDERID
  2 ON ORDERS(ORDERID) TABLESPACE USERS;
```

Index wurde erstellt.

```
SQL> SELECT * FROM ORDERS
  2 WHERE ORDERID BETWEEN 201121322 AND 201121323;
```

Ausführungsplan

Plan hash value: 3208761503

Id	Operation	Name	Rows
0	SELECT STATEMENT		3
1	TABLE ACCESS BY INDEX ROWID	ORDERS	3
* 2	INDEX RANGE SCAN	IDX_ORDERID	3

Predicate Information (identified by operation id):

2 - access("ORDERID">>=201121322 AND "ORDERID"><=201121323)

7.25.6. Funktionsbasierte Indizes

Werden Suchen in SQL-Anweisungen auf eine normal indizierte Spalte mit Funktionen durchgeführt, so wird der Index dieser Spalte nicht verwendet.

Hierzu ein Beispiel:

Auf die Tabelle EMPLOYEES wird auf die Spalte LAST_NAME ein B*-Baum Index erstellt. Im nächsten Schritt werden über diese Spalte alle Mitarbeiter mit dem Namen „King“ gesucht. Der Ausführungsplan zeigt hier, dass der erstellte Index verwendet wird.

```
SQL> CREATE INDEX IDX_LAST_NAME
  2 ON EMPLOYEES(LAST_NAME)
  3 TABLESPACE USERS;
```

Index wurde erstellt.

```
SQL> SET AUTOTRACE TRACEONLY EXPLAIN
```

```
SQL> SELECT * FROM EMPLOYEES
  2 WHERE LAST_NAME='King';
```

Ausführungsplan

```
Plan hash value: 333834246
```

Id	Operation	Name
0	SELECT STATEMENT	
1	TABLE ACCESS BY INDEX ROWID EMPLOYEES	
* 2	INDEX RANGE SCAN	IDX_LAST_NAME

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

```
2 - access("LAST_NAME"='King')
```

Nun wird die SQL-Anweisung abgeändert, wobei die SQL-Funktion UPPER bei der Suche über die Spalte LAST_NAME verwendet wird, um bei der Suche die Berücksichtigung der Groß- und Kleinschreibung zu umgehen.

```
SQL> SELECT * FROM EMPLOYEES
  2 WHERE UPPER(LAST_NAME)='KING';
```

Ausführungsplan

```
Plan hash value: 1445457117
```

Id	Operation	Name	Rows
0	SELECT STATEMENT		1
* 1	TABLE ACCESS FULL EMPLOYEES		1

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

```
1 - filter(UPPER("LAST_NAME")='KING')
```

Der Ausführungsplan der Anweisung zeigt nun, dass der Index nicht mehr verwendet wird. Der Grund liegt darin, dass der Index auf die Spalte LAST_NAME und nicht UPPER(LAST_NAME) gelegt wurde. Da Oracle Groß- und Kleinschreibung unterscheidet, werden die Werte im Index entsprechend sortiert. Die Verwendung der Funktion in der SQL-Anweisung ist somit nicht einsetzbar.

Um dieses Problem zu lösen, wird nun ein Index mit der Funktion erstellt, so dass entsprechend der Funktion die Werte im Index vorliegen.

```
SQL> CREATE INDEX IDX_LAST_NAME
  2 ON EMPLOYEES(UPPER(LAST_NAME))
  3 TABLESPACE USERS;
```

Index wurde erstellt.

```
SQL> SELECT * FROM EMPLOYEES
  2 WHERE UPPER(LAST_NAME)='KING';
```

Ausführungsplan

Plan hash value: 333834246

Id	Operation	Name	Rows
0	SELECT STATEMENT		1
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1
* 2	INDEX RANGE SCAN	IDX_LAST_NAME	1

Predicate Information (identified by operation id):

2 - access(UPPER("LAST_NAME")='KING')

Das Ergebnis ist die Verwendung des Index.

7.26. Bitmap-Indizes

Während B*-Baum Indizes bei Spalten mit Werten eines hohen Unterscheidungsmerkmals Vorteile bieten, können Bitmap-Indizes bei Spalten verwendet werden, bei denen das Unterscheidungsmerkmal gering ist. Bitmap-Indizes sind Bestandteil der Enterprise Version und haben eine andere physikalische Struktur als B*-Baum Indizes. Bei B*-Baum Indizes existiert eine hierarchische Struktur, die bei der Suche nach einem Wert durchlaufen wird, bis dieser in der Blattebene gefunden wird. Mithilfe der dort stehenden ROWID des gesuchten Wertes wird dann die dazugehörige Zeile in der Tabelle lokalisiert. Wird eine B*-Baum Index auf eine Spalte mit Werten von geringem Unterscheidungsmerkmal gelegt, zum Beispiel die Spalte „Anrede“ mit den Werten „Herr“ und „Frau“, so würde die erste Hälfte der Blattebene alle Werte mit „Frau“ und die zweite Hälfte mit „Herr“ beinhalten, was zu einer großen Struktur des Index führt.

Um diese Struktur klein zu halten, verwendet der Bitmap-Index Bitmaps, in denen für ein Vorhandensein eines Wertes der indizierten Tabellenspalte ein korrespondierendes Bit auf 1, bei Nichtvorhandensein auf 0 innerhalb des Index gesetzt wird.

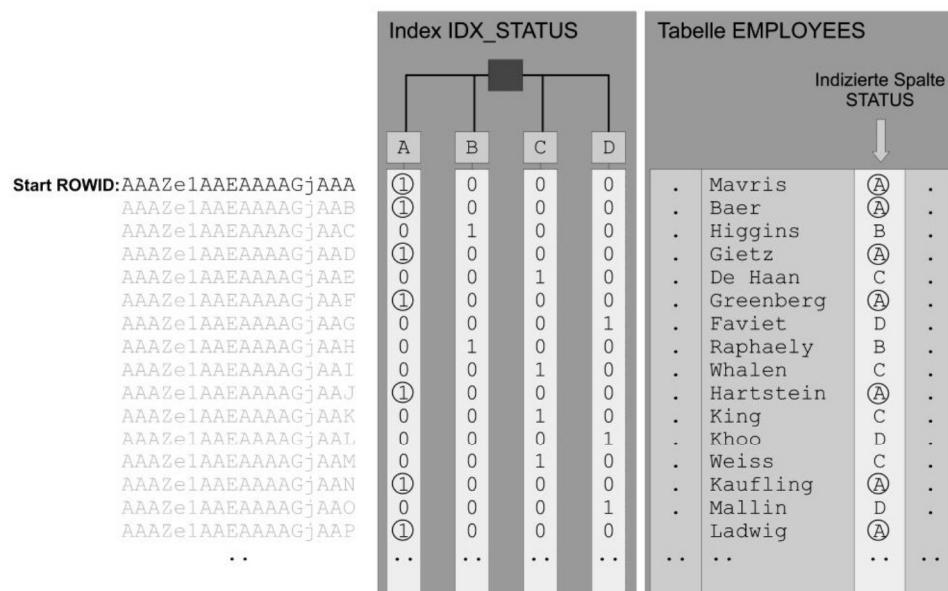


Abbildung 73. Aufbau eines Bitmap-Index

Damit erkannt wird, welches Bit des Bitmaps zur welchem Wert der indizierten Spalte gehört, wird die Start-ROWID und die End-ROWID der Zeilen des Bitmaps aufgezeichnet. Die Differenz von der Start-ROWID bis zum nächsten Vorkommen eines gesuchten Wertes innerhalb des Bitmaps plus die Start-ROWID, ergibt die nächste ROWID, an der sich die gesuchte Zeile innerhalb der Tabelle befinden muss.

Das vorherige Bild zeigt einen Bitmap-Index mit dem Namen `IDX_STATUS` der Spalte `STATUS` und Tabelle `EMPLOYEES`. Die Spalte `STATUS` beinhaltet die Werte „A“, „B“, „C“ und „D“. Betrachten wir mal nur den Wert „A“, so ist ersichtlich, dass das Bitmap für den Wert „A“ in gleichen Abständen ein Bit auf 1 gesetzt hat, wie das Vorkommen des Wertes in der Spalte. Wird also nach dem Wert „A“ gesucht, so wird das Bitmap für „A“ extrahiert und der Abstand des Vorkommens des gesetzten Bit 1 auf die Start-ROWID addiert und die daraus resultierende ROWID verwendet, um die dazugehörige Zeile aus der Tabelle zu erhalten.

7.26.1. Erstellen von Bitmap-Indizes

Die Erstellung von Bitmap-Indizes ist ähnlich der Erstellung eines B*-Baum Index, wobei bei der Erstellung die zusätzliche Klausel `BITMAP` angegeben wird.

Syntax für die Erstellung eines Bitmap-Index:

```
CREATE BITMAP INDEX [Indexname]
ON [Tabellenname] ([Spalte1, Spalte2, ..., SpalteN])
[TABLESPACE [Tablespace-Name]]
    [PCTFREE Wert]
    [MAXTRANS Wert]
    [INITTRANS Wert]
    [LOGGING|NOLOGGING]
[STORAGE
(
    [INITIAL Wert K|M|G]
    [NEXT Wert K|M|G]
    [PCTINCREASE Wert]
    [MINEXTENTS Wert]
    [MAXEXTENTS Wert]
    [FREELISTS Wert]
    [FREELIST GROUPS Wert]
)
]
[ONLINE COMPUTE STATISTICS]
```

Erstellung eines Bitmap-Index auf eine Spalte Status:

```
SQL> CREATE BITMAP INDEX idx_status
  2  ON employees(status)
  3  TABLESPACE Users
  4  PCTFREE 10
  5  MAXTRANS 100
  6  INITTRANS 3
  7  NOLOGGING
  8  STORAGE
  9  (
10      INITIAL 1M
11      NEXT 512K
12      PCTINCREASE 5
13      MINEXTENTS 2
14      MAXEXTENTS 10
15      FREELISTS 1
16      FREELIST GROUPS 1
17  )
18  ONLINE COMPUTE STATISTICS
19 /
```

Index wurde erstellt.

```
SQL> SET AUTOTRACE TRACEONLY EXPLAIN
SQL> SELECT * FROM EMPLOYEES WHERE STATUS='A';
```

Ausführungsplan

Plan hash value: 792930391

Id	Operation	Name	Rows
0	SELECT STATEMENT		13549
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	13549
2	BITMAP CONVERSION TO ROWIDS		
* 3	BITMAP INDEX SINGLE VALUE	IDX_STATUS	

Predicate Information (identified by operation id):

```
3 - access("STATUS"='A')
```

Die Suche in der Tabelle EMPLOYEES nach Mitarbeitern über die Spalte STATUS mit dem Wert „A“ zeigt, dass im Bitmap-Index die Zeilen gesucht, das gefundene Bitmap in die entsprechenden ROWIDs konvertiert und aus der Tabelle zurückgeliefert wird.

7.26.2. Bitmap-OR und Bitmap-AND

Interessant werden Bitmap-Indizes bei Suche nach Zeilen mit mehreren Suchwerten in mehreren indizierten Spalten. In Abhängigkeit, ob eine Verknüpfung der Suchbedingungen in der WHERE-Klausel der SQL-Anweisung über den Operator OR oder AND durchgeführt wird, kann Oracle die extrahierten Bitmaps für die Werte aus dem Index unterschiedlich verarbeiten. Besitzt die Tabelle EMPLOYEES beispielsweise die Spalten STATUS und GRADE, welche beide einen Bitmap-Index besitzen, so werden bei der Suche die Bitmaps des Index für die gesuchten Werte extrahiert und in Abhängigkeit des verwendeten Operators AND oder OR vereinigt, so dass ein neues Gesamtbmap entsteht, aus dem dann die ROWIDs der dazugehörigen Zeilen zurückkonvertiert werden.

Das folgende Bild zeigt die Suche nach Zeilen in der Tabelle EMPLOYEES mit folgender SQL-Anweisung:

```
SELECT * FROM EMPLOYEES WHERE STATUS='A' OR GRADE='2';
```

Hierbei wird das Bitmap der Spalte STATUS für den Wert „A“ und das Bitmap der Spalte GRADE für den Wert „2“ aus den entsprechenden Indizes extrahiert und mit dem Operator OR vereinigt.

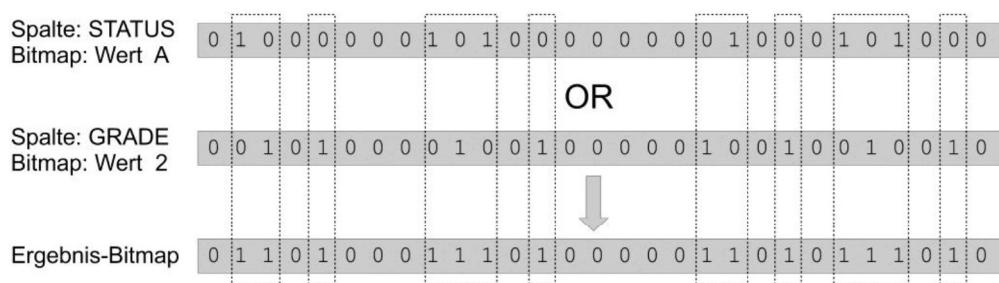


Abbildung 74. Vereinigung zweier Bitmaps mit dem Operator OR

Wird nach Zeilen gesucht, die den Wert „A“ für die Spalte STATUS **oder** den Wert „2“ für die Spalte GRADE besitzen, werden ihre Bitmaps „übereinandergelegt“ und, bei Vorkommen eines der beiden Werte in ihrem Bitmap, das gesetzte Bit für die Position in das Ergebnis-Bitmap übernommen. Aus dem Ergebnis-Bitmap werden dann die dazugehörigen ROWIDs der gesuchten Zeilen zurückkonvertiert, mit denen dann die Zeilen aus der Tabelle ausgelesen werden.

Extraktion der Bitmaps mit nachfolgender Vereinigung mit dem Operator OR:

SQL> SET AUTOTRACE TRACEONLY EXPLAIN

SQL> SELECT * FROM EMPLOYEES WHERE STATUS='A' OR GRADE='2';

Ausführungsplan

Plan hash value: 2962991770

Id	Operation	Name	Rows
0	SELECT STATEMENT		21454
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	21454
2	BITMAP CONVERSION TO ROWIDS		
3	BITMAP OR		
* 4	BITMAP INDEX SINGLE VALUE	IDX_GRADE	
* 5	BITMAP INDEX SINGLE VALUE	IDX_STATUS	

Predicate Information (identified by operation id):

4 - access("GRADE"=2)
5 - access("STATUS"='A')

Wird der Operator AND verwendet, so wird in ähnlicher Weise vorgegangen, allerdings müssen an den gleichen Positionen in den beiden Bitmaps die gesuchten Werte vorhanden sein.

SELECT * FROM EMPLOYEES WHERE STATUS='A' AND GRADE='2';

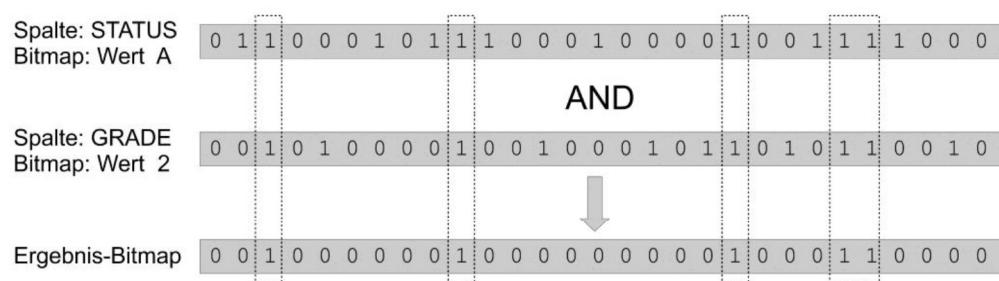


Abbildung 75. Vereinigung zweier Bitmaps mit dem Operator AND

Wird nach Zeilen gesucht, die den Wert „A“ für die Spalte STATUS **und** den Wert „2“ für die Spalte GRADE besitzen, werden ihre Bitmaps „übereinandergelegt“ und, bei Vorkommen beider Werte an gleicher Position in ihren Bitmaps, das gesetzte Bit für die Position in das Ergebnis-Bitmap übernommen.

Extraktion der Bitmaps mit nachfolgender Vereinigung mit dem Operator AND:

SQL> SELECT * FROM EMPLOYEES WHERE STATUS='A' AND GRADE='2';

Ausführungsplan

Plan hash value: 1562305660

Id	Operation	Name	Rows
0	SELECT STATEMENT		1512
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1512
2	BITMAP CONVERSION TO ROWIDS		
3	BITMAP AND		
* 4	BITMAP INDEX SINGLE VALUE	IDX_GRADE	
* 5	BITMAP INDEX SINGLE VALUE	IDX_STATUS	

Predicate Information (identified by operation id):

```
4 - access("GRADE"=2)
5 - access("STATUS"='A')
```

7.26.3. Einsatz von Bitmap-Indizes

Wie schon erwähnt, sind Bitmap-Indizes interessant bei Spalten, deren Werte ein geringes Unterscheidungsmerkmal besitzen. Allerdings haben Bitmap-Indizes auch einen großen Nachteil, da ihr Einfluss auf das Sperrverhalten von Zeilen in einer Tabelle nicht identisch zu anderen Indizes ist.

Hierzu ein Beispiel:

Aus einer Sitzung wird der Wert der Spalte GRADE von „4“ auf „1“ des Mitarbeiters mit der Mitarbeiternummer 100 geändert und die Transaktion nicht abgeschlossen. Kurz danach wird aus einer anderen Sitzung bei einem Mitarbeiter mit der Mitarbeiternummer 234 die Spalte GRADE von „8“ auf „1“ geändert.

Es werden also zwei unterschiedliche Datensätze bearbeitet. Bei Indizierung der Spalte GRADE mit einem B*-Baum Index führt dies zu keinen Problemen, weil eine exklusive Datensatzsperre nur auf den jeweils zu ändernden Datensatz und den Eintrag des Index gelegt wird. Beide Änderungen können durchgeführt werden, weil sie sich durch ihre Sperren nicht gegenseitig behindern. Ist aber nun die Spalte GRADE mit einem Bitmap-Index indiziert, so wird das gesamte Bitmap für den Wert „1“ gesperrt, wenn aus der ersten Sitzung die Änderung durchgeführt wird. Versucht die zweite Sitzung kurz danach ihre Änderung durchzuführen, möchte diese Sitzung ebenfalls das Bitmap des Index für den Wert „1“ für ihren Datensatz abändern, läuft aber auf die Sperre der ersten Sitzung und wird blockiert. Diese Blockierung bleibt solange aufrecht, bis die erste Sitzung ihre Transaktion abschließt.

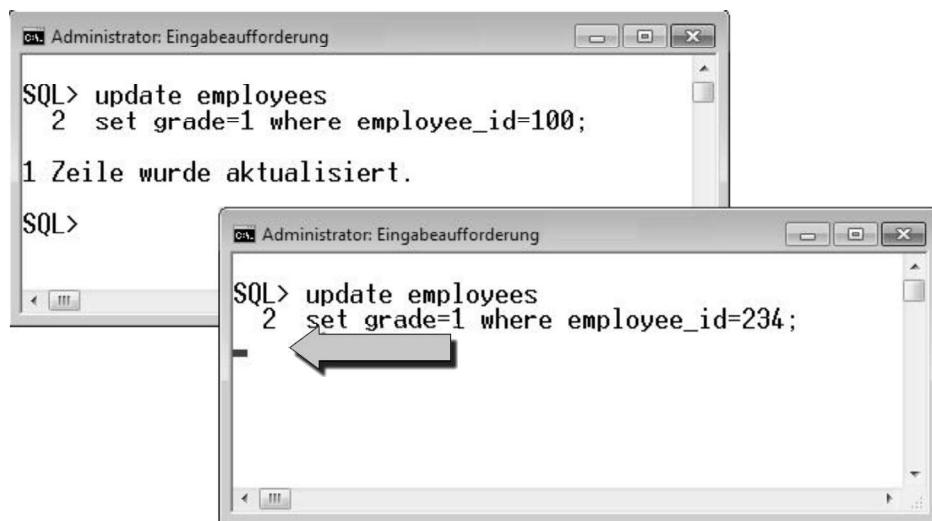


Abbildung 76. Blockierte Sitzung durch eine Sperre durch Änderungen bei einem Bitmap-Index

Aus diesem Grund sind Bitmap-Indizes nicht geeignet bei Tabellenspalten, welche häufig konkurrierend geändert werden. Leider tritt dieses Problem auch bei parallelem Einfügen von Zeilen auf, deren Werte der indizierten Spalte identisch sind. Auch hier wird der zweite Einfügevorgang so lange blockiert, bis die die erste Sitzung ihre Transaktion abschließt.

```

SQL> insert into employees (employee_id, last_name, grade)
  2 values(100001, 'Smith', '1');

1 Zeile wurde erstellt.

SQL>

```

Abbildung 77. Blockierte Sitzung durch eine Sperre durch Einfügen bei einem Bitmap-Index

7.27. Index-organisierte Tabellen (IOT)

Eine Index-organisierte Tabelle ist eine Tabelle und Index zugleich. Während bei einem B*-Baum Index die Blattebene nur die Werte der indizierten Spalte und die ROWID ihrer Zeilen der Tabelle beinhaltet, so ist eine Index-organisierte Tabelle der Index selber und besitzt alle Werte der Spalten in der Blattebene, sortiert nach ihrer indizierten Spalte. Eine Index-organisierte Tabelle kann nur auf Basis eines Primärschlüssels der Tabelle erstellt werden und bietet Vorteile bei einer Suche über die Primärschlüsselspalte mit gleichzeitiger Ausgabe aller Spaltenwerte der Tabelle. Bei einer derartigen Suche entfällt nach dem Durchlaufen der Indexstruktur ein Zugriff auf die Tabelle, weil alle Spaltenwerte direkt aus der Blattebene ausgelesen werden können.

In dem nächsten Beispiel wird eine Index-organisierte Tabelle mit dem Namen EMPLOYEES2 aus der Tabelle EMPLOYEES erstellt. Die Organisation der Tabelle wird durch die Spalte EMPLOYEE_ID und dessen Primärschlüssel erzeugt.

Erstellung der Index-organisierten Tabelle EMPLOYEES2 auf Basis der Tabelle EMPLOYEES:

```

SQL> CREATE TABLE EMPLOYEES2
  2  (EMPLOYEE_ID,
  3  FIRST_NAME,
  4  LAST_NAME,
  5  EMAIL,
  6  PHONE_NUMBER,
  7  HIRE_DATE,
  8  JOB_ID,
  9  SALARY,
 10 COMMISSION_PCT,
 11 MANAGER_ID,
 12 DEPARTMENT_ID,
 13 CONSTRAINT PK_EMPID PRIMARY KEY(EMPLOYEE_ID)
14 ORGANIZATION INDEX
 15 AS SELECT * FROM EMPLOYEES;

```

Tabelle wurde erstellt.

Im zweiten Schritt wird eine Suche des Mitarbeiters mit der Mitarbeiternummer 100 über die Primärschlüsselalte EMPLOYEE_ID durchgeführt, allerdings soll das Ergebnis der SQL-Anweisung alle Spaltenwerte dieses Mitarbeiters anzeigen.

Ausführungsplan bei der Suche in einer Index-organisierten Tabelle:

```
SQL> SET AUTOTRACE TRACEONLY EXPLAIN
```

```
SQL> SELECT * FROM EMPLOYEES2
  2 WHERE EMPLOYEE_ID=100;
```

Ausführungsplan

Plan hash value: 2962278629

Id	Operation	Name	Rows
0	SELECT STATEMENT		1
* 1	INDEX UNIQUE SCAN	PK_EMPID	1

Predicate Information (identified by operation id):

1 - access("EMPLOYEE_ID"=100)

Das Ergebnis zeigt, dass für die Rückgabe der Zeilen nur ein Indexzugriff verwendet wird.

Im Vergleich dazu dieselbe Abfrage auf die Tabelle EMPLOYEES, welche nicht als IOT erstellt wurde, aber einen B*-Baum Index auf der Spalte EMPLOYEE_ID besitzt.

Ausführungsplan bei der Suche in einer nicht Index-organisierten Tabelle:

```
SQL> SELECT * FROM EMPLOYEES
  2 WHERE EMPLOYEE_ID=100;
```

Ausführungsplan

Plan hash value: 1833546154

Id	Operation	Name	Rows
0	SELECT STATEMENT		1
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1
* 2	INDEX UNIQUE SCAN	EMP_EMP_ID_PK	1

Predicate Information (identified by operation id):

2 - access("EMPLOYEE_ID"=100)

Der Ausführungsplan zeigt, dass nach dem Finden des Wertes im Index ein Zugriff auf die Tabelle über die gefundene ROWID erfolgen muss, um alle Spaltenwerte der Zeile anzugeben.

7.27.1. Erstellen von Index-organisierten Tabellen

Die Syntax für die Erstellung einer Index-organisierten Tabelle ist folgende:

Syntax für die Erstellung von IOTs:

```

CREATE TABLE [Name]
(
    [Spalte1] [Datentyp] [DEFAULT Wert] [NOT NULL],
    [Spalte2] [Datentyp] [DEFAULT Wert] [NOT NULL],
    .....
    [SpalteN] [Datentyp] [DEFAULT Wert] [NOT NULL],
    CONSTRAINT [Constraint-Name]
        PRIMARY KEY (Spalte1, Spalte2,...,SpalteN)
)
ORGANIZATION INDEX
[PCTFREE Wert]
[MAXTRANS Wert]
[INITRANS Wert]
[LOGGING|NOLOGGING]

[STORAGE
(
    [INITIAL Wert K|M|G]
    [NEXT Wert K|M|G]
    [PCTINCREASE Wert]
    [MINEXTENTS Wert]
    [MAXEXTENTS Wert]
    [FREELISTS Wert]
    [FREELIST GROUPS Wert]
)
]
[TABLESPACE Tablespace-Name]
[INCLUDING Spalte]
[PCTTHRESHOLD Wert]
[OVERFLOW TABLESPACE Tablespace-Name]

```

PCTFREE	Prozentsatz freien Speichers, der in einem vollen Block verbleibt zur Vermeidung der Zeilenmigration
MAXTRANS	Maximale Anzahl an offenen Transaktionen in einem Block. Der Maximal- und Standardwert ist 255. Ab Oracle 10.2 wird dieser Parameter bei Angabe ignoriert.
INITRANS	Anzahl von Initialtransaktionen in einem Block. Jede Transaktion muss sich in den Blockheader eintragen. Dies hat zur Folge, dass sich der Blockheader vergrößert. Um im Vorhinein Platz für mögliche Transaktionen zu bieten, kann dieser Wert angepasst werden, um Wartezeiten zu reduzieren.
LOGGING	Aktiviert die Protokollierung von Redo- und Undo-Daten bei entsprechenden Ladevorgängen.
NOLOGGING	Deaktiviert die Protokollierung von Redo- und Undo-Daten bei entsprechenden Ladevorgängen.
INITIAL	Größe des ersten zugewiesenen Extents des Segmentes. Wird eine Tabelle erzeugt und soll initial gefüllt werden, so kann das erste Extent die Größe für die zu erhaltenden Daten bekommen, um für die Initialdaten zusammenhängenden Speicher zu erhalten.
NEXT	Größe des zweiten zugewiesenen Extents des Segmentes
PCTINCREASE	Prozentualer Größenwachstum der folgenden Extents auf NEXT
MINEXTNTS	Minimale Anzahl von Extents in diesem Segment bei Dictionary verwalteten Tablespaces. Bei lokal verwalteten Tablespaces wird dieser Parameter für die Berechnung des Initial-Extents verwendet (INITIAL*MINEXTENTS).
MAXEXTENTS	Maximale Anzahl von Extents in diesem Segment. Diese Klausel ist nur in Dictionary verwalteten Tablespaces verwendbar und wird bei ASSM-Tablespaces ignoriert.
FREELISTS	Anzahl der Freispeicherlisten. Die Verwendung dieses Parameters bei ASSM-Tablespaces wird ignoriert.
FREELIST GROUPS	Anzahl der Gruppen der Freispeicherlisten. Die Verwendung dieses Parameters bei ASSM-Tablespaces wird ignoriert.
TABLESPACE	Zu speichernder Tablespace des Segments
INCLUDING	Nachfolgende Spalten werden in einem Overflow-Bereich abgelegt.
PCTTHRESHOLD	Prozentualer Anteil, ab dem pro Datensatz und Block eine Auslagerung in einen Overflow-Bereich erfolgt.
OVERFLOW TABLESPACE	Tablespace für den Overflow-Bereich

Beispiel für die Erstellung einer IOT:

```

SQL> CREATE TABLE EMPLOYEES2
  2  (
  3      EMPLOYEE_ID          NUMBER(6),
  4      FIRST_NAME            VARCHAR2(20),
  5      LAST_NAME              VARCHAR2(25),
  6      EMAIL                  VARCHAR2(25),
  7      PHONE_NUMBER           VARCHAR2(20),
  8      HIRE_DATE               DATE,
  9      JOB_ID                 VARCHAR2(10),
10      SALARY                  NUMBER(8,2),
11      COMMISSION_PCT          NUMBER(2,2),
12      MANAGER_ID              NUMBER(6),
13      DEPARTMENT_ID           NUMBER(4),
14      CONSTRAINT PK_EMPID PRIMARY KEY (EMPLOYEE_ID)
15  )
16  ORGANIZATION INDEX
17  PCTFREE 10
18  MAXTRANS 100
19  INITTRANS 4
20  LOGGING
21  STORAGE
22  (
23      INITIAL 1M
24      NEXT 512 K
25      PCTINCREASE 10
26      MINEXTENTS 1
27      MAXEXTENTS 10
28      FREELISTS 1
29      FREELIST GROUPS 1
30  )
31  TABLESPACE USERS
32  INCLUDING EMAIL
33  PCTTHRESHOLD 20
34  OVERFLOW TABLESPACE USERS;

```

Tabelle wurde erstellt.

7.27.2. Overflow-Bereiche

Des Weiteren besteht die Möglichkeit, Spalten einer IOT in einen Overflow-Bereich auszulagern. Dadurch werden nicht alle Spalten in der Blattebene aufgenommen, wodurch sie kleiner gehalten werden kann. Dies ist sinnvoll, wenn nicht alle Spalten in den SQL-Anweisungen der SELECT-Klausel angegeben werden. Nach welcher Spalte eine Auslagerung der Spaltenwerte in einen Overflow-Bereich zu erfolgen hat, wird über die Klausel INCLUDING definiert.

Angabe des Overflow-Bereichs:

```

SQL> CREATE TABLE EMPLOYEES2
.....
31  TABLESPACE USERS
32  INCLUDING EMAIL
33  PCTTHRESHOLD 20
34  OVERFLOW TABLESPACE USERS;

```

Tabelle wurde erstellt.

Zusätzlich bestimmt die Klausel PCTTHRESHOLD, ab welchem prozentualen Anteil pro Zeile und Block die Auslagerung durchzuführen ist. In dem obigen Beispiel erfolgt die Auslagerung der Spaltenwerte ab der Spalte EMAIL, wenn der Schwellwert pro Zeile und Block von 20 % überschritten wird. Die Klausel OVERFLOW TABLESPACE bestimmt den Tablespace, auf dem der Overflow-Bereich abzulegen ist.