Datenbank Architektur für Fortgeschrittene

Ausarbeitung 1: Anfrageverarbeitung

Daniel Gürber
Stefan Eggenschwiler

02.05.2013

Inhaltsverzeichnis

1 Vorbereitung

1.1 Einleitung

Diese Ausarbeitung behandelt die Übung „SQL Tuning“. Sie wird Schritt für Schritt gelöst. Gezeigt werden das SQL-Statement und den dazugehörigen Ausführungsplan, der von der Oracle Datenbank generiert wird. Bei nennenswerten Erkenntnissen werden diese unterhalb des Ausführungsplan in einer kurzen Reflexion behandelt. Die Nummerierung im Dokument entspricht dabei der Nummern des Übungsblattes. Um die Übung auszuführen haben wir folgende Verbindungsdaten verwendet:

Connection Name hades11gdbarc03
Username dbarc03
Password ¡YouKnowIt¿
Role default
Connection Type Basic
Hostname hades.imvs.technik.fhnw.ch
Port 1521
SID ¡kein Eintrag¿
Service Name hades11g.hades.fhnw.ch

1.2 Einrichten Datenbasis

```
1  CREATE TABLE regions
2  AS SELECT *
3    FROM dbarc00.regions;
4
5  CREATE TABLE nations
6  AS SELECT *
7    FROM dbarc00.nations;
8
9  CREATE TABLE parts
10 AS SELECT *
11   FROM dbarc00.parts;
12
13 CREATE TABLE customers
14 AS SELECT *
15   FROM dbarc00.customers;
16
17 CREATE TABLE suppliers
18 AS SELECT *
19   FROM dbarc00.suppliers;
20
21 CREATE TABLE orders
22 AS SELECT *
23   FROM dbarc00.orders;
24
25 CREATE TABLE partsupps
26 AS SELECT *
27   FROM dbarc00.partsupps;
28
29 CREATE TABLE lineitems
30 AS SELECT *
31   FROM dbarc00.lineitems;
```

2 Statistiken erheben

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS('dbarc00','parts');
END;
```

Tabelle	Anzahl Zeilen	Grösse in Bytes	Anzahl Blöcke	Anzahl Extents
CUSTOMERS	150000	23850000	3494	43
LINEITEMS	6001215	750151875	109217	186
NATIONS	25	2675	4	1
ORDER	1500000	166500000	24284	95
PARTS	200000	26400000	3859	46
PARTSUPPS	800000	114400000	16650	88
REGIONS	5	480	4	1

3 Ausführungsplan

```
EXPLAIN PLAN FOR
SELECT *
FROM parts;
```

```
SELECT plan_table_output
FROM TABLE(DBMS_XPLAN.DISPLAY('plan_table',null,'serial'));
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |   200K|    25M|  1051   (1)| 00:00:13 |
|   1 |  TABLE ACCESS FULL| PARTS |   200K|    25M|  1051   (1)| 00:00:13 |
-------------------------------------------------------------------------------
```

Reflexion
Da weder ein Index, noch ein Filterkriterium gesetzt sind, wird die gesamte Tabelle ausgelesen und zurückgegeben.

4 Versuche ohne Index

4.1 Projektion

```
EXPLAIN PLAN FOR
SELECT *
FROM ORDERS;
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |  1500K|   158M|  6610   (1)| 00:01:20 |
|   1 |  TABLE ACCESS FULL| ORDERS |  1500K|   158M|  6610   (1)| 00:01:20 |
-------------------------------------------------------------------------------
```

Reflexion
Da weder ein Index, noch ein Filterkriterium gesetzt sind, wird die gesamte Tabelle ausgelesen und zurückgegeben.

```
1 EXPLAIN PLAN FOR
2 SELECT o_clerk
3 FROM ORDERS;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  1500K|    22M|  6607   (1)| 00:01:20 |
|   1 |  TABLE ACCESS FULL| ORDERS  |  1500K|    22M|  6607   (1)| 00:01:20 |
-------------------------------------------------------------------------------
```

Reflexion
Da weder ein Index, noch ein Filterkriterium gesetzt sind, wird die gesamte Tabelle ausgelesen, da aber nur die o_clerk Spalte ausgelesen wird, sinkt die Größe von 158 MB auf 22 MB.

```
1 EXPLAIN PLAN FOR
2 SELECT DISTINCT o_clerk
3 FROM ORDERS;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  1000 | 16000 |  6676   (2)| 00:01:21 |
|   1 |  HASH UNIQUE       |         |  1000 | 16000 |  6676   (2)| 00:01:21 |
|   2 |   TABLE ACCESS FULL| ORDERS  |  1500K|    22M|  6607   (1)| 00:01:20 |
-------------------------------------------------------------------------------
```

Reflexion
Da weder ein Index, noch ein Filterkriterium gesetzt sind, wird die gesamte Tabelle ausgelesen. Durch das Schlüsselwort DISTINCT wird ein HASH UNIQUE ausgelöst, wodurch die Kosten auf 6676 steigen, die Grösse aber auf 16000 Bytes sinkt.

4.2 Selektion

Exact Point Query

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM orders
4 WHERE o_orderkey=44444;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |     1 |   111 |  6602   (1)| 00:01:20 |
|*  1 |  TABLE ACCESS FULL| ORDERS  |     1 |   111 |  6602   (1)| 00:01:20 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("O_ORDERKEY"=44444)
```

Reflexion
Da kein Index vorhanden ist, wird die ganze Tabelle ausgelesen und auf o_orderkey gefiltert.

Partial Point Query

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM orders
4 WHERE o_orderkey=44444 OR o_clerk='Clerk#000000286';
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |  1501 |   162K|  6629   (1)| 00:01:20 |
|*  1 |  TABLE ACCESS FULL| ORDERS |  1501 |   162K|  6629   (1)| 00:01:20 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("O_CLERK"='Clerk#000000286' OR "O_ORDERKEY"=44444)
```

Reflexion
Da kein Index vorhanden ist, wird die ganze Tabelle ausgelesen und auf o_orderkey und o_clerk gefiltert, die zusätzliche Bedingung erhöht die Kosten von 6602 auf 6629.

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM orders
4 WHERE o_orderkey=44444 AND o_clerk='Clerk#000000286';
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |     1 |   111 |  6611   (1)| 00:01:20 |
|*  1 |  TABLE ACCESS FULL| ORDERS |     1 |   111 |  6611   (1)| 00:01:20 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("O_ORDERKEY"=44444 AND "O_CLERK"='Clerk#000000286')
```

Reflexion
Da kein Index vorhanden ist, wird die ganze Tabelle ausgelesen und auf o_orderkey und o_clerk gefiltert, die AND Bedingung senkt von die Kosten von 6629 auf 6611 verglichen mit der OR Bedingung.

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM orders
4 WHERE o_orderkey*2=44444 AND o_clerk='Clerk#000000286';
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |    15 |  1665 |  6615   (1)| 00:01:20 |
|*  1 |  TABLE ACCESS FULL| ORDERS |    15 |  1665 |  6615   (1)| 00:01:20 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("O_ORDERKEY"*2=44444 AND "O_CLERK"='Clerk#000000286')
```

Reflexion
Da kein Index vorhanden ist, wird die ganze Tabelle ausgelesen und auf o_orderkey und o_clerk gefiltert, die zusätzliche Multiplikation erhöht die Kosten nur um 4.

Range Query

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM orders
4 WHERE o_orderkey BETWEEN 111111 AND 222222
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         | 27780 |  3011K|  6603   (1)| 00:01:20 |
|*  1 |  TABLE ACCESS FULL| ORDERS  | 27780 |  3011K|  6603   (1)| 00:01:20 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("O_ORDERKEY"<=222222 AND "O_ORDERKEY">=111111)
```

Reflexion

Da kein Index vorhanden ist, wird die ganze Tabelle ausgelesen und auf o_orderkey gefiltert, die Range Query ist nur um 1 teurer als die Abfrage auf den einzelnen Wert. Die Intervallgrösse spielt in diesem Beispiel, abgesehen von der höheren bzw. tieferen Anzahl Bytes keine Rolle.

Partial Range Query

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM orders
4 WHERE o_orderkey BETWEEN 44444 AND 55555
5 AND   o_clerk BETWEEN 'Clerk#000000130' AND 'Clerk#000000139'
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |     6 |   666 |  6611   (1)| 00:01:20 |
|*  1 |  TABLE ACCESS FULL| ORDERS  |     6 |   666 |  6611   (1)| 00:01:20 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("O_ORDERKEY"<=55555 AND "O_CLERK"<='Clerk#000000139' AND
              "O_ORDERKEY">=44444 AND "O_CLERK">='Clerk#000000130')
```

Reflexion

Da kein Index vorhanden ist, wird die ganze Tabelle ausgelesen und auf o_orderkey und o_clerk gefiltert, die Range Query ist nur um 8 teurer als die letzte Abfrage.

4.3 Join

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM orders, customers
4 WHERE o_custkey = c_custkey
5 AND   o_orderkey < 100;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |    25 |  6750 |  7555   (1)| 00:01:31 |
|*  1 |  HASH JOIN         |           |    25 |  6750 |  7555   (1)| 00:01:31 |
|*  2 |   TABLE ACCESS FULL| ORDERS    |    25 |  2775 |  6602   (1)| 00:01:20 |
|   3 |   TABLE ACCESS FULL| CUSTOMERS |   150K|    22M|   951   (1)| 00:00:12 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("O_CUSTKEY"="C_CUSTKEY")
   2 - filter("O_ORDERKEY"<100)
```

Reflexion
Da kein Index vorhanden ist, werden beide Tabellen voll geladen. Die orders Tabelle wird gefiltert und mit Hilfe von custkey word ein HASH JOIN ausgeführt. Die Formulierung mit INNER JOIN spielt

5 Versuch mit Index

```
1 CREATE INDEX o_orderkey_ix ON orders(o_orderkey);
```

```
1 CREATE INDEX o_clerk_ix ON orders(o_clerk);
```

Index Name	Index Grösse	Tabellengrösse in Bytes
o_orderkey_ix	60817408	166500000
o_clerik_ix	96468992	166500000

5.1 Projektion

```
1 EXPLAIN PLAN FOR
2 SELECT DISTINCT o_clerk
3 FROM ORDERS;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation             | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |             |  1000 | 16000 |  1615   (5)| 00:00:20 |
|   1 |  HASH UNIQUE          |             |  1000 | 16000 |  1615   (5)| 00:00:20 |
|   2 |   INDEX FAST FULL SCAN| O_CLERK_IX  |  1500K|    22M|  1546   (1)| 00:00:19 |
-------------------------------------------------------------------------------------
```

Reflexion
Die Verwendung des Index senkt die Kosten dieser abfrage von 6676 auf 1615 (Faktor 4.1).

5.2 Selektion

Exact Point Query

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM ORDERS
4 WHERE o_orderkey=44444
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     1 |   111 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| ORDERS        |     1 |   111 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | O_ORDERKEY_IX |     1 |       |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("O_ORDERKEY"=44444)
```

Reflexion
Die Verwendung des Index senkt die Kosten dieser abfrage von 6602 auf 4 (Faktor 1650).

```
EXPLAIN PLAN FOR
SELECT /*+ FULL(orders) */ *
FROM ORDERS
WHERE o_orderkey=44444
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |     1 |   111 |  6602   (1)| 00:01:20 |
|*  1 |  TABLE ACCESS FULL| ORDERS  |     1 |   111 |  6602   (1)| 00:01:20 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("O_ORDERKEY"=44444)
```

Reflexion
Wenn ein TABLE ACCESS FULL erzwungen wird, entsprechen die Kosten wieder denen ohne Index.

Partial Point Query

```
EXPLAIN PLAN FOR
SELECT *
FROM orders
WHERE o_orderkey=44444 OR o_clerk='Clerk#000000286';
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |  1501 |   162K|   336   (0)| 00:00:05 |
|   1 |  TABLE ACCESS BY INDEX ROWID        | ORDERS        |  1501 |   162K|   336   (0)| 00:00:05 |
|   2 |   BITMAP CONVERSION TO ROWIDS       |               |       |       |            |          |
|   3 |    BITMAP OR                        |               |       |       |            |          |
|   4 |     BITMAP CONVERSION FROM ROWIDS   |               |       |       |            |          |
|*  5 |      INDEX RANGE SCAN               | O_CLERK_IX    |       |       |     8   (0)| 00:00:01 |
|   6 |     BITMAP CONVERSION FROM ROWIDS   |               |       |       |            |          |
|*  7 |      INDEX RANGE SCAN               | O_ORDERKEY_IX |       |       |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access("O_CLERK"='Clerk#000000286')
   7 - access("O_ORDERKEY"=44444)
```

Reflexion
Die Verwendung der Indexes senkt die Kosten von 6611 auf 336 (Faktor 19.7) und führt zu BITMAP Operationen.

```
EXPLAIN PLAN FOR
SELECT *
FROM orders
WHERE o_orderkey=44444 AND o_clerk='Clerk#000000286';
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |     1 |   111 |     4   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID  | ORDERS        |     1 |   111 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN            | O_ORDERKEY_IX |     1 |       |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("O_CLERK"='Clerk#000000286')
   2 - access("O_ORDERKEY"=44444)
```

Reflexion
Die Verwendung des Index senkt die Kosten dieser Abfrage von 6611 auf 4 (Faktor 1652.8) und ist somit stärker optimiert als die OR Abfrage.

```
EXPLAIN PLAN FOR
SELECT *
FROM orders
WHERE o_orderkey*2=44444 AND o_clerk='Clerk#000000286';
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |    15 |  1665 |  1464   (1)| 00:00:18 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| ORDERS     |    15 |  1665 |  1464   (1)| 00:00:18 |
|*  2 |   INDEX RANGE SCAN           | O_CLERK_IX |  1500 |       |     8   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("O_ORDERKEY"*2=44444)
   2 - access("O_CLERK"='Clerk#000000286')
```

Reflexion
Die Verwendung des Index senkt die Kosten dieser Abfrage von 6615 auf 1464 (Faktor 4.5) sie ist aber, wegen der Multiplikation, um den Faktor 366 langsamer als die vorherige AND Abfrage, im Gegensatz zum Faktor 1.0006 ohne Index.

Range Query

```
EXPLAIN PLAN FOR
SELECT *
FROM orders
WHERE o_orderkey BETWEEN 111111 AND 222222
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               | 27780 |  3011K|   932   (1)| 00:00:12 |
|   1 |  TABLE ACCESS BY INDEX ROWID| ORDERS        | 27780 |  3011K|   932   (1)| 00:00:12 |
|*  2 |   INDEX RANGE SCAN           | O_ORDERKEY_IX | 27780 |       |    68   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("O_ORDERKEY">=111111 AND "O_ORDERKEY"<=222222)
```

Reflexion
Die Verwendung des Index senkt die Kosten dieser Abfrage von 6603 auf 932 (Faktor 7.1). Wird der Intervall genug gross gewählt, z.B. 222222222, wird wieder ein TABLE ACCESS FULL ausgeführt, da der aufwand für die Index Berechnungen grösser wäre.

Partial Range Query

```
EXPLAIN PLAN FOR
SELECT *
FROM orders
WHERE o_orderkey BETWEEN 44444 AND 55555
AND   o_clerk BETWEEN 'Clerk#000000130' AND 'Clerk#000000139'
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     6 |   666 |    27  (12)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| ORDERS        |     6 |   666 |    27  (12)| 00:00:01 |
|   2 |   BITMAP CONVERSION TO ROWIDS|               |       |       |            |          |
|   3 |    BITMAP AND                |               |       |       |            |          |
|   4 |     BITMAP CONVERSION FROM ROWIDS|           |       |       |            |          |
|   5 |      SORT ORDER BY           |               |       |       |            |          |
|*  6 |       INDEX RANGE SCAN       | O_ORDERKEY_IX |  2780 |       |     9   (0)| 00:00:01 |
|   7 |     BITMAP CONVERSION FROM ROWIDS|           |       |       |            |          |
|   8 |      SORT ORDER BY           |               |       |       |            |          |
|*  9 |       INDEX RANGE SCAN       | O_CLERK_IX    |  2780 |       |    14   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   6 - access("O_ORDERKEY">=44444 AND "O_ORDERKEY"<=55555)
   9 - access("O_CLERK">='Clerk#000000130' AND "O_CLERK"<='Clerk#000000139')
```

Reflexion
Die Verwendung des Index senkt die Kosten dieser Abfrage von 6611 auf 27 (Faktor 244.9), die Verwendung von 2 Filterkriterien führt wieder zu BITMAP Operationen.

5.3 Join

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM orders, customers
4 WHERE o_custkey = c_custkey;
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |  1500K|   386M|       | 17514   (1)| 00:03:31 |
|*  1 |  HASH JOIN         |           |  1500K|   386M|    24M| 17514   (1)| 00:03:31 |
|   2 |   TABLE ACCESS FULL| CUSTOMERS |   150K|    22M|       |   951   (1)| 00:00:12 |
|   3 |   TABLE ACCESS FULL| ORDERS    |  1500K|   158M|       |  6610   (1)| 00:01:20 |
-----------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("O_CUSTKEY"="C_CUSTKEY")
```

Reflexion
In diesem Statement wird auf beide Tabellen TABLE ACCESS FULL ausgeführt, die gesetzten Indexes haben keinen Einfluss auf den Ausführungsplan.

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM orders, customers
4 WHERE o_custkey = c_custkey
5 AND   o_orderkey < 100;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |    25 |  6750 |   957   (1)| 00:00:12 |
|*  1 |  HASH JOIN                    |               |    25 |  6750 |   957   (1)| 00:00:12 |
|   2 |   TABLE ACCESS BY INDEX ROWID| ORDERS        |    25 |  2775 |     4   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN           | O_ORDERKEY_IX |    25 |       |     3   (0)| 00:00:01 |
|   4 |   TABLE ACCESS FULL           | CUSTOMERS     |   150K|    22M|   951   (1)| 00:00:12 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("O_CUSTKEY"="C_CUSTKEY")
   3 - access("O_ORDERKEY"<100)
```

Reflexion
Die Verwendung des Index senkt die Kosten dieser Abfrage von 7555 auf 957 (Faktor 7.9).

```
CREATE INDEX c_custkey_ix ON customers(c_custkey);
```

```
EXPLAIN PLAN FOR
SELECT *
FROM orders, customers
WHERE o_custkey = c_custkey;
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |  1500K|   386M|       | 17514   (1)| 00:03:31 |
|*  1 |  HASH JOIN         |           |  1500K|   386M|    24M| 17514   (1)| 00:03:31 |
|   2 |   TABLE ACCESS FULL| CUSTOMERS |   150K|    22M|       |   951   (1)| 00:00:12 |
|   3 |   TABLE ACCESS FULL| ORDERS    |  1500K|   158M|       |  6610   (1)| 00:01:20 |
-----------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("O_CUSTKEY"="C_CUSTKEY")
```

Reflexion
Auf diese Abfrage hat das setzen des Indexes keinen Einfluss.

Erzwingen eines Nested Loop Joins:

```
EXPLAIN PLAN FOR
SELECT /*+ USE_NL (o c) */ *
FROM orders o, customers c
WHERE o_custkey = c_custkey;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |  1500K|   386M|  3007K  (1)| 10:01:34 |
|   1 |  NESTED LOOPS                |               |       |       |            |          |
|   2 |   NESTED LOOPS               |               |  1500K|   386M|  3007K  (1)| 10:01:34 |
|   3 |    TABLE ACCESS FULL         | ORDERS        |  1500K|   158M|  6610   (1)| 00:01:20 |
|*  4 |    INDEX RANGE SCAN          | C_CUSTKEY_IX  |     1 |       |     1   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| CUSTOMERS     |     1 |   159 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("O_CUSTKEY"="C_CUSTKEY")
```

Reflexion
Das Erzwingen des NESTED LOOPS führt zwar zur Verwendung des Index, steigert aber die Kosten auf von 17514 auf 3007K.

Erzwingen eines Nicht-Hash Joins:

```
EXPLAIN PLAN FOR
SELECT /*+ NO_USE_HASH (o c) */ *
FROM orders o, customers c
WHERE o_custkey = c_custkey;
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |  1500K|   386M|       | 50568   (1)| 00:10:07 |
|   1 |  MERGE JOIN        |           |  1500K|   386M|       | 50568   (1)| 00:10:07 |
|   2 |   SORT JOIN        |           |   150K|    22M|    52M|  6202   (1)| 00:01:15 |
|   3 |    TABLE ACCESS FULL| CUSTOMERS |   150K|    22M|       |   951   (1)| 00:00:12 |
|*  4 |   SORT JOIN        |           |  1500K|   158M|   390M| 44366   (1)| 00:08:53 |
|   5 |    TABLE ACCESS FULL| ORDERS    |  1500K|   158M|       |  6610   (1)| 00:01:20 |
-----------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("O_CUSTKEY"="C_CUSTKEY")
       filter("O_CUSTKEY"="C_CUSTKEY")
```

Reflexion
Das Erzwingen des NESTED LOOPS führt nicht zur Verwendung des Index, und steigert aber die Kosten auf von 17514 auf 50568.

6 Quiz

```
1 EXPLAIN PLAN FOR
2 SELECT count(*)
3 FROM parts, partsupps, lineitems
4 WHERE p_partkey=ps_partkey
5 AND ps_partkey=l_partkey
6 AND ps_suppkey=l_suppkey
7 AND ((ps_partkey = 5 AND p_type = 'MEDIUM ANODIZED BRASS')
8 OR (ps_partkey = 5 AND p_type = 'MEDIUM BRUSHED COPPER'));
```

Ausgangslage:

```
-------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |     1 |    45 | 35577   (2)| 00:07:07 |
|   1 |  SORT AGGREGATE      |           |     1 |    45 |            |          |
|*  2 |   HASH JOIN          |           |     4 |   180 | 35577   (2)| 00:07:07 |
|*  3 |    HASH JOIN         |           |     4 |   144 |  5872   (6)| 00:01:11 |
|*  4 |     TABLE ACCESS FULL| PARTSUPPS |     4 |    36 |  4525   (1)| 00:00:55 |
|*  5 |     TABLE ACCESS FULL| PARTS     |  2667 | 72009 |  1052   (1)| 00:00:13 |
|   6 |    TABLE ACCESS FULL | LINEITEMS |  6001K|    51M| 29675   (1)| 00:05:57 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("PS_PARTKEY"="L_PARTKEY" AND "PS_SUPPKEY"="L_SUPPKEY")
   3 - access("P_PARTKEY"="PS_PARTKEY")
       filter("PS_PARTKEY"=5 AND "P_TYPE"='MEDIUM ANODIZED BRASS' OR
              "PS_PARTKEY"=5 AND "P_TYPE"='MEDIUM BRUSHED COPPER')
   4 - filter("PS_PARTKEY"=5)
```

Auf ps_partkey wird gefiltert:

```
1 CREATE INDEX ps_partkey_ix ON partsupps(ps_partkey);
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |     1 |    45 | 31055   (2)| 00:06:13 |
|   1 |  SORT AGGREGATE               |               |     1 |    45 |            |          |
|*  2 |   HASH JOIN                   |               |     4 |   180 | 31055   (2)| 00:06:13 |
|*  3 |    HASH JOIN                  |               |     4 |   144 |  1350  (23)| 00:00:17 |
|   4 |     TABLE ACCESS BY INDEX ROWID| PARTSUPPS     |     4 |    36 |     4   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN         | PS_PARTKEY_IX |     4 |       |     3   (0)| 00:00:01 |
|*  6 |     TABLE ACCESS FULL         | PARTS         |  2667 | 72009 |  1052   (1)| 00:00:13 |
|   7 |    TABLE ACCESS FULL          | LINEITEMS     |  6001K|    51M| 29675   (1)| 00:05:57 |
-----------------------------------------------------------------------------------------------

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

   2 - access("PS_PARTKEY"="L_PARTKEY" AND "PS_SUPPKEY"="L_SUPPKEY")
   3 - access("P_PARTKEY"="PS_PARTKEY")
       filter("PS_PARTKEY"=5 AND "P_TYPE"='MEDIUM ANODIZED BRASS' OR "PS_PARTKEY"=5 AND
              "P_TYPE"='MEDIUM BRUSHED COPPER')
```

Tabelle partsubs wird jetzt mit dem Index gescannt, Kosten von 35577 auf 31055 gesenkt.
Auf p_type wird gefiltert:

```
1 CREATE INDEX p_type_ix ON parts(p_type);
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                |     1 |    45 | 29885   (1)| 00:05:59 |
|   1 |  SORT AGGREGATE                        |                |     1 |    45 |            |          |
|*  2 |   HASH JOIN                            |                |     4 |   180 | 29885   (1)| 00:05:59 |
|   3 |    NESTED LOOPS                        |                |       |       |            |          |
|   4 |     NESTED LOOPS                       |                |     4 |   144 |   180   (0)| 00:00:03 |
|   5 |      TABLE ACCESS BY INDEX ROWID       | PARTSUPPS      |     4 |    36 |     4   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN                 | PS_PARTKEY_IX  |     4 |       |     3   (0)| 00:00:01 |
|   7 |      BITMAP CONVERSION TO ROWIDS       |                |       |       |            |          |
|   8 |       BITMAP AND                       |                |       |       |            |          |
|   9 |        BITMAP OR                       |                |       |       |            |          |
|  10 |         BITMAP CONVERSION FROM ROWIDS  |                |       |       |            |          |
|* 11 |          INDEX RANGE SCAN              | P_TYPE_IX      |       |       |     8   (0)| 00:00:01 |
|  12 |         BITMAP CONVERSION FROM ROWIDS  |                |       |       |            |          |
|* 13 |          INDEX RANGE SCAN              | P_TYPE_IX      |       |       |     8   (0)| 00:00:01 |
|  14 |        BITMAP OR                       |                |       |       |            |          |
|  15 |         BITMAP CONVERSION FROM ROWIDS  |                |       |       |            |          |
|* 16 |          INDEX RANGE SCAN              | P_TYPE_IX      |       |       |     8   (0)| 00:00:01 |
|  17 |         BITMAP CONVERSION FROM ROWIDS  |                |       |       |            |          |
|* 18 |          INDEX RANGE SCAN              | P_TYPE_IX      |       |       |     8   (0)| 00:00:01 |
|* 19 |     TABLE ACCESS BY INDEX ROWID        | PARTS          |     1 |    27 |   180   (0)| 00:00:03 |
|  20 |    TABLE ACCESS FULL                   | LINEITEMS      |  6001K|    51M| 29675   (1)| 00:05:57 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("PS_PARTKEY"="L_PARTKEY" AND "PS_SUPPKEY"="L_SUPPKEY")
   6 - access("PS_PARTKEY"=5)
  11 - access("P_TYPE"='MEDIUM ANODIZED BRASS')
  13 - access("P_TYPE"='MEDIUM BRUSHED COPPER')
  16 - access("P_TYPE"='MEDIUM ANODIZED BRASS')
  18 - access("P_TYPE"='MEDIUM BRUSHED COPPER')
  19 - filter("P_PARTKEY"="PS_PARTKEY" AND ("PS_PARTKEY"=5 AND "P_TYPE"='MEDIUM ANODIZED
```

Tabelle parts wird jetzt mit dem Index gescannt, Kosten von 31055 auf 29885 gesenkt.
Auf l_partkey_ix wird gejoined:

```
1 CREATE INDEX l_partkey_ix ON lineitems(l_partkey);
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                |     1 |    45 |   308   (0)| 00:00:04 |
|   1 |  SORT AGGREGATE                        |                |     1 |    45 |            |          |
|   2 |   NESTED LOOPS                         |                |       |       |            |          |
|   3 |    NESTED LOOPS                        |                |     4 |   180 |   308   (0)| 00:00:04 |
|   4 |     NESTED LOOPS                       |                |     4 |   144 |   180   (0)| 00:00:03 |
|   5 |      TABLE ACCESS BY INDEX ROWID       | PARTSUPPS      |     4 |    36 |     4   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN                 | PS_PARTKEY_IX  |     4 |       |     3   (0)| 00:00:01 |
|*  7 |      TABLE ACCESS BY INDEX ROWID       | PARTS          |     1 |    27 |   180   (0)| 00:00:03 |
|   8 |       BITMAP CONVERSION TO ROWIDS      |                |       |       |            |          |
|   9 |        BITMAP AND                      |                |       |       |            |          |
|  10 |         BITMAP OR                      |                |       |       |            |          |
|  11 |          BITMAP CONVERSION FROM ROWIDS |                |       |       |            |          |
|* 12 |           INDEX RANGE SCAN             | P_TYPE_IX      |       |       |     8   (0)| 00:00:01 |
|  13 |          BITMAP CONVERSION FROM ROWIDS |                |       |       |            |          |
|* 14 |           INDEX RANGE SCAN             | P_TYPE_IX      |       |       |     8   (0)| 00:00:01 |
|  15 |         BITMAP OR                      |                |       |       |            |          |
|  16 |          BITMAP CONVERSION FROM ROWIDS |                |       |       |            |          |
|* 17 |           INDEX RANGE SCAN             | P_TYPE_IX      |       |       |     8   (0)| 00:00:01 |
|  18 |          BITMAP CONVERSION FROM ROWIDS |                |       |       |            |          |
|* 19 |           INDEX RANGE SCAN             | P_TYPE_IX      |       |       |     8   (0)| 00:00:01 |
|* 20 |     INDEX RANGE SCAN                   | L_PARTKEY_IX   |    30 |       |     2   (0)| 00:00:01 |
|* 21 |    TABLE ACCESS BY INDEX ROWID         | LINEITEMS      |     1 |     9 |    32   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   6 - access("PS_PARTKEY"=5)
   7 - filter("P_PARTKEY"="PS_PARTKEY" AND ("PS_PARTKEY"=5 AND "P_TYPE"='MEDIUM ANODIZED
              BRASS' OR "PS_PARTKEY"=5 AND "P_TYPE"='MEDIUM BRUSHED COPPER'))
  12 - access("P_TYPE"='MEDIUM ANODIZED BRASS')
  14 - access("P_TYPE"='MEDIUM BRUSHED COPPER')
  17 - access("P_TYPE"='MEDIUM ANODIZED BRASS')
  19 - access("P_TYPE"='MEDIUM BRUSHED COPPER')
  20 - access("PS_PARTKEY"="L_PARTKEY")
  21 - filter("PS_SUPPKEY"="L_SUPPKEY")
```

Tabelle lineitems wird jetzt mit dem Index gescannt, Kosten von 29885 auf 308 gesenkt. Die Kosten wurden im gesamten von 35577 auf 308 gesenkt (Faktor 115.5).

7 Deep Left Join

```
SELECT *
FROM orders, lineitems, partsupps, parts
WHERE orders.o_orderkey = lineitems.l_orderkey
AND lineitems.l_suppkey = partsupps.ps_suppkey
AND partsupps.ps_partkey = parts.p_partkey;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |   482M|   229G|       |  9176K  (1)| 30:35:13 |
|*  1 |  HASH JOIN           |           |   482M|   229G|    27M|  9176K  (1)| 30:35:13 |
|   2 |   TABLE ACCESS FULL  | PARTS     |   200K|    25M|       |  1051   (1)| 00:00:13 |
|*  3 |   HASH JOIN          |           |   486M|   171G|   118M|   168K  (2)| 00:33:39 |
|   4 |    TABLE ACCESS FULL | PARTSUPPS |   800K|   109M|       |  4526   (1)| 00:00:55 |
|*  5 |    HASH JOIN         |           |  6086K|  1369M|   175M| 84027   (1)| 00:16:49 |
|   6 |     TABLE ACCESS FULL| ORDERS    |  1500K|   158M|       |  6610   (1)| 00:01:20 |
|   7 |     TABLE ACCESS FULL| LINEITEMS |  6001K|   715M|       | 29752   (1)| 00:05:58 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("PARTSUPPS"."PS_PARTKEY"="PARTS"."P_PARTKEY")
   3 - access("LINEITEMS"."L_SUPPKEY"="PARTSUPPS"."PS_SUPPKEY")
   5 - access("ORDERS"."O_ORDERKEY"="LINEITEMS"."L_ORDERKEY")
```

Nach einigem Suchen konnten wir anhand folgenden Statements einen Bushy Tree erzeugen:
(Quelle: http://dboptimizer.com/2011/12/09/right-deep-left-deep-and-bushy-joins/)

```
SELECT  *
FROM
(
SELECT /*+ no_merge */ *
FROM ORDERS, LIMEITEMS
WHERE ORDERS.o_orderkey = LINEITEMS.l_orderkey
)
,
(
SELECT /*+ no_merge */ *
FROM PARTSUPPS, PARTS
WHERE PARTSUPPS.ps_partkey = PARTS.p_partkey
)
WHERE l_suppkey = ps_suppkey;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |   482M|   286G|       |   211K  (2)| 00:42:23 |
|*  1 |  HASH JOIN           |           |   482M|   286G|   234M|   211K  (2)| 00:42:23 |
|   2 |   VIEW               |           |   792K|   225M|       | 12812   (1)| 00:02:34 |
|*  3 |    HASH JOIN         |           |   792K|   207M|    27M| 12812   (1)| 00:02:34 |
|   4 |     TABLE ACCESS FULL| PARTS     |   200K|    25M|       |  1051   (1)| 00:00:13 |
|   5 |     TABLE ACCESS FULL| PARTSUPPS |   800K|   109M|       |  4526   (1)| 00:00:55 |
|   6 |   VIEW               |           |  6086K|  1967M|       | 84027   (1)| 00:16:49 |
|*  7 |    HASH JOIN         |           |  6086K|  1369M|   175M| 84027   (1)| 00:16:49 |
|   8 |     TABLE ACCESS FULL| ORDERS    |  1500K|   158M|       |  6610   (1)| 00:01:20 |
|   9 |     TABLE ACCESS FULL| LINEITEMS |  6001K|   715M|       | 29752   (1)| 00:05:58 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("L_SUPPKEY"="PS_SUPPKEY")
   3 - access("PARTSUPPS"."PS_PARTKEY"="PARTS"."P_PARTKEY")
   7 - access("ORDERS"."O_ORDERKEY"="LINEITEMS"."L_ORDERKEY")
```

Reflexion
Beim Vergleichen der beiden Ausführungspläne erkennt man schnell, dass die erste Version eines Deep Left Joins sehr hohe Kosten mit sich bringt. Diese werden v.A. durch das mehrmalige Joinen derselben Tabellen verursacht.
Im zweiten Statement wird durch das Verwenden von Hints eine Reihenfolge erzwingt, in der die Tabellen gejoined werden sollen und das Mergen wird unterbunden.
Durch den Ausführungsplan sieht man schnell, dass so ein Bushy Tree erstellt wird. Ausserdem merkt man, dass die Kosten um ein Vielfaches geringer sind als beim ersten Statement. (Um den Faktor 43)

Um die Anfrage weiter zu Beschleunigen haben wir folgende Indizes erstellt:

```
CREATE INDEX o_orderkey_ix ON orders(o_orderkey);
```

```
CREATE INDEX l_orderkey_ix ON lineitems(l_orderkey);
```

```
CREATE INDEX l_suppkey_ix ON lineitems(l_suppkey);
```

```
CREATE INDEX ps_suppkey_ix ON partsupps(ps_suppkey);
```

```
CREATE INDEX ps_partkey_ix ON partsupps(ps_partkey);
```

```
CREATE INDEX p_partkey_ix ON parts(p_partkey);
```

Deep Left Join:

```
-------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |   482M|   229G|       |  9176K  (1)| 30:35:13 |
|*  1 |  HASH JOIN           |           |   482M|   229G|    27M|  9176K  (1)| 30:35:13 |
|   2 |   TABLE ACCESS FULL  | PARTS     |   200K|    25M|       |  1051   (1)| 00:00:13 |
|*  3 |   HASH JOIN          |           |   486M|   171G|   118M|   168K  (2)| 00:33:39 |
|   4 |    TABLE ACCESS FULL | PARTSUPPS |   800K|   109M|       |  4526   (1)| 00:00:55 |
|*  5 |    HASH JOIN         |           |  6086K|  1369M|   175M| 84027   (1)| 00:16:49 |
|   6 |     TABLE ACCESS FULL| ORDERS    |  1500K|   158M|       |  6610   (1)| 00:01:20 |
|   7 |     TABLE ACCESS FULL| LINEITEMS |  6001K|   715M|       | 29752   (1)| 00:05:58 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("PARTSUPPS"."PS_PARTKEY"="PARTS"."P_PARTKEY")
   3 - access("LINEITEMS"."L_SUPPKEY"="PARTSUPPS"."PS_SUPPKEY")
   5 - access("ORDERS"."O_ORDERKEY"="LINEITEMS"."L_ORDERKEY")
```

Bushy Tree:

```
-------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |   482M|   286G|       |   211K  (2)| 00:42:23 |
|*  1 |  HASH JOIN           |           |   482M|   286G|   234M|   211K  (2)| 00:42:23 |
|   2 |   VIEW               |           |   792K|   225M|       | 12812   (1)| 00:02:34 |
|*  3 |    HASH JOIN         |           |   792K|   207M|    27M| 12812   (1)| 00:02:34 |
|   4 |     TABLE ACCESS FULL| PARTS     |   200K|    25M|       |  1051   (1)| 00:00:13 |
|   5 |     TABLE ACCESS FULL| PARTSUPPS |   800K|   109M|       |  4526   (1)| 00:00:55 |
|   6 |   VIEW               |           |  6086K|  1967M|       | 84027   (1)| 00:16:49 |
|*  7 |    HASH JOIN         |           |  6086K|  1369M|   175M| 84027   (1)| 00:16:49 |
|   8 |     TABLE ACCESS FULL| ORDERS    |  1500K|   158M|       |  6610   (1)| 00:01:20 |
|   9 |     TABLE ACCESS FULL| LINEITEMS |  6001K|   715M|       | 29752   (1)| 00:05:58 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("L_SUPPKEY"="PS_SUPPKEY")
   3 - access("PARTSUPPS"."PS_PARTKEY"="PARTS"."P_PARTKEY")
   7 - access("ORDERS"."O_ORDERKEY"="LINEITEMS"."L_ORDERKEY")
```

Reflexion
Wir haben festgestellt, dass der Oracle Optimizer die Verwendung von Indizes sowohl beim Bushy Tree, als auch beim Left Deep Join unterbunden hat, weswegen die Performance nicht weiter verbessert werden konnte.

8 Eigene SQL-Abfragen

```
1 EXPLAIN PLAN FOR
2 SELECT *
3 FROM lineitems, orders
4 WHERE l_orderkey=o_orderkey
5 AND o_orderkey BETWEEN 100 AND  1000;
```

Ergibt ohne Index folgenden Aufsführungsplan:

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |   227 | 53572 | 36282   (1)| 00:07:16 |
|*  1 |  HASH JOIN         |           |   227 | 53572 | 36282   (1)| 00:07:16 |
|*  2 |   TABLE ACCESS FULL| ORDERS    |   227 | 25197 |  6602   (1)| 00:01:20 |
|*  3 |   TABLE ACCESS FULL| LINEITEMS |   908 |   110K| 29679   (1)| 00:05:57 |
-------------------------------------------------------------------------------

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

   1 - access("L_ORDERKEY"="O_ORDERKEY")
   2 - filter("O_ORDERKEY"<=1000 AND "O_ORDERKEY">=100)
   3 - filter("L_ORDERKEY"<=1000 AND "L_ORDERKEY">=100)
```

Nun setzten wir je einen Index auf o_orderkey und l_orderkey.

```
1 CREATE INDEX l_orderkey_ix ON lineitems(l_orderkey);
2 CREATE INDEX o_orderkey_ix ON orders(o_orderkey);
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |   227 | 53572 |    51   (2)| 00:00:01 |
|*  1 |  HASH JOIN                    |               |   227 | 53572 |    51   (2)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| ORDERS        |   227 | 25197 |    11   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN           | O_ORDERKEY_IX |   227 |       |     3   (0)| 00:00:01 |
|   4 |   TABLE ACCESS BY INDEX ROWID| LINEITEMS     |   908 |   110K|    39   (0)| 00:00:01 |
|*  5 |    INDEX RANGE SCAN           | L_ORDERKEY_IX |   908 |       |     5   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   1 - access("L_ORDERKEY"="O_ORDERKEY")
   3 - access("O_ORDERKEY">=100 AND "O_ORDERKEY"<=1000)
   5 - access("L_ORDERKEY">=100 AND "L_ORDERKEY"<=1000)
```

Durch Verwendung der Indexes konnten die Kost von 36282 auf 51 gesenkt weden (Faktor 711).