



THE AUSTRALIAN ORACLE USER GROUP IN ASSOCIATION WITH INTEL PRESENTS

# master class series

CUT TO THE CHASE USING TIPS AND PRACTICAL EXAMPLES FROM EXPERT USERS



FOR DEVELOPERS



FOR DATABASE ADMINISTRATORS

## SQL Optimisation

**SAGE Computing Services**

**Customised Oracle Training Workshops  
and Consulting**

**[www.sagecomputing.com.au](http://www.sagecomputing.com.au)**

**Penny Cookson - Managing Director**

**[www.sagecomputing.com.au](http://www.sagecomputing.com.au)**

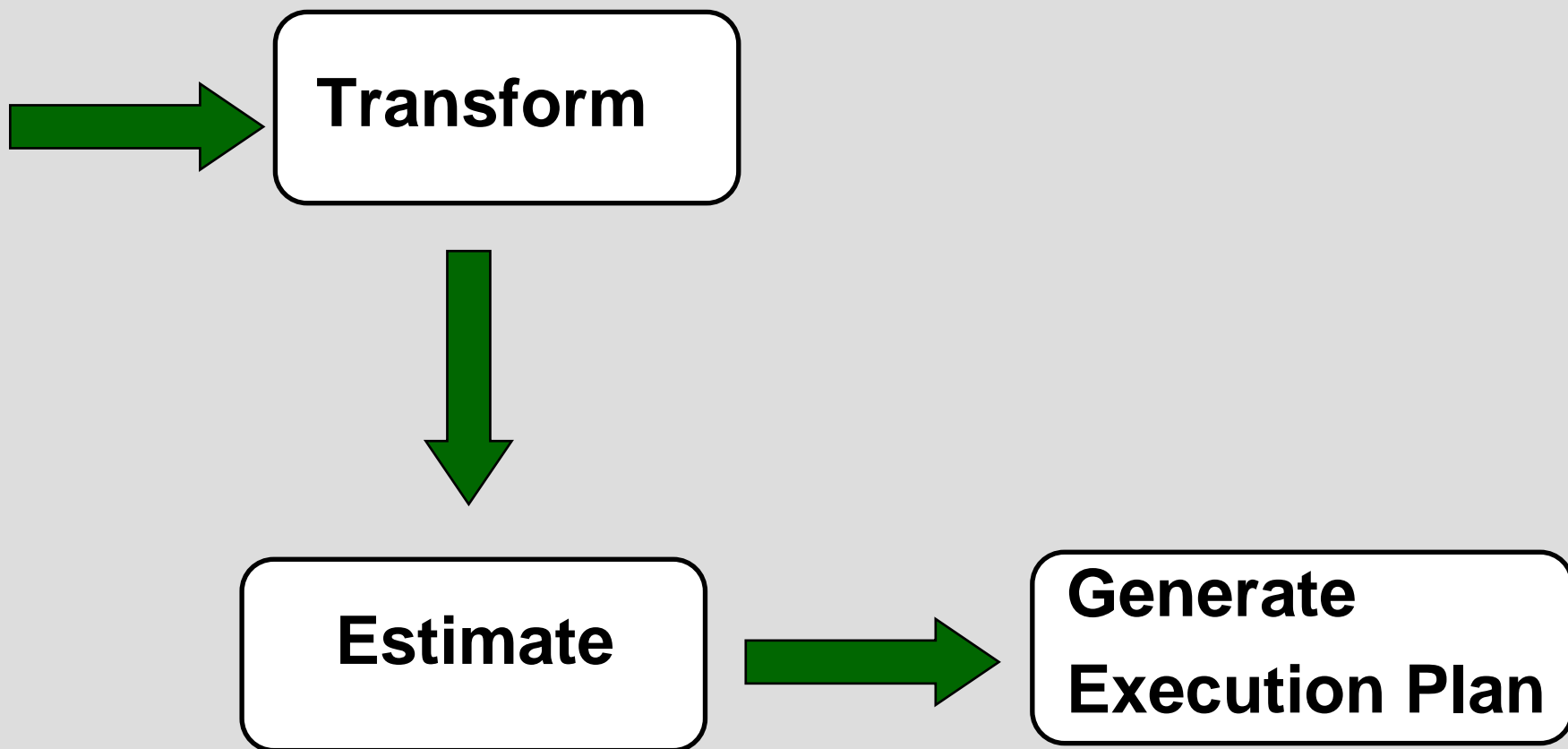
- **How does the optimizer work?**
- **Performance enhancements in version 10g**
- **System Statistics**
- **Recommendations on gathering statistics**
- **How bind variables affect SQL tuning**
- **Tuning Tools – finding out what it did and why**
- **Hints and when to use them**
- **Tuning – making it do what we want it to**
- **Common SQL tuning issues and how to resolve them**

**How does the optimizer work?**

# History of the Optimiser

<u>Version</u>	<u>Feature</u>
5	Rule based
6	Rule based
7	Rule or Cost but cost is pretty dodgy
8i	Rule or Cost and its OK to use cost
9i	Rule or Cost and its OK to use cost I/O and CPU Costing
10g	Cost only – System statistics

# What does the optimizer do?



# Transform

## ➡ Based on rules

View merging

Unnest Subqueries

IN to OR

NOT < to >=

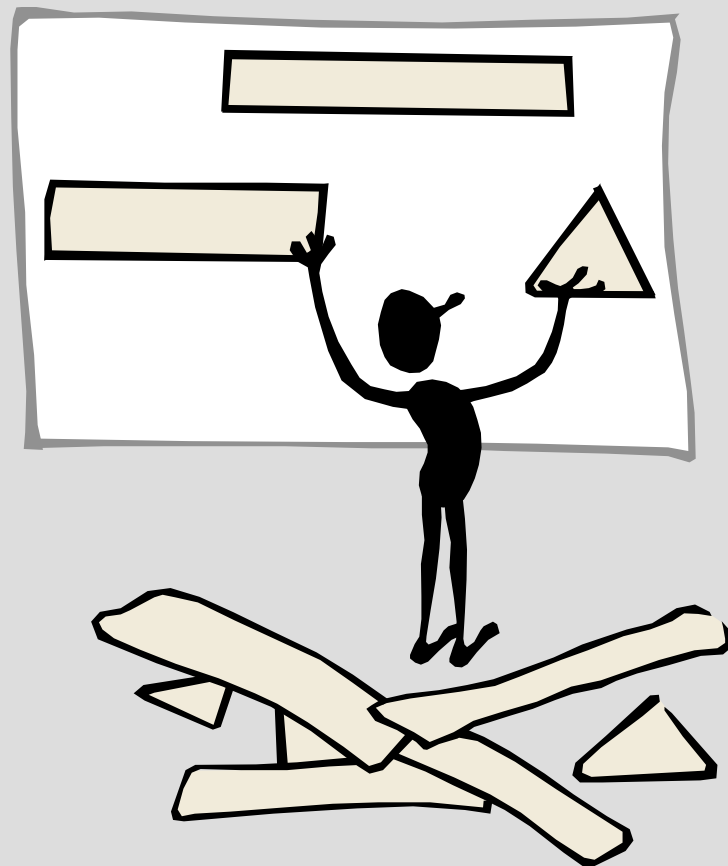
BETWEEN to Range Checks

## ➡ Based on reducing cost

OR to UNION

Star transformation

Materialized view rewrite



# Transitivity

- ➡ The optimiser can utilise transitivity to infer additional information

**WHERE      a.columnA = 'X'**  
**AND         b.columnB = a.columnA**

- ➡ can be used to infer the conditions :

**WHERE      a.columnA = 'X'**  
**AND         b.columnB = a.columnA**  
**AND         b.columnB = 'X'**

# Transformations

EXPLAIN PLAN FOR  
SELECT booking\_no  
FROM bookings  
WHERE cost < (100 + 54) / 2;



PLAN\_TABLE\_OUTPUT

-----  
1 - filter("COST"<77)

EXPLAIN PLAN FOR  
SELECT \*  
FROM organisations  
WHERE state IN ('WA','VIC');



PLAN\_TABLE\_OUTPUT

-----  
1 - filter("STATE"='VIC' OR  
"STATE"='WA')

EXPLAIN PLAN FOR  
SELECT resource\_code  
FROM bookings  
WHERE cost BETWEEN 10 and 100



PLAN\_TABLE\_OUTPUT

-----  
1 - filter("COST"<=100 AND  
"COST">=10)

EXPLAIN PLAN FOR  
SELECT resource\_code  
FROM bookings  
WHERE NOT cost < 100;



PLAN\_TABLE\_OUTPUT

-----  
1 - filter("COST">=100)



# Transformations - Subqueries

EXPLAIN PLAN FOR

```
SELECT r.description, b.cost
FROM   bookings b, resources r
WHERE  r.code = b.resource_code
AND    b.event_no IN
      (SELECT e.event_no
       FROM   events e
       WHERE  start_date =
            TO_DATE('01-JAN-1995','dd-mon-yyyy'));
```

Predicate Information (identified by operation id):

-----

- 1 - access("R"."CODE"="B"."RESOURCE\_CODE")
- 4 - filter("START\_DATE"=TO\_DATE('1995-01-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))
- 5 - access("B"."EVENT\_NO"="E"."EVENT\_NO")

```
SELECT r.description, b.cost
FROM   bookings b, resources r, events e
WHERE  r.code = b.resource_code
AND    b.event_no = e.event_no
AND    e.start_date = TO_DATE('01-JAN-1995','dd-mon-yyyy');
```

# Transformations - Subqueries

## EXPLAIN PLAN FOR

```
SELECT  r.description, b.cost
FROM    bookings b, resources r
WHERE   r.code = b.resource_code
AND     b.cost >
        (SELECT avg(b1.cost)
         FROM    bookings b1
         WHERE   b1.resource_code =
                 b.resource_code);
```

## Predicate Information (identified by operation id):

```
-----
2 - access("RESOURCE_CODE"=
           "B"."RESOURCE_CODE")
    filter("B"."COST">"VW_COL_1")
9 - access("R"."CODE"="B"."RESOURCE_CODE")
```



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	45	6 (17)	00:00:01
1	NESTED LOOPS		1	45	6 (17)	00:00:01
* 2	HASH JOIN		1	25	5 (20)	00:00:01
3	VIEW	VW_SQ_1	9	153	2 (0)	00:00:01
4	SORT GROUP BY		9	72	2 (0)	00:00:01
5	TABLE ACCESS BY INDEX ROWID	BOOKINGS	22	176	2 (0)	00:00:01
6	INDEX FULL SCAN	BK_RES	22		1 (0)	00:00:01
7	TABLE ACCESS FULL	BOOKINGS	22	176	2 (0)	00:00:01
8	TABLE ACCESS BY INDEX ROWID	RESOURCES	1	20	1 (0)	00:00:01
* 9	INDEX UNIQUE SCAN	RES_PK	1		0 (0)	00:00:01

# Transformations - OR

EXPLAIN PLAN FOR

```
SELECT b.booking_no, b.cost, e.comments
FROM bookings_large b, events_large e
WHERE b.event_no = e.event_no
AND (e.org_id = 1000
OR b.resource_code = 'VCR1')
```

```
SELECT b.booking_no, b.cost, e.comments
FROM bookings_large b, events_large e
WHERE b.event_no = e.event_no
AND e.org_id = 1000
UNION
SELECT b.booking_no, b.cost, e.comments
FROM bookings_large b, events_large e
WHERE b.event_no = e.event_no
AND b.resource_code = 'VCR1'
```

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time
0	SELECT STATEMENT		6784	212K	7269	(1)	00:01:28
1	CONCATENATION						
2	NESTED LOOPS		6478	202K	6684	(1)	00:01:21
3	TABLE ACCESS BY INDEX ROWID	BOOKINGS_LARGE	6478	107K	196	(0)	00:00:03
* 4	INDEX RANGE SCAN	EVT_RES	6478		15	(0)	00:00:01
5	TABLE ACCESS BY INDEX ROWID	EVENTS_LARGE	1	15	1	(0)	00:00:01
* 6	INDEX UNIQUE SCAN	EVTLG_PK	1		0	(0)	00:00:01
7	NESTED LOOPS		306	9792	585	(1)	00:00:08
* 8	TABLE ACCESS FULL	BOOKINGS_LARGE	563	9571	22	(5)	00:00:01
* 9	TABLE ACCESS BY INDEX ROWID	EVENTS_LARGE	1	15	1	(0)	00:00:01
* 10	INDEX UNIQUE SCAN	EVTLG_PK	1		0	(0)	00:00:01

# Prevent OR Transformation

EXPLAIN PLAN FOR

```
SELECT --+ NO_EXPAND
      b.booking_no, b.cost, e.comments
FROM   bookings_large b, events_large e
WHERE  b.event_no = e.event_no
AND    (e.org_id = 1000
OR      b.resource_code = 'VCR1')
```

Predicate Information (identified by operation id):

-----  
3 - filter("E"."ORG\_ID"=1000 OR  
          "B"."RESOURCE\_CODE"='VCR1')  
4 - access("B"."EVENT\_NO"="E"."EVENT\_NO")



Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time
0	SELECT STATEMENT		7276	227K	11301	(1)	00:02:16
1	NESTED LOOPS		7276	227K	11301	(1)	00:02:16
2	TABLE ACCESS FULL	BOOKINGS_LARGE	11264	187K	21	(0)	00:00:01
* 3	TABLE ACCESS BY INDEX ROWID	EVENTS_LARGE	1	15	1	(0)	00:00:01
* 4	INDEX UNIQUE SCAN	EVTLG_PK	1		0	(0)	00:00:01

# Force OR Transformation

EXPLAIN PLAN FOR

```
SELECT booking_no, cost, comments
FROM   bookings_large
WHERE  event_no = 1000
OR      booking_no = 1000
```



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		9	189	4 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	BOOKINGS_LARGE	9	189	4 (0)	00:00:01
2	BITMAP CONVERSION TO ROWIDS					
3	BITMAP OR					
4	BITMAP CONVERSION FROM ROWIDS					
* 5	INDEX RANGE SCAN	BK_EVTS			1 (0)	00:00:01
6	BITMAP CONVERSION FROM ROWIDS					
* 7	INDEX RANGE SCAN	BOOKLG_PK			1 (0)	00:00:01

# Force OR Transformation

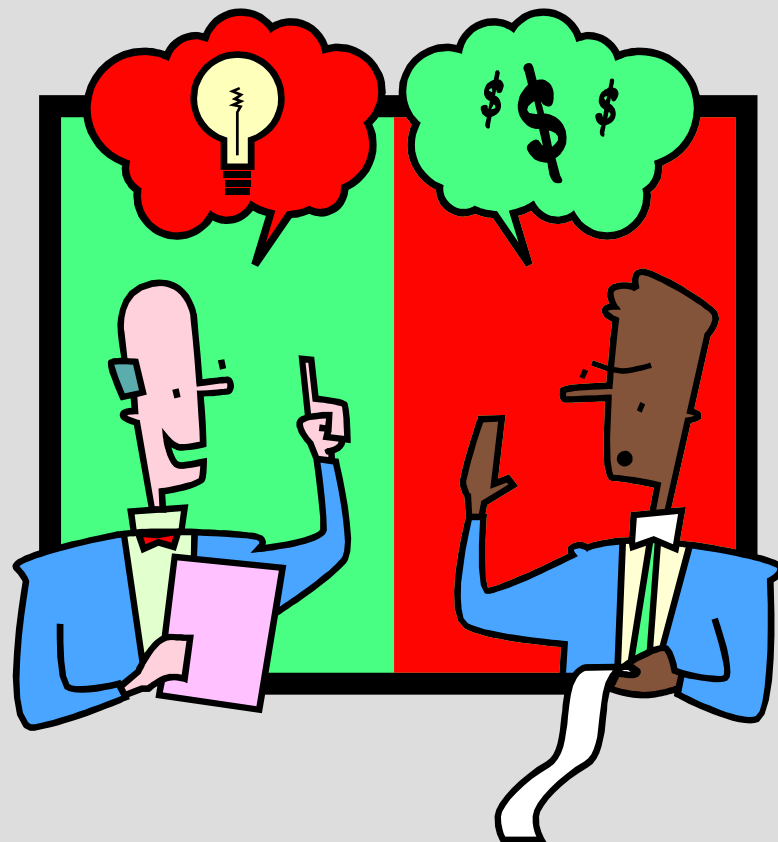
```
EXPLAIN PLAN FOR
SELECT  --+ USE_CONCAT
        booking_no, cost, comments
FROM    bookings_large
WHERE   event_no = 1000
OR      booking_no = 1000
```



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		2	42	12 (0)	00:00:01
1	CONCATENATION					
2	TABLE ACCESS BY INDEX ROWID	BOOKINGS_LARGE	1	21	2 (0)	00:00:01
* 3	INDEX UNIQUE SCAN	BOOKLG_PK	1		1 (0)	00:00:01
* 4	TABLE ACCESS BY INDEX ROWID	BOOKINGS_LARGE	1	21	10 (0)	00:00:01
* 5	INDEX RANGE SCAN	BK_EVTS	8		1 (0)	00:00:01

# Estimate

- ➡ **Single table access paths**
- ➡ **Join order and mechanism**
- ➡ **Selectivity**
- ➡ **Cardinality**
- ➡ **Cost**





# Single Table Access

- ➡ Full table scan
- ➡ Sample table scan
- ➡ Partition scan
- ➡ Index scan
  - Unique
  - Range
  - Fast Full scan
  - Full scan
  - Skip scan
  - Index join
- ➡ Access using rowid
- ➡ Hash scan
- ➡ Index cluster scan





# Single Table Access

## SINGLE TABLE ACCESS PATH

**COLUMN:** ORG\_ID(NUMBER) Col#: 2 Table: EVENTS\_LARGE Alias: E

Size: 4 NDV: 6 Nulls: 0 Density: 1.6667e-001 Min: 1000 Max: 3843

No Histogram: #BKT: 1

(1 uncompressed buckets and 2 endpoint values)

**TABLE:** EVENTS\_LARGE Alias: E

Original Card: 1536 Rounded Card: 256 Computed Card: 256.00

Access Path: table-scan Resc: 5 Resp: 5

Access Path: index (equal)

Index: EVT\_ORGN

rsc\_cpu: 215784 rsc\_io: 16

ix\_sel: 0.0000e+000 ix\_sel\_with\_filters: 1.6667e-001

Access Path: index (equal)

Index: EVT\_ORGN

rsc\_cpu: 65443 rsc\_io: 1

ix\_sel: 1.6667e-001 ix\_sel\_with\_filters: 1.6667e-001

Access Path: index (no start/stop keys)

Index: EVTLG\_PK

rsc\_cpu: 335686 rsc\_io: 4

ix\_sel: 1.0000e+000 ix\_sel\_with\_filters: 1.0000e+000

**BEST\_CST:** 5.04 **PATH:** 2 **Degree:** 1

# Full Table Scan

- ➡ Uses number of used blocks
- ➡ Uses System Statistic MBRC rather than DB\_FILE\_MULTIBLOCK\_READ\_COUNT

```
BASE STATISTICAL INFORMATION
*****
Table stats      Table: BOOKINGS_LARGE3  Alias: BOOKINGS_LARGE3
TOTAL :: CDN: 10270  NBLKS: 82  AVG_ROW_LEN: 42
_OPTIMIZER_PERCENT_PARALLEL = 0
*****
SINGLE TABLE ACCESS PATH
TABLE: BOOKINGS_LARGE3  Alias: BOOKINGS_LARGE3
Original card: 10270  Rounded Card: 10270  Computed Card: 10270.00
Access Path: table-scan  Resc: 20  Resp: 20
BEST_CST: 20.18  PATH: 2  Degree: 1
=====
```

# Cardinality

► **Column = Value (No histogram)**

Number of rows expected (cardinality) =  
(Cardinality of table – Number of null rows) \* Density of column

**Example:**

**SELECT \* FROM ORGANISATIONS WHERE STATE = 'WA'**

$$3.33 = (12 - 2) * 3.3333e-001$$

SINGLE TABLE ACCESS PATH

COLUMN: STATE (VARCHAR2) Col#: 7 Table: ORGANISATIONS Alias: ORGANISATIONS

Size: 4 NDV: 3 Nulls: 2 Density: 3.3333e-001

No Histogram: #BKT: 1

(1 uncompressed buckets and 2 endpoint values)

TABLE: ORGANISATIONS Alias: ORGANISATIONS

Original Card: 12 Rounded Card: 3 Computed Card: 3.33

# Cardinality

## ► Column > Value (No histogram)

Number of rows expected (cardinality) =  
 (Cardinality of table – Number of null rows)  
 \* (Max – Value) / Max – Min)

### Example:

**SELECT \* FROM bookings\_large WHERE booking\_no > 11000**

$$263.88 = (11264 - 6) * \\ (11264 - 11000) / (11264 - 1)$$


```
COLUMN: BOOKING_NO(NUMBER)  Col#: 1      Table: BOOKINGS_LARGE  Alias: BOOKINGS_LARGE
Size: 5  NDV: 11258  Nulls: 6  Density: 8.8826e-005  Min: 1  Max: 11264
No Histogram: #BKT: 1
(1 uncompressed buckets and 2 endpoint values)
TABLE: BOOKINGS_LARGE  Alias: BOOKINGS_LARGE
Original Card: 11264  Rounded Card: 264  Computed Card: 263.88|
```

# Cardinality

## ➡ Column $\geq$ Value (No histogram)

Number of rows expected (cardinality) =

$(\text{Cardinality of table} - \text{Number of null rows}) * (\text{Max} - \text{Value}) / \text{Max} - \text{Min}$   
 $+ (\text{Cardinality of table} - \text{Number of null rows}) / \text{num distinct values}$

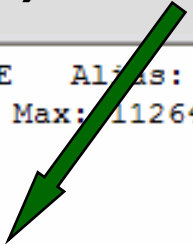
**Example:**

**SELECT \* FROM bookings\_large WHERE booking\_no  $\geq$  11000**

$$264.88 = ((11264 - 6) * (11264 - 11000) / (11264 - 1)) + (11264 - 6) / 11258$$

```

COLUMN: BOOKING_NO(NUMBER)  Col#: 1      Table: BOOKINGS_LARGE  Alias: BOOKINGS_LARGE
Size: 5  NDV: 11258  Nulls: 6  Density: 8.8826e-005  Min: 1  Max: 11264
No Histogram: #BKT: 1
(1 uncompressed buckets and 2 endpoint values)
TABLE: BOOKINGS_LARGE  Alias: BOOKINGS_LARGE
Original Card: 11264  Rounded Card: 265  Computed Card: 264.88
    
```



# Generate Plan - Join Order

## Example:

```
SELECT b.resource_code, b.cost
FROM  events e, bookings b, resources r, resource_types t
WHERE e.event_no = b.event_no AND b.resource_code = r.code
AND r.type_code = t.code AND e.org_id = 1000
```

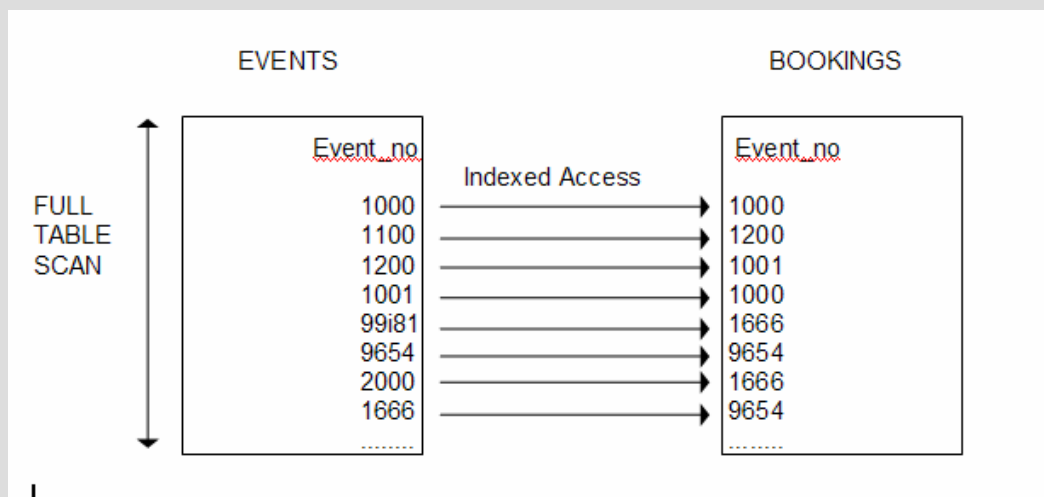
```
Join order [1]:  EVENTS[E]#0  RESOURCE_TYPES[T]#1  RESOURCES[R]#2  BOOKINGS[B]#3
Join order [2]:  EVENTS[E]#0  RESOURCE_TYPES[T]#1  BOOKINGS[B]#3  RESOURCES[R]#2
Join order [3]:  EVENTS[E]#0  RESOURCES[R]#2  RESOURCE_TYPES[T]#1  BOOKINGS[B]#3
Join order [4]:  EVENTS[E]#0  RESOURCES[R]#2  BOOKINGS[B]#3  RESOURCE_TYPES[T]#1
Join order [6]:  EVENTS[E]#0  BOOKINGS[B]#3  RESOURCES[R]#2  RESOURCE_TYPES[T]#1
Join order [5]:  EVENTS[E]#0  BOOKINGS[B]#3  RESOURCE_TYPES[T]#1  RESOURCES[R]#2
Join order [7]:  RESOURCE_TYPES[T]#1  EVENTS[E]#0  RESOURCES[R]#2  BOOKINGS[B]#3
Join order [8]:  RESOURCE_TYPES[T]#1  EVENTS[E]#0  BOOKINGS[B]#3  RESOURCES[R]#2
Join order [9]:  RESOURCE_TYPES[T]#1  RESOURCES[R]#2  EVENTS[E]#0  BOOKINGS[B]#3
Join order [10]: RESOURCE_TYPES[T]#1  RESOURCES[R]#2  BOOKINGS[B]#3  EVENTS[E]#0
Join order [11]: RESOURCE_TYPES[T]#1  BOOKINGS[B]#3  EVENTS[E]#0  RESOURCES[R]#2
Join order [12]: RESOURCE_TYPES[T]#1  BOOKINGS[B]#3  RESOURCES[R]#2  EVENTS[E]#0
Join order [13]: RESOURCES[R]#2  EVENTS[E]#0  RESOURCE_TYPES[T]#1  BOOKINGS[B]#3
Join order [14]: RESOURCES[R]#2  RESOURCE_TYPES[T]#1  EVENTS[E]#0  BOOKINGS[B]#3
Join order [15]: RESOURCES[R]#2  RESOURCE_TYPES[T]#1  BOOKINGS[B]#3  EVENTS[E]#0
Join order [16]: RESOURCES[R]#2  BOOKINGS[B]#3  EVENTS[E]#0  RESOURCE_TYPES[T]#1
Join order [17]: RESOURCES[R]#2  BOOKINGS[B]#3  RESOURCE_TYPES[T]#1  EVENTS[E]#0
Join order [18]: BOOKINGS[B]#3  EVENTS[E]#0  RESOURCE_TYPES[T]#1  RESOURCES[R]#2
Join order [19]: BOOKINGS[B]#3  EVENTS[E]#0  RESOURCES[R]#2  RESOURCE_TYPES[T]#1
Join order [20]: BOOKINGS[B]#3  RESOURCE_TYPES[T]#1  EVENTS[E]#0  RESOURCES[R]#2
Join order [21]: BOOKINGS[B]#3  RESOURCES[R]#2  EVENTS[E]#0  RESOURCE_TYPES[T]#1
Join order [22]: BOOKINGS[B]#3  RESOURCES[R]#2  RESOURCE_TYPES[T]#1  EVENTS[E]#0
```

```
Final - All Rows Plan:
JOIN ORDER: 6
```

# Join Mechanism

## ➤ Nested Loop

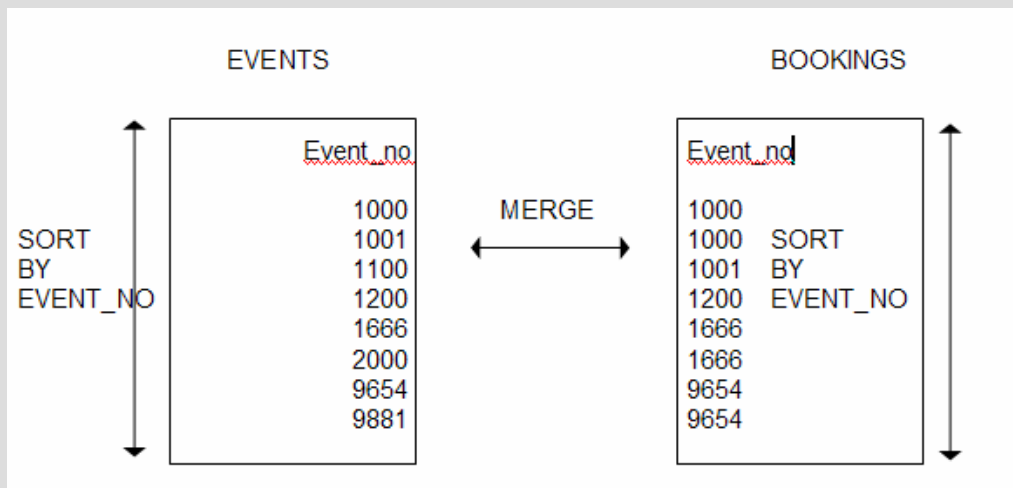
- At least one side of the join condition is indexed
- One table is designated as the driving (outer) table
- The second table (inner table) is accessed for each row in the driving table
- Cost = Cardinality of outer table X accessing inner table



# Join Mechanism

## ➡ Sort Merge

- Qualifying rows in each table are sorted according to the join column or columns
- A merge of the two sets of rows is carried out
- Cost = table access of inner and outer + sort of inner and outer + merge

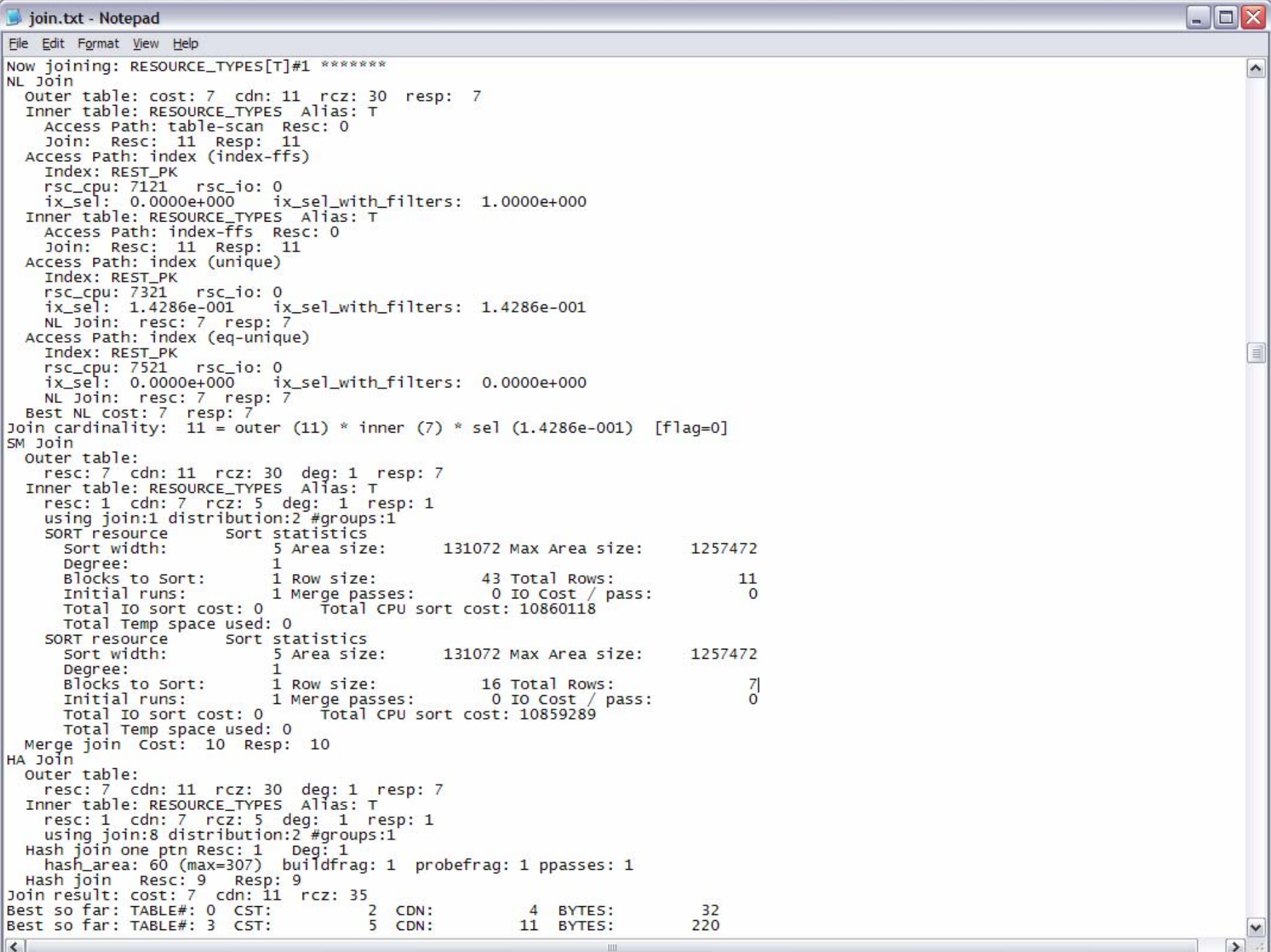




# Join Mechanism

## ➡ Hash Join

- Access rows from outer table
- Inner table used to create a hash table
- Inner table probed from outer table
- Cost = table access of inner and outer + hash table creation
- Avoids the sort associated with sort merge joins



# When Will it Make the Wrong Decision

- ➡ It does not have accurate statistics
- ➡ It does not have enough statistics
  - ➡ Indexes
  - ➡ histograms
- ➡ It does not have up to date system statistics
- ➡ The optimiser mode or other initialisation parameters are wrong
- ➡ Bind variables are hiding the cardinality
- ➡ There are complex data dependencies it does not know about

# **Performance Enhancements in 10g**



# New Performance Features

- ➡ **Optimizer mode**
- ➡ **Statistics gathering**
- ➡ **New optimizer behaviour**
- ➡ **Explain plan**
- ➡ **Outlines**
- ➡ **Trcsess utility**
- ➡ **New hints**
- ➡ **New DUAL access**
- ➡ **New tuning tools**



# No Rule Based Optimiser

## ➡ OPTIMIZER\_MODE

- ALL\_ROWS

Throughput

- FIRST\_ROWS

Response

- FIRST\_ROWS\_N

Response + I hate full scans

# Statistics Gathering

- ➡ Number of INSERT, UPDATE, and DELETE operations recorded
- ➡ `SELECT * FROM DBA_TAB_MODIFICATIONS`
- ➡ Automatic statistics gathered on STALE objects
  - Tables and indexes
  - Job Name = `GATHER_STATS_JOB`
  - Program =  
`DBMS_STATS.GATHER_DATABASE_STATS_JOB_PROC`
  - Schedule = `MAINTENANCE_WINDOW_GROUP`
  - Nights and weekend
- ➡ Can be disabled using `EXECUTE DBMS_SCHEDULER.DISABLE('GATHER_STATS_JOB');`

# Statistics Management

- ➡ **Statistics history**
- ➡ **Stored by default for a period of 31 days**
- ➡ **If statistics\_level is TYPICAL or ALL they are purged automatically**

**DBA\_OPTSTAT\_OPERATIONS**

**USER\_TAB\_STATS\_HISTORY**

**DBMS\_STATS.ALTER\_STATS\_HISTORY\_RETENTION**

**DBMS\_STATS.PURGE\_STATS**

**DBMS\_STATS.RESTORE**

**DBMS\_STATS.GET\_STATS\_HISTORY\_AVAILABILITY**



# Statistics Management

## ➤ Ability to lock statistics

- `DBMS_STATS.LOCK_SCHEMA_STATS`
- `DBMS_STATS.LOCK_TABLE_STATS`

## ➤ Statistics can be unlocked

- `DBMS_STATS.UNLOCK_SCHEMA_STATS`
- `DBMS_STATS.UNLOCK_TABLE_STATS`

## ➤ Dynamic sampling

- For very dynamic data or missing statistics
- `OPTIMIZER_DYNAMIC_SAMPLING`
- `--+ DYNAMIC_SAMPLING(tablealias level)`
- Levels 0 to 10



# New Optimizer Behaviour

- **New costing model**
  - **CPU + IO**
  - **Uses MBRC system parameter for full scans**
  - **Statistics with or without workload**
- **Optimizer can run in tuning mode**
- **SQL profiles contain additional statistics**
- **SQL Access Advisor recommends indexes and materialised views**



# Explain Plan

- ➡ **Global temporary table SYS.PLAN\_TABLE\$ can be used**
- ➡ **New Columns**

**PROJECTION**                      Expressions created by the operation

**TIME**                                The estimated elapsed time in seconds

**QBLOCK\_NAME**                    The name of a query block.  
This can be provided by a hint or system generated

# Public v Private Outlines

- ➡ **Public outlines are stored in the tables owned by the OUTLN user**
- ➡ **Public Outline tables are available for use by all users of the database**
- ➡ **Private outlines are stored in a separate set of global temporary tables located in the system schema**
- ➡ **They only persist for the user's session**

# Trcess

➡ End to end tracing

➡ trcssess used to consolidate a number of trace files

trcssess            output=output\_file  
                     session=session\_id            e.g. 152.3 This is the SID and  
   SERIAL# from v\$session  
  
                     clientid=client\_id  
                     service=service\_name  
                     action=action\_name  
                     module=module\_name  
                     trace\_files                    the \* wildcard can be used

➡ DBMS\_MONITOR package can be used to gather statistics for a specific Client ID, Service or service, module and action

➡ DBMS\_APPLICATION\_INFO package can be used to set the module and action from within an application

# New Hints

- ➡ **Specify index with name or column**  
--+ INDEX(org (org\_id name))
- ➡ **Use nested loop with a specific index**  
--+ USE\_NL\_WITH\_INDEX (alias (indexname))
- ➡ **Name a query block**  
--+ QB\_NAME (q1).
- ➡ **Do not perform query transformation**  
--+ NO\_QUERY\_TRANSFORMATION
- ➡ **Perform index skip scan**  
--+ INDEX\_SS



# New/Changed Hints

## ➡ Prevent operations

NO\_USE\_NL  
NO\_USE\_MERGE  
NO\_USE\_HASH  
NO\_INDEX\_FFS  
NO\_INDEX\_SS

## ➡ Deprecated Hints

AND\_EQUAL  
HASH\_AJ  
MERGE\_AJ  
HASH\_SJ  
MERGE\_SJ  
STAR  
ORDERED\_PREDICATES

## ➡ Renamed Hints

New	Old
NO_PARALLEL	NOPARALLEL
NO_PARALLEL_INDEX	NOPARALLEL_INDEX
NO_REWRITE	NOREWRITE

# Access DUAL V9

```
ORA92>SELECT booking_seq.nextval FROM dual;
```

```
NEXTVAL
```

```
-----  
11283
```

## Execution Plan

```
-----  
0  SELECT STATEMENT Optimizer=ALL_ROWS (Cost=2 Card=1)  
1  0  SEQUENCE OF 'BOOKING_SEQ'  
2  1  TABLE ACCESS (FULL) OF 'DUAL' (Cost=2 Card=1)
```

## Statistics

```
-----  
0 recursive calls  
0 db block gets  
3 consistent gets  
0 physical reads  
0 redo size  
380 bytes sent via SQL*Net to client  
499 bytes received via SQL*Net from client  
2 SQL*Net roundtrips to/from client  
0 sorts (memory)  
0 sorts (disk)  
1 rows processed
```



# FAST DUAL Operation

```
ORA10G>SELECT booking_seq.nextval FROM dual;
```

```
NEXTVAL
```

```
-----  
11282
```

```
Execution Plan
```

```
-----  
0   SELECT STATEMENT Optimizer=ALL_ROWS (Cost=2 Card=1)  
1   0   SEQUENCE OF 'BOOKING_SEQ' (SEQUENCE)  
2   1   FAST DUAL (Cost=2 Card=1)
```

```
Statistics
```

```
-----  
0 recursive calls  
0 db block gets  
0 consistent gets  
0 physical reads  
0 redo size  
394 bytes sent via SQL*Net to client  
508 bytes received via SQL*Net from client  
2 SQL*Net roundtrips to/from client  
0 sorts (memory)  
0 sorts (disk)  
1 rows processed
```



# New Tuning Tools

- ➡ **Oracle Enterprise Manager**
- ➡ **SQL Tuning Advisor**
- ➡ **SQL Access Advisor**
- ➡ **Automatic Database Diagnostics Monitor**
- ➡ **Automatic Workload Repository**
- ➡ **SQL Tuning Sets**
- ➡ **SQL Profiling**
- ➡ **QBQL**
- ➡ **SQL\*Repository**



# System Statistics

# System Statistics

## ➡ WORKLOAD

**SREADTIM**

Avg time (in milliseconds) to read a single block

**MREADTIM**

Avg time (in milliseconds) for a multiblock read

**CPUSPEED**

Avg CPU cycles per second

**MBRC**

Avg number of blocks read in a multiblock read

**MAXTHR**

Maximum IO throughput (for parallel queries)

**SLAVETHR**

Maximum slave throughput (for parallel queries)

## ➡ NON WORKLOAD

**CPUSPEEDNW**

Avg CPU cycles per second

**IOSEEKTIM**

Seek time + latency time + OS overhead time

**IOTFRSPEED**

Time for a single read request

# System Statistics

```
SELECT *
FROM sys.aux_stats$
```

SNAME	PNAME	PVAL1	PVAL2
SYSSTATS_INFO	STATUS		COMPLETED
SYSSTATS_INFO	DSTART		05-14-2005 10:25
SYSSTATS_INFO	DSTOP		05-14-2005 16:39
SYSSTATS_INFO	FLAGS	1	
SYSSTATS_MAIN	CPUSPEEDNW	904.86697	
SYSSTATS_MAIN	IOSEEKTIM	10	
SYSSTATS_MAIN	IOTFRSPEED	4096	
SYSSTATS_MAIN	SREADTIM	18.474	
SYSSTATS_MAIN	MREADTIM	11.958	
SYSSTATS_MAIN	CPUSPEED	1264	
SYSSTATS_MAIN	MBRC	7	
SYSSTATS_MAIN	MAXTHR	610304	
SYSSTATS_MAIN	SLAVETHR		

13 rows selected.



# Gathering System Statistics

**DBMS\_STATS.GATHER\_SYSTEM\_STATS (**

<b>gathering_mode</b>	<b>VARCHAR2 DEFAULT 'NOWORKLOAD',</b>
<b>interval</b>	<b>INTEGER DEFAULT NULL,</b>
<b>stattab</b>	<b>VARCHAR2 DEFAULT NULL,</b>
<b>statid</b>	<b>VARCHAR2 DEFAULT NULL,</b>
<b>statown</b>	<b>VARCHAR2 DEFAULT NULL);</b>

**dbms\_stats.gather\_system\_stats(gathering\_mode=>'start');**

**dbms\_stats.gather\_system\_stats(gathering\_mode=>'stop');**

# System Statistics

- ➡ Optimiser cost model uses system statistics for CPU and IO
- ➡ MBRC used rather than `db_file_miltblock_read_count`
- ➡ Some operations use CPU not IO, e.g filters, merges, hash joins
- ➡ Gather for different workloads  
`EXPORT_SYSTEM_STATS`  
`IMPORT_SYSTEM_STATS`

# **Recommendations on gathering statistics**





# Solving Performance Problems

- ➡ Check that statistics are up to date on all objects
  - ➡ Tables
  - ➡ Indexes
  - ➡ Columns
- ➡ Gather new stats
- ➡ Use automatic stats job

# Histograms

- Create histograms on skewed columns
- Cost based optimiser assumes an even distribution of values
- METHOD\_OPT parameter

FOR COLUMNS col1,col2 SIZE n

FOR ALL COLUMNS SIZE n

FOR ALL INDEXED COLUMNS SIZE n

integer

Number of histogram buckets

REPEAT

already have histograms

**AUTO**

**data distribution and workload**

**SKEWONLY**

**data distribution**



# Gathering Stats

```
DBMS_STATS.GATHER_TABLE_STATS (  
  ownname          VARCHAR2,  
  tabname          VARCHAR2,  
  partname         VARCHAR2 DEFAULT NULL,  
  estimate_percent  NUMBER  DEFAULT to_estimate_percent_type  
                      (get_param('ESTIMATE_PERCENT')),  
  
  block_sample     BOOLEAN DEFAULT FALSE,  
  method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),  
  degree           NUMBER  DEFAULT  
                      to_degree_type(get_param('DEGREE')),  
  
  granularity      VARCHAR2 DEFAULT 'AUTO',  
  cascade          BOOLEAN DEFAULT  
                      to_cascade_type(get_param('CASCADE')),  
  
  stattab          VARCHAR2 DEFAULT NULL,  
  statid           VARCHAR2 DEFAULT NULL,  
  statown          VARCHAR2 DEFAULT NULL,  
  no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type (  
                      get_param('NO_INVALIDATE')));
```



# Managing Stats

- ➡ Manually set and copy statistics
- ➡ Export and Import statistics
- ➡ Store statistics in a user owned table
- ➡ `DBMS_STATS.CREATE_STAT_TABLE`
- ➡ Keep multiple sets of statistics
- ➡ `STATID` identifies a set of statistics



# Manually Setting Stats

- ➡ **DBMS\_STATS.SET\_TABLE\_STATS**
- ➡ **Set table, column or index stats manually**
- ➡ **Regathering will overwrite**
- ➡ **Freezing fixes stats – won't respond to changes**
- ➡ **Use only for jobs where data is very volatile and a fixed result gives good results**

# Export a Set of Statistics

BEGIN

```
dbms_stats.export_schema_stats  
(ownname=>'EOUG',  
stattab=>'STATS',  
statid=>'S2',  
statown=>'EOUG');  
END;
```

# Import a Set of Statistics

BEGIN

```
dbms_stats.import_schema_stats  
(ownname=>'EOUG',  
stattab=>'STATS',  
statid=>'S2',  
statown=>'EOUG');
```

END;

# Getting Table Statistics

```
DBMS_STATS.GET_TABLE_STATS (  
    ownname          VARCHAR2,  
    tabname          VARCHAR2,  
    partname         VARCHAR2 DEFAULT NULL,  
    stattab          VARCHAR2 DEFAULT NULL,  
    statid           VARCHAR2 DEFAULT NULL,  
    numrows          OUT NUMBER,  
    numblks          OUT NUMBER,  
    avgrlen          OUT NUMBER,  
    statown          VARCHAR2 DEFAULT NULL);
```



# Setting Table Statistics

```
DBMS_STATS.SET_TABLE_STATS (  
    ownname          VARCHAR2,  
    tabname          VARCHAR2,  
    partname         VARCHAR2 DEFAULT NULL,  
    stattab          VARCHAR2 DEFAULT NULL,  
    statid           VARCHAR2 DEFAULT NULL,  
    numrows          NUMBER  DEFAULT NULL,  
    numblks          NUMBER  DEFAULT NULL,  
    avgrlen          NUMBER  DEFAULT NULL,  
    flags            NUMBER  DEFAULT NULL,  
    statown          VARCHAR2 DEFAULT NULL);
```

# **How bind variables affect SQL tuning**

# Problems Calculating Cardinality

- ➡ The CBO can work out cardinality for single table access when it has:
  - ➡ Data not skewed + col = literal or col = bind variable
  - ➡ Data not skewed + range check using literal
  - ➡ Data skewed + histogram + col = literal
- ➡ The CBO cannot work out cardinality for single table access when it has:
  - ➡ Data skewed + col = bind variable
  - ➡ Data not skewed + range check using bind variable
- ➡ The CBO's solution to this is to use bind peeking

# Bind Variables

- ➡ Recommended to optimise performance of the library cache
- ➡ Can use `CURSOR_SHARING = SIMILAR` to represent literals as bind variables

# Example Data

```
SELECT resource_code, COUNT(*)
FROM bookings_large
GROUP BY resource_code
```



RESO	COUNT ( * )
----	-----
BRLG	297763
BRSM	148879
CONF	99255
FLPC	99252
LNCH	148881
PC1	9629
TAP1	49626
VCR1	489252
VCR2	99255

```
SELECT num_distinct, low_value,
       high_value, histogram
FROM   user_tab_col_statistics
WHERE  table_name = 'BOOKINGS_LARGE'
AND    column_name = 'RESOURCE_CODE'
```



NUM_DISTINCT	LOW_VALUE	HIGH_VALUE	HISTOGRAM
-----	-----	-----	-----
9	42524C47	56435232	NONE

# Cardinality

► **Column = bind variable (No histogram)**

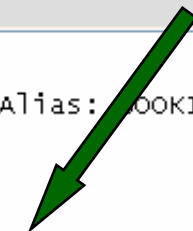
**Number of rows expected (cardinality) =  
(Cardinality of table – Number of null rows) \* Density of column**

**Example:**

**SELECT \* FROM bookings\_large  
WHERE resource\_code = :b1**

$$174727.67 = 1572549 / 9$$

```
*****
SINGLE TABLE ACCESS PATH
  COLUMN: RESOURCE_C(VARCHAR2) Col#: 3      Table: BOOKINGS_LARGE  Alias: BOOKINGS_LARGE
    Size: 5 NDV: 9 Nulls: 0 Density: 1.1111e-001
    No Histogram: #BKT: 1
      (1 uncompressed buckets and 2 endpoint values)
  TABLE: BOOKINGS_LARGE Alias: BOOKINGS_LARGE
    Original Card: 1572549 Rounded Card: 174728 Computed Card: 174727.67
  Access Path: table-scan Resc: 1961 Resp: 1961
  Access Path: index (equal)
    Index: BK_RES2
    rsc_cpu: 142551196 rsc_io: 7525
    ix_sel: 0.0000e+000 ix_sel_with_filters: 1.1111e-001
  BEST_CST: 1961.44 PATH: 2 Degree: 1
*****
```

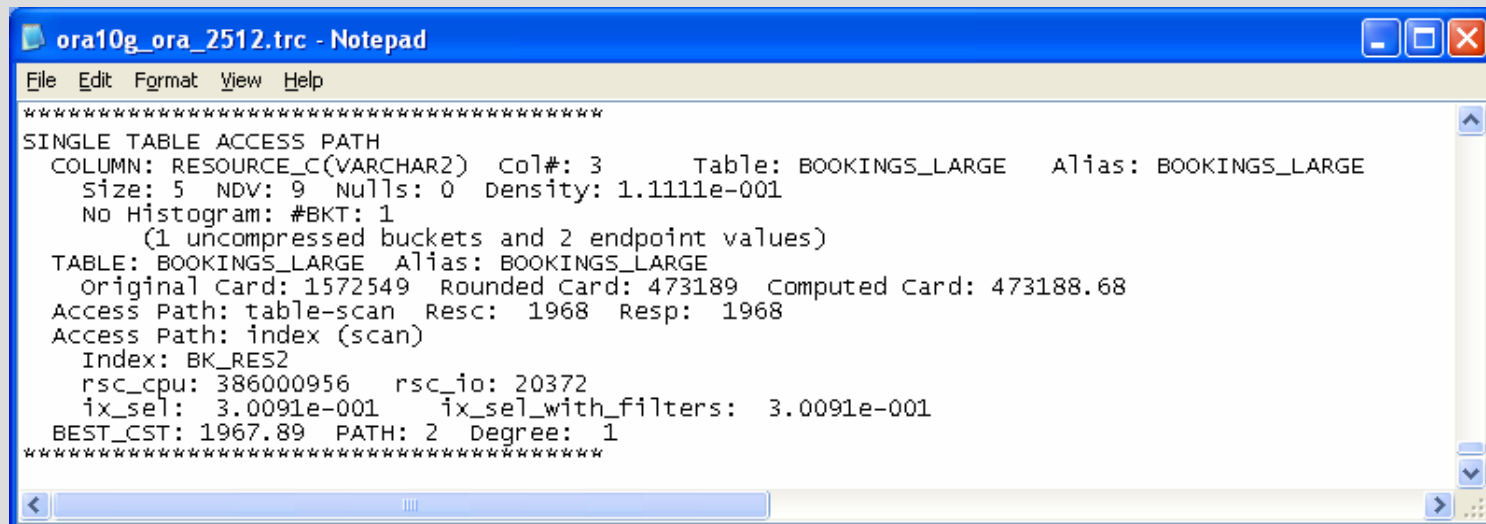


# Selectivity

➡ **Column > bind variable (No histogram)**

**Example:**

**SELECT \* FROM bookings\_large WHERE booking\_no > :b1**



```

ora10g_ora_2512.trc - Notepad
File Edit Format View Help
*****
SINGLE TABLE ACCESS PATH
  COLUMN: RESOURCE_C(VARCHAR2) Col#: 3      Table: BOOKINGS_LARGE  Alias: BOOKINGS_LARGE
    Size: 5 NDV: 9 Nulls: 0 Density: 1.1111e-001
    No Histogram: #BKT: 1
      (1 uncompressed buckets and 2 endpoint values)
  TABLE: BOOKINGS_LARGE Alias: BOOKINGS_LARGE
    Original Card: 1572549 Rounded Card: 473189 Computed Card: 473188.68
    Access Path: table-scan Resc: 1968 Resp: 1968
    Access Path: index (scan)
      Index: BK_RES2
        rsc_cpu: 386000956 rsc_io: 20372
        ix_sel: 3.0091e-001 ix_sel_with_filters: 3.0091e-001
    BEST_CST: 1967.89 PATH: 2 Degree: 1
*****
  
```

# Bind Peeking

- ➡ Version 9i/10g use “Bind Variable Peeking”
- ➡ Plan is based on the value of the first bind value used when statement is hard parsed
- ➡ To disable set `_optim_peek_user_binds` to FALSE



# Bind Peeking

```
ORA92> SELECT resource_code, count(*)
        FROM   bookings_large
        GROUP BY resource_code;
```

RESO	COUNT ( * )
----	-----
BRLG	297763
BRSM	148879
CONF	99255
FLPC	99252
LNCH	148881
PC1	9629
TAP1	49626
VCR1	489252
VCR2	99255



Minority value



Majority value

# Bind Peeking

```
CREATE OR REPLACE procedure testbind  
  (p_code IN resources.code%TYPE)  
IS  
  v_count NUMBER;  
BEGIN  
  SELECT count(*)  
  INTO    v_count  
  FROM    bookings_large b  
  WHERE   b.resource_code = p_code;  
END;
```

# Bind Peeking

BEGIN

testbind('VCR1');

END;

Oracle SQL\*Plus

File Edit Search Options Help

ORA92>@plan\_Real

Enter value for address: 7B12014C

Enter value for hash\_value: 666553192

Enter value for child\_number: 0

Id	Parent	Object	Operation	Options	Pos	Rows	Cpu	IO
Cost: 4			SELECT STATEMENT					
1	0		SORT	AGGREGATE	1	1		
2	1	BOOK_RES_LG	INDEX	FAST FULL SCAN	1	6478		4

ORA92>|



# Bind Peeking

BEGIN

testbind('PC1');

END;

Oracle SQL\*Plus window showing the execution plan for plan\_Real. The window title is "Oracle SQL\*Plus". The menu bar includes File, Edit, Search, Options, and Help. The command prompt shows the user entering the command "ORA92>@plan\_Real". The prompt then asks for the address, hash\_value, and child\_number, which are entered as 7B12014C, 666553192, and 0 respectively. The execution plan is displayed as a table with columns: Id, Parent, Object, Operation, Options, Pos, Rows, Cpu, and IO.

Id	Parent	Object	Operation	Options	Pos	Rows	Cpu	IO
1	0		SELECT STATEMENT					
		Cost: 4						
2	1	BOOK_RES_LG	INDEX	FAST FULL SCAN	1	6478		4

ORA92>|

# Bind Peeking

```
CREATE OR REPLACE procedure testbind2
  (p_code IN resources.code%TYPE)
IS
  v_count NUMBER;
BEGIN
  SELECT count(*)
  INTO    v_count
  FROM    bookings_large c
  WHERE   c.resource_code = p_code;
END;
```

# Bind Peeking

BEGIN

testbind2('PC1');

END;

Oracle SQL\*Plus

File Edit Search Options Help

ORA92>@plan\_real  
Enter value for address: 7AEC76F4  
Enter value for hash\_value: 3762386321  
Enter value for child\_number: 0

Id	Parent	Object	Operation	Options	Pos	Rows	Cpu	IO
-----								
		Cost: 2	SELECT STATEMENT					
1	0		SORT	AGGREGATE	1	1		
2	1	BOOK_RES_LG	INDEX	RANGE SCAN	1	478		2

# Bind Peeking

ora10g\_ora\_1248.trc - Notepad

```

File Edit Format View Help
*****
SINGLE TABLE ACCESS PATH
  COLUMN: RESOURCE_C(VARCHAR2) Col#: 3      Table: BOOKINGS_LARGE  Alias: B
    Size: 5 NDV: 9 Nulls: 0 Density: 3.1078e-007
    Frequency Histogram: #BKT: 9
      (5560 uncompressed buckets and 9 endpoint values)
  TABLE: BOOKINGS_LARGE Alias: B
    Original Card: 1571418 Rounded Card: 11588 Computed Card: 11587.79
  Access Path: table-scan Rsc: 1958 Resp: 1958
  Access Path: index (index-ffs)
    Index: BK_RES2
      rsc_cpu: 28086959 rsc_io: 750
      ix_sel: 0.0000e+000 ix_sel_with_filters: 1.0000e+000
  Access Path: index-ffs Rsc: 755 Resp: 755
  Access Path: index (equal)
    Index: BK_RES2
      rsc_cpu: 2543286 rsc_io: 32
      ix_sel: 0.0000e+000 ix_sel_with_filters: 7.3741e-003
  BEST_CST: 32.46 PATH: 4 Degree: 1
*****
OPTIMIZER STATISTICS AND COMPUTATIONS
*****
GENERAL PLANS
*****
Join order[1]: BOOKINGS_LARGE[B]#0
Best so far: TABLE#: 0 CST: 32 CDN: 11588 BYTES: 57940
(newjo-stop-1) k:0, spcnt:0, perm:1, maxperm:80000
prefetching is on for BK_RES2
Final - All Rows Plan:
  JOIN ORDER: 1
  CST: 32 CDN: 11588 RSC: 32 RSP: 32 BYTES: 57940
  IO-RSC: 32 IO-RSP: 32 CPU-RSC: 2543286 CPU-RSP: 2543286
*** 2005-05-24 16:54:32.984

```

CBO knows it's a minority value

## **Tuning Tools – finding out what it did, and why**





# Explain

```
EXPLAIN PLAN SET STATEMENT_ID = 'Q1'  
FOR  
SELECT      b.resource_code, b.cost  
FROM        events e, bookings b  
WHERE       e.event_no = b.event_no  
AND         e.org_id = 1000;
```

# Explain Output

## ➡ OPERATION

SELECT STATEMENT  
UPDATE STATEMENT  
INSERT STATEMENT  
DELETE STATEMENT  
AND-EQUAL  
CONNECT BY  
CONCATENATION  
COUNT  
FILTER  
INDEX

INLIST ITERATOR  
HASH JOIN  
MERGE JOIN  
NESTED LOOPS  
SORT  
TABLE ACCESS  
UNION  
INTERSECTION  
MINUS  
VIEW



# Explain Output

➡ <b>OPTIONS</b>	
TABLE ACCESS	FULL BY INDEX ROWID
INDEX ACCESS	UNIQUE SCAN RANGE SCAN RANGE SCAN DESCENDING FULL SCAN FAST FULL SCAN SKIP SCAN
SORT	AGGREGATE GROUP BY JOIN ORDER BY UNIQUE

# Explain Output

@utlxpls

PLAN\_TABLE\_OUTPUT

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		11	220	5 (20)	00:00:01
* 1	HASH JOIN		11	220	5 (20)	00:00:01
* 2	TABLE ACCESS FULL	EVENTS	4	32	2 (0)	00:00:01
3	TABLE ACCESS FULL	BOOKINGS	30	360	2 (0)	00:00:01

Predicate Information (identified by operation id):

- 1 - access("E"."EVENT\_NO"="B"."EVENT\_NO")
- 2 - filter("E"."ORG\_ID"=1000)

Aim to reduce the number of rows as soon as possible

# Access and Filter

Used to access the table

PLAN\_TABLE\_OUTPUT

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		11	220	5 (20)	00:00:01
* 1	HASH JOIN		11	220	5 (20)	00:00:01
* 2	TABLE ACCESS FULL	EVENTS	4	32	2 (0)	00:00:01
3	TABLE ACCESS FULL	BOOKINGS	30	360	2 (0)	00:00:01

Predicate Information (identified by operation id):

1 - access("E"."EVENT\_NO"="B"."EVENT\_NO")  
 2 - filter("E"."ORG\_ID">1000)

Used to filter rows



# AUTOTRACE

## SET AUTOTRACE ON

```
SELECT      e.start_date,b.resource_code, b.cost
FROM        events_large e, bookings_large b
WHERE       b.event_no = e.event_no
```

### Execution Plan

```
-----
0          SELECT STATEMENT Optimizer=ALL_ROWS (Cost=29 Card=11264 Bytes=270336)

1    0      HASH JOIN (Cost=29 Card=11264 Bytes=270336)
2    1        TABLE ACCESS (FULL) OF 'EVENTS_LARGE' (TABLE) (Cost=6 Card=1536 Bytes=18432)

3    1        TABLE ACCESS (FULL) OF 'BOOKINGS_LARGE' (TABLE) (Cost=22 Card=11264 Bytes=135168)
```



# AUTOTRACE

## Statistics

---

395	recursive calls
0	db block gets
926	consistent gets
16	physical reads
0	redo size
314886	bytes sent via SQL*Net to client
8757	bytes received via SQL*Net from client
752	SQL*Net roundtrips to/from client
4	sorts (memory)
0	sorts (disk)
11264	rows processed

# Use the V\$ tables

## ➡ Identify high disk reads

```
SELECT      u.username,  
            a.disk_reads,  
            a.executions,  
            a.disk_reads / decode(a.executions,0,null,a.executions),  
            a.sql_text  
FROM        dba_users u, v$sqlarea a  
WHERE       a.parsing_user_id  
            = u.user_id  
AND         a.disk_reads / decode(a.executions,0,null,a.executions) > &reads;  
|
```



# Use the V\$ tables

## ➡ Identify high memory reads

```
SELECT      u.username,  
            a.buffer_gets,  
            a.executions,  
            a.buffer_gets / decode(a.executions,0,null,a.executions),  
            a.sql_text  
FROM        dba_users u, v$sqlarea a  
WHERE       a.parsing_user_id  
            = u.user_id  
AND         a.buffer_gets / decode(a.executions,0,null,a.executions) > &reads;
```



# Statistic Levels

➡ **BASIC**

No advisories or statistics are enabled

➡ **TYPICAL**

Default value. Enables a number of advisories of interest to database administrators, for example the buffer cache and shared pool sizing advisories  
Results in the collection of the following statistics:

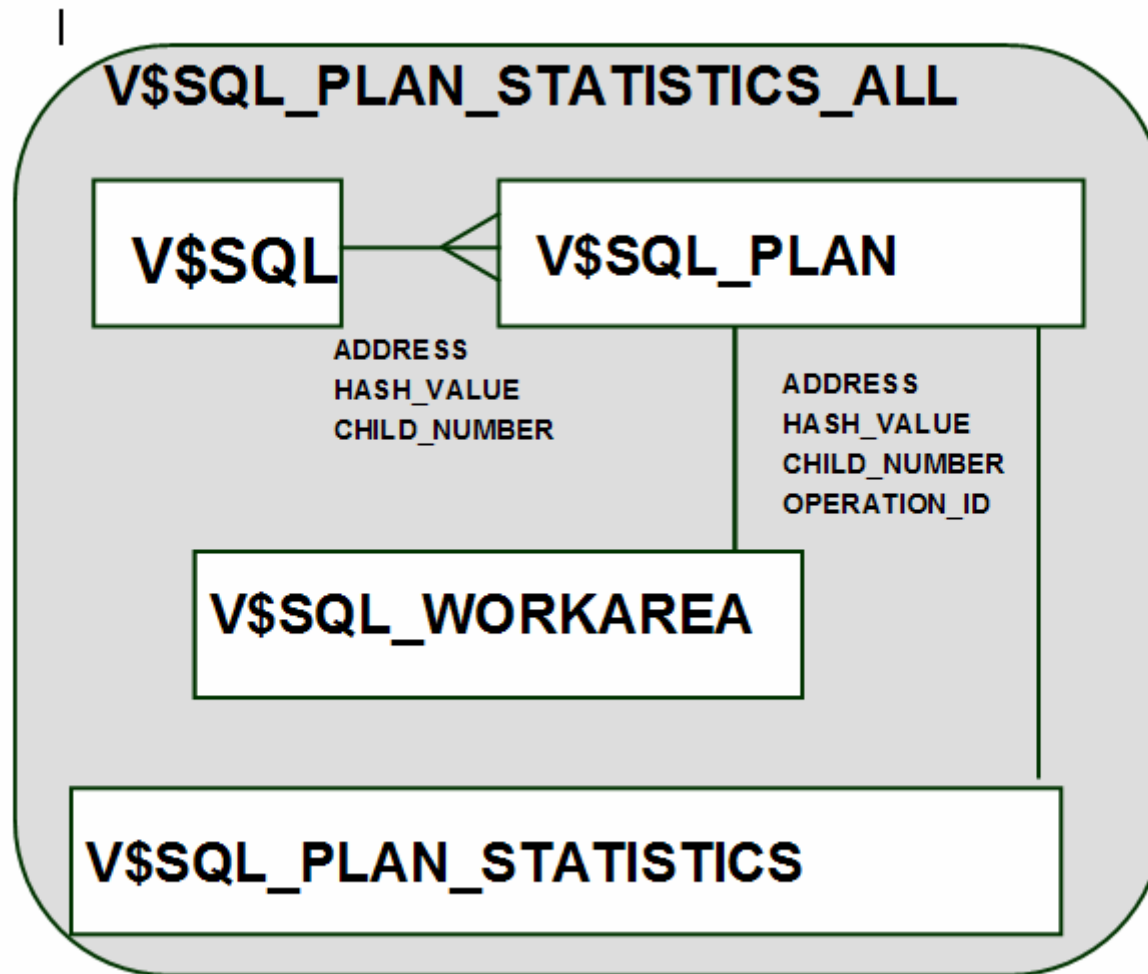
Segment level statistics

Timed Statistics

◆ **ALL**

Includes all the statistics and advisories gathered for the TYPICAL option and also timed operating system statistics and row source execution statistics

# V\$SQL



# Identify statements of interest

```
SELECT      s.sql_text, u.username,
            s.address,
            s.hash_value,
            s.child_number

FROM        v$sql s, dba_users u
WHERE       upper(s.sql_text) like '%'||upper('&sqltext')||'%'
AND         s.parsing_user_id = u.user_id;
```

```
temp.LST - Notepad
File Edit Format Help

SQL> @get_sql
Enter value for sqltext: events
old 6: WHERE          upper(s.sql_text) like '%'||upper('&sqltext')||'%'
new 6: WHERE          upper(s.sql_text) like '%'||upper('events')||'%'

SQL_TEXT
-----
USERNAME          ADDRESS  HASH_VALUE CHILD_NUMBER
-----
SELECT  s.sql_text, u.username,  s.address,  s.hash_value,  s.child_number F
ROM      v$sql s, dba_users u WHERE  upper(s.sql_text) like '%'||upper('events')
||'%' AND          s.parsing_user_id = u.user_id
TRAIN          7A9E86B8 3256518038          0

SELECT COUNT(*) FROM EVENTS
TRAIN          7B1EA484 3804003270          0

SELECT COUNT(*) FROM EVENTS
SYSTEM        7B1EA484 3804003270          1

|
```

# Access V\$SQL\_PLAN

```

plan_real.sql - Notepad
File Edit Format Help
Rem      Filename:      plan_real.sql
Rem      Purpose:       Display results from the V$SQL_PLAN.
Rem      Steps are displayed in an indented format according to
Rem      their level in the execution tree.

set verify off
set linesize 180
set pagesize 60

col ID      format a6      heading "Id"
col PT      format 99      heading "Par
col OB      format a10     heading "Obj
col OP      format a18     heading "Ope
col OT      format a30     heading "Opt
col PS      format 999     heading "Pos
col CD      format 999     heading "Row
col CP      format 99999   heading "Cpu
col IO      format 999     heading "IO"

spool plan_real

SELECT      lpad(to_char(p.id),4,' ') PT,
            decode(p.id,0,'Cost: '||p.cost,p.object_name||' OP',
            p.operation OP,
            p.options OT,
            to_number(decode(p.id,0,null,p.position)) PS,
            p.cardinality CD, p.cpu_cost CP,p.io_cost IO
FROM        v$sql_plan p
WHERE       p.address = HEXTORAW('&address')
AND        p.hash_value = &hash_value
AND        p.child_number = &child_number
/

set verify on
spool off
set linesize 80

```

Oracle SQL\*Plus

File Edit Search Options Help

SQL> @plan\_real

Enter value for address: 7B1EA484

Enter value for hash\_value: 3804003270

Enter value for child\_number: 0

Id	Parent	Object	Operation	Options	Pos	Rows	Cpu	IO
1	0		Cost: 2 SELECT STATEMENT SORT		1	1		
2	1	EUT_PK	INDEX	AGGREGATE FULL SCAN	1	12	10372	1

SQL> |

# Access V\$SQL\_PLAN\_STATISTICS\_ALL

```
plan_real_stats.sql - Notepad
File Edit Format Help
Rem Filename: plan_real_stats.sql
Rem Purpose: Display results from the V$SQL_PLAN_STATISTICS_ALL view.
Rem Steps are displayed in an indented format according to
Rem their level in the execution tree.

set verify off
set linesize 180
set pagesize 60
set recsep off
break on tx page
```

```
col TX noprint new_value tx
col ID format a6 head
col PT format 99 head
col OB format a20 head
col OP format a18 head
col OT format a20 head
col PS format 999 head
col CD format 9999999 head
col CP format 9999999 head
col LR format 99999 head
col EX format 9999 head
col TP format 999999 head

ttitle "Statement:" txt
spool plan_real_stats

SELECT sql_text TX
       lpad(to_char(p.parent_id), 6, ' ') PT,
       decode(p.id, 0, 'Cost: ' || p.cost.p.object_name) OB,
       p.operation OP,
       p.options OT,
       to_number(decode(p.id, 0, null, p.position)) PS,
       p.cardinality CD, p.cpu_cost CP,
       nvl(p.cr_buffer_gets, 0) + nvl(p.cu_buffer_gets, 0) LR,
       p.executions EX, p.max_temptseg_size TP
FROM v$sql s, v$sql_plan_statistics_all p
WHERE p.address = s.address
AND p.hash_value = s.hash_value
AND p.child_number = s.child_number
AND upper(s.sql_text) like '%' || upper('&tablename') || '%';

set verify on
set recsep wrap
spool off
set linesize 80
```

plan\_real\_stats\$1 - Notepad

File Edit Format Help

@plan\_real\_stats  
Enter value for tablename: bookings\_large

Statement: SELECT COUNT(DISTINCT RESOURCE\_CODE) FROM bookings\_large

Id	Parent Object	Operation	Options	Pos	Rows	Cpu	Reads	Exec	Max Temp
1	0	SORT	GROUP BY	1	1		60	1	
2	1 BOOKINGS_LARGE	TABLE ACCESS	FULL	1	11264	2603054	60	1	

Statement: SELECT e.event\_no, b.booking\_no FROM events\_large e, bookings\_large b WHERE e.event\_no = b.event\_no AND e.event\_no = 1000

Id	Parent Object	Operation	Options	Pos	Rows	Cpu	Reads	Exec	Max Temp
1	0	NESTED LOOPS		1	8	2955417	71	1	
2	1 EVTIG_PK	INDEX	UNIQUE SCAN	1	1	14443	2	1	
3	1 BOOKINGS_LARGE	TABLE ACCESS	FULL	2	8	2940974	69	1	

# Trace 10046 and tkprof

## ➡ Trace levels

- 1 Trace resources
- 4 Trace resources and bind variable values
- 8 Trace resources and wait events
- 12 Trace resources, bind variables and wait events

➡ **ALTER SYSTEM SET EVENTS '10046 trace name context forever, level 12'**

➡ **ALTER SESSION SET EVENTS '10046 trace name context forever, level 12'**



# Trace 10046 Example

```
ALTER session SET EVENTS '10046 trace  
name context forever, level 12'
```

```
DECLARE  
v_event_no NUMBER :=2264;  
v_count    NUMBER;  
BEGIN  
SELECT      COUNT(e.comments)  
INTO        v_count  
FROM        events e, bookings b  
WHERE       e.event_no = b.event_no  
AND         e.org_id = v_event_no;  
END;
```





# Trace 10046 Example

```
=====
PARSING IN CURSOR #16 len=100 dep=1 uid=67 oct=3 lid=67 tim=1820067602 hv=3289882752 ad='671e4a68'
SELECT COUNT(E.COMMENTS) FROM EVENTS E, BOOKINGS B WHERE E.EVENT_NO = B.EVENT_NO AND E.ORG_ID = :B1
END OF STMT
PARSE #16:c=0,e=327,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=1,tim=1820067596
=====
PARSING IN CURSOR #8 len=210 dep=2 uid=0 oct=3 lid=0 tim=1820074126 hv=864012087 ad='6a3b2014'
select /*+ rule */ bucket_cnt, row_cnt, cache_cnt, null_cnt, timestamp#, sample_size, minimum, maximum, distcnt, lowval, h
END OF STMT
PARSE #8:c=0,e=89,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=3,tim=1820074121
BINDS #8:
  bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=08 oacfl2=0001 size=24 offset=0
    bfp=05bfcd48 bln=22 avl=04 flg=05
    value=53686
  bind 1: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=08 oacfl2=0001 size=24 offset=0
    bfp=062c029c bln=24 avl=02 flg=05
    value=7
EXEC #8:c=0,e=8682,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=3,tim=1820089289
FETCH #8:c=0,e=78,p=0,cr=3,cu=0,mis=0,r=1,dep=2,og=3,tim=1820090928
BINDS #16:
  bind 0: dty=2 mxl=22(21) mal=00 scl=00 pre=00 oacflg=13 oacfl2=206001 size=24 offset=0
    bfp=05e4bc8c bln=22 avl=03 flg=09
    value=2264
EXEC #16:c=31250,e=25972,p=0,cr=3,cu=0,mis=1,r=0,dep=1,og=1,tim=1820099318
WAIT #16: nam='db file sequential read' ela= 105980 p1=6 p2=130 p3=1
FETCH #16:c=0,e=107747,p=1,cr=5,cu=0,mis=0,r=1,dep=1,og=1,tim=1820208431
EXEC #14:c=31250,e=143977,p=1,cr=8,cu=0,mis=0,r=1,dep=0,og=1,tim=1820209995
WAIT #14: nam='SQL*Net message to client' ela= 5 p1=1413697536 p2=1 p3=0
```

# tkprof

tkprof ora10g\_ora\_456.trc trace1.prf

```

SQL*Net message to client                      1          0.00          0.00
*****

SELECT COUNT(E.COMMENTS)
FROM
  EVENTS E, BOOKINGS B WHERE E.EVENT_NO = B.EVENT_NO AND E.ORG_ID = :B1

call      count          cpu    elapsed        disk    query    current    rows
-----
Parse         1         0.00         0.00          0         0         0         0
Execute       1         0.03         0.01          0         0         0         0
Fetch         1         0.00         0.10          1         5         0         1
-----
total         3         0.03         0.12          1         5         0         1

Misses in library cache during parse: 1
Misses in library cache during execute: 1
Optimizer goal: ALL_ROWS
Parsing user id: 67      (recursive depth: 1)

Elapsed times include waiting on following events:
  Event waited on                      Times    Max. Wait    Total Waited
  -----
  db file sequential read                1         0.10         0.10
*****

```



# Trace 10053

- ➡ Trace optimiser decisions
- ➡ Not formatted with tkprof
- ➡ **ALTER SESSION SET EVENTS '10053 trace name context forever, level 1'**
- ➡ **ALTER SESSION SET EVENTS '10053 trace name context off';**
- ➡ File in user\_dump\_dest
- ➡ Only on hard parse



# Trace 10053 - Example

**ALTER SESSION**

**SET EVENTS**

**'10053 trace name context forever, level 1';**

**SELECT b.resource\_code, b.cost**

**FROM events e, bookings b**

**WHERE e.event\_no = b.event\_no**

**AND e.org\_id = 1030;**



**ora10g\_ora\_3028.trc**

# Trace 10053 File – Initialisation Parameters

```
*****
PARAMETERS USED BY THE OPTIMIZER
*****
*****
PARAMETERS WITH ALTERED VALUES
*****
db_file_multiblock_read_count      = 16
*****
PARAMETERS WITH DEFAULT VALUES
*****
optimizer_mode_hinted               = false
optimizer_features_hinted           = 0.0.0
parallel_execution_enabled          = true
parallel_query_forced_dop           = 0
parallel_dml_forced_dop             = 0
parallel_ddl_forced_degree          = 0
parallel_ddl_forced_instances       = 0
_query_rewrite_fudge                = 90
optimizer_features_enable           = 10.1.0
_optimizer_search_limit             = 5
cpu_count                           = 2
active_instance_count               = 1
parallel_threads_per_cpu            = 2
hash_area_size                     = 131072
bitmap_merge_area_size             = 1048576
sort_area_size                     = 65536
sort_area_retained_size             = 0
_sort_elimination_cost_ratio        = 0
_optimizer_block_size               = 8192
_sort_multiblock_read_count         = 2
_hash_multiblock_io_count           = 0
_optimizer_max_permutations         = 2000
pga_aggregate_target               = 24576 KB
```

Check for Optimizer Mode and other session variables



# Trace 10053 File – Table, Column and Index Stats

```
Column Usage Monitoring is ON: tracking level = 1
*****
QUERY BLOCK SIGNATURE
*****
qb name was generated
signature (optimizer): qb_name=SEL$1 nbfros=2 flg=0
  fro(0): flg=0 objn=53694 hint_alias="B"@SEL$1"
  fro(1): flg=0 objn=53686 hint_alias="E"@SEL$1"
*****
BASE STATISTICAL INFORMATION
*****
Table stats      Table: BOOKINGS      Alias: B
TOTAL ::  CDN: 30  NBLKS: 1  AVG_ROW_LEN: 35
COLUMN:  EVENT_NO(NUMBER)  Col#: 2      Table: BOOKINGS      Alias: B
Size: 4  NDV: 11  Nulls: 0  Density: 1.6667e-002
Frequency Histogram: #BKT: 11
(30 uncompressed buckets and 11 endpoint values)
Index stats
Index: BK_EVT  COL#: 2
TOTAL ::  LVLS: 0  #LB: 1  #DK: 11  LB/K: 1  DB/K: 1  CLUF: 1
Index: BK_RES  COL#: 3
TOTAL ::  LVLS: 0  #LB: 1  #DK: 9  LB/K: 1  DB/K: 1  CLUF: 1
Index: BOOK_PK  COL#: 1
TOTAL ::  LVLS: 0  #LB: 1  #DK: 30  LB/K: 1  DB/K: 1  CLUF: 1
*****
Table stats      Table: EVENTS      Alias: E
TOTAL ::  CDN: 12  NBLKS: 1  AVG_ROW_LEN: 62
COLUMN:  EVENT_NO(NUMBER)  Col#: 1      Table: EVENTS      Alias: E
Size: 4  NDV: 12  Nulls: 0  Density: 8.3333e-002  Min: 100  Max: 226
No Histogram: #BKT: 1
(1 uncompressed buckets and 2 endpoint values)
Index stats
Index: EVT_PK  COL#: 1
TOTAL ::  LVLS: 0  #LB: 1  #DK: 12  LB/K: 1  DB/K: 1  CLUF: 1
__OPTIMIZER_PERCENT_PARALLEL = 0
*****
```



# Trace 10053 File – Single Table Access Paths

```
*****
SINGLE TABLE ACCESS PATH
COLUMN:      ORG_ID(NUMBER)  Col#: 2      Table: EVENTS  Alias: E
Size: 4  NDV: 6  Nulls: 0  Density: 4.1667e-002
Frequency Histogram: #BKT: 6
      (12 uncompressed buckets and 6 endpoint values)
TABLE: EVENTS  Alias: E
Original Card: 12  Rounded Card: 1  Computed Card: 0.50
Access Path: table-scan  Resc: 2  Resp: 2
BEST_CST: 2.00  PATH: 2  Degree: 1
*****
SINGLE TABLE ACCESS PATH
TABLE: BOOKINGS  Alias: B
Original Card: 30  Rounded Card: 30  Computed Card: 30.00
Access Path: table-scan  Resc: 2  Resp: 2
BEST_CST: 2.00  PATH: 2  Degree: 1
*****
```

# Trace 10053 File – Join Order and Mechanism [1]

```

*****
GENERAL PLANS
*****
Join order[1]:  EVENTS[E]#0  BOOKINGS[B]#1
Now joining: BOOKINGS[B]#1 *****
NL Join
  Outer table: cost: 2  cdn: 1  rcz: 8  resc: 2
  Inner table: BOOKINGS Alias: B
    Access Path: table-scan Resc: 2
    Join: Resc: 4  Resp: 4
  Access Path: index (join index)
    Index: BK_EVT
    rsc_cpu: 8441  rsc_io: 1
    ix_sel: 0.0000e+000  ix_sel_with_filters: 9.0909e-002
  NL Join: resc: 3  resp: 3
  Best NL cost: 3  resp: 3
Join cardinality: 1 = outer (1) * inner (30) * sel (9.0909e-002) [flag=0]
SM Join
  Outer table:
    resc: 2  cdn: 1  rcz: 8  deg: 1  resp: 2
  Inner table: BOOKINGS Alias: B
    resc: 2  cdn: 30  rcz: 12  deg: 1  resp: 2
    using join:1 distribution:2 #groups:1
  SORT resource      Sort statistics
    Sort width:      5 Area size:      131072 Max Area size:      1257472
    Degree:          1
    Blocks to Sort:  1 Row size:          19 Total Rows:          1
    Initial runs:    1 Merge passes:      0 IO Cost / pass:      0
    Total IO sort cost: 0      Total CPU sort cost: 23351136
    Total Temp space used: 0
  SORT resource      Sort statistics
    Sort width:      5 Area size:      131072 Max Area size:      1257472
    Degree:          1
    Blocks to Sort:  1 Row size:          24 Total Rows:          30
    Initial runs:    1 Merge passes:      0 IO Cost / pass:      0
    Total IO sort cost: 0      Total CPU sort cost: 23357768
    Total Temp space used: 0
  Merge join Cost: 6  Resp: 6
SM Join (with index on outer)
  Access Path: index (no start/stop keys)

```



# Trace 10053 File – Join Order and Mechanism [2]

```

Join order[2]: BOOKINGS[B]#1 EVENTS[E]#0
Now joining: EVENTS[E]#0 *****
NL Join
  Outer table: cost: 2   cdn: 30   rcz: 12   resp: 2
  Inner table: EVENTS Alias: E
    Access Path: table-scan   Resc: 0
    Join: Resc: 12   Resp: 12
  Access Path: index (unique)
    Index: EVI_PK
    rsc_cpu: 14663   rsc_io: 1
    ix_sel: 8.3333e-002   ix_sel_with_filters: 8.3333e-002
    NL Join: resc: 32   resp: 32
  Access Path: index (eq-unique)
    Index: EVI_PK
    rsc_cpu: 14863   rsc_io: 1
    ix_sel: 0.0000e+000   ix_sel_with_filters: 0.0000e+000
    NL Join: resc: 32   resp: 32
  Best NL cost: 12   resp: 12
Join cardinality: 1 = outer (30) * inner (1) * sel (9.0909e-002) [flag=0]
SM Join
  Outer table:
    resc: 2   cdn: 30   rcz: 12   deg: 1   resp: 2
  Inner table: EVENTS Alias: E
    resc: 2   cdn: 1   rcz: 8   deg: 1   resp: 2
    using join:1 distribution:2 #groups:1
  SORT resource      Sort statistics
    Sort width:      5 Area size:      131072 Max Area size:      1257472
    Degree:          1
    Blocks to Sort:  1 Row size:      24 Total Rows:      30
    Initial runs:    1 Merge passes:    0 IO Cost / pass:      0
    Total IO sort cost: 0      Total CPU sort cost: 23357768
    Total Temp space used: 0
  SORT resource      Sort statistics
    Sort width:      5 Area size:      131072 Max Area size:      1257472
    Degree:          1
    Blocks to Sort:  1 Row size:      19 Total Rows:      1
    Initial runs:    1 Merge passes:    0 IO Cost / pass:      0
    Total IO sort cost: 0      Total CPU sort cost: 23351136
    Total Temp space used: 0
  Merge join Cost: 6   Resp: 6

```



# Trace 10053 File – Best So Far

```
Merge join Cost: 5 Resp: 5
HA Join
Outer table:
  resc: 2 cdn: 1 rcz: 8 deg: 1 resp: 2
Inner table: BOOKINGS Alias: B
  resc: 2 cdn: 30 rcz: 12 deg: 1 resp: 2
  using join:8 distribution:2 #groups:1
Hash join one ptn Resc: 1 Deg: 1
  hash_area: 32 (max=307) buildfrag: 1 probefrag: 1 ppasses: 1
Hash join Resc: 5 Resp: 5
Join result: cost: 3 cdn: 1 rcz: 20
Best so far: TABLE#: 0 CST: 2 CDN: 1 BYTES: 8
Best so far: TABLE#: 1 CST: 3 CDN: 1 BYTES: 20
*****
```



# Trace 10053 File – Final Plan

```
HA Join
  Outer table:
    resc: 2  cdn: 30  rcz: 12  deg: 1  resp: 2
  Inner table: EVENTS  Alias: E
    resc: 2  cdn: 1  rcz: 8  deg: 1  resp: 2
    using join:8 distribution:2 #groups:1
  Hash join one ptn Resc: 1  Deg: 1
    hash_area: 32 (max=307)  buildfrag: 1  probefrag: 1  ppasses: 1
  Hash join  Resc: 5  Resp: 5
(newjo-stop-1) k:0, spcnt:0, perm:2, maxperm:2000
(newjo-save)    [1 0 ]
Final - All Rows Plan:
  JOIN ORDER: 1
  CST: 3  CDN: 1  RSC: 3  RSP: 3  BYTES: 20
  IO-RSC: 3  IO-RSP: 3  CPU-RSC: 18203  CPU-RSP: 18203
QUERY
SELECT  b.resource_code, b.cost
FROM    events e, bookings b
WHERE   e.event_no = b.event_no
AND     e.org_id = 1030
```

## **Hints and when to use them**

# Hints

- ➡ Remember hints fix the execution plan regardless of changes to data
- ➡ Consider other options first
- ➡ Use full not partial hints
- ➡ Incorrect hints are ignored
- ➡ Use the table alias not the table name
- ➡ Provide a complete set of hints
- ➡ Hints apply to a statement block

# Hints

- Hints only refer to directly referenced tables
- To refer to tables within a view use global hints

**VIEWNAME.TABLENAME**

**Example**

**SELECT --+ FULL (contact\_view.contacts)**



# Transformation Hints

**NO\_QUERY\_TRANSFORMATION** Do not transform the statement

**USE\_CONCAT** Use UNION for ORs

**NO\_EXPAND** Do not expand ORs to UNIONS

**STAR\_TRANSFORMATION** Use a star transformation and bitmap indexes  
– if it seems like a good idea

**NO\_STAR\_TRANSFORMATION** Do not use a star transformation

**UNNEST** Merge a subquery with a main query

**NO\_UNNEST** Do not merge a subquery with a main query

**MERGE** Merge a view with the main query

**NO\_MERGE** Do not merge a view with the main query



# Access Hints

**FULL(alias)**

**Perform a full scan of the table**

**INDEX (alias index1 index2 )**

**Use any of the indexes specified**

**INDEX\_ASC (alias index1 index2)**

**As above**

**INDEX\_DESC(alias index1 index2)**

**Use any of the indexes in descending order**

**NO\_INDEX (alias index1 index2 )**

**Do not use any of the indexes specified**

**INDEX\_COMBINE(alias index1 index2)**

**Use to combine bitmap indexes**

**INDEX\_FFS(alias indexname)**

**Perform a fast full index scan**

**INDEX\_JOIN**

**Resolve the query by performing a hash join of a number of indexes**

**[www.sagecomputing.com.au](http://www.sagecomputing.com.au)**



# Access Hint

**COL1 = :b1**



**T2**

**SELECT   --+ INDEX(t2 (col1))**

.....

**FROM     t2**

**WHERE   COL1 = :b1**



# Join Hints

**LEADING(alias alias)**

**Join the tables in the given order**

**ORDERED**

**Join the tables according to the FROM clause**

**USE\_NL**

**Use a nested loop join**

**USE\_NL\_WITH\_INDEX**

**Use a nested loop join with a specific index**

**NO\_USE\_NL**

**Do not use a nested loop join**

**USE\_MERGE**

**Use a sort merge join**

**NO\_USE\_MERGE**

**Do not use a sort merge join**

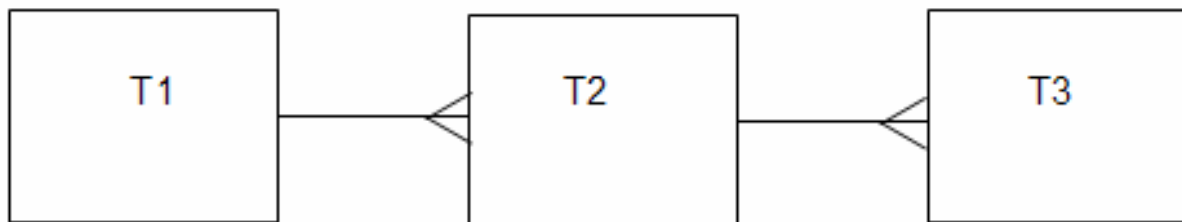
**USE\_HASH**

**Use a hash join**

**NO\_USE\_HASH**

**Do not use a hash join**

# Hints for Nested Loop Joins



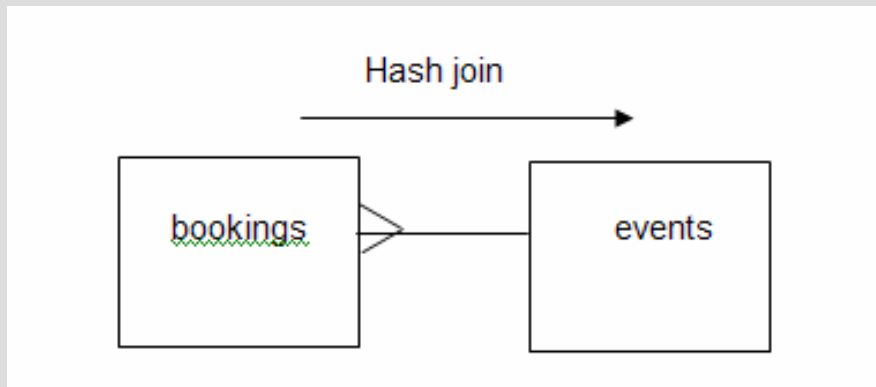
**SELECT   --+ORDERED USE\_NL(t2 t3)**

.....

**FROM     t1, t2, t3**

**WHERE    .....**

# Hints for Hash Joins



```
SELECT      --+ ORDERED USE_HASH (e)
            e.start_date,
            b.resource_code, b.cost
FROM        bookings b, events e
WHERE       b.event_no = e.event_no;
```

**Tuning – making it do what we want it to**

# Initial Checks

- ➡ Are statistics up to date and gathered on all objects?
- ➡ Are histograms gathered on skewed data?
- ➡ Are system statistics collected?
- ➡ Are the initialisation parameters appropriate

**Fix any of these and then rerun the statement**

**If it still doesn't run well**

- ➡ Trace the statement
- ➡ Decide what you want it to do

# Fixing It

- ➡ **Change session settings**
- ➡ **Consider rewriting the statement**
- ➡ **Consider hints**
- ➡ **Consider new indexes**
- ➡ **Use SQL Tuning Advisor (for packaged applications)**
- ➡ **Use SQL Access Advisor**
- ➡ **Run the statement overnight**
- ➡ **Consider materialised views**
- ➡ **Consider partitioning**



# Favouring Index Access

- ➡ If the optimizer is favouring too many full scans / hash joins
- ➡ **OPTIMIZER\_INDEX\_COST\_ADJ**
  - ➡ Default 100
  - ➡ Set lower
- ➡ **OPTIMIZER\_INDEX\_CACHING**
  - ➡ Default 0
  - ➡ Set higher

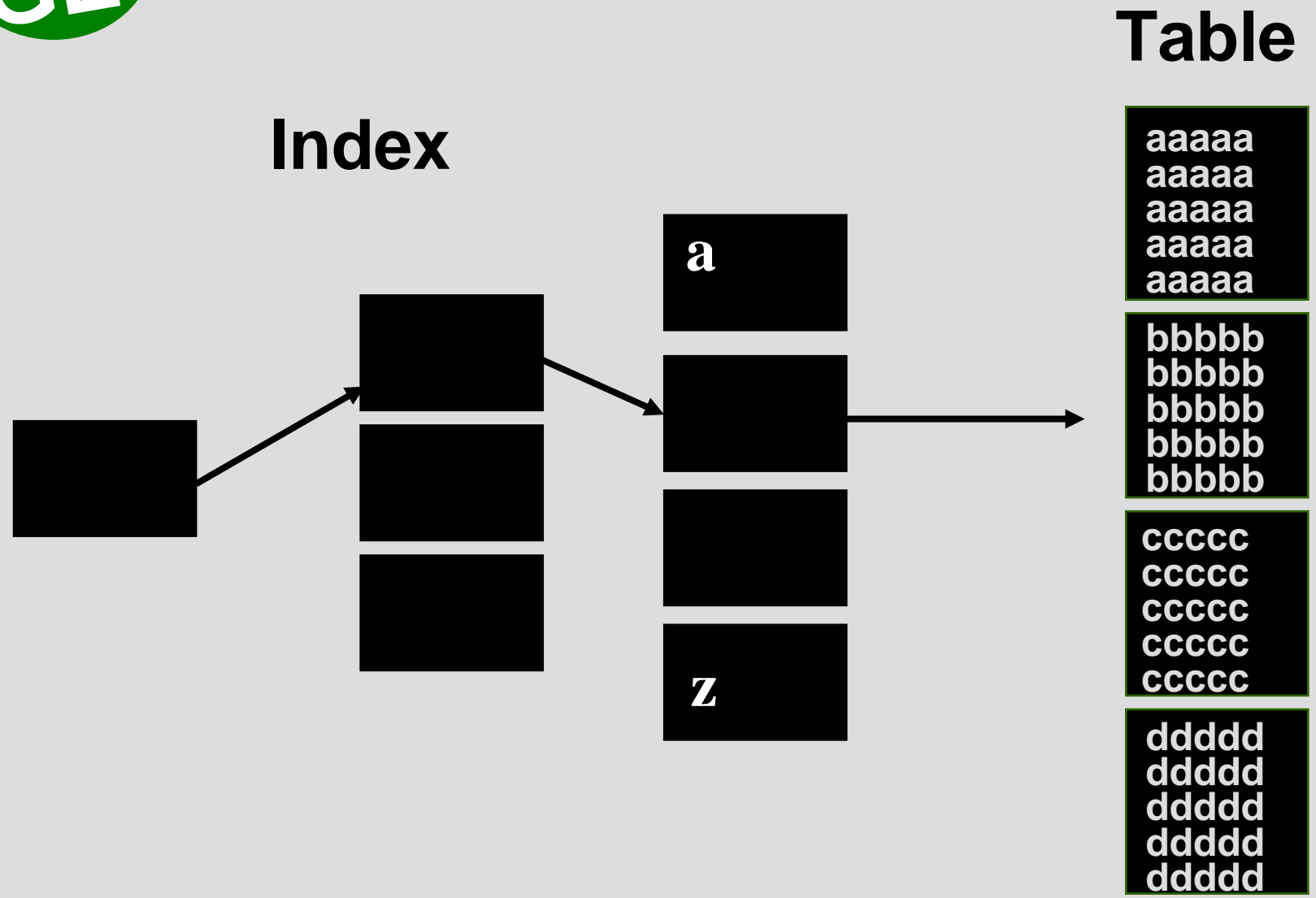




# Should it Use an Index

- ➡ **An index will be efficient if the condition returns a small proportion of data**
- ➡ **The proportion depends on how well clustered the data values are**

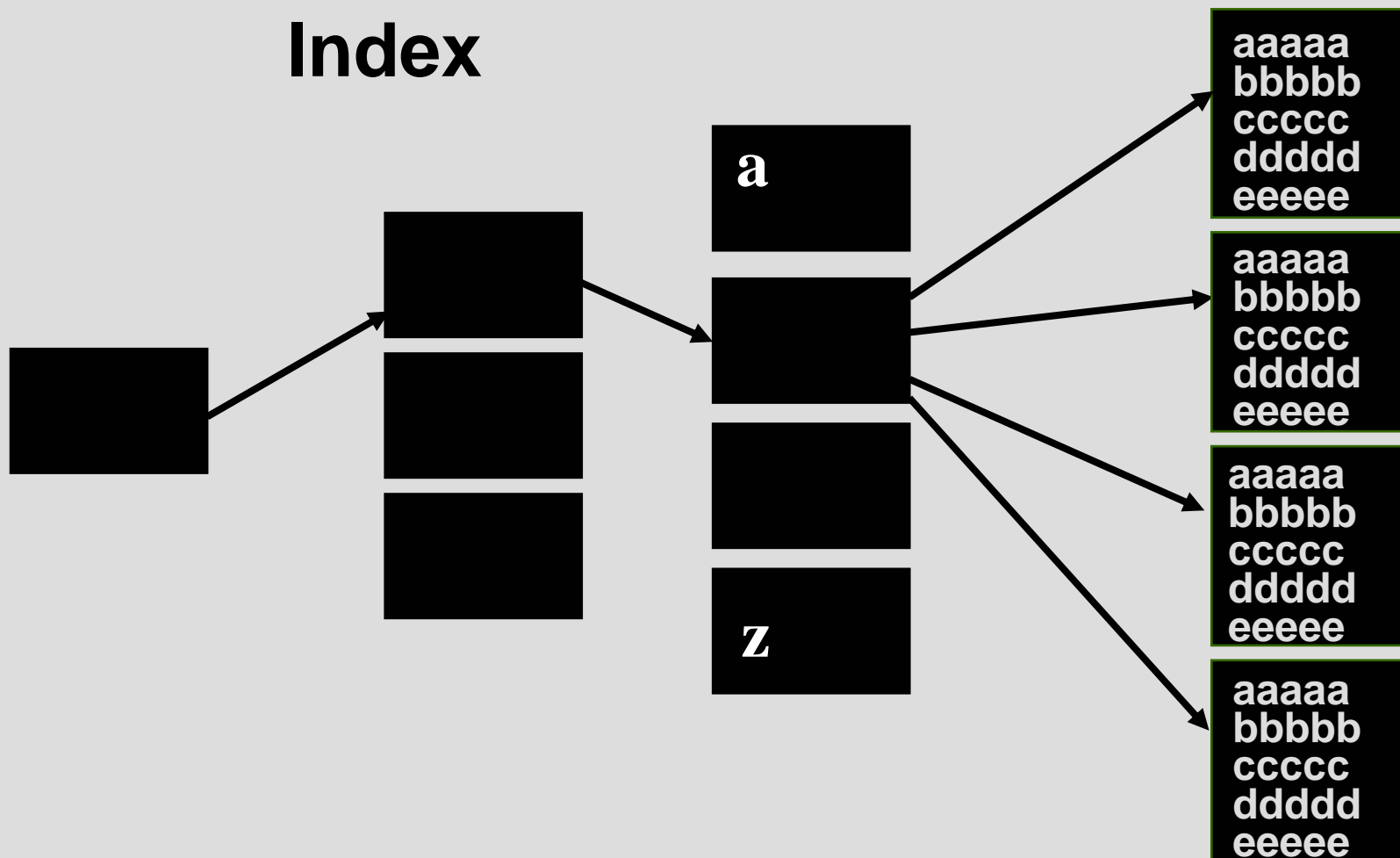
# Well Clustered



# Poorly Clustered

Table

Index



# Clustering Factor

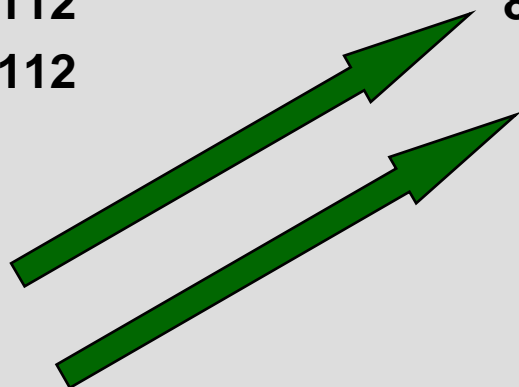
```
SELECT      i.index_name, t.blocks, i.leaf_blocks,  
            i.num_rows, i.clustering_factor  
FROM        dba_tables t, dba_indexes i  
WHERE       i.table_name = t.table_name  
AND         i.table_owner = t.owner  
AND         t.table_name = '&p_tab'
```

# Clustering Factor

INDEX_NAME	BLOCKS	LEAF_BLOCKS	NUM_ROW	CLUSTERING_FACTOR
BOOK_EVTLG	526	191	90112	89682
BOOK_EVTLG2	526	191	90112	430

Poorly clustered

Well clustered



# An Approach - Print the Statement

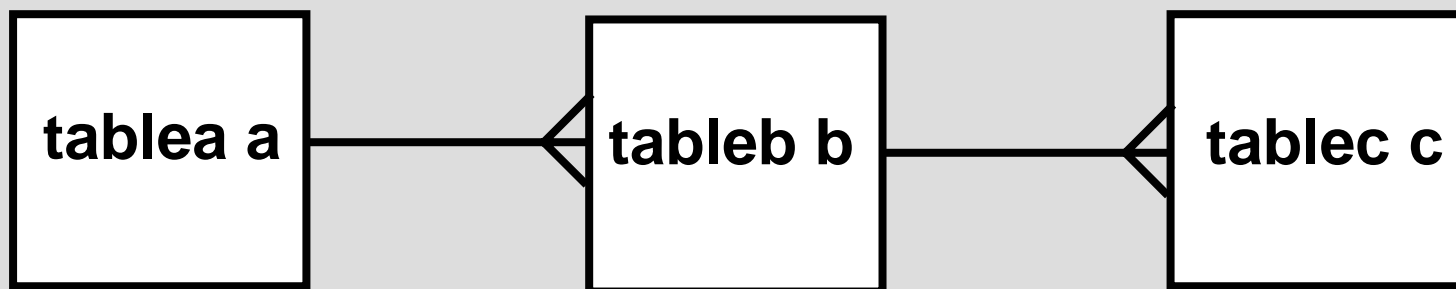
```
SELECT      a.col1, b.col2, b.col3, c.col3  
FROM        tablea a, tableb b, tablec c  
WHERE       a.col1 = b.col1  
AND         b.col2 = c.col2  
AND         a.col4 = 'charvalue'  
AND         c.col5 = numvalue  
AND         c.col6 between :p_1 and :p_2;
```

# Draw the Model

Write in the join conditions

$a.col1 = b.col1$

$b.col2 = c.col2$



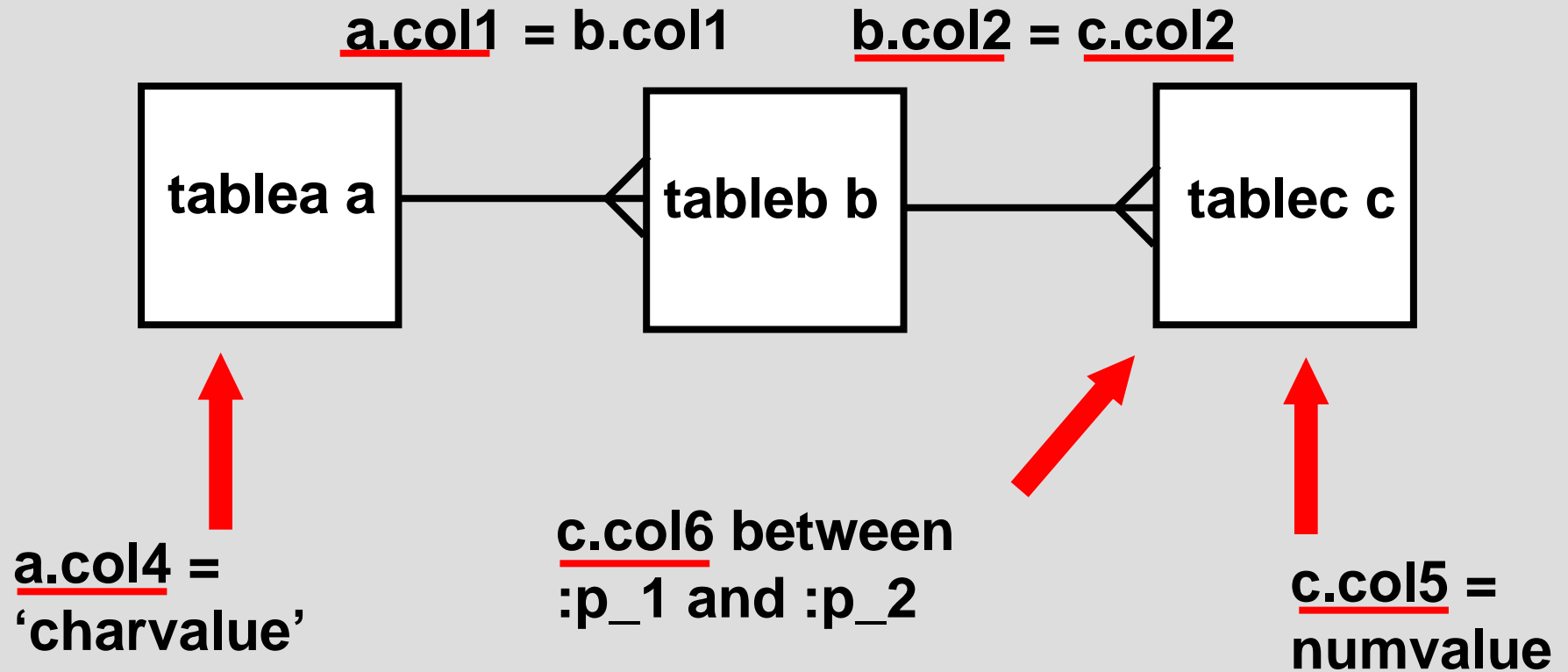
$a.col4 =$   
'charvalue'

$c.col6$  between  
:p\_1 and :p\_2

$c.col5 =$   
numvalue

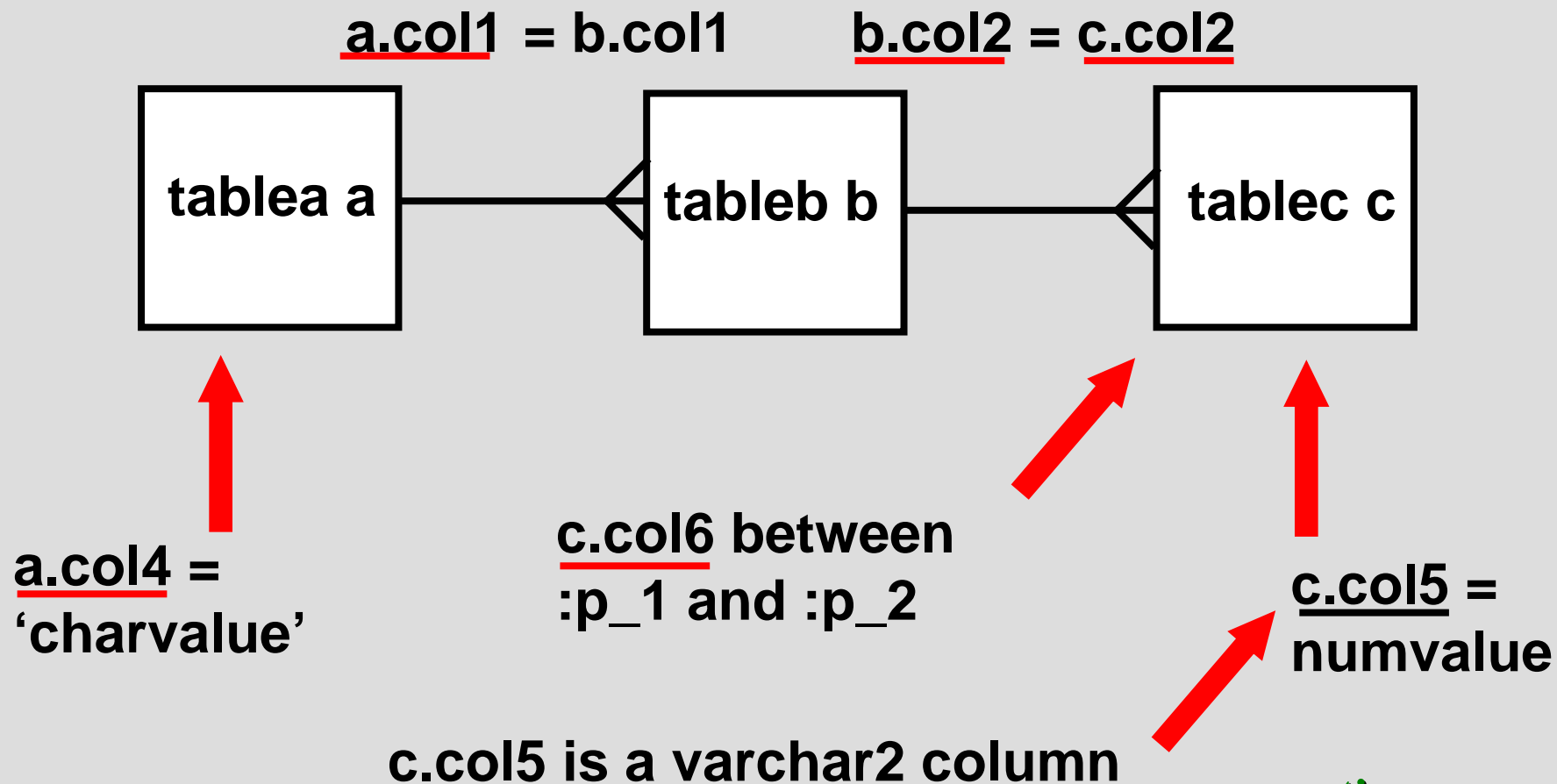
Write in the access points

# Underline Indexed Columns

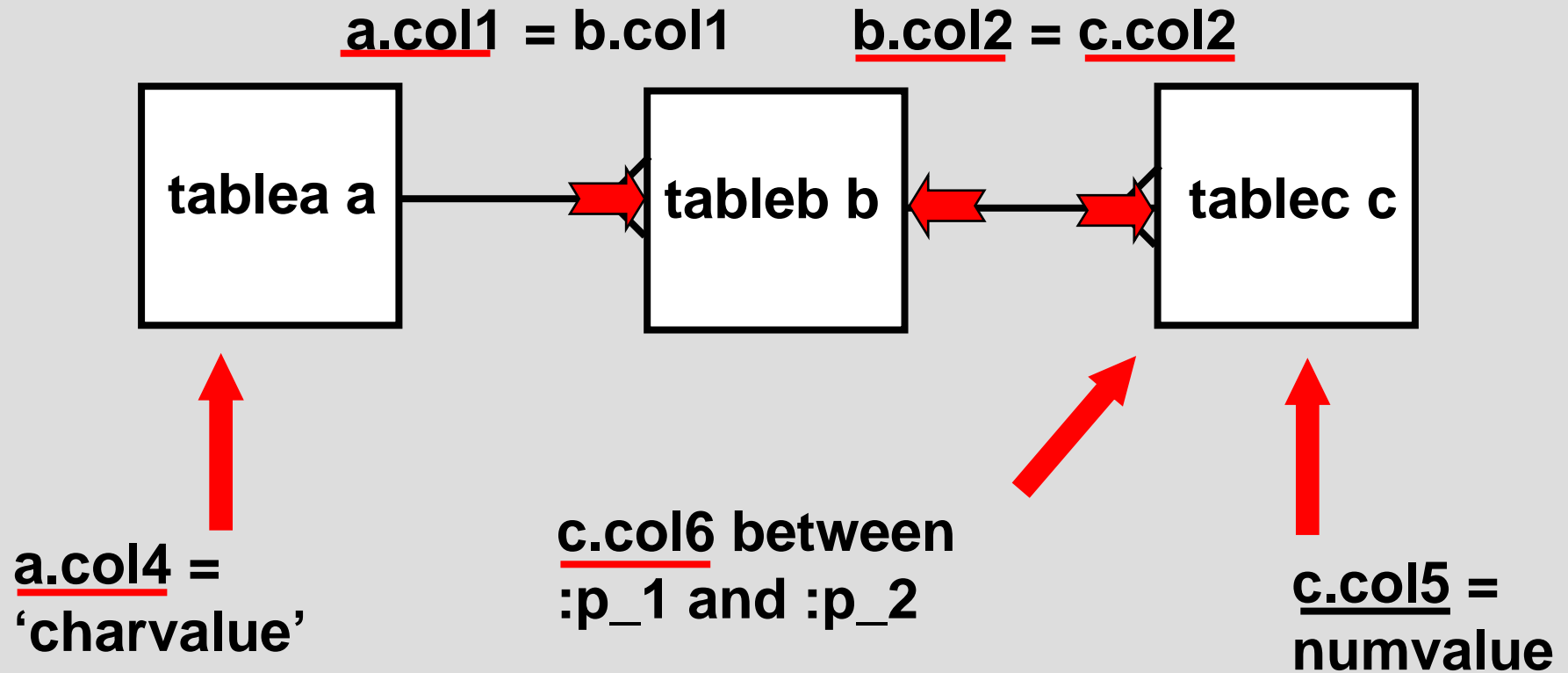




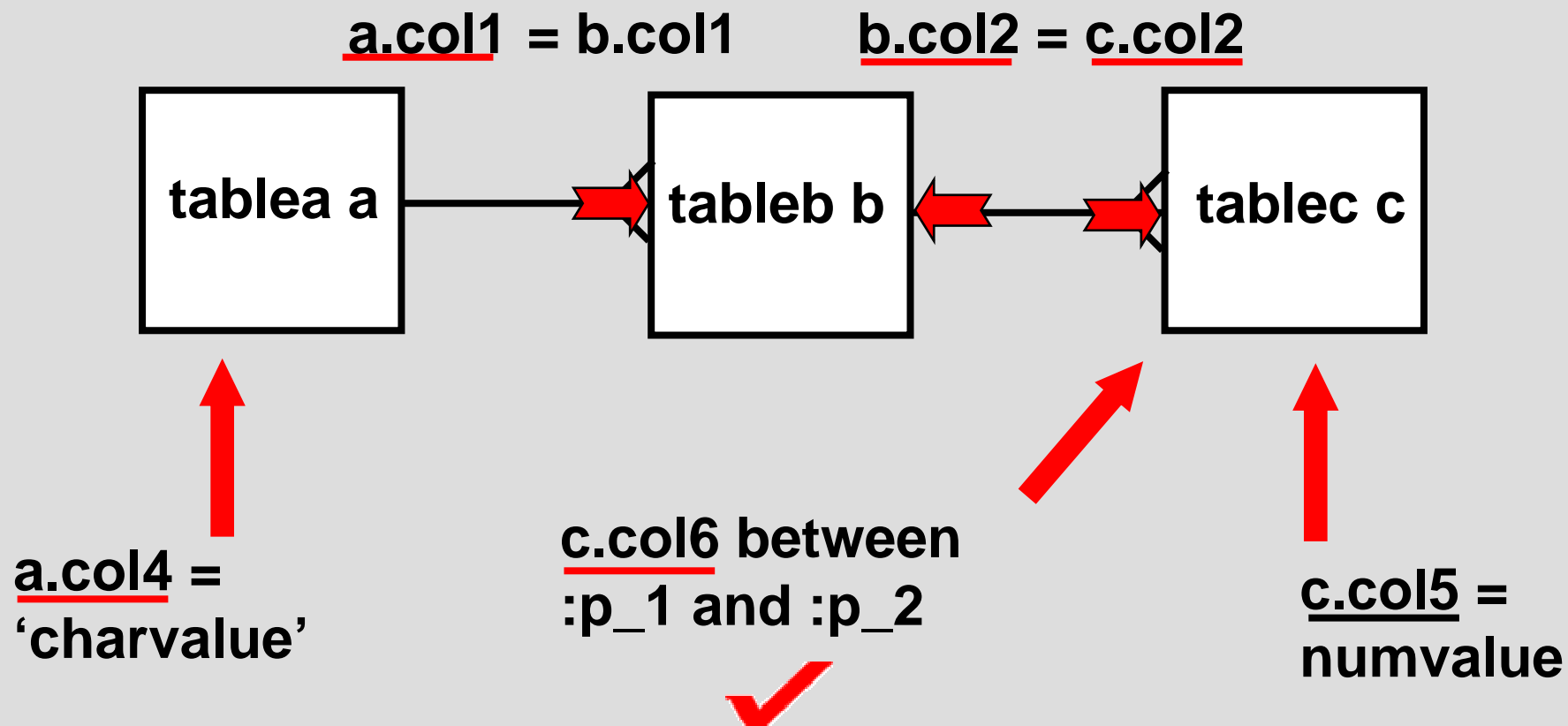
# Remove Unusable Indexes



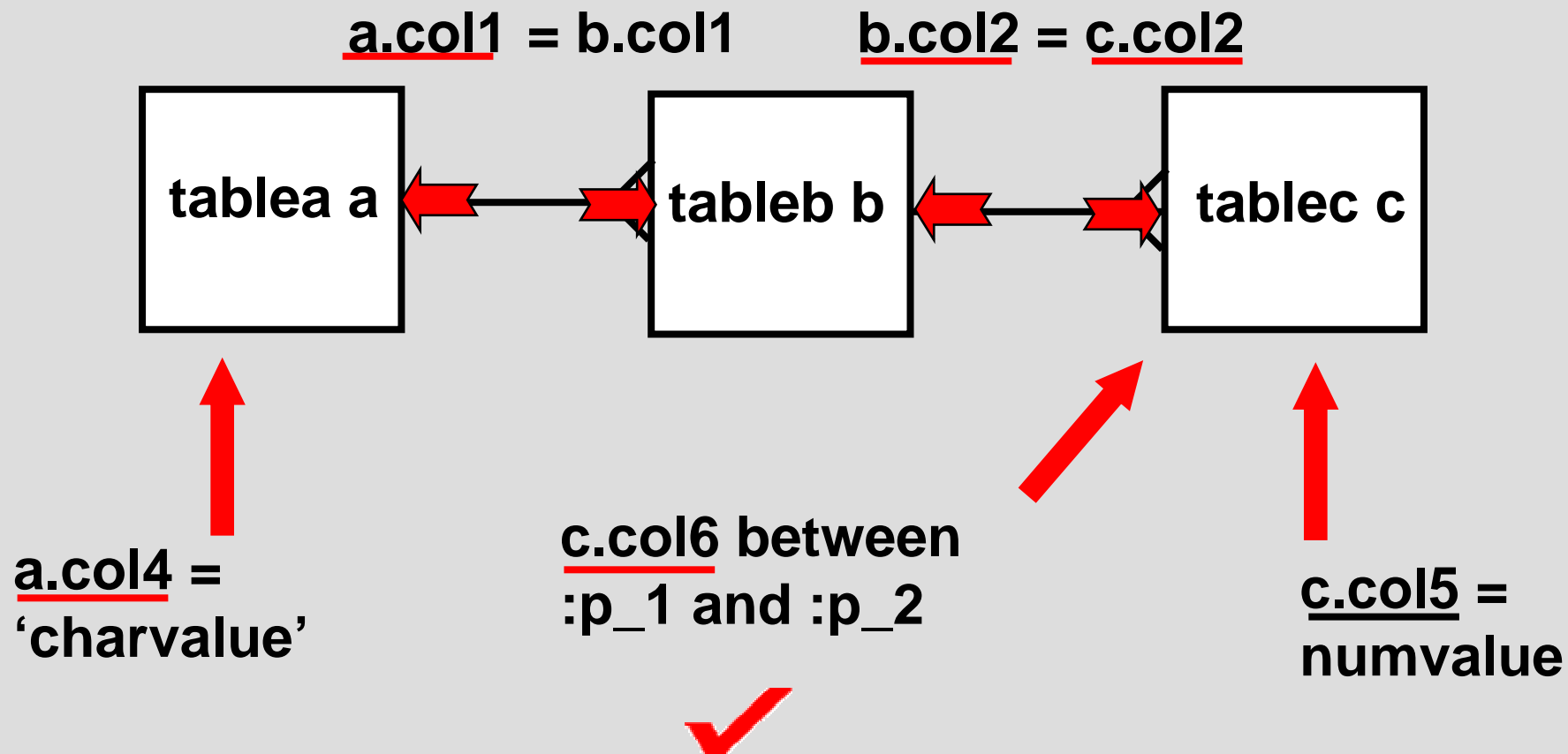
# Mark Possible Indexed Join Directions



# Determine Selectivity



# Add any Required Indexes

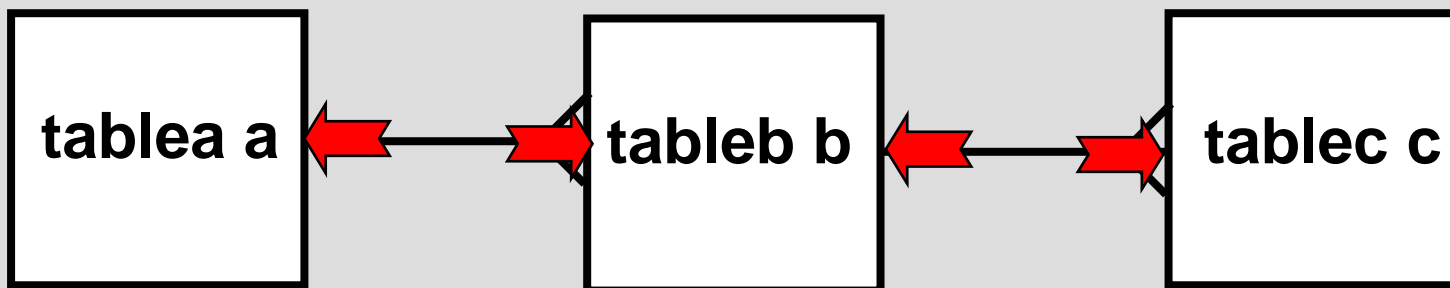


# Determine the Hints

```
SELECT /*+ ORDERED USE_NL (TABLEB TABLEA)
        INDEX(c (col6)) */
FROM TABLEC, TABLEB, TABLEA
.....
```

a.col1 = b.col1

b.col2 = c.col2



a.col4 =  
'charvalue'

c.col6 between  
:p\_1 and :p\_2



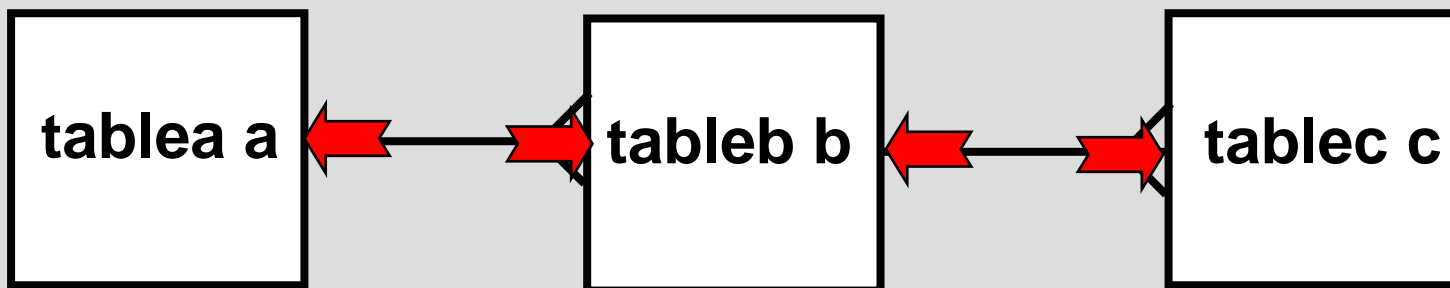
c.col5 =  
numvalue

# Determine the Hints

```
SELECT /*+ ORDERED USE_HASH (TABLEB TABLEA)
        INDEX(c (col6)) */
FROM TABLEC, TABLEB, TABLEA
.....
```

a.col1 = b.col1

b.col2 = c.col2



a.col4 =  
'charvalue'

c.col6 between  
:p\_1 and :p\_2

c.col5 =  
numvalue

# Common SQL tuning issues



# Bad Index Use

```
SELECT  MIN(event_no), MAX(event_no),  
        COUNT(booking_no)  
FROM    bookings_large
```

MIN(EVENT_NO)	MAX(EVENT_NO)	COUNT(BOOKING_NO)
200	1555	11258



# Bad Index Use

**EXPLAIN PLAN FOR**  
**SELECT \* FROM bookings\_large**  
**WHERE event\_no < 1500**

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		10807	443K	10839 (1)	00:02:11
1	TABLE ACCESS BY INDEX ROWID	BOOKINGS_LARGE	10807	443K	10839 (1)	00:02:11
* 2	INDEX RANGE SCAN	BK_EVT5	10807		25 (0)	00:00:01

Predicate Information (identified by operation id):

2 - access("EVENT\_NO"<1500)

# Bad Index Use

- ➡ Optimizer knows the index scan is not the cheapest plan

```
*****
SINGLE TABLE ACCESS PATH
  COLUMN:  EVENT_NO(NUMBER)  Col#: 2      Table: BOOKINGS_LARGE  Alias: BOOKINGS_LARGE
  Size: 4  NDV: 1356  Nulls: 0  Density: 7.3746e-004  Min: 200  Max: 1555
  No Histogram: #BKT: 1
    (1 uncompressed buckets and 2 endpoint values)
  TABLE: BOOKINGS_LARGE  Alias: BOOKINGS_LARGE
  Original Card: 11264  Rounded Card: 10807  Computed Card: 10806.79
  Access Path: table-scan  Rsc: 17  Resp: 17
  Access Path: index (scan)
    Index: BK_EVT5
    rsc_cpu: 82860027  rsc_io: 10831
    ix_sel: 9.5941e-001  ix_sel_with_filters: 9.5941e-001
  BEST_CST: 10838.63  PATH: 4  Degree: 1
*****
OPTIMIZER STATISTICS AND COMPUTATIONS
*****
GENERAL PLANS
*****
Join order[1]:  BOOKINGS_LARGE[BOOKINGS_LARGE]#0
Best so far: TABLE#: 0  CST: 10839  CDN: 10807  BYTES: 453894
(newjo-stop-1) k:0, spcnt:0, perm:1, maxperm:80000
prefetching is on for BK_EVT5
Final - First Rows Plan:
  JOIN ORDER: 1
  CST: 10839  CDN: 10807  RSC: 10839  RSP: 10839  BYTES: 453894
  IO-RSC: 10831  IO-RSP: 10831  CPU-RSC: 82860027  CPU-RSP: 82860027
*****
```

**FIRST\_ROWS**



# ALL\_ROWS

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		10807	443K	17 (0)	00:00:01
* 1	TABLE ACCESS FULL	BOOKINGS_LARGE	10807	443K	17 (0)	00:00:01

Predicate Information (identified by operation id):

1 - filter("EVENT\_NO"<1500)

## SINGLE TABLE ACCESS PATH

COLUMN: EVENT\_NO(NUMBER) Col#: 2 Table: BOOKINGS\_LARGE Alias: BOOKINGS\_LARGE  
Size: 4 NDV: 1356 Nulls: 0 Density: 7.3746e-004 Min: 200 Max: 1555

No Histogram: #BKT: 1

(1 uncompressed buckets and 2 endpoint values)

TABLE: BOOKINGS\_LARGE Alias: BOOKINGS\_LARGE

Original Card: 11264 Rounded Card: 10807 Computed Card: 10806.79

Access Path: table-scan Rsc: 17 Rsp: 17

Access Path: index (scan)

Index: BK\_EVTS

rsc\_cpu: 82860027 rsc\_io: 10831

ix\_sel: 9.5941e-001 ix\_sel\_with\_filters: 9.5941e-001

BEST\_CST: 17.43 PATH: 2 Degree: 1

\*\*\*\*\*

## OPTIMIZER STATISTICS AND COMPUTATIONS

\*\*\*\*\*

## GENERAL PLANS

\*\*\*\*\*

Join order[1]: BOOKINGS\_LARGE[BOOKINGS\_LARGE]#0

Best so far: TABLE#: 0 CST: 17 CDN: 10807 BYTES: 453894

(newjo-stop-1) k:0, spcnt:0, perm:1, maxperm:80000

Final - All Rows Plan:

JOIN ORDER: 1

CST: 17 CDN: 10807 RSC: 17 RSP: 17 BYTES: 453894

IO-RSC: 17 IO-RSP: 17 CPU-RSC: 4712822 CPU-RSP: 4712822

# ALL\_ROWS

# FIRST\_ROWS\_N

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		2	84	2 (0)	00:00:01
* 1	TABLE ACCESS FULL	BOOKINGS_LARGE	2	84	2 (0)	00:00:01

Predicate Information (identified by operation id):

1 - filter("EVENT\_NO"<1500)

```
*****
Join order[1]: BOOKINGS_LARGE[BOOKINGS_LARGE]#0
Best so far: TABLE#: 0 CST: 17 CDN: 10815 BYTES: 454230
*****
```

```
SINGLE TABLE ACCESS PATH (First K Rows)
TABLE: BOOKINGS_LARGE Alias: BOOKINGS_LARGE
Original Card: 2 Rounded Card: 2 Computed Card: 2.00
Access Path: table-scan Resc: 2 Resp: 2
Access Path: index (scan)
Index: BK_EVT5
rsc_cpu: 29546 rsc_io: 4
ix_sel: 1.0000e+000 ix_sel_with_filters: 1.0000e+000
BEST_CST: 2.00 PATH: 2 Degree: 1
First K Rows: unchanged join prefix len = 1
*****
```

```
Join order[1]: BOOKINGS_LARGE[BOOKINGS_LARGE]#0
Best so far: TABLE#: 0 CST: 2 CDN: 2 BYTES: 84
(newjo-stop-1) k:0, spcnt:0, perm:1, maxperm:80000
Final - First K Rows Plan:
JOIN ORDER: 1
CST: 2 CDN: 2 RSC: 2 RSP: 2 BYTES: 84
IO-RSC: 2 IO-RSP: 2 CPU-RSC: 7881 CPU-RSP: 7881
```

## FIRST\_ROWS\_1

# Ignore Cost in Explain

**This statement is slow.**

**I've tried Explain with  
other access paths but  
they all have a higher cost**





# Ignore Cost in Explain

- ➡ The optimiser is going to use the plan which it thinks has the lowest cost
- ➡ If the optimiser has got it right we don't need to Explain
- ➡ If its wrong we need to ignore what it thinks

# Ignore Cost in Explain

```

Document - WordPad
File Edit View Insert Format Help

SQL>
1 SELECT count(made_by)
2 FROM bookings_large b
3* WHERE resource_code = 'TAP1';

COUNT(MADE_BY)
-----
239

Execution Plan
-----
0  SELECT STATEMENT Optimizer=FIRST_ROWS (Cost=8 Card=1 Bytes=11)
1  0  SORT (AGGREGATE)
2  1  TABLE ACCESS (FULL) OF 'BOOKINGS_LARGE' (Cost=8 Card=1252 Bytes=13772)

Statistics
-----
0 recursive calls
0 db block gets
68 consistent gets
0 physical reads
0 redo size
386 bytes sent via SQL*Net to client
499 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from clientSQL>
1 SELECT count(made_by)
0 sorts (memory)
0 sorts (disk)
1 rows processed

```

# Ignore Cost in Explain

```

Document - WordPad
File Edit View Insert Format Help

SQL> SELECT /*+INDEX (b BKLG_RES) */
2 count(made_by)
3 FROM bookings_large b
4* WHERE resource_code = 'TAP1';

COUNT(MADE_BY)
-----
10

Execution Plan
-----
0  SELECT STATEMENT Optimizer=FIRST_ROWS (Cost=38 Card=1 Bytes=14)
1  0  SORT (AGGREGATE)
2  1  TABLE ACCESS (BY INDEX ROWID) OF 'BOOKINGS_LARGE' (Cost=38 Card=1252 Bytes=17528)
3  2  INDEX (RANGE SCAN) OF 'BKLG_RES' (NON-UNIQUE) (Cost=3 Card=1252)

Statistics
-----
0 recursive calls
0 db block gets
3 consistent gets
0 physical reads
0 redo size
386 bytes sent via SQL*Net to client
499 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1 rows processed

```





# OPTIMIZER\_MODE

- ➡ Use **FIRST\_ROWS\_N** for OLTP
- ➡ Use **ALL\_ROWS** for Reporting/Batch
- ➡ Only use **FIRST\_ROWS** to force index use when Oracle doesn't want to



# Tuning Functions

- ➡ Database functions in SQL executed once per row in row set
- ➡ Reduce row set before executing functions



# Avoid PL/SQL

- ➡ Use set processing
- ➡ Avoid row processing
- ➡ For cursor loop to insert or update 12,000 rows = 1759 secs
- ➡ Equivalent UPDATE and INSERT = 79 secs

# Bitmapped Indexes

Index on Gender  
column



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

Index on Country  
column

Row	NZ	AUS	USA
1	1	0	0
2	0	1	0
3	0	1	0
4	1	0	0
5	0	0	1
6	0	0	0

**GENDER = 'M' and COUNTRY = 'NZ'**

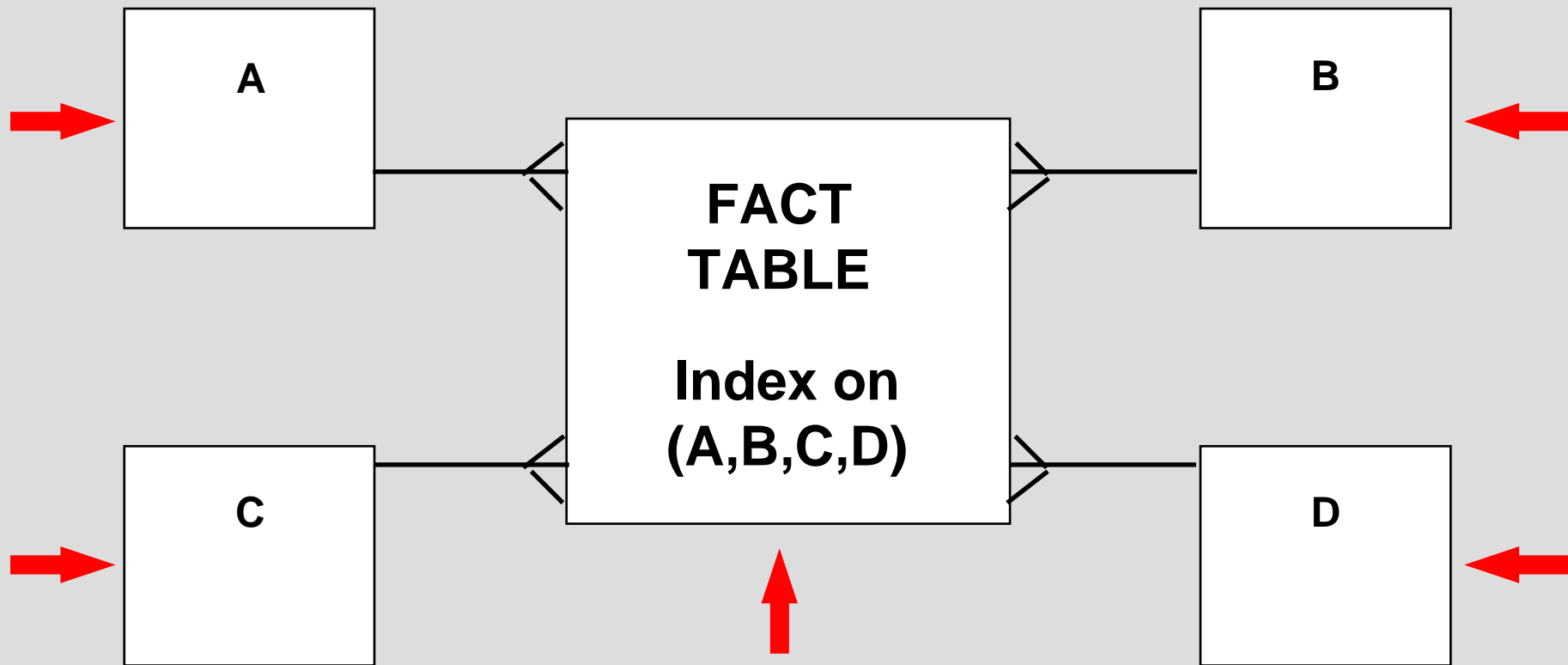


# Bitmap Indexes

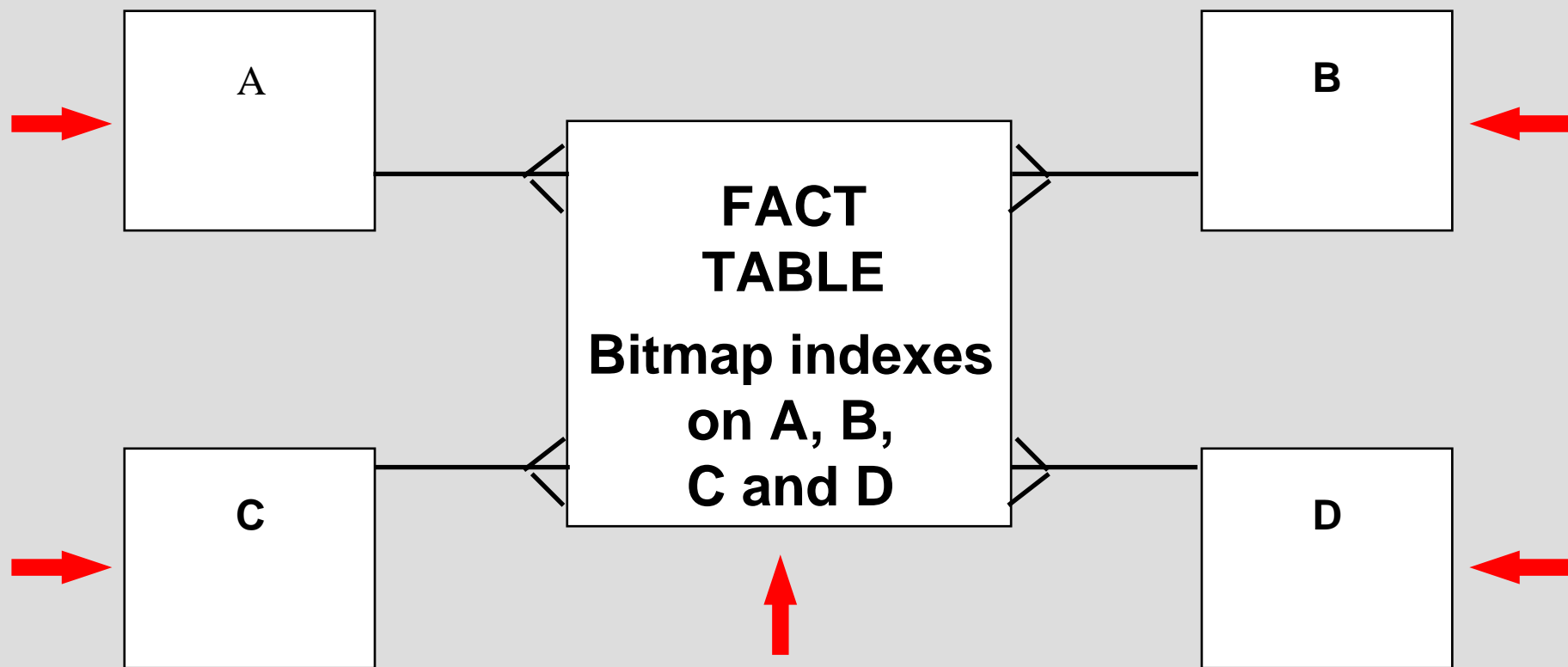
- ➡ Significantly less space
- ➡ Use for DWH applications
- ➡ Not for OLTP
- ➡ Reduce storage
- ➡ Higher level of locking Contains null values
- ➡ `STAR_TRANSFORMATION= TRUE`

# DSS Queries

- ➡ Star schemas
- ➡ Large number of dimensions
- ➡ Unknown queries



# STAR Transformation



**Convert each dimension condition to a subquery**  
**Access the fact table using bitmap indexes**



# Thank You For Your Attention

**SAGE Computing Services**  
**Customised Oracle Training Workshops**  
**and Consulting**  
**[www.sagecomputing.com.au](http://www.sagecomputing.com.au)**

Enquiries@[sagecomputing.com.au](http://sagecomputing.com.au)

**[www.sagecomputing.com.au](http://www.sagecomputing.com.au)**