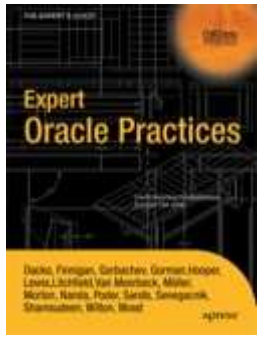
Execution Plan Interpretation

Jože Senegačnik
Oracle ACE Director
joze.senegacnik@dbprof.com

About the Speaker

Jože Senegačnik

- Registered private researcher
- First experience with Oracle Version 4 in 1988
- 21+ years of experience with Oracle RDBMS.
- Proud member of the OakTable Network www.oaktable.net
- Oracle ACE Director
- Co-author of the OakTable book “Expert Oracle Practices” by Apress (Jan 2010)
- VP of Slovenian OUG (SIOUG) board
- CISA – Certified IS auditor
- Blog about Oracle: http://joze-senegacnik.blogspot.com



- PPL(A) – private pilot license / night qualified
- Blog about flying: http://jsenegacnik.blogspot.com
- Blog about Building Ovens, Baking and Cooking: http://senegacnik.blogspot.com

Agenda

- How to find Execution Plan
- Execution Plan Interpretation
- SQL Monitor Feature (11g)

Explain Plan Utility

```
explain plan for select * from my_table;
```

- Result stored in PLAN_TABLE which is Oracle version dependant!
- $ORACLE_HOME/rdbms/admin/utlxplan.sql creates PLAN_TABLE
- Use DBMS_XPLAN package utility to format results of a explain plan
- Autotrace uses the same package since 10.2

```
select * from table(dbms_xplan.display);
```

DBMS_XPLAN

- Displaying explain plan from PLAN_TABLE

```
DBMS_XPLAN.DISPLAY( table_name IN VARCHAR2 DEFAULT 'PLAN_TABLE',
                    statement_id IN VARCHAR2 DEFAULT NULL,
                    format IN VARCHAR2 DEFAULT 'TYPICAL',
                    filter_preds IN VARCHAR2 DEFAULT NULL);
```

- To display a statement's plan from AWR

```
DBMS_XPLAN.DISPLAY_AWR( sql_id IN VARCHAR2,
                        plan_hash_value IN NUMBER DEFAULT NULL,
                        db_id IN NUMBER DEFAULT NULL,
                        format IN VARCHAR2 DEFAULT TYPICAL);
```

DBMS_XPLAN (2)

- Displaying from cursor cache (VSQL, VSQL_PLAN)
- We can obtain different run-time statistics like IOSTATS, MEMSTATS, or statistics for the LAST run only.
- Use hint **/*+ gather_plan_statistics */**

```
DBMS_XPLAN.DISPLAY_CURSOR( sql_id IN VARCHAR2 DEFAULT NULL,
                           child_number IN NUMBER DEFAULT NULL,
                           format IN VARCHAR2 DEFAULT 'TYPICAL');
```

DBMS_XPLAN(3)

- Used to display the execution plan of a given statement stored in a SQL tuning set.
- Use DBMS_SQLTUNE.SELECT_SQLSET to get SQL_IDs of SQL statements in SQLSet.

```
DBMS_XPLAN.DISPLAY_SQLSET(sqlset_name IN VARCHAR2,
                          sql_id IN VARCHAR2,
                          plan_hash_value IN NUMBER := NULL,
                          format IN VARCHAR2 := 'TYPICAL',
                          sqlset_owner IN VARCHAR2 := NULL);
```

DBMS_XPLAN(4) – 11g

- Used to display one or more execution plans for the specified SQL handle of a SQL plan baseline

```
DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE (
              sql_handle IN VARCHAR2 := NULL,
              plan_name IN VARCHAR2 := NULL,
              format IN VARCHAR2 := 'TYPICAL');
```

Execution Plan Differences (1)

```
dbms_xplan.diff_plan_cursor(
      sql_id              IN VARCHAR2,
      cursor_child_num1 IN NUMBER,
      cursor_child_num2 IN NUMBER)
  RETURN VARCHAR2;

dbms_xplan.diff_plan_awr(
      sql_id IN VARCHAR2,
      plan_hash_value1 IN NUMBER,
      plan_hash_value2 IN NUMBER)
  RETURN VARCHAR2;
```

Execution Plan Differences (2)

- ```
  dbms_xplan.diff_plan(
      sql_text  IN CLOB,
      outline   IN CLOB,
      user_name IN VARCHAR2 := NULL)
  RETURN VARCHAR2;
  ```

- ```
  dbms_xplan.diff_plan_outline(
      sql_text  IN CLOB,
      outline1  IN CLOB,
      outline2  IN CLOB,
      user_name IN VARCHAR2 := NULL)
  RETURN VARCHAR2;
  ```

DBMS_XPLAN Parameters

- DBMS_XPLAN format parameter: (basic, typical, serial, all, advanced)
- DBMS_XPLAN format modifiers: (alias,bytes,cost,outline,parallel,partition, peeked_binds,predicate,projection,remote,rows)

```
select * from table(
   DBMS_XPLAN.DISPLAY_CURSOR
     ('8wccfj60yh0us',null,'TYPICAL +OUTLINE')
 );
```

Event 10053 trace

- Event 10053 switches on CBO trace

```
SQL> alter session set events
     '10053 trace name context forever, level 1';
```

- Useful to find out why CBO has chosen certain execution plan.
- Hard to interpret – no tools available!
- See Metalink Note:338137.1 "CASE STUDY: Analyzing 10053 Trace Files" for explanation
- Jonathan Lewis: "Cost Based Oracle"
- Wolfgang Breitling's site http://www.centrexcc.com

Optimizer trace

- Alternative way for CBO tracing in Oracle10g - _optimizer_trace parameter

```
SQL> alter session set "_optimizer_trace" = <value>
```

- Possible values are:
  - HINT
  - ENVIRONMENT
  - PHYSICAL
  - LOGICAL
  - MEDIUM
  - NONE
  - HIGH
  - ALL
  - LOW

Interpreting Execution Plans

Old Instructions For Plan Interpretation

- We start to interpret execution plan from right most operation and from the bottom to top.

```
SQL> explain plan for select * from t1 where c1=2000;
Explained.
SQL> select * from table(dbms_xplan.display());
-------------------------------------------------------------------------
| Id  | Operation                    |  Name | Rows | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |       |    1 |    56 |   13   (8)|
|   1 |  TABLE ACCESS BY INDEX ROWID | T1    |    1 |    56 |   13   (8)|
|*  2 |   INDEX SKIP SCAN            | T1_I1 |    1 |       |   12   (9)|
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("T1"."C1"=2000)
       filter("T1"."C1"=2000)
```

Interpretation Rules (1)

RULE 1:

```
Id  PId Execution path
N       This action happens second
N+1  N    This action sends result to the action above and left
```

Example:

```
-------------------------------------------------------------------
| Id |PId| Operation                                | Name   | Rows  |
-------------------------------------------------------------------
|   0|   | SELECT STATEMENT                         |        |     1 |
|   1|  0|  TABLE ACCESS BY INDEX ROWID             | T3     |     1 |
|   2|  1|   INDEX UNIQUE SCAN                      | T3_I1  |     1 |
-------------------------------------------------------------------
```

Interpretation Rules (2)

RULE 2:

```
ID     PID POS Execution path
N              Description of combining the data sets
N+1     N   1    Row source 1
N+x     N   2    Row source 2
...
N+y     N   z    Row source n

EXAMPLE:
-----------------------------------------------
| Id  | Operation              | Name   | Rows  |
-----------------------------------------------
|   0 | SELECT STATEMENT       |        |   918K|
|*  1 |  HASH JOIN             |        |   918K|
|   2 |   INDEX FAST FULL SCAN| T1_I1  |   918K|
|   3 |   INDEX FAST FULL SCAN| T2_I1  |   918K|
-----------------------------------------------
```

Interpretation Rules (3)

- RULE 2 interpretation instructions:
  - no automatic interpretation
  - interpretation depends on how the parent operation combines children (see next slide)
  - Each child may be executed to completion

    OR
  - it may get a row from child 1 and then execute child 2 etc. And then get next row from child 1
  - Execution of child 2 or 3 (...) may be omitted if not required.
  - Parent operation behaves according to the circumstances.

Interpretation Example (1)

SQL Statement:

```
SELECT featureid, geometry FROM
  (SELECT izpp.ID featureid, grkx.raba_id, grkx.gerk_pid,
                sbv_gerk_util.url_encode( ste_export.ret_val_intersect_ (izpp.ID, 'E') ) raba_ukrepi,
                sdo_geom.sdo_intersection (izpp.geometry, :1, 0.0005) geometry
     FROM izm_poligon_poljin izpp,
                izm_poligon_zahtevki izpz,
                sbv_gk_e gk
    WHERE izpp.obrazec = 'E'
         AND izpz.izpp_id = izpp.ID
         AND izpz.sifra_ukrepa = 'SOR'
         AND gk.ID = izpp.gk_id
         AND sdo_relate (izpp.geometry, :2,  'mask=CONTAINS+COVERS') = 'TRUE')
```

Interpretation Example (2)

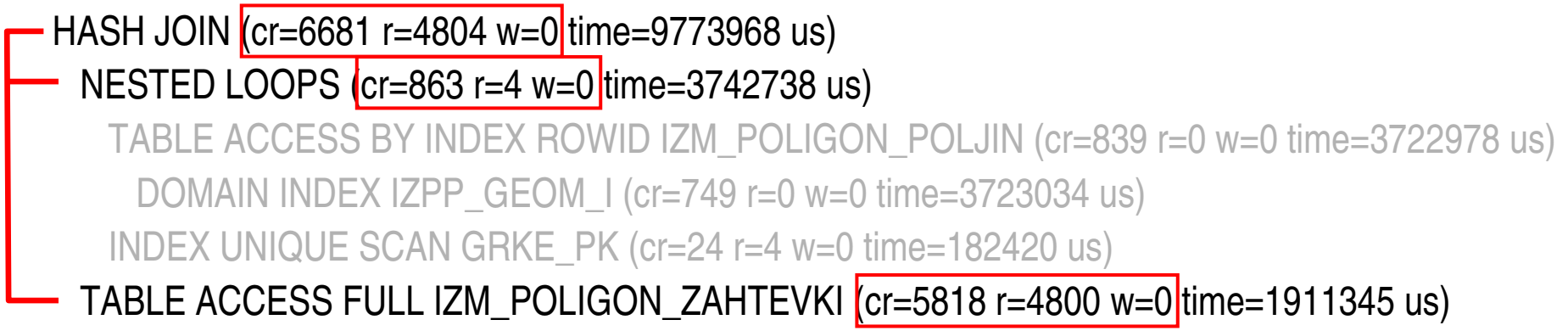
Execution Plan:

```
HASH JOIN (cr=6681 r=4804 w=0 time=9773968 us)
  NESTED LOOPS (cr=863 r=4 w=0 time=3742738 us)
    TABLE ACCESS BY INDEX ROWID IZM_POLIGON_POLJIN (cr=839 r=0 w=0 time=3722978 us)
      DOMAIN INDEX IZPP_GEOM_I (cr=749 r=0 w=0 time=3723034 us)
    INDEX UNIQUE SCAN GRKE_PK (cr=24 r=4 w=0 time=182420 us)
  TABLE ACCESS FULL IZM_POLIGON_ZAHTEVKI (cr=5818 r=4800 w=0 time=1911345 us)
```

- cr = consistent reads
- r = physical reads
- w = physical writes
- time = time elapsed in microseconds (0.000001s)

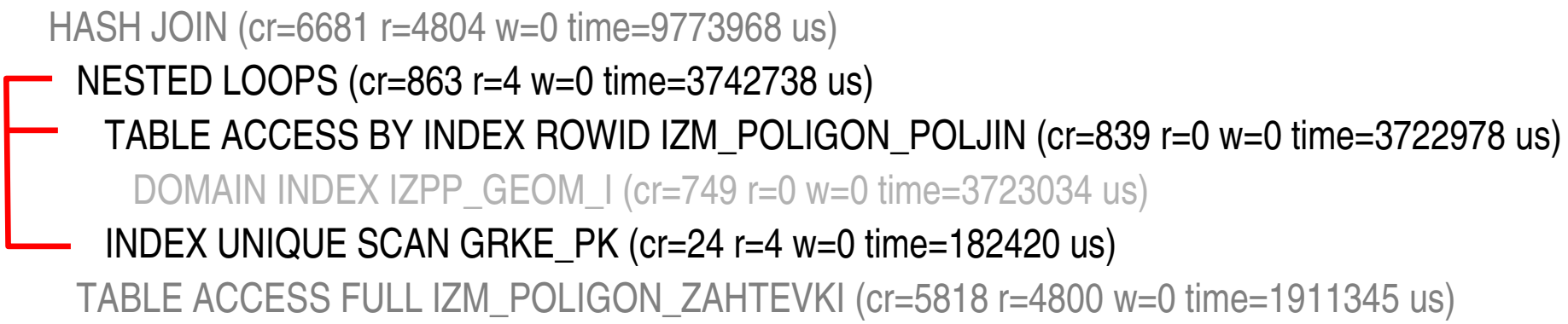
- The values are reported cumulatively for the current and all previous steps.

Interpretation Example (3)



- Interpretation Step 1: We apply Rule 2
- cr, r and w values for parent operation is sum of child operations values

Interpretation Example (4)



- Interpretation Step 2: We apply Rule 2

Interpretation Example (5)

```
HASH JOIN (cr=6681 r=4804 w=0 time=9773968 us)
  NESTED LOOPS (cr=863 r=4 w=0 time=3742738 us)
    TABLE ACCESS BY INDEX ROWID IZM_POLIGON_POLJIN (cr=839 r=0 w=0 time=3722978 us)
      DOMAIN INDEX IZPP_GEOM_I (cr=749 r=0 w=0 time=3723034 us)
    INDEX UNIQUE SCAN GRKE_PK (cr=24 r=4 w=0 time=182420 us)
  TABLE ACCESS FULL IZM_POLIGON_ZAHTEVKI (cr=5818 r=4800 w=0 time=1911345 us)
```

- Interpretation Step 3: We apply Rule 1

Interpretation Example (6)

```
SQL>   select count(*) from (
  2    select /*+ no_merge */ id
  3    from t3
  4*   order by id )
```

```
-----------------------------------------------------------------
| Id  | Operation                | Name  | Rows  | Cost (%CPU)|
-----------------------------------------------------------------
|   0 | SELECT STATEMENT         |       |     1 |   543   (2)|
|   1 |  SORT AGGREGATE          |       |     1 |            |
|   2 |   VIEW                   |       |   918K|   543   (2)|
|   3 |    INDEX FAST FULL SCAN| T3_I1 |   918K|   543   (2)|
-----------------------------------------------------------------
```

- ORDER BY operation is missing – CBO sees that the operation can be omitted

Optional Steps In Execution Plan

```
select * from t3 where 1 != 1

-----------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |     1 |   108 |     0   (0)|
|*  1 |  FILTER             |      |       |       |            |
|   2 |   PARTITION HASH ALL|      |   918K|    94M|  3318   (1)|
|   3 |    TABLE ACCESS FULL| T3   |   918K|    94M|  3318   (1)|
-----------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(NULL IS NOT NULL)

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

NOT EXECUTED AT ALL!!

Major Types Of Operations

- Christian Antognini in his book defines three major types of operations:
  - Standalone operations
  - Unrelated-combine operations
  - Related-combine operations

Stand-alone operations (1)

- Standalone operations have at most one child
- Most operations are of this type.
- The rules governing the working of these operations are the following:
  - Children are executed before their parents. (there are some exeptions)
  - Every child is executed at most once.
  - Every child feeds its parent.

Stand-alone Operation (2)

```
SQL> explain plan for
  2  select * from t2
  3  where  c1 = 1201;

-------------------------------------------------------------------------------
| Id  | Operation                    | Name  | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |       |     1 |    25 |     2   (0)|
|   1 |  TABLE ACCESS BY INDEX ROWID| T2    |     1 |    25 |     2   (0)|
|*  2 |   INDEX RANGE SCAN           | T2_I1 |     1 |       |     1   (0)|
-------------------------------------------------------------------------------

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

   2 - access("C1"=1201)
```

Unrelated-combine operations (1)

- Operations having multiple children that are independently executed are unrelated-combine operations.
- Some typical operations of this type: HASH JOIN, INTERSECTION, MERGE JOIN, MINUS, UNION-ALL, AND-EQUAL, BITMAP AND, BITMAP OR, BITMAP MINUS, CONCATENATION, CONNECT BY WITHOUT FILTERING,, MULTI-TABLE INSERT, SQL MODEL, TEMP TABLE TRANSFORMATION
- The characteristics of these operations are the following:
  - Children are executed before their parents.
  - Children are executed sequentially.
  - Every child is executed at most once and independently of the others.
  - Every child feeds its parent.

Unrelated-combine operations (2)

```
SQL> explain plan for
  2  select t1.c1, t1.c2, t2.c2
  3  from t1, t2
  4  where  t1.c1 = t2.c1
  5  and    t1.c1 between 1501 and 1600;

-------------------------------------------------------------------------------
| Id  | Operation                     | Name  | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |       |   100 |  1200 |     7  (15)|
|*  1 |  HASH JOIN                    |       |   100 |  1200 |     7  (15)|
|   2 |   TABLE ACCESS BY INDEX ROWID| T1    |   101 |   606 |     3   (0)|
|*  3 |    INDEX RANGE SCAN           | T1_C1 |   101 |       |     2   (0)|
|   4 |   TABLE ACCESS BY INDEX ROWID| T2    |   101 |   606 |     3   (0)|
|*  5 |    INDEX RANGE SCAN           | T2_C1 |   101 |       |     2   (0)|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("T1"."C1"="T2"."C1")
   3 - access("T1"."C1">=1501 AND "T1"."C1"<=1600)
   5 - access("T2"."C1">=1501 AND "T2"."C1"<=1600)
```

Related-Combine Operations (1)

- Operations with multiple children where one of the children controls the execution of all other children are related-combine operations.
- The most common operation of this type are NESTED LOOPS and FILTER
- Characteristics:
  - Children are executed before their parents.
  - The child with the smallest id controls the execution of the other children.
  - Children are not executed sequentially. Instead, a kind of interleaving is performed.
  - Only the first child is executed at most once. All other children may be executed several times or not executed at all.
  - Not every child feeds its parent.

Related-Combine Operations – Nested Loop

```
SQL> explain plan for
  2  select t1.c2,t2.c2
  3  from t1,t2
  4  where t1.c1 = t2.c1(+)
  5  and t1.c1 in (100, 101, 103);

-------------------------------------------------------------------------------
| Id  | Operation                     | Name  | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |       |     3 |    36 |     8   (0)|
|   1 |  NESTED LOOPS OUTER           |       |     3 |    36 |     8   (0)|
|   2 |   INLIST ITERATOR             |       |       |       |            |
|   3 |    TABLE ACCESS BY INDEX ROWID| T1    |     3 |    18 |     5   (0)|
|*  4 |     INDEX RANGE SCAN          | T1_C1 |     3 |       |     4   (0)|
|   5 |   TABLE ACCESS BY INDEX ROWID | T2    |     1 |     6 |     1   (0)|
|*  6 |    INDEX UNIQUE SCAN          | T2_C1 |     1 |       |     0   (0)|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("T1"."C1"=100 OR "T1"."C1"=101 OR "T1"."C1"=103)
   6 - access("T1"."C1"="T2"."C1"(+))
```

V$SQL_PLAN_STATISTICS

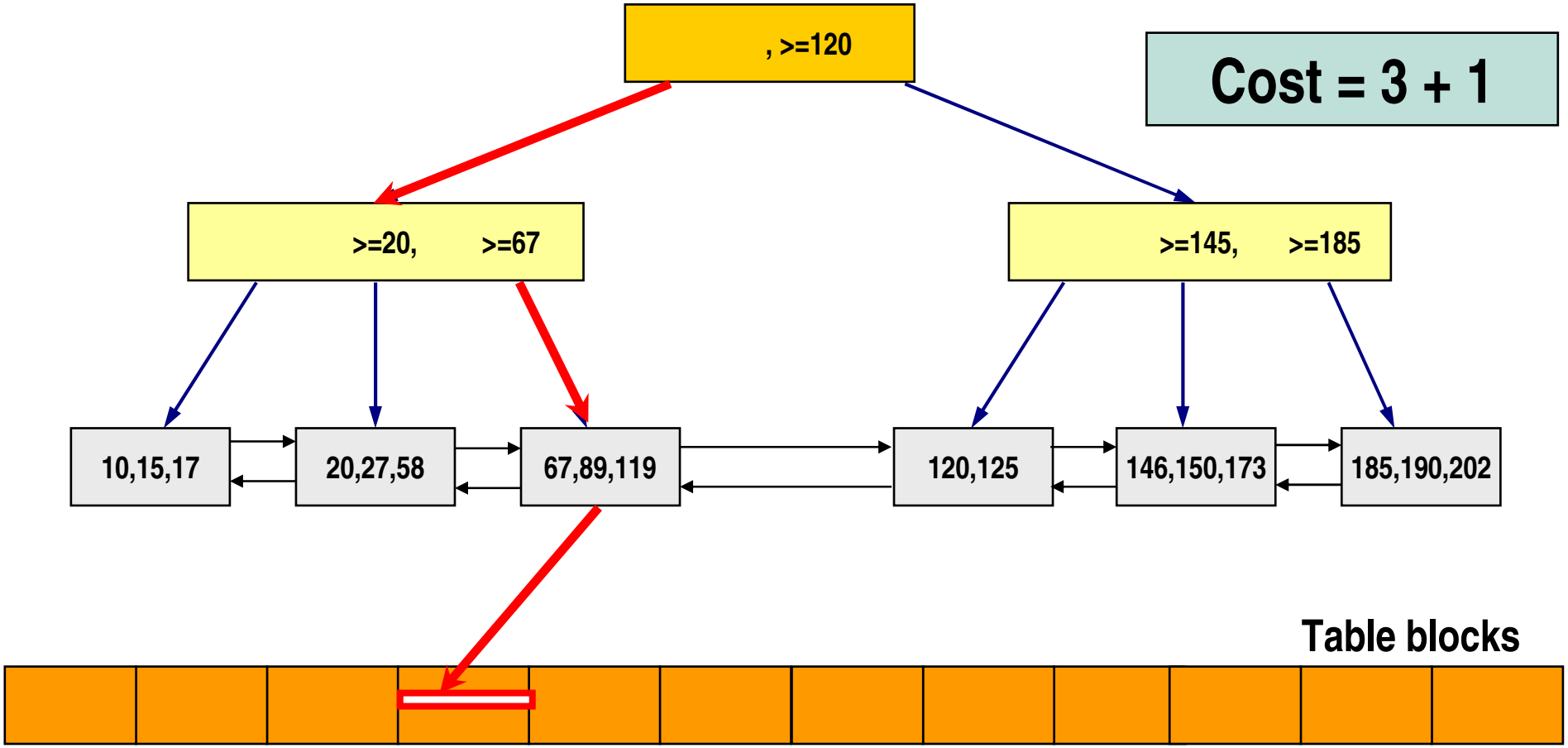
- **V$SQL_PLAN_STATISTICS**
  - This view provides, for each cached cursor, the **execution statistics of each operation in the execution plan**.
  - To view row source statistics in this view, the DBA must set the parameter STATISTICS_LEVEL to ALL.
  - use hint /*+ **gather_plan_statistics** */
- **V$SQL_PLAN_STATISTICS_ALL**
  - This table concatenates information from V$SQL_PLAN with execution statistics from V$SQL_PLAN_STATISTICS and V$SQL_WORKAREA. V$SQL_WORKAREA contains memory usage statistics for row sources that use SQL memory (for example, hash-join and sort).

EXAMPLE OPERATION:

INDEX ACCESS PATHS

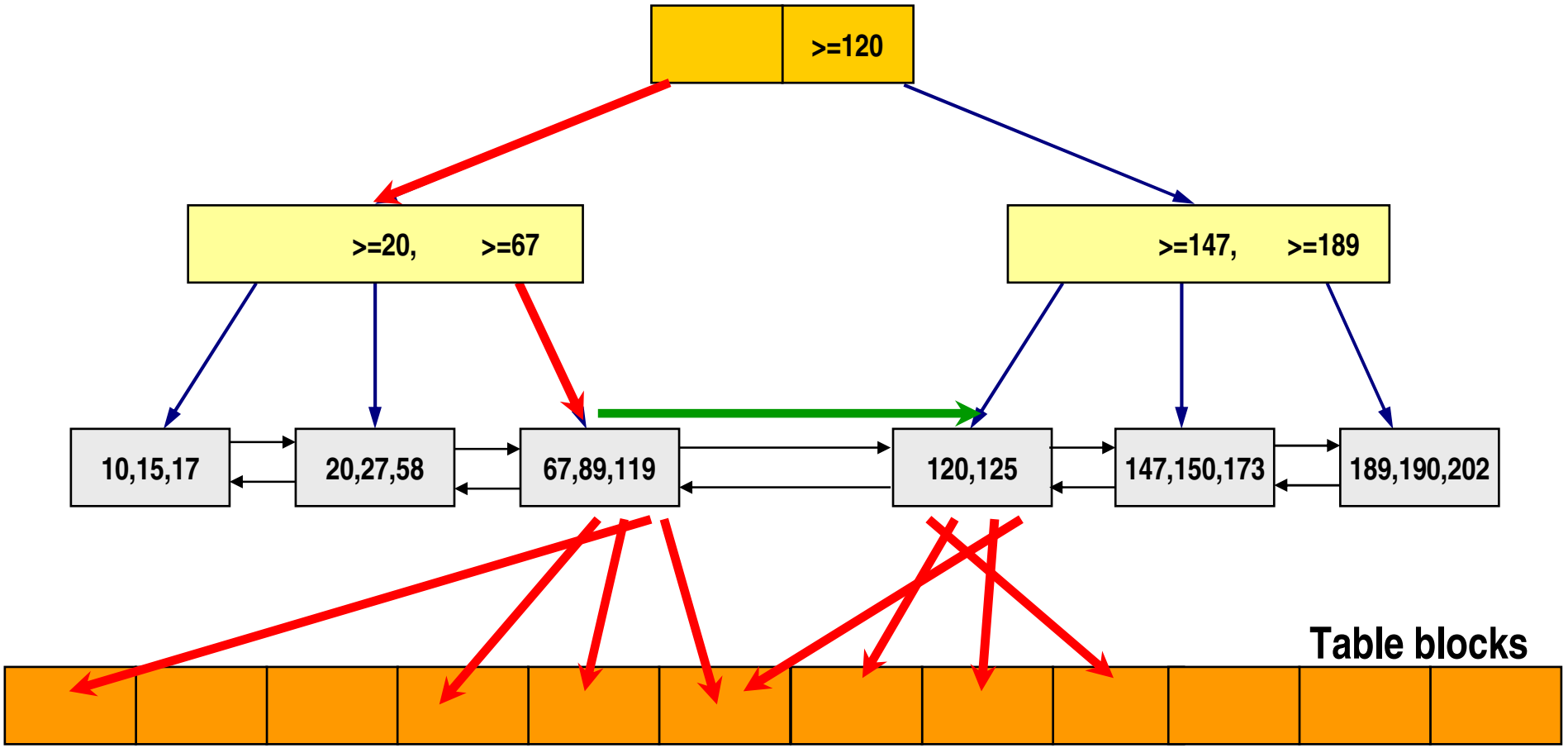
INDEX UNIQUE SCAN (2)

```
select * from t1 where c1=90
```



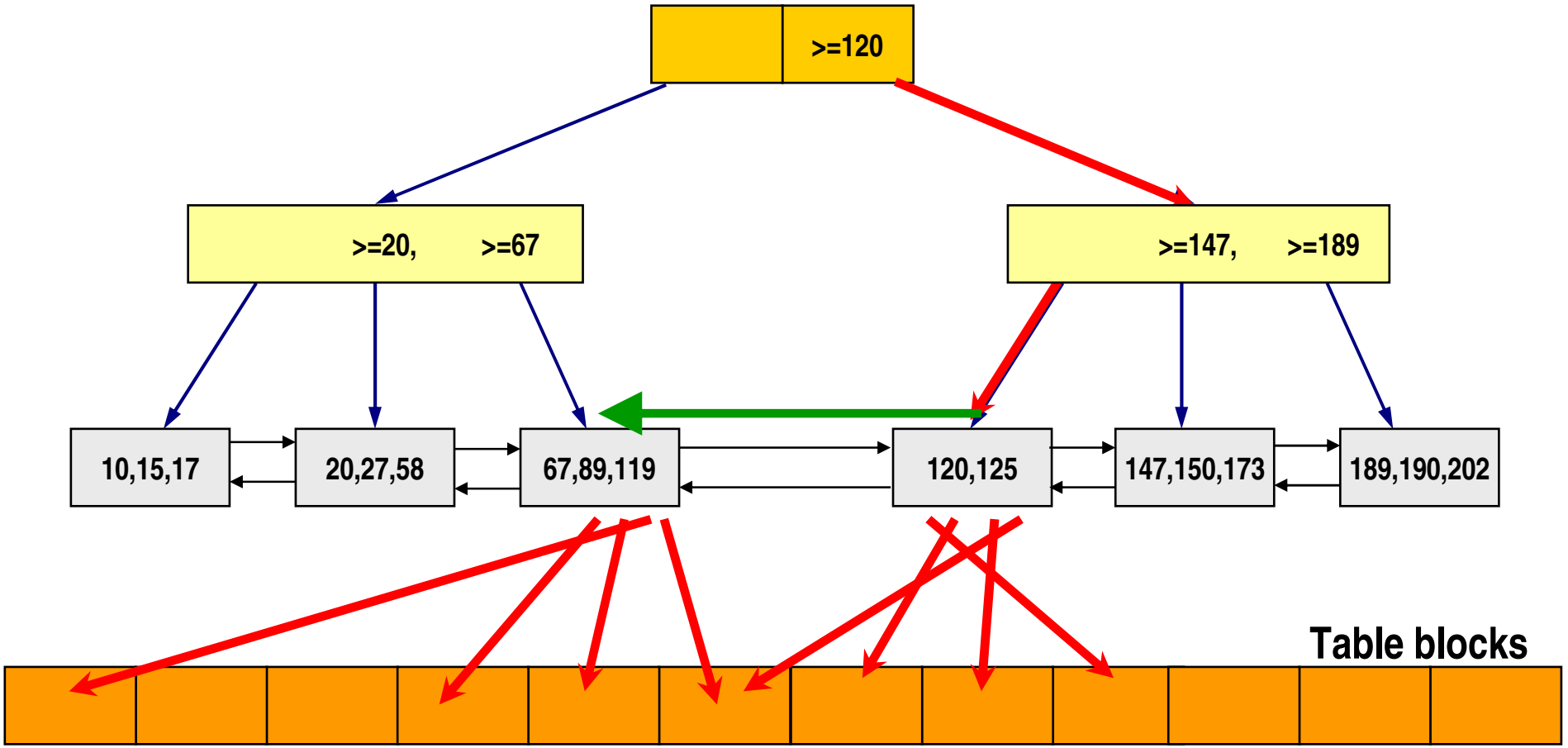
INDEX RANGE SCAN (2)

```
select * from t1 where c1 between 90 and 123
```



INDEX RANGE SCAN DESCENDING (2)

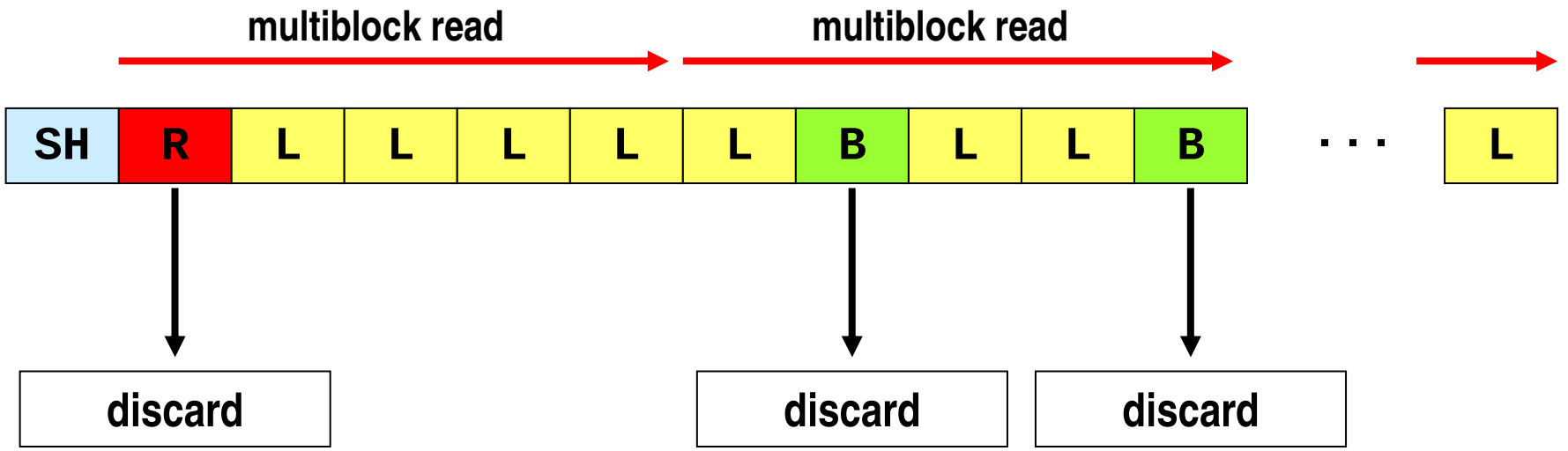
```
select * from t1 where c2 between 90 and 123
                order by c2 desc
```



Index Fast Full Scan

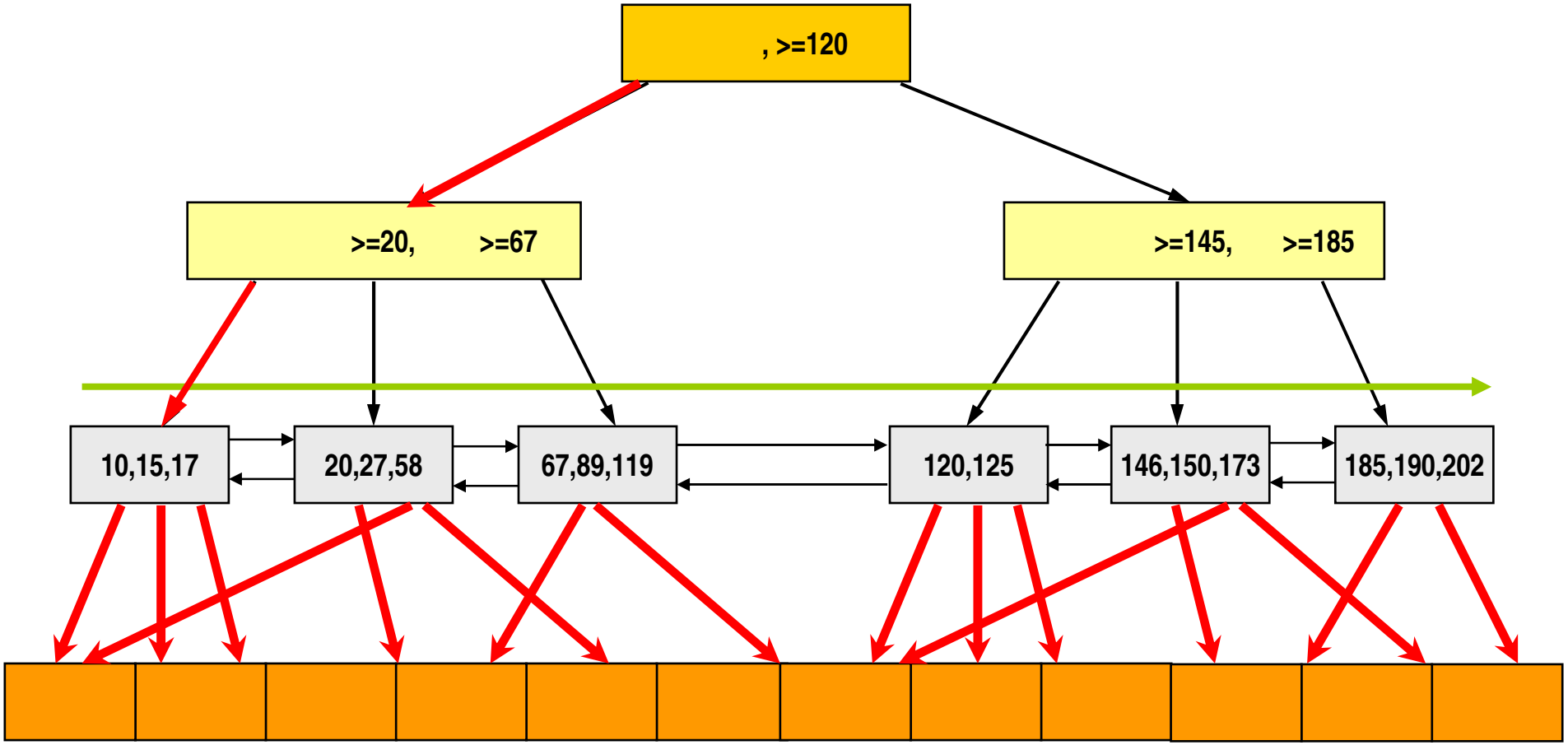
db_file_multiblock_read_count=5

LEGEND:
SH=segment header
R=root block
L=leaf block
B=branch block



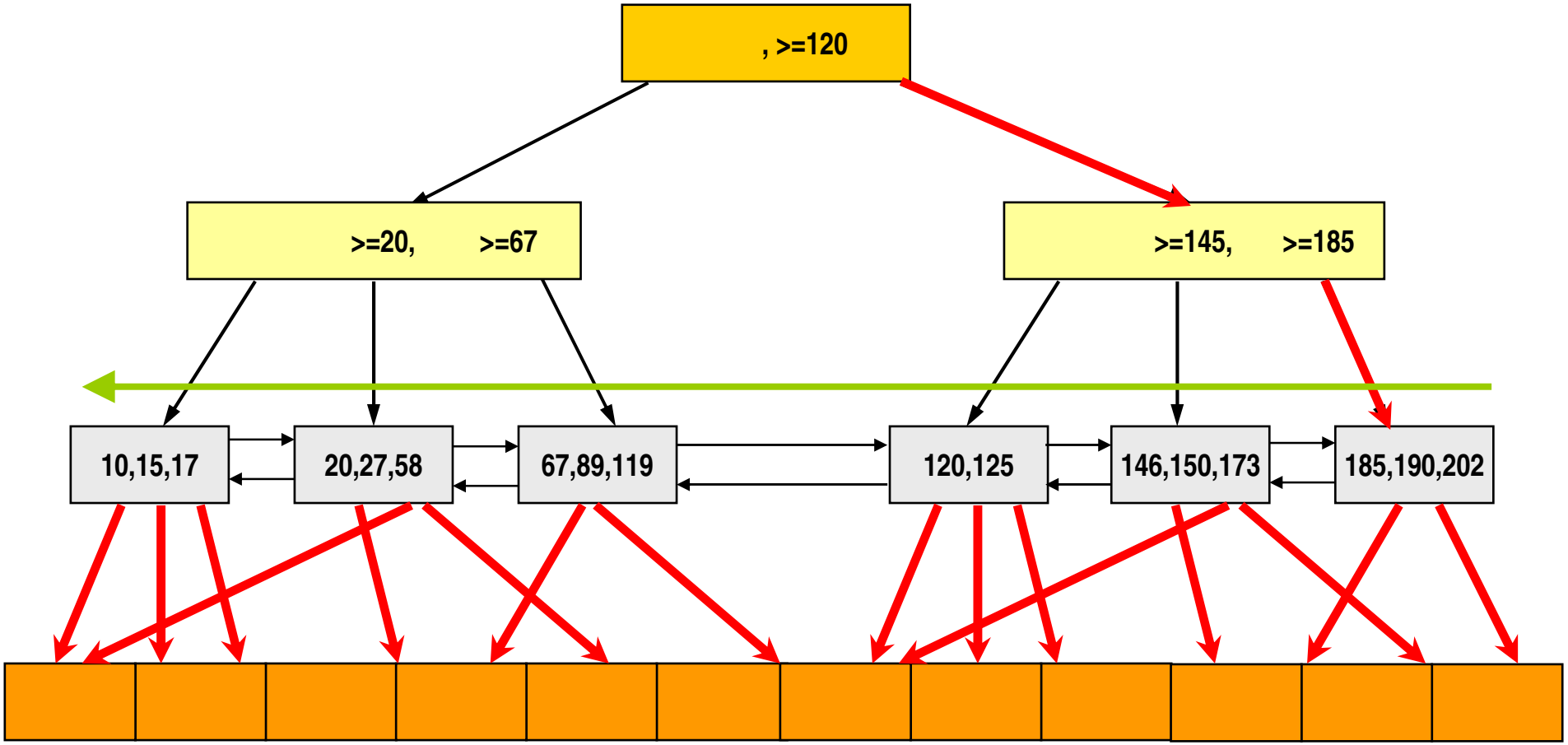
INDEX FULL SCAN

```
select key from t3 order by key
```



INDEX FULL SCAN DESCENDING

```
select key from t3 order by key DESC
```



SQL Monitor

11g New Feature

SQL Plan Monitor (1)

- New 11gR1 feature – requires Tuning pack licensing
- New views **V$SQL_MONITOR, V$SQL_PLAN_MONITOR**
- Captures statistics about SQL execution every second
- For parallel execution every process involved gets separate entries in V$SQL_MONITOR and V$SQL_PLAN_MONITOR
- Enabled by default for long running statements if parameter CONTROL_MANAGEMENT_PACK_ACCESS if it is set to "DIAGNOSTIC+TUNING" and STATISTICS_LEVEL=ALL|TYPICAL

SQL Plan Monitor (2)

- Can be enabled at statement level with /*+ MONITOR */ hint or disabled with /*+ NO_MONITOR */ hint
- Defaults (hidden parameters):
  - **_sqlmon_max_plan** - Maximum number of plans entry that can be monitored. Defaults to 20 per CPU
  - **_sqlmon_max_planlines** - Number of plan lines beyond which a plan cannot be monitored (default 300)

SQL Plan Monitoring - Demo

```
SQL_ID  b0zm3w4h1hbff, child number 0
-------------------------------------
select count(*) from obj$,obj$,obj$

Plan hash value: 3679021907

-----------------------------------------------------
| Id  | Operation                    | Name    | E-Rows |
-----------------------------------------------------
|   1 |  SORT AGGREGATE              |         |      1 |
|   2 |   MERGE JOIN CARTESIAN       |         |   341T|
|   3 |    MERGE JOIN CARTESIAN      |         |  4886M|
|   4 |     INDEX FAST FULL SCAN     | I_OBJ1  |  69901 |
|   5 |     BUFFER SORT              |         |  69901 |
|   6 |      INDEX FAST FULL SCAN    | I_OBJ1  |  69901 |
|   7 |    BUFFER SORT               |         |  69901 |
|   8 |     INDEX FAST FULL SCAN     | I_OBJ1  |  69901 |
-----------------------------------------------------
```

performance killer

V$SQL_MONITOR, V$SQL_PLAN_MONITOR

```
SQL> SELECT status, KEY, SID, sql_id, elapsed_time, cpu_time, fetches, buffer_gets,
  2         disk_reads
  3    FROM v$sql_monitor;

STATUS            KEY SID SQL_ID        ELAPSED_TIME   CPU_TIME FETCHES BUFFER_GETS DISK_READS
--------- ----------- --- ------------- ------------ ---------- ------- ----------- ----------
EXECUTING 21474836481 170 b0zm3w4h1hbff    674281628  624578125       0           0          0

SQL> SELECT   plan_line_id, plan_operation || ' ' || plan_options operation,
  2           starts, output_rows
  3      FROM v$sql_plan_monitor
  4  ORDER BY plan_line_id;

PLAN_LINE_ID OPERATION                      STARTS OUTPUT_ROWS (A-Rows)
------------ -------------------- ---------- -----------
           0 SELECT STATEMENT                1           0
           1  SORT AGGREGATE                 1           0
           2   MERGE JOIN CARTESIAN          1  4283731363
           3    MERGE JOIN CARTESIAN         1      156731
           4     INDEX FAST FULL SCAN        1           3
           5     BUFFER SORT                 3      156731
           6      INDEX FAST FULL SCAN       1       70088
           7    BUFFER SORT                156731  4283731363
           8     INDEX FAST FULL SCAN        1       70088
```

SQL Monitoring Output (1)

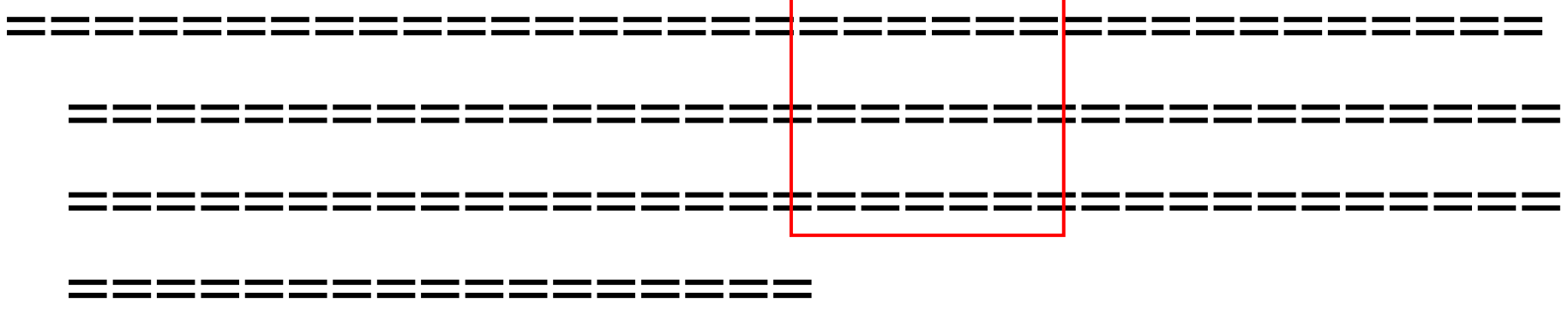
- dbms_sqltune.report_sql_monitor

SQL Plan Monitoring Details (Plan Hash Value=2056254005)

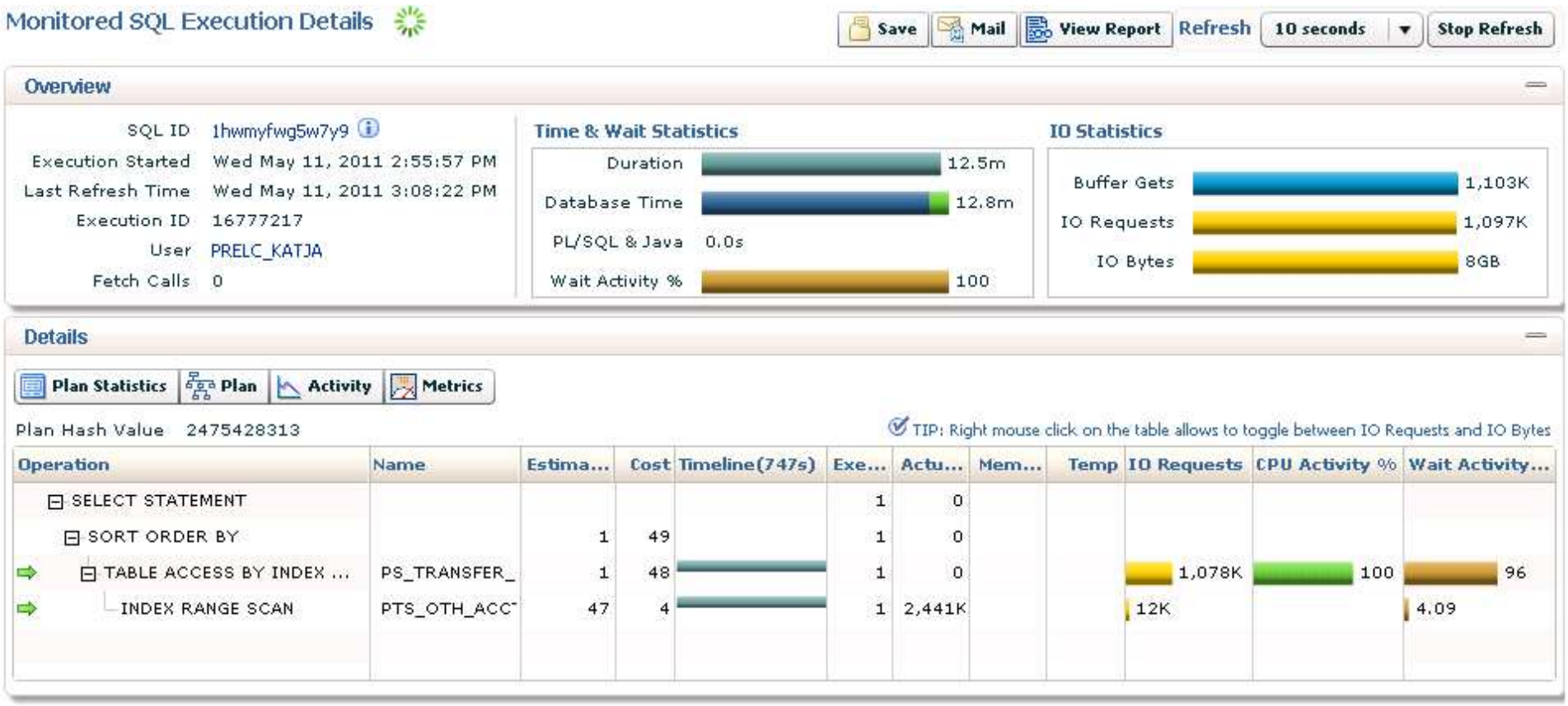
Id	Operation	Name	Rows (Estim)	Cost	Time Active(s)	Start Active	Execs
0	SELECT STATEMENT				1	+4	1
1	SORT ORDER BY		24	39	1	+4	1
2	VIEW	AB_STMTEND	24	38	1	+4	1
3	SORT UNIQUE		24	38	1	+4	1
4	UNION-ALL				1	+4	1
5	NESTED LOOPS				1	+4	1
6	NESTED LOOPS		1	6	1	+4	1
7	NESTED LOOPS		1	5	1	+4	1
8	TABLE ACCESS BY INDEX ROWID	AB_ACCOUNT_BAL	1	3	1	+4	1
9	INDEX UNIQUE SCAN	AB_ACCT_BAL_UIDX	1	2	1	+4	1
10	TABLE ACCESS BY INDEX ROWID	RB_STMT_MAST_SK	1	2	1	+4	1
11	INDEX RANGE SCAN	RXM_INTERNAL_KEY_PK	1	1	1	+4	1
12	INDEX UNIQUE SCAN	RSM_INTERNAL_KEY_PK	1		1	+4	2
13	TABLE ACCESS BY INDEX ROWID	RB_STMT	1	1	1	+4	2
14	NESTED LOOPS				1	+4	1
15	NESTED LOOPS		23	30	1	+4	1
16	NESTED LOOPS		23	7	1	+4	1
17	TABLE ACCESS BY INDEX ROWID	AB_ACCOUNT_BAL	1	3	1	+4	1
18	INDEX UNIQUE SCAN	AB_ACCT_BAL_UIDX	1	2	1	+4	1
19	TABLE ACCESS BY INDEX ROWID	RB_STMT_MAST_HIST_SK	23	4	4	+1	1
20	INDEX RANGE SCAN	RB_STMT_MAST_HIST_SK_I1	2	1	1	+4	1
21	INDEX UNIQUE SCAN	RSM_INTERNAL_KEY_PK	1		1	+4	1133
22	TABLE ACCESS BY INDEX ROWID	AB_STMT	1	1	1	+4	1133

SQL Monitoring Output (2)

- dbms_sqltune.report_sql_monitor



SQL Monitor (in Grid Control)



References:

- Christian Antognini, Interpreting Execution Plans, 2009

Thank you for your interest!

Q&A