

Interpreting Execution Plans

Christian Antognini
Trivadis AG, Zurich (CH)
17 October 2009

Abstract

An execution plan describes the operations carried out by the SQL engine to execute a SQL statement. Every time you have to analyze a performance problem related to a SQL statement, or simply question the decisions taken by the query optimizer, you must know the execution plan. Whenever you deal with an execution plan, you carry out three basic actions: you obtain it, you interpret it, and you judge its efficiency. The aim of this presentation is to describe how the second action is carried out. Chapter 6 of my book, *Troubleshooting Oracle Performance*, covers all of them in detail.

Table of Contents

1	Parent-Child Relationship.....	4
2	Types of Operations.....	6
3	Stand-Alone Operations	7
3.1	Stand-Alone Operations – COUNT STOPKEY.....	9
3.2	Stand-Alone Operations – FILTER.....	10
4	Unrelated-Combine Operations.....	11
5	Related-Combine Operations	13
5.1	Related-Combine Operations – NESTED LOOPS	14
5.2	Related-Combine Operations – FILTER.....	15
5.3	Related-Combine Operations – UPDATE.....	16
5.4	Related-Combine Operations – CONNECT BY	17
6	Divide and Conquer	19
7	Core Messages.....	23

Who Am I

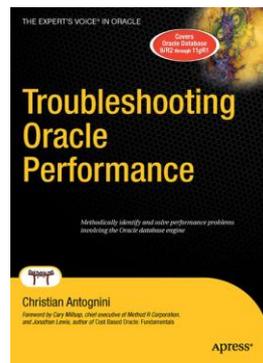


- Principal consultant, trainer and partner at Trivadis in Zurich (CH)
 - Email: christian.antognini@trivadis.com
 - Blog: antognini.ch/blog
- Focus: get the most out of Oracle
 - Logical and physical database design
 - Query optimizer
 - Application performance management and tuning
 - Integration of databases with Java applications
- Proud member of
 - Trivadis Performance Team
 - OakTable Network



Since 1995, Christian Antognini has been focusing on understanding how the Oracle database engine works. His main interests range from logical and physical database design, the integration of databases with Java applications, to the query optimizer and basically everything else related to performance management and tuning. He is currently working as a principal consultant and trainer at Trivadis AG (<http://www.trivadis.com>) in Zurich, Switzerland. If he is not helping one of his customers to get the most out of Oracle, he is somewhere lecturing on optimization or new Oracle database features for developers. He is member of the Trivadis Performance Team and of the OakTable Network.

Troubleshooting Oracle Performance, Apress 2008



<http://top.antognini.ch>

- Foundations
 1. Performance Problems
 2. Key Concepts
- Identification
 3. Identifying Performance Problem
- Query Optimizer
 4. System and Object Statistics
 5. Configuring the Query Optimizer
 6. Execution Plans
 7. SQL Tuning Techniques
- Tuning
 8. Parsing
 9. Optimizing Data Access
 10. Optimizing Joins
 11. Beyond Access and Joins Optimization
 12. Optimizing the Physical Design

Troubleshooting Oracle Performance - Execution Plans

3

© 2009 trivadis
makes IT easier. ■ ■ ■

Troubleshooting Oracle Performance
by Christian Antognini

Hardcover: 616 pages

Publisher: Apress

Publishing date: June 2008

Language: English

ISBN-10: 1-59059-917-9

ISBN-13: 978-1-59059-917-4

ISBN-13 (electronic): 978-1-4302-0498-5

1 Parent-Child Relationship

Parent-child relationship

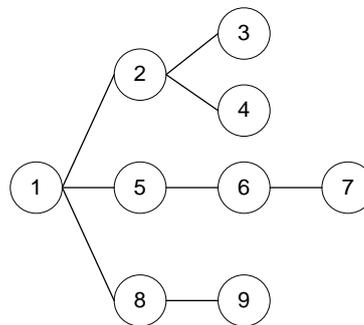


- An execution plan is a tree.
- Each node in the tree is an operation.
- Between operations (nodes) there is a parent-child relationship.
- The rules governing the parent-child relationship are the following:
 - A parent has one or multiple children.
 - A child has a single parent.
 - The only operation without a parent is the root of the tree.
 - When an execution plan is displayed, the children are indented right, with respect to their parent.
 - A parent is placed before its children (ID of the parent < ID of the children).

Parent-child relationship – Example



Id	Operation
1	UPDATE
2	NESTED LOOPS
* 3	TABLE ACCESS FULL
* 4	INDEX UNIQUE SCAN
5	SORT AGGREGATE
6	TABLE ACCESS BY INDEX ROWID
* 7	INDEX RANGE SCAN
8	TABLE ACCESS BY INDEX ROWID
* 9	INDEX UNIQUE SCAN



Using the rules described in the previous slide, you can conclude the following.

- Operation 1 is the root of the tree. It has three children: 2, 5, and 8.
- Operation 2 has two children: 3 and 4.
- Operations 3 and 4 have no children.
- Operation 5 has one child: 6.
- Operation 6 has one child: 7.
- Operation 7 has no children.
- Operation 8 has one child: 9.
- Operation 9 has no children.

2 Types of Operations

Types of operations



- The number of possible operations is high (about 200).
- To fully understand an execution plan, you should know what each operation it is made of does.
- For our purpose of walking through an execution plan, you need to consider only three major types of operations:
 - Standalone operations
 - Unrelated-combine operations
 - Related-combine operations

Caution: I coined the terms used here for the three types of operations while writing a presentation about the query optimizer a few years ago. Don't expect to find these terms used elsewhere.

3 Stand-Alone Operations

Stand-alone operations



- All operations having at most one child are *stand-alone operations*.
- Most operations are of this type.
- The rules governing the working of these operations are the following:
 - Children are executed before their parents. Two optimization techniques presented later however, lead to exceptions to this rule.
 - Every child is executed at most once.
 - Every child feeds its parent.

Stand-alone operations – Example

```
SELECT deptno, count(*)
FROM emp
WHERE job = 'CLERK' AND sal < 1200
GROUP BY deptno
```

Id	Operation	Name	Starts	A-Rows
1	HASH GROUP BY		1	2
* 2	TABLE ACCESS BY INDEX ROWID	EMP	1	3
* 3	INDEX RANGE SCAN	EMP_JOB_I	1	4

```
2 - filter("SAL"<1200)
3 - access("JOB"='CLERK')
```



This execution plan consists only of stand-alone operations. By applying the rules described earlier, you find out that the execution plan carries out the operations as follows:

- Operations 1 and 2 have a single child each (2 and 3, respectively); they cannot be the first operations being executed. Therefore, the execution starts with operation 3.
- Operation 3 scans the index emp_job_i by applying the access predicate "JOB" = 'CLERK'. In doing so, it extracts four rowids (this information is given in the column A-Rows) from the index and passes them to its parent operation (2).
- Operation 2 accesses the table emp through the four rowids passed from operation 3. For each rowid, a row is read. Then, it applies the filter predicate "SAL" < 1200. This filter leads to the exclusion of one row. The data of the remaining three rows are passed to its parent operation (1).
- Operation 1 performs a GROUP BY on the rows passed from operation 2. The resulting set is reduced to two rows. Since this is the last operation, the data is sent to the caller.

3.1 Stand-Alone Operations – COUNT STOPKEY

Stand-alone operations – COUNT STOPKEY



- The operation COUNT STOPKEY is commonly used to execute the top-n queries.
- Its aim is to stop the processing as soon as the required number of rows has been returned to the caller.

```
SELECT *
FROM emp
WHERE rownum <= 10
```

Id	Operation	Name	Starts	A-Rows
* 1	COUNT STOPKEY		1	10
2	TABLE ACCESS FULL	EMP	1	10

1 - filter (ROWNUM<=10)

The important thing to notice in this execution plan is that the number of rows returned by operation 2 is limited to ten. This is true even if operation 2 is a full table scan of a table containing more than ten rows (actually the table contains 14 rows). What happens is that operation 1 stops the processing of operation 2 as soon as the necessary number of rows has been processed. Be careful, though, because blocking operations (e.g. ORDER BY) cannot be stopped. In fact, they need to be fully processed before returning rows to their parent operation.

3.2 Stand-Alone Operations – FILTER

Stand-alone operations – FILTER



- The operation FILTER applies a filter when its child passes data to it. In addition, it could decide to completely avoid the execution of a child and all the dependent operations as well.

```
SELECT *
FROM emp
WHERE job = 'CLERK' AND 1 = 2
```

Id	Operation	Name	Starts	A-Rows
* 1	FILTER		1	0
2	TABLE ACCESS BY INDEX ROWID	EMP	0	0
* 3	INDEX RANGE SCAN	EMP_JOB_I	0	0

```
1 - filter(NULL IS NOT NULL)
3 - access("JOB"='CLERK')
```

According to the rules described earlier, such an execution plan should be carried out by starting the processing of operation 3. In reality, looking at the column Starts tells you that only operation 1 is executed. This optimization simply avoids processing operations 2 and 3 because the data has no chance of going through the filter applied by operation 1 anyway.

4 Unrelated-Combine Operations

Unrelated-combine operations

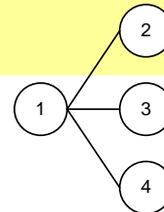


- All operations having multiple children that are independently executed are *unrelated-combine operations*.
- The following operations are of this type: AND-EQUAL, BITMAP AND, BITMAP OR, BITMAP MINUS, CONCATENATION, CONNECT BY WITHOUT FILTERING, HASH JOIN, INTERSECTION, MERGE JOIN, MINUS, MULTI-TABLE INSERT, SQL MODEL, TEMP TABLE TRANSFORMATION and UNION-ALL
- 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 the others.
 - Every child feeds its parent.

Unrelated-combine operations – Example

```
SELECT ename FROM emp
UNION ALL
SELECT dname FROM dept
UNION ALL
SELECT '%' FROM dual
```

Id	Operation	Name	Starts	A-Rows
1	UNION-ALL		1	19
2	TABLE ACCESS FULL	EMP	1	14
3	TABLE ACCESS FULL	DEPT	1	4
4	FAST DUAL		1	1



In this execution plan, the unrelated-combine operation is the UNION-ALL. The other three are stand-alone operations. By applying the rules described earlier, you see that the execution plan carries out the operations as follows:

1. Operation 1 has three children, and, among them, operation 2 is the first in ascending order. Therefore, the execution starts with operation 2.
2. Operation 2 scans the table emp and returns 14 rows to its parent operation (1).
3. When operation 2 is completely executed, operation 3 is started.
4. Operation 3 scans the table dept and returns four rows to its parent operation (1).
5. When operation 3 is completely executed, operation 4 is started.
6. Operation 4 scans the table dual and returns one row to its parent operation (1).
7. Operation 1 builds a single result set of 19 rows based on all data received from its children and sends the data to the caller.

Notice how the column Starts clearly shows that each operation is executed only once.

5 Related-Combine Operations

Related-combine operations



- All operations having multiple children where one of the children controls the execution of all other children are related-combine operations.
- The following operations are of this type: NESTED LOOPS, UPDATE, FILTER, CONNECT BY WITH FILTERING and BITMAP KEY ITERATION.
- The following are the characteristics of these operations:
 - 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.

5.1 Related-Combine Operations – NESTED LOOPS

Related-combine operations – NESTED LOOPS
■ ■ ■

```

SELECT *
FROM emp, dept
WHERE emp.deptno = dept.deptno
AND emp.comm IS NULL
AND dept.dname != 'SALES'
  
```

Id	Operation	Name	Starts	A-Rows
1	NESTED LOOPS		1	8
* 2	TABLE ACCESS FULL	EMP	1	10
* 3	TABLE ACCESS BY INDEX ROWID	DEPT	10	8
* 4	INDEX UNIQUE SCAN	DEPT_PK	10	10

```

2 - filter("EMP"."COMM" IS NULL)
3 - filter("DEPT"."DNAME"<>'SALES')
4 - access("EMP"."DEPTNO"="DEPT"."DEPTNO")
  
```

```

graph TD
    1((1)) --- 2((2))
    1 --- 3((3))
    3 --- 4((4))
  
```

Troubleshooting Oracle Performance - Execution Plans 15 © 2009 trivadis

The execution plan carries out the operations as follows:

1. Operation 1 has two children (2 and 3), and among them, operation 2 is the first in ascending order. Therefore, the execution starts with operation 2.
2. Operation 2 scans the table emp, applies the filter predicate "EMP"."COMM" IS NULL, and returns the data of ten rows to its parent operation (1).
3. For each row returned by operation 2, the second child of the operation NESTED LOOPS is executed once. This is confirmed by comparing the column A-Rows of operation 2 with the column Starts of operations 3 and 4.
4. Based on the rules that apply to stand-alone operations, operation 4 is executed before operation 3.
5. Operation 4 scans the index dept_pk by applying the access predicate "EMP"."DEPTNO" = "DEPT"."DEPTNO". In doing so, it extracts ten rowids from the index over the ten executions and passes them to its parent operation (3).
6. Operation 3 accesses the table dept through the ten rowids passed from operation 4. For each rowid, a row is read. Then it applies the filter predicate "DEPT"."DNAME" <> 'SALES'. This filter leads to the exclusion of two rows. It passes the data of the remaining eight rows to its parent operation (1).
7. Operation 1 sends the data of eight rows to the caller.

5.2 Related-Combine Operations – FILTER

Related-combine operations – FILTER

■■■

```

SELECT *
FROM emp
WHERE NOT EXISTS (SELECT 0 FROM dept WHERE dept.dname = 'SALES'
                  AND dept.deptno = emp.deptno)
AND NOT EXISTS (SELECT 0 FROM bonus WHERE bonus.ename = emp.ename);

```

Id	Operation	Name	Starts	A-Rows
* 1	FILTER		1	8
2	TABLE ACCESS FULL	EMP	1	14
* 3	TABLE ACCESS BY INDEX ROWID	DEPT	3	1
* 4	INDEX UNIQUE SCAN	DEPT_PK	3	3
* 5	TABLE ACCESS FULL	BONUS	8	0

1 - filter(NOT EXISTS (SELECT 0 FROM "DEPT" "DEPT" WHERE "DEPT"."DEPTNO"=:B1 AND "DEPT"."DNAME"='SALES') AND NOT EXISTS (SELECT 0 FROM "BONUS" "BONUS" WHERE "BONUS"."ENAME"=:B2))

3 - filter("DEPT"."DNAME"='SALES')

4 - access("DEPT"."DEPTNO"=:B1)

5 - filter("BONUS"."ENAME"=:B1)

```

graph TD
    1((1)) --- 2((2))
    1((1)) --- 3((3))
    1((1)) --- 5((5))
    2((2)) --- 4((4))
    3((3)) --- 4((4))
    4((4)) --- 5((5))

```

Troubleshooting Oracle Performance - Execution Plans 16 © 2009 trivadis

The execution plan carries out the operations in the following manner:

1. Operation 1 has three children (2, 3 and 5), and operation 2 is the first of them in ascending order. Therefore, the execution starts with operation 2.
2. Operation 2 scans the table emp and returns 14 rows to its parent operation (1).
3. For each row returned by operation 2, the second and third children of the operation FILTER should be executed once. In reality, a kind of caching is implemented to reduce executions to a minimum. Operation 3 is executed three times, once for each distinct value in the column deptno in the table emp. Operation 5 is executed eight times, once for each distinct value in the column empno in the table emp after applying the filter imposed by the operation
4. According to the rules for stand-alone operations, operation 4, which is executed before operation 3, scans the index dept_pk by applying the access predicate "DEPT"."DEPTNO" = :B1.
5. Operation 3 accesses the table dept through the rowids passed from its child operation (4) and applies the filter predicate "DEPT"."DNAME" = 'SALES'.
6. Operation 5 scans the table bonus and applies the filter predicate "BONUS"."ENAME" = :B1.
7. Operation 1, after applying the filter predicate implemented with operations 3 and 5, sends the data of eight rows to the caller.

5.3 Related-Combine Operations – UPDATE

Related-combine operations – UPDATE

■ ■ ■

```
UPDATE emp e1
SET sal = (SELECT avg(sal) FROM emp e2 WHERE e2.deptno = e1.deptno),
    comm = (SELECT avg(comm) FROM emp e3)
```

Id	Operation	Name	Starts	A-Rows
1	UPDATE	EMP	1	0
2	TABLE ACCESS FULL	EMP	1	14
3	SORT AGGREGATE		3	3
* 4	TABLE ACCESS FULL	EMP	3	14
5	SORT AGGREGATE		1	1
6	TABLE ACCESS FULL	EMP	1	14

4 - filter("E2"."DEPTNO"=:B1)

Troubleshooting Oracle Performance - Execution Plans 17 © 2009 trivadis

The execution plan carries out the operations as follows:

1. Operation 1 has three children (2, 3, and 5), and operation 2 is the first of the three in ascending order. Therefore, the execution starts with operation 2.
2. Operation 2 scans the table emp and returns 14 rows to its parent operation (1).
3. The second and third child (3 and 5) might be executed several times (at most, as many times as the number of rows returned by operation 2). Since both these operations are stand-alone, and each has a child, their execution starts with the child operations (4 and 6).
4. For each distinct value in the column deptno returned by operation 2, operation 4 scans the table emp and applies the filter predicate "E2"."DEPTNO"=:B1. In doing so over the three executions, it extracts 14 rows and passes them to its parent operation (3).
5. Operation 3 computes the average salary of the rows passed to it from operation 4 and returns the result to its parent operation (1).
6. Operation 6 scans the table emp, extracts 14 rows, and passes them to its parent operation (5). Note that this subquery is executed only once because it is not correlated to the main query.
7. Operation 5 computes the average commission of the rows passed to it from operation 6 and returns the result to its parent operation (1).
8. Operation 1 updates each row passed by operation 2 with the value returned by its other children (3 and 5). Note that even if the UPDATE statement modifies the 14 rows, column A-Rows shows 0 for this operation.

5.4 Related-Combine Operations – CONNECT BY

Related-combine operations – CONNECT BY
■ ■ ■

```

SELECT level, ename, prior ename AS manager
FROM emp
START WITH mgr IS NULL
CONNECT BY PRIOR empno = mgr
    
```

Id	Operation	Name	Starts	A-Rows
* 1	CONNECT BY WITH FILTERING		1	14
* 2	TABLE ACCESS FULL	EMP	1	1
3	NESTED LOOPS		4	13
4	CONNECT BY PUMP		4	14
5	TABLE ACCESS BY INDEX ROWID	EMP	14	13
* 6	INDEX RANGE SCAN	EMP_MGR_I	14	13

1 - access("MGR"=PRIOR "EMPNO")
2 - filter("MGR" IS NULL)
6 - access("MGR"=PRIOR "EMPNO")

Troubleshooting Oracle Performance - Execution Plans 18 © 2009 trivadis

The execution plan carries out the operations as follows:

1. Operation 1 has two children (2 and 3), and operation 2 is the first of them in ascending order. Therefore, the execution starts with operation 2.
2. Operation 2 scans the table emp, applies the filter predicate "MGR" IS NULL, and returns the root of the hierarchy to its operation (1).
3. Operation 3 is the second child of operation 1. It is therefore executed for each level of the hierarchy—in this case, four times. Naturally, the rules previously discussed for the operation NESTED LOOPS apply for operation 3. The first child, operation 4, is executed, and for each row it returns, the inner loop (composed of operation 5 and its child operation 6) is executed once.
4. For the first execution, operation 4 gets the root of the hierarchy through the operation CONNECT BY PUMP. In this case, there is a single row (KING) at level 1. With the value in the column mgr, operation 6 does a scan of the index emp_mgr_i by applying the access predicate "MGR"=PRIOR "EMPNO", extracts the rowids, and returns them to its parent operation (5). Operation 5 accesses the table emp with the rowids and returns the rows to its parent operation (3).
5. For the second execution of operation 4, everything works the same as for the first execution. The only difference is that the data from level 2 (JONES, BLAKE, and CLARK) is passed to operation 4 for the processing.
6. For the third execution of operation 4, everything works like in the first one. The only difference is that level 3 data (SCOTT, FORD, ALLEN, WARD, MARTIN, TURNER, JAMES, and MILLER) is passed to operation 4 for the processing.

7. For the fourth and last execution of operation 4, everything works like in the first one. The only difference is that level 4 data (ADAMS and SMITH) is passed to operation 4 for the processing.
8. Operation 3 gets the rows passed from its children and returns them to its parent operation (1).
9. Operation 1 applies the access predicate "MGR" = PRIOR "EMPNO" and sends the 14 rows to the caller.

6 Divide and Conquer

Divide and conquer (1)



- Reading long execution plans is no different from reading short ones.
- All you need is to methodically apply the rules provided in the previous slides.
- With them, it does not matter how many lines an execution plan has. You simply proceed in the same way.

Divide and conquer (2)



0	SELECT STATEMENT	
1	FILTER	
2	SORT GROUP BY	
3	FILTER	
4	HASH JOIN OUTER	G
5	NESTED LOOPS OUTER	E
6	NESTED LOOPS	C
7	TABLE ACCESS FULL	A
8	TABLE ACCESS BY INDEX ROWID	
9	INDEX UNIQUE SCAN	B
10	TABLE ACCESS BY INDEX ROWID	
11	INDEX UNIQUE SCAN	D
12	TABLE ACCESS FULL	F
13	SORT UNIQUE	J
14	UNION-ALL	
15	TABLE ACCESS FULL	H
16	TABLE ACCESS FULL	I

- To read an execution plan it is necessary to both decompose the execution plan into basic blocks and recognize the order of execution.

- Each combine operations (both related and unrelated) must be identified.

→ 3, 4, 5, 6, and 14

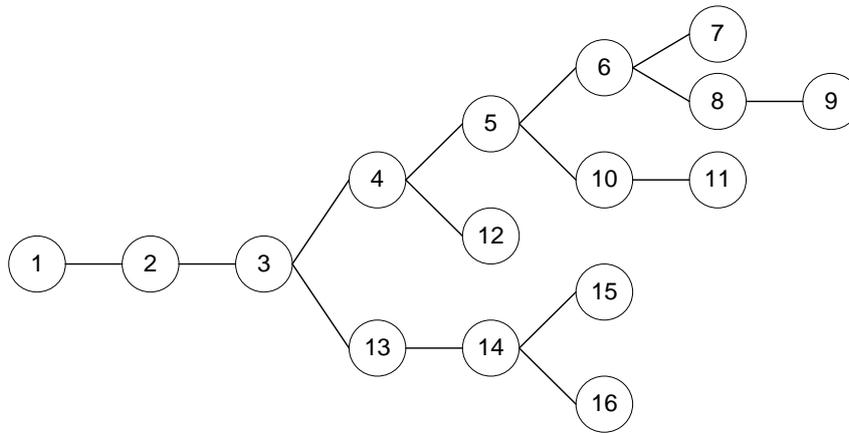
- For each child operation of each combine operation, a block is defined.

To find out in what order the blocks are executed, you have to apply the rules discussed previously discussed:

1. Operation 1 is a stand-alone operation, and its child (2) is executed before it.
2. Operation 2 is a stand-alone operation, and its child (3) is executed before it.
3. Operation 3 is a related-combine operation, and its children are executed before it. Since the first child block (G) is executed before the second child block (J), let's continue with the first operation (4) of the first child block (G).
4. Operation 4 is an unrelated-combine operation, and its children are executed before it. Since the first child block (E) is executed before the second child block (F), let's continue with the first operation (5) of the first child block (E).
5. Operation 5 is a related-combine operation, and its children are executed before it. Since the first child block (C) is executed before the second child block (D), let's continue with the first operation (6) of the first child block (C).
6. Operation 6 is a related-combine operation, and its children are executed before it. Since the first child block (A) is executed before the second child block (B), let's continue with the first operation (7) of the first child block (A).
7. Operation 7 is a stand-alone operation and has no children. This means that you have finally found the first operation to be executed (hence it's in block A). The operation scans a table and returns the rows to its parent operation (6).
8. Block B is executed for each row returned by block A. In this block, operation 9 scans an index at first, and operation 8 accesses a table with the returned rowids and finally returns the rows to its parent operation (6).

9. Operation 6 performs the join between the rows returned by blocks A and B and then returns the result to its parent operation (5).
10. Block D is executed for each row returned by block C. In other words, it is executed for each row returned by operation 6 to its parent operation (5). In this block, operation 11 scans an index initially. Then, operation 10 accesses a table with the returned rowids and returns the rows to its parent operation (5).
11. Operation 5 performs the join between the rows returned by the blocks C and D and then returns the result to its parent operation (4).
12. Operation 12 (block F) is executed only once. It scans a table and returns the result to its parent operation (4).
13. Operation 4 performs the join between the rows returned by the blocks E and F and then returns the result to its parent operation (3).
14. Block J is basically executed for each row returned by block G. In other words, it is executed for each row returned by operation 4 to its parent operation (3). In this block, operation 15 scans a table at first and returns the rows to its parent operation (14). Then, operation 16 scans a table and returns the rows to its parent operation (14). After that, operation 14 puts the rows returned by its children together and returns the result to its parent operation (13). Finally, operation 13 removes some duplicate rows.
15. Once operation 3 has applied the filter with the block J, it returns the result to its parent operation (2).
16. Operation 2 performs a GROUP BY and returns the result to its parent operation (1).
17. Operation 1 applies a filter and returns the result to the caller.

Divide and conquer (3)



7 Core Messages

Core messages
■ ■ ■

- Simple rules can be applied for interpreting execution plans.



Knowledge transfer is only the beginning. Knowledge application is what counts.

Troubleshooting Oracle Performance - Execution Plans 22 © 2009 **trivadis**
makes IT easier. ■ ■ ■