ORACLE CERTIFIED SOLUTION PARTNER

Speeding Up Queries with Semi-Joins and Anti-Joins: How Oracle Evaluates EXISTS, NOT EXISTS, IN, and NOT IN

Roger Schrag
Database Specialists, Inc.
www.dbspecialists.com

DatabaseSpecialists

Today's Session

- "Semi-join" and "anti-join" defined
- EXISTS and IN clauses
  - **How Oracle evaluates them**
  - **Prerequisites and hints**
  - **Examples**
- NOT EXISTS and NOT IN clauses
  - **How Oracle evaluates them**
  - **Prerequisites and hints**
  - **Examples**

DatabaseSpecialists

White Paper

- Fourteen pages of details I can't possibly cover in a one hour presentation.
- Lots of sample code, execution plans, and TKPROF reports that you will see are probably not readable when I put them up on PowerPoint slides—but they are readable in the white paper.
- Download: www.dbspecialists.com/presentations

Semi-Joins and Anti-Joins

- Two special types of joins with efficient access paths.
- Can dramatically speed up certain classes of queries.
- Can only be used by Oracle when certain prerequisites are met.

“Semi-Join” Defined

A semi-join between two tables returns rows from the first table where one or more matches are found in the second table.

The difference between a semi-join and a conventional join is that rows in the first table will be returned at most once. Even if the second table contains two matches for a row in the first table, only one copy of the row will be returned.

Semi-joins are written using EXISTS or IN.

A Simple Semi-Join Example

"Give me a list of departments with at least one employee."

Query written with a conventional join:

```
SELECT   D.deptno, D.dname
FROM     dept D, emp E
WHERE    E.deptno = D.deptno
ORDER BY D.deptno;
```

- **A department with N employees will appear in the list N times.**
- **You could use a DISTINCT keyword to get each department to appear only once.**
- **Oracle will do more work than necessary.**

A Simple Semi-Join Example

“Give me a list of departments with at least one employee.”

Query written with a semi-join:

```
SELECT   D.deptno, D.dname
FROM     dept D
WHERE    EXISTS
         (SELECT 1
          FROM   emp E
          WHERE  E.deptno = D.deptno)
ORDER BY D.deptno;
```

- **No department appears more than once.**
- **Oracle stops processing each department as soon as the first employee in that department is found.**

“Anti-Join” Defined

An anti-join between two tables returns rows from the first table where no matches are found in the second table. An anti-join is essentially the opposite of a semi-join.

Anti-joins are written using the NOT EXISTS or NOT IN constructs. These two constructs differ in how they handle nulls—a subtle but very important distinction which we will discuss later.

DatabaseSpecialists

A Simple Anti-Join Example

“Give me a list of empty departments.”

Query written without an anti-join:

```
SELECT   D1.deptno, D1.dname
FROM     dept D1
MINUS
SELECT   D2.deptno, D2.dname
FROM     dept D2, emp E2
WHERE    D2.deptno = E2.deptno
ORDER BY 1;
```

A Simple Anti-Join Example

“Give me a list of empty departments.”

Query written with an anti-join:

```
SELECT   D.deptno, D.dname
FROM     dept D
WHERE    NOT EXISTS
         (
         SELECT 1
         FROM   emp E
         WHERE  E.deptno = D.deptno
         )
ORDER BY D.deptno;
```

Semi-Joins in Greater Detail

- How Oracle evaluates EXISTS and IN clauses.
- Semi-join access path prerequisites.
- Hints that affect semi-joins.
- Examples.

DatabaseSpecialists

How Oracle Evaluates EXISTS and IN

- Oracle transforms the subquery into a join if at all possible (according to Metalink document 144967.1). Oracle does not consider cost when deciding whether or not to do this transformation.
- Oracle can perform a semi-join in a few different ways:
  - **Semi-join access path.**
  - **Conventional join access path followed by a sort to remove duplicate rows.**
  - **Scan of first table with a filter operation against the second table.**

How Oracle Evaluates EXISTS and IN

- Rule of thumb from the Oracle 8i/9i/10g Performance Tuning Guide (highly simplified):
  - **Use EXISTS when outer query is selective.**
  - **Use IN when subquery is selective and outer query is not.**
- My personal experience:
  - **Rule of thumb is valid for Oracle 8i.**
  - **Oracle 9i often does the right thing regardless of whether EXISTS or IN is used.**

Semi-Join Access Path Prerequisites

- Oracle cannot use a semi-join access path in queries that:
  - **use the DISTINCT keyword.**
  - **perform a UNION (involves an implicit DISTINCT).**
  - **have the EXISTS or IN clause on an OR branch.**
- Oracle 8i will only use a semi-join access path if the always_semi_join instance parameter is set to "hash" or "merge", or if a hint is used.
- Oracle 9i and later evaluate the cost of a nested loops, merge, and hash semi-join and choose the least expensive.

DatabaseSpecialists

Hints that Affect Semi-Joins

- Apply the HASH_SJ, MERGE_SJ, and NL_SJ hints to the subquery of an EXISTS or IN clause to tell Oracle which semi-join access path to use.
- Oracle will disregard semi-join hints if you ask for the impossible. (eg: A HASH_SJ hint in a query with a DISTINCT keyword will be ignored.)
- In my experience Oracle is good about knowing when to use a semi-join access path. However, I have seen cases where Oracle chose a nested loops semi-join where a hash semi-join was much more efficient.

DatabaseSpecialists

Semi-Join Example #1

“List the gold-status customers who have placed an order within the last three days.”

```
SELECT   DISTINCT C.short_name, C.customer_id
FROM     customers C, orders O
WHERE    C.customer_type = 'Gold'
AND      O.customer_id = C.customer_id
AND      O.order_date > SYSDATE - 3
ORDER BY C.short_name;
```

```
 Rows     Row Source Operation
-------   ---------------------------------------------------
      2   SORT UNIQUE (cr=33608 r=30076 w=0 time=6704029 us)
     20    HASH JOIN  (cr=33608 r=30076 w=0 time=6703101 us)
     10     TABLE ACCESS FULL CUSTOMERS (cr=38 r=36 w=0 time=31718 us)
   2990     TABLE ACCESS FULL ORDERS (cr=33570 r=30040 w=0 time=6646420 us)
```

Semi-Join Example #1

What we see on the previous slide:

- The query was written with a conventional join and DISTINCT instead of an EXISTS or IN clause.
- Oracle performed a conventional hash join followed by a sort for uniqueness in order to remove the duplicates. (18 of the 20 rows resulting from the hash join were apparently duplicates.)
- The query took 6.70 seconds to complete and performed 33,608 logical reads.

Semi-Join Example #1

Rewritten with a semi-join:

```
SELECT   C.short_name, C.customer_id
FROM     customers C
WHERE    C.customer_type = 'Gold'
AND      EXISTS
         (
         SELECT 1
         FROM   orders O
         WHERE  O.customer_id = C.customer_id
         AND    O.order_date > SYSDATE - 3
         )
ORDER BY C.short_name;

Rows     Row Source Operation
-------  ---------------------------------------------------
      2  SORT ORDER BY (cr=33608 r=29921 w=0 time=6422770 us)
      2   HASH JOIN SEMI (cr=33608 r=29921 w=0 time=6422538 us)
     10    TABLE ACCESS FULL CUSTOMERS (cr=38 r=0 w=0 time=61290 us)
   2990    TABLE ACCESS FULL ORDERS (cr=33570 r=29921 w=0 time=6345754 us)
```

Semi-Join Example #1

What we see on the previous slide:

- An EXISTS clause was used to specify a semi-join.
- Oracle performed a hash semi-join instead of a conventional hash join. This offers two benefits:
  - Oracle can move on to the next customer record as soon as the first matching order record is found.
  - There is no need to sort out duplicate records.
- The query took 6.42 seconds to complete and performed 33,608 logical reads.

DatabaseSpecialists

Semi-Join Example #1

Adding a hint to specify a nested loops semi-join:

```
SELECT   C.short_name, C.customer_id
FROM     customers C
WHERE    C.customer_type = 'Gold'
AND      EXISTS
         (
         SELECT /*+ NL_SJ */ 1
         FROM   orders O
         WHERE  O.customer_id = C.customer_id
         AND    O.order_date > SYSDATE - 3
         )
ORDER BY C.short_name;

Rows     Row Source Operation
-------  ---------------------------------------------------
      2  SORT ORDER BY (cr=833 r=725 w=0 time=358431 us)
      2   NESTED LOOPS SEMI (cr=833 r=725 w=0 time=358232 us)
     10    TABLE ACCESS FULL CUSTOMERS (cr=38 r=0 w=0 time=2210 us)
      2    TABLE ACCESS BY INDEX ROWID ORDERS (cr=795 r=725
    780     INDEX RANGE SCAN ORDERS_N1 (cr=15 r=13 w=0 time=5601 us)
```

Semi-Join Example #1

What we see on the previous slide:

- **The NL_SJ hint in the subquery suggests a nested loops semi-join.**
- **Oracle performed a nested loops semi-join as requested.**
- **The same benefits as with the hash semi-join apply here, but are now more pronounced.**
- **The query took 0.36 seconds to complete and performed 833 logical reads.**

Semi-Join Example #2

“List the assignments for projects owned by a specified person that involve up to five specified people.”

```
SELECT   DISTINCT A.name, A.code, A.description,
                  A.item_id, A.assignment_id, FI.string0, FI.string1
FROM     relationships R, assignments A, format_items FI,
         relationships R1, relationships R2, relationships R3,
         relationships R4, relationships R5
WHERE    R.user_id = 134546
AND      R.account_id = 134545
AND      R.type_code = 0
AND      A.item_id = R.item_id
AND      FI.item_id = A.item_id
AND      R1.item_id = A.item_id AND R1.status = 5 AND R1.user_id = 137279
AND      R2.item_id = A.item_id AND R2.status = 5 AND R2.user_id = 134555
AND      R3.item_id = A.item_id AND R3.status = 5 AND R3.user_id = 134546
AND      R4.item_id = A.item_id AND R4.status = 5 AND R4.user_id = 137355
AND      R5.item_id = A.item_id AND R5.status = 5 AND R5.user_id = 134556
ORDER BY A.name ASC;
```

Semi-Join Example #2

```
Rows     Row Source Operation
-------  ---------------------------------------------------
    642  SORT UNIQUE (cr=23520269 r=34 w=0 time=2759937104 us)
64339104  TABLE ACCESS BY INDEX ROWID RELATIONSHIPS (cr=23520269 r=34 w=0 time=
95184881   NESTED LOOPS  (cr=7842642 r=23 w=0 time=907238095 us)
2710288     NESTED LOOPS  (cr=2266544 r=23 w=0 time=103840003 us)
 317688      NESTED LOOPS  (cr=484734 r=11 w=0 time=23494451 us)
  50952       NESTED LOOPS  (cr=43280 r=10 w=0 time=2688237 us)
   4146        NESTED LOOPS  (cr=19016 r=3 w=0 time=988374 us)
   1831         NESTED LOOPS  (cr=13353 r=0 w=0 time=608296 us)
   4121          HASH JOIN  (cr=7395 r=0 w=0 time=399488 us)
   2046           TABLE ACCESS BY INDEX ROWID RELATIONSHIPS (cr=7211 r=0 w=0 ti
  17788            INDEX RANGE SCAN RELATIONSHIPS_N3 (cr=71 r=0 w=0 time=81158
   3634           TABLE ACCESS FULL ASSIGNMENTS (cr=184 r=0 w=0 time=25536 us)
   1831          TABLE ACCESS BY INDEX ROWID FORMAT_ITEMS (cr=5958 r=0 w=0 time
   1831           INDEX RANGE SCAN FORMAT_ITEMS_N1 (cr=4127 r=0 w=0 time=115113
   4146         TABLE ACCESS BY INDEX ROWID RELATIONSHIPS (cr=5663 r=3 w=0 time
   4264          INDEX RANGE SCAN RELATIONSHIPS_N2 (cr=3678 r=0 w=0 time=224390
  50952        TABLE ACCESS BY INDEX ROWID RELATIONSHIPS (cr=24264 r=7 w=0 time
  70976         INDEX RANGE SCAN RELATIONSHIPS_N2 (cr=8428 r=0 w=0 time=630831
    ...        ...
```

Semi-Join Example #2

What we see on the previous slides:

- The query was written with conventional joins and DISTINCT instead of semi-joins.
- Oracle performed a conventional nested loops joins to the relationships tables.
- A substantial Cartesian product situation developed, yielding 64,339,104 rows before duplicates were eliminated.
- The query took 2759.94 seconds to complete and performed 23,520,269 logical reads.

Semi-Join Example #2

Rewritten with semi-joins:

```
SELECT   /*+ NO_MERGE (M) */
         DISTINCT M.name, M.code, M.description,
                  M.item_id, M.assignment_id, M.string0, M.string1
FROM     (
         SELECT A.name, A.code, A.description,
                A.item_id, A.assignment_id, FI.string0, FI.string1
         FROM   relationships R, assignments A, format_items FI
         WHERE  R.user_id = 134546
         AND    R.account_id = 134545
         AND    R.type_code = 0
         AND    A.item_id = R.item_id
         AND    FI.item_id = A.item_id
         AND    EXISTS
                (SELECT 1 FROM relationships R1
                 WHERE R1.item_id = A.item_id AND R1.status = 5
                 AND   R1.user_id = 137279)
         AND    EXISTS
                (SELECT 1 FROM relationships R2 ...
         ) M
ORDER BY M.name ASC;
```

Semi-Join Example #2

```
Rows     Row Source Operation
-------  ---------------------------------------------------
    642  SORT UNIQUE (cr=36315 r=89 w=0 time=1107054 us)
   1300   VIEW  (cr=36315 r=89 w=0 time=1085116 us)
   1300    NESTED LOOPS SEMI (cr=36315 r=89 w=0 time=1082232 us)
   1314     NESTED LOOPS SEMI (cr=32385 r=89 w=0 time=1002330 us)
   1314      NESTED LOOPS SEMI (cr=28261 r=89 w=0 time=904654 us)
   1314       NESTED LOOPS SEMI (cr=22822 r=89 w=0 time=737705 us)
   1322        NESTED LOOPS SEMI (cr=18730 r=89 w=0 time=651196 us)
   1831         NESTED LOOPS  (cr=13353 r=89 w=0 time=530670 us)
   4121          HASH JOIN  (cr=7395 r=89 w=0 time=347584 us)
   2046           TABLE ACCESS BY INDEX ROWID RELATIONSHIPS (cr=7211 r=0 w=0 t
  17788            INDEX RANGE SCAN RELATIONSHIPS_N3 (cr=71 r=0 w=0 time=43770
   3634           TABLE ACCESS FULL ASSIGNMENTS (cr=184 r=89 w=0 time=91899 us
   1831          TABLE ACCESS BY INDEX ROWID FORMAT_ITEMS (cr=5958 r=0 w=0 tim
   1831           INDEX RANGE SCAN FORMAT_ITEMS_N1 (cr=4127 r=0 w=0 time=10020
   1322         TABLE ACCESS BY INDEX ROWID RELATIONSHIPS (cr=5377 r=0 w=0 tim
   2472          INDEX RANGE SCAN RELATIONSHIPS_N2 (cr=3664 r=0 w=0 time=61077
   1314        TABLE ACCESS BY INDEX ROWID RELATIONSHIPS (cr=4092 r=0 w=0 time
   1582         INDEX RANGE SCAN RELATIONSHIPS_N2 (cr=2647 r=0 w=0 time=40433
   1314       TABLE ACCESS BY INDEX ROWID RELATIONSHIPS (cr=5439 r=0 w=0 time=
  11011        INDEX RANGE SCAN RELATIONSHIPS_N2 (cr=2639 r=0 w=0 time=65312 u
    ...      ...
```

DatabaseSpecialists

Semi-Join Example #2

What we see on the previous slides:

- Five joins to the relationships table were replaced with EXISTS clauses.
- An in-line view and NO_MERGE hint were used to isolate the DISTINCT keyword so the semi-join access paths would not be defeated.
- Oracle chose nested loops semi-joins to access the relationships tables.
- There were only 1300 candidate rows going into the final sort instead of 64 million.
- The query took 1.11 seconds to complete and performed 36,315 logical reads.

Anti-Joins in Greater Detail

- An important difference between NOT EXISTS and NOT IN.
- How Oracle evaluates NOT EXISTS and NOT IN clauses.
- Anti-join access path prerequisites.
- Hints that affect anti-joins.
- Examples.

DatabaseSpecialists

How NOT EXISTS Treats Nulls

- Null values do not impact the output of a NOT EXISTS clause:
  - **The contents of the select list in a NOT EXISTS subquery do not impact the result.**
  - **A row in the table referenced by the NOT EXISTS subquery will never match a null value because a null value is never equal to another value in Oracle.**

How NOT EXISTS Treats Nulls

- If we added a row to the emp table with a null deptno, the output of the following query would not change. This is because NOT EXISTS effectively ignores null values.

```
SELECT   D.deptno, D.dname
FROM     dept D
WHERE    NOT EXISTS
         (SELECT 1
          FROM   emp E
          WHERE  E.deptno = D.deptno)
ORDER BY D.deptno
```

How NOT IN Treats Nulls

- If the subquery of a NOT IN clause returns at least one row with a null value, the entire NOT IN clause evaluates to false for all rows.

```
SELECT   D.deptno, D.dname
FROM     dept D
WHERE    D.deptno NOT IN
         (SELECT E.deptno
          FROM   emp E)
ORDER BY D.deptno;
```

- If we added a row to the emp table with a null deptno, the above query would retrieve no rows. This is not a bug. See Metalink document 28934.1.

Null Values and NOT EXISTS and NOT IN

- Although you can write a query with NOT EXISTS or NOT IN, the results may not be the same.
- You can make NOT IN treat nulls like NOT EXISTS by adding an extra predicate to the subquery “AND column IS NOT NULL”.

DatabaseSpecialists

Null Values and NOT EXISTS and NOT IN

- If the subquery of a NOT IN clause is capable of retrieving a null value, indexes may get defeated by an implicit query rewrite:

```
SELECT   D.deptno, D.dname
FROM     dept D
WHERE    NOT EXISTS
         (
         SELECT 1
         FROM   emp E
         WHERE  NVL (E.deptno, D.deptno) = D.deptno
         )
ORDER BY D.deptno;
```

How Oracle Evaluates NOT EXISTS and NOT IN

- If a NOT IN subquery is capable of retrieving a null value, Oracle adds the implicit NVL().
- Qualifying NOT IN clauses will usually get an anti-join access path.
- Qualifying NOT EXISTS clauses will occasionally get an anti-join access path.
- In the absence of an anti-join access path Oracle will usually scan the first table and execute the subquery as a filter operation once for each candidate row.

DatabaseSpecialists

Anti-Join Access Path Prerequisites

- Oracle can use an anti-join access path in NOT IN clauses that:
  - **select only columns with NOT NULL constraints, or**
  - **have predicates in the WHERE clause ensuring each selected column is not null.**
- Oracle can sometimes use an anti-join access path in a NOT EXISTS clause, but this behavior does not appear to be documented.

Anti-Join Access Path Prerequisites

- Oracle 8i will only use an anti-join access path if the always_anti_join instance parameter is set to “hash” or “merge”, or if a hint is used.
- Oracle 9i and later evaluate the cost of a nested loops, merge, and hash anti-join and choose the least expensive.

Hints that Affect Anti-Joins

- You can apply the HASH_AJ, MERGE_AJ, and NL_AJ hints to the subquery of a NOT EXISTS or NOT IN clause to tell Oracle which anti-join access path to use. (NL_AJ is not available in Oracle 8i.)
- As with other hints, Oracle will disregard anti-join hints if you ask for the impossible.

Anti-Join Example #1

“List the customers who have not placed an order within the last ten days.”

```
SELECT   C.short_name, C.customer_id
FROM     customers C
WHERE    NOT EXISTS
         (SELECT 1
          FROM   orders O
          WHERE  O.customer_id = C.customer_id
          AND    O.order_date > SYSDATE - 10)
ORDER BY C.short_name;

Rows     Row Source Operation
-------  ---------------------------------------------------
     11  SORT ORDER BY (cr=18707 r=301 w=0 time=22491917 us)
     11   FILTER  (cr=18707 r=301 w=0 time=22491555 us)
   1000    TABLE ACCESS FULL CUSTOMERS (cr=38 r=36 w=0 time=15277 us)
    989    VIEW  (cr=18669 r=265 w=0 time=22365647 us)
    989     HASH JOIN  (cr=18669 r=265 w=0 time=22347234 us)
 100000      INDEX RANGE SCAN ORDERS_N1_CUST_ID (cr=2207 r=208 w=0 time=
5338680      INDEX RANGE SCAN ORDERS_N2_ORD_DATE (cr=16462 r=57 w=0 time
```

Anti-Join Example #1

What we see on the previous slide:

- Oracle chose a filter approach instead of an anti-join access path. (Not surprising because NOT EXISTS was used instead of NOT IN.)
- For each of the 1000 customers, Oracle retrieved all of the customer's orders placed in the last 10 days. (Oracle cleverly hash joined two indexes together to do this.)
- The query took 22.49 seconds to complete and performed 18,707 logical reads.

Anti-Join Example #1

Rewritten with a NOT IN clause:

```
SELECT   C.short_name, C.customer_id
FROM     customers C
WHERE    C.customer_id NOT IN
         (SELECT O.customer_id
          FROM   orders O
          WHERE  O.order_date > SYSDATE - 10)
ORDER BY C.short_name;

Rows     Row Source Operation
-------  ---------------------------------------------------
     11  SORT ORDER BY (cr=5360749 r=4870724 w=0 time=695232973 us)
     11   FILTER  (cr=5360749 r=4870724 w=0 time=695232475 us)
   1000    TABLE ACCESS FULL CUSTOMERS (cr=38 r=129 w=0 time=61614 us)
    989    TABLE ACCESS BY INDEX ROWID ORDERS (cr=5360711 r=4870595 w=0 tim
5359590     INDEX RANGE SCAN ORDERS_N2_ORD_DATE (cr=16520 r=0 w=0 time=2229
```

DatabaseSpecialists

Anti-Join Example #1

What we see on the previous slide:

- **Oracle still chose a filter approach instead of an anti-join access path. (The customer_id column in the orders table is nullable.)**
- **For each of the 1000 customers, Oracle retrieved all orders placed in the last 10 days and searched for a customer match.**
- **The query took 695.23 seconds to complete and performed 5,360,749 logical reads.**

Anti-Join Example #1

Added exclusion of null values:

```
SELECT   C.short_name, C.customer_id
FROM     customers C
WHERE    C.customer_id NOT IN
         (SELECT O.customer_id
          FROM   orders O
          WHERE  O.order_date > SYSDATE - 10
          AND    O.customer_id IS NOT NULL)
ORDER BY C.short_name;

Rows     Row Source Operation
-------  ---------------------------------------------------
     11  SORT ORDER BY (cr=311 r=132 w=98 time=1464035 us)
     11   HASH JOIN ANTI (cr=311 r=132 w=98 time=1463770 us)
   1000    TABLE ACCESS FULL CUSTOMERS (cr=38 r=34 w=0 time=37976 us)
  20910    VIEW  (cr=273 r=98 w=98 time=1318222 us)
  20910     HASH JOIN  (cr=273 r=98 w=98 time=1172207 us)
  20910      INDEX RANGE SCAN ORDERS_N2_ORD_DATE (cr=58 r=0 w=0
 100000      INDEX FAST FULL SCAN ORDERS_N1_CUST_ID (cr=215 r=0 w=0
```

DatabaseSpecialists

Anti-Join Example #1

What we see on the previous slide:

- **The query contains a NOT IN clause with a subquery that cannot return a null value.**
- **Oracle chose to perform a hash anti-join.**
- **Oracle builds a list just one time of customers who placed orders within the last 10 days and then anti-joins this against all customers.**
- **The query took 1.46 seconds to complete and performed 311 logical reads.**

Anti-Join Example #2

“How many calls to the customer service center were placed by users who did not belong to corporate customers?”

```
SELECT COUNT(*)
FROM   calls C
WHERE  C.requestor_user_id NOT IN
       (
       SELECT CM.member_user_id
       FROM   corp_members CM
       );
```

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=12784272 r=1864678 w=0 time=1978321835 us)
      0   FILTER  (cr=12784272 r=1864678 w=0 time=1978321817 us)
 184965    TABLE ACCESS FULL CALLS (cr=3588 r=1370 w=0 time=979769 us)
  61032    TABLE ACCESS FULL CORP_MEMBERS (cr=12780684 r=1863308 w=0 time=
```

Anti-Join Example #2

What we see on the previous slide:

- Oracle chose a filter approach instead of an anti-join access path. (The member_user_id column in the corp_members table is nullable.)
- For each of the 184,965 calls, Oracle had to scan the corp_members table to see if the caller belonged to a corporate customer.
- The query took 1978.32 seconds to complete and performed 12,784,272 logical reads.

Anti-Join Example #2

Added exclusion of null values:

```
SELECT COUNT(*)
FROM   calls C
WHERE  C.requestor_user_id NOT IN
       (
       SELECT CM.member_user_id
       FROM   corp_members CM
       WHERE  CM.member_user_id IS NOT NULL
       );

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=3790 r=3906 w=420 time=5450615 us)
      0   HASH JOIN ANTI (cr=3790 r=3906 w=420 time=5450591 us)
 184965    TABLE ACCESS FULL CALLS (cr=3588 r=3480 w=0 time=646554 us)
  81945    INDEX FAST FULL SCAN CORP_MEMBERS_USER_ID (cr=202 r=6 w=0 time=
```

DatabaseSpecialists

Anti-Join Example #2

What we see on the previous slide:

- Oracle can now use an anti-join access path because the NOT IN subquery can no longer return a null value. Oracle chose a hash anti-join.
- Oracle scans the calls table and an index on the corp_members table once each, instead of thousands of times.
- The query took 5.45 seconds to complete and performed 3,790 logical reads.

Anti-Join Example #2

Let’s try a few other query changes for the sake of learning:

- **Add a MERGE_AJ hint. Results: Same number of logical reads as a hash join, but a little more CPU time used for sorting.**
- **Use NOT EXISTS instead of NOT IN. Results on the next slide.**

Anti-Join Example #2

Rewritten with a NOT EXISTS clause:

```
SELECT COUNT(*)
FROM   calls C
WHERE  NOT EXISTS
       (
       SELECT 1
       FROM   corp_members CM
       WHERE  CM.member_user_id = C.requestor_user_id
       );

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=125652 r=3489 w=0 time=3895569 us)
      0   FILTER  (cr=125652 r=3489 w=0 time=3895547 us)
 184965    TABLE ACCESS FULL CALLS (cr=3588 r=3489 w=0 time=906699 us)
  61032    INDEX RANGE SCAN CORP_MEMBERS_USER_ID (cr=122064 r=0 w=0
```

Anti-Join Example #2

What we see on the previous slide:

- **Oracle performs a filter operation instead of using an anti-join access path.**
- **The possibility of null values in the corp_members table does not defeat the index.**
- **Oracle scans an index on the corp_members table once for each row in the calls table. This makes for lots of logical reads, but in this case was actually faster because the sort required for a hash join has been eliminated.**
- **The query took 3.89 seconds to complete and performed 125,652 logical reads.**

Wrapping Up

- The semi-join and anti-join are two special ways of joining data from multiple tables in a query.
- We use EXISTS, IN, NOT EXISTS, and NOT IN clauses to denote these special types of joins.
- Oracle has special access paths that can make semi-joins and anti-joins extremely efficient.
- Understanding how semi-joins and anti-joins work—and how Oracle implements the relevant data access paths—will enable you to write very efficient SQL and dramatically speed up an entire class of queries.

White Paper

- All of the code samples and TKPROF reports you couldn’t read in these slides.
- More explanation that we didn’t have time to discuss today.
- Additional examples and points of interest.
- Download: www.dbspecialists.com/presentations

DatabaseSpecialists

ORACLE CERTIFIED SOLUTION PARTNER

Contact Information

Roger Schrag

Database Specialists, Inc.
388 Market Street, Suite 400
San Francisco, CA 94111

Tel: 415/344-0500
Email: rschrag@dbspecialists.com
Web: www.dbspecialists.com

DatabaseSpecialists