

[Sign In/Register](#) [Help](#) [Country](#) [Communities](#) [I am a...](#) [I want to...](#)[Products](#) [Solutions](#) [Downloads](#) [Store](#) [Support](#) [Training](#) [Partners](#) [About](#)[OTN](#)[Oracle Technology Network](#)[Oracle Magazine Issue Archive](#)

2006

September 2006

[Oracle Magazine Online](#)[2015](#)[2014](#)[2013](#)[2012](#)[2011](#)[2010](#)[2009](#)**TECHNOLOGY: Ask Tom**

## On ROWNUM and Limiting Results

*By Tom Kyte* **Our technologist explains how ROWNUM works and how to make it work for you.**

This issue's Ask Tom column is a little different from the typical column. I receive many questions about how to perform top- N and pagination queries in Oracle Database, so I decided to provide an excerpt from the book *Effective Oracle by Design* (Oracle Press, 2003) in hopes of answering a lot of these questions with this one column. Note that the content here has been modified from the original to fit the space and format.

As Published In



September/October 2006

### Limiting Result Sets

ROWNUM is a magic column in Oracle Database that gets many people into trouble. When you learn what it is and how it works, however, it can be very useful. I use it for two main things:

To perform top- N processing. This is similar to using the LIMIT clause, available in some other databases.

To paginate through a query, typically in a stateless environment such as the Web. I use this technique on the [asktom.oracle.com](http://asktom.oracle.com) Web site.

I'll take a look at each of these uses after I review how ROWNUM works.

### How ROWNUM Works

ROWNUM is a pseudocolumn (not a real column) that is available in a query. ROWNUM will be assigned the numbers 1, 2, 3, 4, ... N , where N is the number of rows in the set ROWNUM is used with. A ROWNUM value is not assigned permanently to a row (this is a common misconception). A row in a table does not have a number; you cannot ask for row 5 from a table—there is no such thing.

Also confusing to many people is when a ROWNUM value is actually assigned. A ROWNUM value is assigned to a row after it passes the predicate phase of the query but before the query does any sorting or aggregation. Also, a ROWNUM value is incremented only after it is assigned, which is why the following query will never return a row:

```
select *
  from t
 where ROWNUM > 1;
```

Because ROWNUM > 1 is not true for the first row, ROWNUM does not advance to 2. Hence, no ROWNUM value ever gets to be greater than 1. Consider a query with this structure:

```
select ..., ROWNUM
  from t
 where <where clause>
 group by <columns>
 having <having clause>
 order by <columns>;
```

Think of it as being processed in this order:

1. The FROM/WHERE clause goes first.
2. ROWNUM is assigned and incremented to each output row from the FROM/WHERE clause.
3. SELECT is applied.
4. GROUP BY is applied.
5. HAVING is applied.
6. ORDER BY is applied.

That is why a query in the following form is almost certainly an error:

```
select *
  from emp
 where ROWNUM <= 5
 order by sal desc;
```

The intention was most likely to get the five highest-paid people—a top- N query. What the query will return is five random records (the first five the query happens to hit), sorted by salary. The procedural pseudocode for this query is as follows:

```
ROWNUM = 1
for x in
( select * from emp )
loop
  exit when NOT(ROWNUM <= 5)
  OUTPUT record to temp
  ROWNUM = ROWNUM+1
end loop
SORT TEMP
```

It gets the first five records and then sorts them. A query with WHERE ROWNUM = 5 or WHERE ROWNUM > 5 doesn't make sense. This is because a ROWNUM value is assigned to a row during the predicate evaluation and gets incremented only after a row passes the WHERE clause.

Here is the correct version of this query:

```
select *
  from
( select *
  from emp
  order by sal desc )
where ROWNUM <= 5;
```

This version will sort EMP by salary descending and then return the first five records it encounters (the top-five records). As you'll see in the top-N discussion coming up shortly, Oracle Database doesn't really sort the entire result set—it is smarter than that—but conceptually that is what takes place.

#### Top- N Query Processing with ROWNUM

In a top- N query, you are generally interested in taking some complex query, sorting it, and then retrieving just the first N rows (the top N rows). ROWNUM has a top- N optimization that facilitates this type of query. You can use ROWNUM to avoid a massive sort of large sets. I'll discuss how it does this conceptually and then look at an example.

Suppose you have a query in this form:

```
select ...
  from ...
 where ...
 order by columns;
```

Assume that this query returns a lot of data: thousands, hundreds of thousands, or more rows. However, you are interested only in the top N—say the top 10 or top 100. There are two ways to approach this:

Have the client application run that query and fetch just the first N rows.

Use that query as an inline view, and use ROWNUM to limit the results, as in SELECT \* FROM ( your\_query\_here ) WHERE ROWNUM <= N.

The second approach is by far superior to the first, for two reasons. The lesser of the two reasons is that it requires less work by the client, because the database takes care of limiting the result set. The more important reason is the special processing the database can do to give you just the top N rows. Using the top- N query means that you have given the database extra information. You have told it, "I'm interested only in getting N rows; I'll never consider the rest." Now, that doesn't sound too earth-shattering until you think about sorting—how sorts work and what the server would need to do. Let's walk through the two approaches with a sample query:

```
select *
  from t
 order by unindexed_column;
```

Now, assume that T is a big table, with more than one million records, and each record is "fat"—say, 100 or more bytes. Also assume that UNINDEXED\_COLUMN is, as its name implies, a column that is not indexed. And assume that you are interested in getting just the first 10 rows. Oracle Database would do the following:

1. Run a full-table scan on T.
2. Sort T by UNINDEXED\_COLUMN. This is a full sort.
3. Presumably run out of sort area memory and need to swap temporary extents to disk.
4. Merge the temporary extents back to get the first 10 records when they are requested.
5. Clean up (release) the temporary extents as you are finished with them.

Now, that is a lot of I/O. Oracle Database has most likely copied the entire table into TEMP and written it out, just to get the first 10 rows.

Next, let's look at what Oracle Database can do conceptually with a top- N query:

```
select *
  from
(select *
  from t
  order by unindexed_column)
where ROWNUM < :N;
```

In this case, Oracle Database will take these steps:

1. Run a full-table scan on T, as before (you cannot avoid this step).
2. In an array of : N elements (presumably in memory this time), sort only : N rows.

The first N rows will populate this array of rows in sorted order. When the N +1 row is fetched, it will be compared to the last row in the array. If it would go into slot N +1 in the array, it gets thrown out. Otherwise, it is added to this array and sorted and one of the existing rows is discarded. Your sort area holds N rows maximum, so instead of sorting one million rows, you sort N rows.

This seemingly small detail of using an array concept and sorting just N rows can lead to huge gains in performance and resource usage. It takes a lot less RAM to sort 10 rows than it does to sort one million rows (not to mention TEMP space usage).

Using the following table T, you can see that although both approaches get the same results, they use radically different amounts of resources:

```
create table t
as
select dbms_random.value(1,1000000)
id,
```

```

        rpad('*',40,'*') data
    from dual
connect by level <= 100000;

begin
dbms_stats.gather_table_stats
( user, 'T');
end;
/

```

Now enable tracing, via

```

exec
dbms_monitor.session_trace_enable
(waits=>true);

```

And then run your top- N query with ROWNUM:

```

select *
  from
(select *
  from t
 order by id)
where rownum <= 10;

```

And finally run a "do-it-yourself" query that fetches just the first 10 records:

```

declare
cursor c is
select *
  from t
 order by id;
l_rec c%rowtype;
begin
  open c;
  for i in 1 .. 10
    loop
      fetch c into l_rec;
      exit when c%notfound;
    end loop;
  close c;
end;
/

```

After executing this query, you can use TKPROF to format the resulting trace file and review what happened. First examine the top- N query, as shown in Listing 1.

**Code Listing 1:** Top- N query using ROWNUM

```

select *
  from
(select *
  from t
 order by id)
where rownum <= 10

```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	2	0.04	0.04	0	949	0	10
total	4	0.04	0.04	0	949	0	10

Rows	Row	Source Operation
10		COUNT STOPKEY (cr=949 pr=0 pw=0 time=46997 us)
10		VIEW (cr=949 pr=0 pw=0 time=46979 us)
10		SORT ORDER BY STOPKEY (cr=949 pr=0 pw=0 time=46961 us)
100000		TABLE ACCESS FULL T (cr=949 pr=0 pw=0 time=400066 us)

The query read the entire table (because it had to), but by using the SORT ORDER BY STOPKEY step, it was able to limit its use of temporary space to just 10 rows. Note the final Row Source Operation line—it shows that the query did 949 logical I/Os in total (cr=949), performed no physical reads or writes (pr=0 and pw=0), and took 400066 millionths of a second (0.04 seconds). Compare that with the do-it-yourself approach shown in Listing 2.

**Code Listing 2:** Do-it-yourself query without ROWNUM

```

SELECT * FROM T ORDER BY ID

```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	2	0.00	0.00	0	0	0	0
Fetch	10	0.35	0.40	155	949	6	10
total	13	0.36	0.40	155	949	6	10

Rows	Row	Source Operation
10		SORT ORDER BY (cr=949 pr=155 pw=891 time=401610 us)
100000		TABLE ACCESS FULL T (cr=949 pr=0 pw=0 time=400060 us)

Elapsed times include waiting for the following events:

Event waited on	Times
direct path write temp	33
direct path read temp	5

As you can see, this result is very different. Notably, the elapsed/CPU times are significantly higher, and the final Row Source Operation lines provide insight into why this is. You had to perform a sort to disk, which you can see with the pw=891 (physical writes). Your query performed some direct path reads and writes—the sort of 100,000 records (instead of just the 10 we are ultimately interested in) took place on disk—adding considerably to the runtime/resource usage of your query.

#### Pagination with ROWNUM

My all-time-favorite use of ROWNUM is pagination. In this case, I use ROWNUM to get rows N through M of a result set. The general form is as follows:

```
select *
  from ( select /*+ FIRST_ROWS(n) */
        a.*, rownum rnum
          from ( your_query_goes_here,
                with order by ) a
         where rownum <=
               :MAX_ROW_TO_FETCH )
 where rnum >= :MIN_ROW_TO_FETCH;
```

where

FIRST\_ROWS(N) tells the optimizer, "Hey, I'm interested in getting the first rows, and I'll get N of them as fast as possible."

:MAX\_ROW\_TO\_FETCH is set to the last row of the result set to fetch—if you wanted rows 50 to 60 of the result set, you would set this to 60.

:MIN\_ROW\_TO\_FETCH is set to the first row of the result set to fetch, so to get rows 50 to 60, you would set this to 50.

The concept behind this scenario is that an end user with a Web browser has done a search and is waiting for the results. It is imperative to return the first result page (and second page, and so on) as fast as possible. If you look at that query closely, you'll notice that it incorporates a top- N query (get the first :MAX\_ROW\_TO\_FETCH rows from your query) and hence benefits from the top- N query optimization I just described. Further, it returns over the network to the client only the specific rows of interest—it removes any leading rows from the result set that are not of interest.

One important thing about using this pagination query is that the ORDER BY statement should order by something unique. If what you are ordering by is not unique, you should add something to the end of the ORDER BY to make it so. If you sort 100 records by SALARY, for example, and they all have the same SALARY value, then specifying rows 20 to 25 does not really have any meaning. In order to see this, use a small table with lots of duplicated ID values:

```
SQL> create table t
2 as
3 select mod(level,5) id,
4      trunc(dbms_random.value(1,100)) data
5      from dual
6 connect by level <= 10000;
Table created.
```

And then query rows 148 to 150 and 151 after sorting by the ID column:

```
SQL> select *
2   from
3   (select a.*, rownum rnum
4     from
5     (select id, data
6       from t
7      order by id) a
8    where rownum <= 150
9   )
10  where rnum >= 148;
```

ID	DATA	RNUM
0	38	148
0	64	149
0	53	150

```
SQL>
SQL> select *
2   from
3   (select a.*, rownum rnum
4     from
5     (select id, data
6       from t
7      order by id) a
8    where rownum <= 151
```

```

9 )
10 where rnum >= 148;

```

ID	DATA	RNUM
0	59	148
0	38	149
0	64	150
0	53	151

Note in this case that one time for row 148, the result returned DATA=38, and that the next time, the result returned DATA=59. Both queries are returning exactly the right answer, given what you've requested: Sort the data by ID, throw out the first 147 rows, and return the next 3 or 4 rows. Both of them do that, but because ID has so many duplicate values, the query cannot do it deterministically—the same sort order is not assured from run to run of the query. In order to correct this, you need to add something unique to the ORDER BY. In this case, just use ROWID:

```

SQL> select *
2   from
3   (select a.*, rownum rnum
4     from
5     (select id, data
6       from t
7       order by id, rowid) a
8     where rownum <= 150
9   )
10  where rnum >= 148;

```

ID	DATA	RNUM
0	45	148
0	99	149
0	41	150

```

SQL>
SQL> select *
2   from
3   (select a.*, rownum rnum
4     from
5     (select id, data
6       from t
7       order by id, rowid) a
8     where rownum <= 151
9   )
10  where rnum >= 148;

```

ID	DATA	RNUM
0	45	148
0	99	149
0	41	150
0	45	151

Now the query is very deterministic. ROWID is unique within a table, so if you use ORDER BY ID and then within ID you use ORDER BY ROWID, the rows will have a definite, deterministic order and the pagination query will deterministically return the rows as expected.

### ROWNUM Wrap-Up

I'll hazard a guess that you and many other readers now have a newfound respect for ROWNUM and understand these aspects:

How ROWNUM is assigned, so you can write bug-free queries that use it

How it affects the processing of your query, so you can use it to paginate a query on the Web

How it can reduce the work performed by your query, so that top- N queries that once consumed a lot of TEMP space now use none and return results much faster.

### Oracle OpenWorld Plans

This is the Oracle OpenWorld issue of Oracle Magazine. I really enjoy the weeklong conference—I get to meet a lot of people face-to-face that I've only met electronically. So if you're at the conference, I hope to see you at the talk I'll be giving there. (If you haven't already guessed, my talk will be about the database and development.) In addition, I plan to be at a "Meet the Experts" event hosted by OTN—the Oracle Technology Network. I've done this event in years past, and it's always been a great forum for one-on-one discussions as well as group talks. Check your program schedule for the dates and times of my talk and OTN events.

In addition, I'll of course be blogging at [tkyte.blogspot.com](http://tkyte.blogspot.com)—with pictures—about what is going on at the conference. In addition to my blog, be sure to check out [OTN](#), which will be full of downloadable show content including podcasts, video streams, and presentation material.

**Tom Kyte** has worked for Oracle since 1993. He is a vice president in the Oracle Public Sector group and the author of *Expert Oracle Database Architecture: 9i and 10g Programming Techniques and Solutions* (Apress, 2005) and *Effective Oracle by Design* (Oracle Press, 2003), among other books.

[Send us your comments](#)

### Next Steps



#### ASK Tom

Oracle Vice President Tom Kyte answers your most difficult technology questions. Highlights from that forum appear in this column.

[asktom.oracle.com](http://asktom.oracle.com)

#### READ more Tom

[Expert Oracle Database Architecture: 9i and 10g Programming Techniques and Solutions](#)  
[Effective Oracle By Design](#)

 E-mail this page  Printer View**ORACLE CLOUD**

[Learn About Oracle Cloud Computing](#)  
[Get a Free Trial](#)  
[Learn About DaaS](#)  
[Learn About SaaS](#)  
[Learn About PaaS](#)  
[Learn About IaaS](#)  
[Learn About Private Cloud](#)  
[Learn About Managed Cloud](#)

**JAVA**

[Learn About Java](#)  
[Download Java for Consumers](#)  
[Download Java for Developers](#)  
[Java Resources for Developers](#)  
[Java Cloud Service](#)  
[Java Magazine](#)

**CUSTOMERS AND EVENTS**

[Explore and Read Customer Stories](#)  
[All Oracle Events](#)  
[Oracle OpenWorld](#)  
[JavaOne](#)

**COMMUNITIES**

[Blogs](#)  
[Discussion Forums](#)  
[Wikis](#)  
[Oracle ACEs](#)  
[User Groups](#)  
[Social Media Channels](#)

**SERVICES AND STORE**

[Log In to My Oracle Support](#)  
[Training and Certification](#)  
[Become a Partner](#)  
[Find a Partner Solution](#)  
[Purchase from the Oracle Store](#)

**CONTACT AND CHAT**

**Phone: +1.800.633.0738**  
[Global Contacts](#)  
[Oracle Support](#)  
[Partner Support](#)

---

[Subscribe](#) [Careers](#) [Contact Us](#) [Site Maps](#) [Legal Notices](#) [Terms of Use](#) [Privacy](#) [Cookie Preferences](#) [Oracle Mobile](#)