Complex Queries

7 min. read · View original

Suppose that you want to start lumping together information from multiple rows. For example, you're interested in JOINing users with their classified ads. That will give you one row per ad posted. But you want to mush all the rows together for a particular user and just look at the most recent posting time. What you need is the GROUP BY construct:

```
select users.user_id, users.email,
max(classified_ads.posted)
from users, classified_ads
where users.user_id = classified_ads.user_id
group by users.user_id, users.email
order by upper(users.email);

   USER_ID EMAIL
MAX(CLASSI
---------- ----------------------------------- --
--------
     39406 102140.1200@compuserve.com
1998-10-08
     39842 102144.2651@compuserve.com
1998-12-13
     41426 50@seattle.va.gov
1997-01-13
     37428 71730.345@compuserve.com
1998-11-24
     35970 aaibrahim@earthlink.net
1998-11-08
     36671 absolutsci@aol.com
1998-10-06
     35781 alevy@agtnet.com
1997-07-14
```

```
     40111 alexzorba@aol.com
1998-09-25
     39060 amchiu@worldnet.att.net
1998-12-11
     35989 andrew.c.beckner@bankamerica.com
1998-08-13
     33923 andy_roo@mit.edu
1998-12-10
```

The `group by users.user_id, users.email` tells SQL to "lump together all the rows that have the same values in these two columns." In addition to the grouped by columns, we can run aggregate functions on the columns that aren't being grouped. For example, the MAX above applies to the posting dates for the rows in a particular group. We can also use COUNT to see at a glance how active and how recently active a user has been:

```

select users.user_id, users.email, count(*),
max(classified_ads.posted)
from users, classified_ads
where users.user_id = classified_ads.user_id
group by users.user_id, users.email
order by upper(users.email);

   USER_ID EMAIL
COUNT(*) MAX(CLASSI
---------- ----------------------------------- --
-------- ----------
     39406 102140.1200@compuserve.com
3 1998-10-08
     39842 102144.2651@compuserve.com
3 1998-12-13
     41426 50@seattle.va.gov
1 1997-01-13
     37428 71730.345@compuserve.com
3 1998-11-24
     35970 aaibrahim@earthlink.net
1 1998-11-08
     36671 absolutsci@aol.com
2 1998-10-06
     35781 alevy@agtnet.com
1 1997-07-14
```

```
     40111 alexzorba@aol.com
1 1998-09-25
     39060 amchiu@worldnet.att.net
1 1998-12-11
     35989 andrew.c.beckner@bankamerica.com
1 1998-08-13
     33923 andy_roo@mit.edu
1 1998-12-10
```

A publisher who was truly curious about this stuff probably wouldn't be interested in these results alphabetically. Let's find our most recently active users. At the same time, let's get rid of the unsightly "MAX(CLASSI" at the top of the report:

```
select users.user_id,
       users.email,
       count(*) as how_many,
       max(classified_ads.posted) as how_recent
from users, classified_ads
where users.user_id = classified_ads.user_id
group by users.user_id, users.email
order by how_recent desc, how_many desc;

   USER_ID EMAIL
HOW_MANY HOW_RECENT
---------- ----------------------------------- --
-------- ----------
     39842 102144.2651@compuserve.com
3 1998-12-13
     39968 mkravit@mindspring.com
1 1998-12-13
     36758 mccallister@mindspring.com
1 1998-12-13
     38513 franjeff@alltel.net
1 1998-12-13
     34530 nverdesoto@earthlink.net
3 1998-12-13
     34765 jrl@blast.princeton.edu
1 1998-12-13
     38497 jeetsukumaran@pd.jaring.my
1 1998-12-12
     38879 john.macpherson@btinternet.com
```

```
5 1998-12-12
     37808 eck@coastalnet.com
1 1998-12-12
     37482 dougcan@arn.net
1 1998-12-12
```

Note that we were able to use our *correlation names* of "how_recent" and "how_many" in the ORDER BY clause. The `desc` ("descending") directives in the ORDER BY clause instruct Oracle to put the largest values at the top. The default sort order is from smallest to largest ("ascending").

Upon close inspection, the results are confusing. We instructed Oracle to rank first by date and second by number of postings. Yet for 1998-12-13 we don't see both users with three total postings at the top. That's because Oracle dates are precise to the second even when the hour, minute, and second details are not displayed by SQL*Plus. A better query would include the clause

```
order by trunc(how_recent) desc, how_many desc
```

where the built-in Oracle function `trunc` truncates each date to midnight on the day in question.

Finding co-moderators: The HAVING Clause

The WHERE clause restricts which rows are returned. The HAVING clause operates analogously but on groups of rows. Suppose, for example, that we're interested in finding those users who've contributed heavily to our discussion forum:

```
select user_id, count(*) as how_many
from bboard
group by user_id
order by how_many desc;

   USER_ID   HOW_MANY
---------- ----------
```

```
     34474       1922
     35164        985
     41112        855
     37021        834
     34004        823
     37397        717
     40375        639
...
     33963          1
     33941          1
     33918          1

7348 rows selected.
```

Seventy three hundred rows. That's way too big considering that we are only interested in nominating a couple of people. Let's restrict to more recent activity. A posting contributed three years ago is not necessarily evidence of interest in the community right now.

```
select user_id, count(*) as how_many
from bboard
where posting_time + 60 > sysdate
group by user_id
order by how_many desc;

   USER_ID   HOW_MANY
---------- ----------
     34375         80
     34004         79
     37903         49
     41074         46
...

1120 rows selected.
```

We wanted to kill rows, not groups, so we did it with a WHERE clause. Let's get rid of the people who are already serving as maintainers.

```
select user_id, count(*) as how_many
from bboard
```

```
where not exists (select 1 from
                  bboard_authorized_maintainers
bam
                  where bam.user_id =
bboard.user_id)
and posting_time + 60 > sysdate
group by user_id
order by how_many desc;
```

The concept of User ID makes sense for both rows and groups, so we can restrict our results either with the extra WHERE clause above or by letting the relational database management system produce the groups and then we'll ask that they be tossed out using a HAVING clause:

```
select user_id, count(*) as how_many
from bboard
where posting_time + 60 > sysdate
group by user_id
having not exists (select 1 from
                  bboard_authorized_maintainers
bam
                  where bam.user_id =
bboard.user_id)
order by how_many desc;
```

This doesn't get to the root cause of our distressingly large query result: we don't want to see groups where `how_many` is less than 30. Here's the SQL for "show me users who've posted at least 30 messages in the past 60 days, ranked in descending order of volubility":

```
select user_id, count(*) as how_many
from bboard
where posting_time + 60 > sysdate
group by user_id
having count(*) >= 30
order by how_many desc;

   USER_ID   HOW_MANY
---------- ----------
     34375         80
     34004         79
     37903         49
     41074         46
     42485         46
     35387         30
     42453         30


7 rows selected.
```

We had to do this in a HAVING clause because the number of rows in a group is a concept that doesn't make sense at the per-row level on which WHERE clauses operate.

Oracle 8's SQL parser is too feeble to allow you to use the `how_many` correlation variable in the HAVING clause. You therefore have to repeat the `count(*)` incantation.

Set Operations: UNION, INTERSECT, and MINUS

Oracle provides set operations that can be used to combine rows produced by two or more separate SELECT statements. UNION pools together the rows returned by two queries, removing any duplicate rows. INTERSECT combines the result sets of two queries by removing any rows that are not present in both. MINUS combines the results of two queries by taking the the first result set and subtracting from it any rows that are also found in the second. Of the three, UNION is the most useful in practice.

In the ArsDigita Community System ticket tracker, people reporting a bug or requesting a feature are given a menu of potential deadlines. For some projects, common project deadlines are stored in the `ticket_deadlines` table. These should appear in an HTML SELECT form element. We also, however, want some options like "today", "tomorrow", "next week", and "next month". The easiest way to handle these is to query the `dual` table to perform some date arithmetic. Each of these queries will return one row and if we UNION four of them together with the query from `ticket_deadlines`, we can have the basis for a very simple Web script to present the options:

```
select
  'today - ' || to_char(trunc(sysdate),'Mon
FMDDFM'),
  trunc(sysdate) as deadline
from dual
UNION
select
  'tomorrow - '|| to_char(trunc(sysdate+1),'Mon
FMDDFM'),
  trunc(sysdate+1) as deadline
from dual
UNION
select
  'next week - '|| to_char(trunc(sysdate+7),'Mon
FMDDFM'),
  trunc(sysdate+7) as deadline
from dual
UNION
select
  'next month - '||
to_char(trunc(ADD_MONTHS(sysdate,1)),'Mon
FMDDFM'),
  trunc(ADD_MONTHS(sysdate,1)) as deadline
from dual
UNION
select
  name || ' - ' || to_char(deadline, 'Mon
FMDDFM'),
  deadline
from ticket_deadlines
where project_id = :project_id
and deadline >= trunc(sysdate)
order by deadline
```

will produce something like

The INTERSECT and MINUS operators are seldom used. Here are some contrived examples. Suppose that you collect contest entries by Web users, each in a separate table:

```
create table trip_to_paris_contest (
        user_id         references users,
        entry_date      date not null
);

create table camera_giveaway_contest (
        user_id         references users,
        entry_date      date not null
);
```

Now let's populate with some dummy data:

```
-- all three users love to go to Paris
insert into trip_to_paris_contest values
(1,'2000-10-20');
insert into trip_to_paris_contest values
(2,'2000-10-22');
insert into trip_to_paris_contest values
(3,'2000-10-23');

-- only User #2 is a camera nerd
insert into camera_giveaway_contest values
(2,'2000-05-01');
```

Suppose that we've got a new contest on the site. This time we're giving away a trip to Churchill, Manitoba to photograph polar bears. We assume that the most interested users will be those who've entered both the travel and the camera contests. Let's get their user IDs so that we can notify them via email (spam) about the new contest:

```
select user_id from trip_to_paris_contest
intersect
```

```
select user_id from camera_giveaway_contest;

   USER_ID
----------
         2
```

Or suppose that we're going to organize a personal trip to Paris and want to find someone to share the cost of a room at the Crillon. We can assume that anyone who entered the Paris trip contest is interested in going. So perhaps we should start by sending them all email. On the other hand, how can one enjoy a quiet evening with the absinthe bottle if one's companion is constantly blasting away with an electronic flash? We're interested in people who entered the Paris trip contest but who *did not* enter the camera giveaway:

```

select user_id from trip_to_paris_contest
minus
select user_id from camera_giveaway_contest;

   USER_ID
----------
         1
         3
```

Next: Transactions

---

philg@mit.edu

Reader's Comments

> In less trivial uses of UNION, you can use UNION ALL, instructing Oracle not to remove duplicates and saving the sort if you know there aren't going to be any duplicate rows(or maybe don't care)
>
> -- Neal Sidhwaney, December 10, 2002

> Another example of using MINUS is shown in the following crazy-looking (and Oracle-specific [1]) query which selects the 91st through 100th rows of a subquery.

```
with subq as (select * from my_table order by
my_id)

select * from subq
where rowid in (select rowid from subq
                where rownum <= 100
                    MINUS
                select rowid from subq
                where rownum <= 90)
```

[1] The Oracle dependencies in this query are rowid and rownum. Other databases have other means of limiting query results by row position.

-- Kevin Murphy, February 10, 2003

And in PostgreSQL (and MySQL too for that matter) it is as simple as:

select * from my_table order by my_id limit 90,10

An easier way for Oracle (according to a random post in a devshed.com forum I googled) would be like this:

select * from my_table order by my_id where rownum between 90,100

(Though the whole point about how to use MINUS is well taken)

-- Gabriel Ricard, February 26, 2003

Oops. I was wrong. Phil emailed me and explained that my rownum example won't work (just goes to show that not everything you find on the internet is right!).

-- Gabriel Ricard, March 17, 2003

Add a comment