

CS 325 - SQL Reading Packet 5: "order by, group by, and having"

SOURCES:

- * [Oracle9i Programming: A Primer](#), Rajshekhar Sunderraman, Addison Wesley.
- * Classic Oracle example tables `emp1` and `dept`, adapted somewhat over the years

more select clauses: order by, group by, and having

The `select` statement can have some additional optional clauses, in addition to the clauses discussed thus far. In this lab, we'll be discussing three such clauses: `order by`, `group by`, and `having`.

order by

As you have written your queries, have you ever wished that the rows in the result would appear in a different order? That's all that `order by` does -- it has **absolutely no effect** on what is stored in the database (since, indeed, a `select` never effects what is stored in a database), but it does allow the user to specify the order in which they would like the resulting rows to be displayed.

This should always be the final clause of a `select` (and indeed, syntactically, it only belongs on an outer-`select` - since it is really just specifying a final row-display order, it wouldn't make sense inside of a subselect, if you think about it.) And, in its simplest form, you just follow the `order by` by the column (or the projected expression) that you want the rows to be ordered by.

(If you have ever used SORT BY to change the order of rows in an Excel database, it is a similar idea -- you specify the column you want to sort the Excel table's rows by, and then the rows are sorted in order of that column's values.)

For example, say that you want to select all of the rows of the `emp1` table, but displaying those selected rows in order of increasing salary. Then, you'd `order by salary`:

```
select  *
from    emp1
order  by salary;
```

...resulting in:

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7369	Smith	Clerk	7902	17-DEC-12	800		200
7900	James	Clerk	7698	03-DEC-17	950		300
7876	Adams	Clerk	7788	23-SEP-18	1100		400
7654	Martin	Sales	7698	28-SEP-18	1250	1400	300

7521	Ward	Sales	7698	22-FEB-19	1250	500	300
7934	Miller	Clerk	7782	23-JAN-16	1300		100
7844	Turner	Sales	7698	08-SEP-19	1500	0	300
7499	Michaels	Sales	7698	20-FEB-18	1600	300	300
7782	Raimi	Manager	7839	09-JUN-12	2450		100
7698	Blake	Manager	7839	01-MAY-13	2850		300
7566	Jones	Manager	7839	02-APR-12	2975		200

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7788	Scott	Analyst	7566	09-NOV-18	3000		200
7902	Ford	Analyst	7566	03-DEC-12	3000		200
7839	King	President		17-NOV-11	5000		500

14 rows selected.

If you'd like to see the selected rows in order of increasing hiredate,

```
select *
from     empl
order by hiredate;
```

...and the displayed results for this query will be:

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7839	King	President		17-NOV-11	5000		500
7566	Jones	Manager	7839	02-APR-12	2975		200
7782	Raimi	Manager	7839	09-JUN-12	2450		100
7902	Ford	Analyst	7566	03-DEC-12	3000		200
7369	Smith	Clerk	7902	17-DEC-12	800		200
7698	Blake	Manager	7839	01-MAY-13	2850		300
7934	Miller	Clerk	7782	23-JAN-16	1300		100
7900	James	Clerk	7698	03-DEC-17	950		300
7499	Michaels	Sales	7698	20-FEB-18	1600	300	300
7876	Adams	Clerk	7788	23-SEP-18	1100		400
7654	Martin	Sales	7698	28-SEP-18	1250	1400	300

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7788	Scott	Analyst	7566	09-NOV-18	3000		200
7521	Ward	Sales	7698	22-FEB-19	1250	500	300
7844	Turner	Sales	7698	08-SEP-19	1500	0	300

14 rows selected.

Or, if you'd like to see the empl rows in order of job_title:

```
select *
from     empl
```

```
order by job_title;
```

...and this query has the results:

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7902	Ford	Analyst	7566	03-DEC-12	3000		200
7788	Scott	Analyst	7566	09-NOV-18	3000		200
7369	Smith	Clerk	7902	17-DEC-12	800		200
7934	Miller	Clerk	7782	23-JAN-16	1300		100
7876	Adams	Clerk	7788	23-SEP-18	1100		400
7900	James	Clerk	7698	03-DEC-17	950		300
7698	Blake	Manager	7839	01-MAY-13	2850		300
7782	Raimi	Manager	7839	09-JUN-12	2450		100
7566	Jones	Manager	7839	02-APR-12	2975		200
7839	King	President		17-NOV-11	5000		500
7844	Turner	Sales	7698	08-SEP-19	1500	0	300

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7654	Martin	Sales	7698	28-SEP-18	1250	1400	300
7521	Ward	Sales	7698	22-FEB-19	1250	500	300
7499	Michaels	Sales	7698	20-FEB-18	1600	300	300

14 rows selected.

So, using `order by`, you can see the columns projected from the rows selected in any order that you would like.

Note that what you choose to order by does not affect which columns are projected, or the order in which the projected columns appear -- that is determined completely by the `select` clause.

For example, I don't even have to project the column I'm ordering by:

```
select  empl_last_name
from    empl
order  by salary;
```

Here, then, I would get:

```
EMPL_LAST_NAME
-----
Smith
James
Adams
Martin
Ward
Miller
```

```
Turner  
Michaels  
Raimi  
Blake  
Jones
```

```
EMPL_LAST_NAME  
-----  
Scott  
Ford  
King
```

14 rows selected.

And, just to make sure this is clear: `order by` just affects the order that the rows selected by the rest of the `select` are displayed; if you only select a few rows, then only those rows are in the ordered result:

```
select  salary, empl_last_name  
from    empl  
where   job_title = 'Manager'  
order  by empl_last_name;
```

This query has the results:

```
      SALARY  EMPL_LAST_NAME  
-----  -  
      2850  Blake  
      2975  Jones  
      2450  Raimi
```

Multiple attributes in an order by clause

What happens if you give **multiple** attributes (or expressions) in the `order by` clause, separated by commas? Then you are specifying additional ordering information -- you are saying what to sort by in case of TIES in the previous expression(s) given in the `order by`.

Say that I want to select all the rows of `empl`, displaying the rows in order of `job_title`, and if they have the same `job_title`, display the rows within that `job_title` by `mgr`, and if they have the same `job_title` and `mgr`, display the rows within that `job_title` and `mgr` by `hiredate`:

```
select  *  
from    empl  
order  by job_title, mgr, hiredate;
```

...and you can see that this is indeed the case in the resulting rows:

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7902	Ford	Analyst	7566	03-DEC-12	3000		200
7788	Scott	Analyst	7566	09-NOV-18	3000		200
7900	James	Clerk	7698	03-DEC-17	950		300
7934	Miller	Clerk	7782	23-JAN-16	1300		100
7876	Adams	Clerk	7788	23-SEP-18	1100		400
7369	Smith	Clerk	7902	17-DEC-12	800		200
7566	Jones	Manager	7839	02-APR-12	2975		200
7782	Raimi	Manager	7839	09-JUN-12	2450		100
7698	Blake	Manager	7839	01-MAY-13	2850		300
7839	King	President		17-NOV-11	5000		500
7499	Michaels	Sales	7698	20-FEB-18	1600	300	300

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7654	Martin	Sales	7698	28-SEP-18	1250	1400	300
7521	Ward	Sales	7698	22-FEB-19	1250	500	300
7844	Turner	Sales	7698	08-SEP-19	1500	0	300

14 rows selected.

You could say that we are displaying the rows in primary order by `job_title`, in secondary order by `mgr`, and in 3rd-level order by `hiredate`.

NULL columns and order by

You might wonder: what happens with NULL columns if you order by those columns?

```
select  empl_last_name, job_title, commission
from    empl
order  by commission;
```

Try it, and you'll see that the result is:

EMPL_LAST_NAME	JOB_TITLE	COMMISSION
Turner	Sales	0
Michaels	Sales	300
Ward	Sales	500
Martin	Sales	1400
Ford	Analyst	
Smith	Clerk	
Miller	Clerk	
Jones	Manager	
Raimi	Manager	
Scott	Analyst	
King	President	

```
EMPL_LAST_NAME JOB_TITLE COMMISSION
-----
Adams           Clerk
James           Clerk
Blake           Manager
```

14 rows selected.

Similarly, if you perform the query:

```
select *
from   empl
order by mgr;
```

This has the results:

```
EMPL EMPL_LAST_NAME JOB_TITLE MGR HIREDATE SALARY COMMISSION DEP
-----
7788 Scott Analyst 7566 09-NOV-18 3000 200
7902 Ford Analyst 7566 03-DEC-12 3000 200
7499 Michaels Sales 7698 20-FEB-18 1600 300 300
7900 James Clerk 7698 03-DEC-17 950 300
7844 Turner Sales 7698 08-SEP-19 1500 0 300
7654 Martin Sales 7698 28-SEP-18 1250 1400 300
7521 Ward Sales 7698 22-FEB-19 1250 500 300
7934 Miller Clerk 7782 23-JAN-16 1300 100
7876 Adams Clerk 7788 23-SEP-18 1100 400
7566 Jones Manager 7839 02-APR-12 2975 200
7782 Raimi Manager 7839 09-JUN-12 2450 100

EMPL EMPL_LAST_NAME JOB_TITLE MGR HIREDATE SALARY COMMISSION DEP
-----
7698 Blake Manager 7839 01-MAY-13 2850 300
7369 Smith Clerk 7902 17-DEC-12 800 200
7839 King President 17-NOV-11 5000 500
```

14 rows selected.

In these results, notice that the row for President King is the last displayed, as it is the only row containing a value of NULL for mgr.

order by: DESC option and ASC default

Have you noticed that all of our orderings have been in ascending order so far? That's the default for order by. You can order rows in **descending** order of some expression by writing a blank, and then **DESC**, after the expression you want to order in descending order.

So, if you'd like to select the rows of the empl table, displaying the resulting rows by salary, with

the HIGHEST salary first (in descending order of salary), you just write:

```
select *
from empl
order by salary desc;
```

This query has the results:

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7839	King	President		17-NOV-11	5000		500
7902	Ford	Analyst	7566	03-DEC-12	3000		200
7788	Scott	Analyst	7566	09-NOV-18	3000		200
7566	Jones	Manager	7839	02-APR-12	2975		200
7698	Blake	Manager	7839	01-MAY-13	2850		300
7782	Raimi	Manager	7839	09-JUN-12	2450		100
7499	Michaels	Sales	7698	20-FEB-18	1600	300	300
7844	Turner	Sales	7698	08-SEP-19	1500	0	300
7934	Miller	Clerk	7782	23-JAN-16	1300		100
7521	Ward	Sales	7698	22-FEB-19	1250	500	300
7654	Martin	Sales	7698	28-SEP-18	1250	1400	300

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7876	Adams	Clerk	7788	23-SEP-18	1100		400
7900	James	Clerk	7698	03-DEC-17	950		300
7369	Smith	Clerk	7902	17-DEC-12	800		200

14 rows selected.

Make sure this is clear: you put DESC after EACH attribute that you want in descending order; if you are specifying secondary or additional orderings, you must put DESC after each expression that you want to be displayed in DESC order. For example, if you are selecting the rows of the empl table and you'd like to display the rows:

- * in descending alphabetical order by job_title,
- * but for rows with the same job_title, in ascending order by mgr,
- * but for rows with the same job_title and mgr, in descending order by hiredate,

...you'd put:

```
select *
from empl
order by job_title desc, mgr, hiredate desc;
```

...resulting in:

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7844	Turner	Sales	7698	08-SEP-19	1500	0	300

7521	Ward	Sales	7698	22-FEB-19	1250	500	300
7654	Martin	Sales	7698	28-SEP-18	1250	1400	300
7499	Michaels	Sales	7698	20-FEB-18	1600	300	300
7839	King	President		17-NOV-11	5000		500
7698	Blake	Manager	7839	01-MAY-13	2850		300
7782	Raimi	Manager	7839	09-JUN-12	2450		100
7566	Jones	Manager	7839	02-APR-12	2975		200
7900	James	Clerk	7698	03-DEC-17	950		300
7934	Miller	Clerk	7782	23-JAN-16	1300		100
7876	Adams	Clerk	7788	23-SEP-18	1100		400

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7369	Smith	Clerk	7902	17-DEC-12	800		200
7788	Scott	Analyst	7566	09-NOV-18	3000		200
7902	Ford	Analyst	7566	03-DEC-12	3000		200

14 rows selected.

And, if you'd like to display those those rows in primary order of increasing salary, and in secondary order of decreasing hiredate, you'd put:

```
select *
from empl
order by salary, hiredate desc;
```

...resulting in:

7369	Smith	Clerk	7902	17-DEC-12	800		200
7900	James	Clerk	7698	03-DEC-17	950		300
7876	Adams	Clerk	7788	23-SEP-18	1100		400
7521	Ward	Sales	7698	22-FEB-19	1250	500	300
7654	Martin	Sales	7698	28-SEP-18	1250	1400	300
7934	Miller	Clerk	7782	23-JAN-16	1300		100
7844	Turner	Sales	7698	08-SEP-19	1500	0	300
7499	Michaels	Sales	7698	20-FEB-18	1600	300	300
7782	Raimi	Manager	7839	09-JUN-12	2450		100
7698	Blake	Manager	7839	01-MAY-13	2850		300
7566	Jones	Manager	7839	02-APR-12	2975		200

7788	Scott	Analyst	7566	09-NOV-18	3000		200
7902	Ford	Analyst	7566	03-DEC-12	3000		200
7839	King	President		17-NOV-11	5000		500

14 rows selected.

order by style warning

One final comment with regard to `order by`: do not use it in a nested select! First, it is not good style, and second, it doesn't make sense, anyway, if you really think about it. It is only reasonable at the

END of a top-level (or "outermost") select. This will be a Course SQL Coding Standard, that order by clauses must only be given for top-level/"outermost" selects.

For example, then, it will go AFTER and OUTSIDE a nested select (and thus as part of the top-level select):

```
select *
from   empl
where  salary >
      (select min(salary)
       from   empl
       where  job_title = 'Manager')
order by salary;
```

...resulting in:

EMPL	EMPL_LAST_NAME	JOB_TITLE	MGR	HIREDATE	SALARY	COMMISSION	DEP
7698	Blake	Manager	7839	01-MAY-13	2850		300
7566	Jones	Manager	7839	02-APR-12	2975		200
7788	Scott	Analyst	7566	09-NOV-18	3000		200
7902	Ford	Analyst	7566	03-DEC-12	3000		200
7839	King	President		17-NOV-11	5000		500

group by

group by is a clause that takes more effort to get comfortable with than order by, but allows for some quite nifty queries of your data. group by provides a way to "group" rows sharing common characteristics, usually so you can perform aggregate function computations on rows within those "groups".

The easiest way to get used to group by is by example. You already know how to get the average salary of all employees, or for all employees whose job_title is 'Manager', or for all employees who work in the 'Research' department -- respectively:

```
select avg(salary)
from   empl;
```

...resulting in:

```
AVG (SALARY)
-----
 2073.21429
```

```
select avg(salary)
from   empl
where  job_title = 'Manager';
```

...resulting in:

```
AVG (SALARY)
-----
 2758.33333
```

and:

```
select avg(salary)
from   empl e, dept d
where  e.dept_num = d.dept_num
       and dept_name = 'Research';
```

...resulting in:

```
AVG (SALARY)
-----
 2443.75
```

But each of these queries returns just a single result, just a single row.

What `group by` provides is a way to get computations for different groups of rows from a single query -- if you would like to get the average `salary` for employees who are Managers, AND for employees who are Clerks, AND for employees who are Salesmen, etc., for all `job_titles`, then you `group by job_title`:

```
select  avg(salary)
from    empl
group by job_title;
```

That is, this says, get the rows of `empl`, group those rows by `job_title`, and project the average `salary` for each of those groups. So, the above query results in:

```
AVG (SALARY)
-----
 2758.33333
    3000
 1037.5
    5000
    1400
```

You are also allowed to project a column you are grouping by along with any computations on those groups -- so, if you `group by job_title`, then you can also project `job_title`, if you would like, and result is that you see WHICH `job_title` has each average:

```
select  job_title, avg(salary)
from    empl
group by job_title;
```

...resulting in:

```
JOB_TITLE  AVG (SALARY)
-----
Manager    2758.33333
```

```
Analyst      3000
Clerk        1037.5
President     5000
Sales        1400
```

Where does `group by` "fit" in terms of the `select` statement syntax?

- * You still perform any Cartesian products given in the `FROM` clause first,
- * and then you select those rows from that Cartesian product that satisfy the condition(s) given in the `WHERE` clause.
- * Then, if there is a `group by` clause, you group only the selected rows by the expression given in the `group by` clause,
- * and then you project what is specified in the `select` clause, usually the desired computations for each of those groups, and the expression you are grouping by if desired,
- * ordering the rows as specified by the `order by` clause if it is there!

So, consider this query:

```
select  dept_name, avg(salary)
from    empl e, dept d
where   e.dept_num = d.dept_num
group by dept_name
order by avg(salary);
```

This will:

- * perform a Cartesian product of the `empl` and `dept` tables,
- * then select those rows of the Cartesian product in which `e.dept_num = d.dept_num` (thus performing an equi-join!),
- * then, only in the rows for which `e.dept_num = d.dept_num`, it will group the rows by `dept_name`,
- * then it will project the `dept_name` and the average `salary` for each set of rows grouped by `dept_name`,
- * displaying the resulting rows in order of (increasing) average `salary`.

So, you would see the following:

DEPT_NAME	AVG(SALARY)
Operations	1100
Sales	1566.66667
Accounting	1875
Research	2443.75
Management	5000

What if you would like the minimum and maximum salaries, minimum and maximum `hiredates`, salary totals, and number of employees for each value of `dept_num`? Then this would do the trick (although I'm projecting this information in a different order than stated above, just to make the point that you can project these columns in any order you want, and I'm happening to order the resulting rows by minimum `salary`):

```
select    count(*), dept_num, min(salary), max(salary), min(hiredate),
          max(hiredate), sum(salary)
from      empl
group by  dept_num
order by  min(salary);
```

...resulting in:

COUNT (*)	DEP	MIN (SALARY)	MAX (SALARY)	MIN (HIRED	MAX (HIRED	SUM (SALARY)
4	200	800	3000	02-APR-12	09-NOV-18	9775
6	300	950	2850	01-MAY-13	08-SEP-19	9400
1	400	1100	1100	23-SEP-18	23-SEP-18	1100
2	100	1300	2450	09-JUN-12	23-JAN-16	3750
1	500	5000	5000	17-NOV-11	17-NOV-11	5000

I cannot stress the following two points enough:

- * If you want to project MORE than one row in a query involving a projected aggregate function call, then you MUST use `group by`; otherwise, you can ONLY get one row in the result.
- * When you DO use `group by`, you get ONE row for EACH value of the attribute(s) or expression(s) you are grouping by. You can only project, then, either computations on the attributes of the rows within each group, or the attribute(s) or expression(s) you are grouping by.

Oracle is a stickler on the second part of that second point -- when using `group by`, you really cannot project anything except the expression(s) you are grouping by or aggregate function calls for those groups. (Think about it -- since `group by` essentially gives you "one row" per group, what would it mean to try to project another attribute? To project `empl_last_name` when grouping by `job_title`?) So, it is an error to try to do so, even if you know that the attribute's value happens to be the same for all rows in a group. For example, this query will FAIL:

```
select    dept_num, empl_last_name, min(salary), max(salary),
          min(hiredate), max(hiredate)
from      empl
group by  dept_num;
```

...giving the error message:

```
ERROR at line 1:
ORA-00979: not a group by expression
```

When you see this error message, chances are good you are using `group by` and trying to project something that is not an aggregate function call and not what you are grouping by.

Likewise, this fails with the same error message; even though I know that `dept_name` is the same for all rows with a given `dept_num`, Oracle doesn't know that:

```
select    d.dept_num, dept_name, min(salary), max(salary),
          min(hiredate), max(hiredate)
```

```
from      empl e, dept d
where     e.dept_num = d.dept_num
group by d.dept_num;
```

...giving the error message:

```
ERROR at line 1:
ORA-00979: not a group by expression
```

Now, you CAN use multiple expressions after group by, separated by commas -- when you do that, you will get a group for each distinct collection of values of those expressions. So, you could project both dept_num and dept_name if you were to group by both dept_num and dept_name:

```
select    d.dept_num, dept_name, min(salary), max(salary),
          min(hiredate), max(hiredate)
from      empl e, dept d
where     e.dept_num = d.dept_num
group by d.dept_num, dept_name;
```

...resulting in:

DEP	DEPT_NAME	MIN(SALARY)	MAX(SALARY)	MIN(HIRED)	MAX(HIRED)
200	Research	800	3000	02-APR-12	09-NOV-18
500	Management	5000	5000	17-NOV-11	17-NOV-11
100	Accounting	1300	2450	09-JUN-12	23-JAN-16
400	Operations	1100	1100	23-SEP-18	23-SEP-18
300	Sales	950	2850	01-MAY-13	08-SEP-19

But remember -- each distinct combination of values of those expressions is considered a separate group:

```
select    job_title, mgr, avg(salary), count(*)
from      empl
group by job_title, mgr;
```

...resulting in:

JOB_TITLE	MGR	AVG(SALARY)	COUNT(*)
Analyst	7566	3000	2
Clerk	7782	1300	1
Manager	7839	2758.33333	3
Sales	7698	1400	4
President		5000	1
Clerk	7902	800	1
Clerk	7788	1100	1
Clerk	7698	950	1

8 rows selected.

Each distinct (job_title, mgr) pair is a SEPARATE group, as you can see in the above results.

Finally, it is Course SQL Style Standard that you should only use `group by` for a reason (usually, because you want some computation for the rows in each group). If you aren't performing some computation on the rows in each group, do not use `group by`. In particular, don't use it just to suppress duplicate rows -- that is what `DISTINCT` is for!

-- POOR style: (you will lose points for this!)

```
select  dept_name, job_title
from    empl e, dept d
where   e.dept_num = d.dept_num
group by dept_name, job_title;
```

-- BETTER style:

```
select  distinct dept_name, job_title
from    empl e, dept d
where   e.dept_num = d.dept_num;
```

...which has the results:

DEPT_NAME	JOB_TITLE
Accounting	Clerk
Management	President
Sales	Manager
Accounting	Manager
Research	Clerk
Sales	Sales
Research	Manager
Operations	Clerk
Sales	Clerk
Research	Analyst

10 rows selected.

-- ALSO good:

```
select  dept_name, job_title, count(*), avg(salary)
from    empl e, dept d
where   e.dept_num = d.dept_num
group by dept_name, job_title;
```

...which has the results:

DEPT_NAME	JOB_TITLE	COUNT (*)	AVG (SALARY)
Accounting	Clerk	1	1300
Management	President	1	5000
Sales	Manager	1	2850
Accounting	Manager	1	2450
Research	Clerk	1	800
Sales	Sales	4	1400

Research	Manager	1	2975
Operations	Clerk	1	1100
Sales	Clerk	1	950
Research	Analyst	2	3000

10 rows selected.

group by can be part of any select, including a nested select, although you should be careful to use a proper operator in the condition including the nested select in this case. In particular, note that, IF you are using group by, you can have an aggregate function call whose expression is another aggregate function -- you'd like the minimum of all of the averages, or the count of all of the maximums, or the sum of all of the minimums, etc.

For example, to see which employees make more than or equal to the average salary for any one department (even if not their own), you could write:

```
select empl_last_name, salary
from   empl
where  salary >=
      (select min(avg(salary))
       from   empl
       group by dept_num);
```

...which has the results:

EMPL_LAST_NAME	SALARY
King	5000
Jones	2975
Blake	2850
Raimi	2450
Ford	3000
Michaels	1600
Ward	1250
Martin	1250
Scott	3000
Turner	1500
Adams	1100

EMPL_LAST_NAME	SALARY
Miller	1300

12 rows selected.

That subquery would work on its own, too -- if you just want to know the minimum average salary for the employees with the same value of dept_num, this would do it:

```
select min(avg(salary))
from   empl
group by dept_num;
```

...resulting in:

```
MIN (AVG (SALARY))  
-----  
1100
```

Do not confuse order by and group by!

If you want your resulting rows to be displayed in a certain order, you **STILL** need `order by` -- it is quite common for a query to have both `group by` and `order by` clauses. Using `group by` does not guarantee that the resulting rows will be projected in a certain order; if you want a particular order for the resulting rows of any query, `order by` is needed.

So, if I would like various statistics for each department with the resulting rows ordered by minimum salary, this should be used:

```
select dept_num, min(salary), max(salary), min(hiredate),  
       max(hiredate), sum(salary)  
from   empl  
group by dept_num  
order by min(salary);
```

...resulting in:

DEP	MIN (SALARY)	MAX (SALARY)	MIN (HIRED)	MAX (HIRED)	SUM (SALARY)
200	800	3000	02-APR-12	09-NOV-18	9775
300	950	2850	01-MAY-13	08-SEP-19	9400
400	1100	1100	23-SEP-18	23-SEP-18	1100
100	1300	2450	09-JUN-12	23-JAN-16	3750
500	5000	5000	17-NOV-11	17-NOV-11	5000

Finally, it is important to remember that the selection of rows specified by a `WHERE` clause is done **BEFORE** the grouping of the resulting rows specified by a `group by` clause. For example, what if you'd like the average salary by department, but only for employees hired after January 15, 2016? Then this will do that, because the selection based on `hiredate` will be done **BEFORE** the grouping based on `dept_num`:

```
select dept_num, avg(salary), count(*)  
from   empl  
where  hiredate > '15-Jan-2016'  
group by dept_num  
order by count(*);
```

...resulting in:

DEP	AVG (SALARY)	COUNT (*)
100	1300	1
400	1100	1
200	3000	1
300	1310	5

That query's results will be different from the average salary by department overall:

```
select dept_num, avg(salary), count(*)
from empl
group by dept_num
order by count(*);
```

...which has the results:

DEP	AVG (SALARY)	COUNT (*)
400	1100	1
500	5000	1
100	1875	2
200	2443.75	4
300	1566.66667	6

having

Our final new `select` clause for this lab has a direct relationship to the `group by` clause. We've discussed how a `select` statement's `WHERE` clause lets you specify which rows you want to select. What if, however, you are using `group by`, but you don't really want to see the results for all of the groups? What if you only want to see the results for some of the resulting groups?

That's what the `having` clause lets you do. `having` is to groups what `WHERE` is to rows -- it simply gives you a way limit which groups you see in your result.

For example, what if I want `dept_nums` and average salaries for employees in each `dept_num`, but I'm only interested in `dept_nums` with an average salary greater than 1500. You must use `having` to get this:

```
select dept_num, avg(salary)
from empl
group by dept_num
having avg(salary) > 1500;
```

...which has the results:

DEP	AVG (SALARY)
100	1875
200	2443.75
300	1566.66667
500	5000

And, of course, if you'd like to see those results in order of descending average salary, you could use:

```
select dept_num, avg(salary)
from empl
group by dept_num
having avg(salary) > 1500
order by avg(salary) desc;
```

...which has the results:

DEP	AVG (SALARY)
500	5000
200	2443.75
100	1875
300	1566.66667

You can limit the groups in your result based on a variety of criteria, BUT those criteria have to be "related" to the group, based on the grouped-by attributes, or on expressions using those attributes, or to computations on the group.

So, to see the department number and average salary of those with that department number, but only for dept_nums whose latest employee hiredate is after January 1, 2017, you could use:

```
select dept_num, avg(salary)
from empl
group by dept_num
having max(hiredate) > '01-Jan-2017'
order by avg(salary) desc;
```

...which has the results:

DEP	AVG (SALARY)
200	2443.75
300	1566.66667
400	1100

And, to see the department number and average salary of those with that department number, but only for dept_nums only for dept_nums 300, 400, and 500, you could use (noting that dept_num is actually of type char (3)):

```
select dept_num, avg(salary)
from empl
group by dept_num
having dept_num in ('300', '400', '500')
order by avg(salary) desc;
```

...which has the results:

DEP	AVG (SALARY)
500	5000
300	1566.66667
400	1100

What if I am interested in the average salaries within each department of employees hired before January 1, 2013, and only for departments with average salary greater than 1500, displaying the resulting rows in order of decreasing average salary? You could use:

```
select dept_num, avg(salary)
from empl
where hiredate < '01-Jan-2013'
group by dept_num
having avg(salary) > 1500
order by avg(salary) desc;
```

...which has the results:

DEP	AVG (SALARY)
500	5000
100	2450
200	2258.33333

If we'd like the above, except projecting the department name instead of the department number, we could use:

```
select dept_name, avg(salary)
from empl e, dept d
where e.dept_num = d.dept_num
and hiredate < '01-Jan-2013'
group by dept_name
having avg(salary) > 1500
order by avg(salary) desc;
```

...which has the results:

DEPT_NAME	AVG (SALARY)
Management	5000
Accounting	2450
Research	2258.33333

And, a having clause can be as interesting as we'd like...consider:

```
select dept_name, avg(salary)
from empl e, dept d
where e.dept_num = d.dept_num
group by dept_name
having avg(salary) > 1500
and min(salary) < 4000
order by avg(salary) desc;
```

...which has the results:

DEPT_NAME	AVG(SALARY)
Research	2443.75
Accounting	1875
Sales	1566.66667

distinct with Aggregate Functions

One final little SQL tidbit: you know that `DISTINCT` can be used in a `select` to get a "pure" relational projection, to get a result with no duplicate rows. It turns out that you can use `DISTINCT` *within* an aggregate function call, inside its parentheses, to get that function's results just for each distinct value of that attribute.

For example, this simply counts how many rows of `empl` have non-NULL values for the attribute `job_title`:

```
select count(job_title)
from   empl;
```

...which happens to be

```
COUNT(JOB_TITLE)
-----
14
```

If I instead put:

```
select count(distinct job_title)
from   empl;
```

...this will instead count how many distinct, or different, `job_titles` there are, and since there are 5 different `job_title` values amongst the 14 rows of `empl`, this query returns:

```
COUNT(DISTINCTJOB_TITLE)
-----
5
```

And this gives you a slightly prettier result:

```
select count(distinct job_title) "How Many Job-titles"
from   empl;
```

...resulting in:

```
How Many Job-titles
-----
5
```