Group functions, also called multiple- row functions, return one result per group of rows processed. Multiple- row functions covered in this chapter include SUM, AVG, COUNT, MIN, and MAX. This chapter also explains using the GROUP BY clause to identify groups of records to process and the HAVING clause to restrict groups returned in the query results.

When using the GROUP BY clause, remember the following:

- If a group function is used in the SELECT clause, any single (non-aggregate) columns listed in the SELECT clause must also be listed in the GROUP BY clause.
- Columns used to group data in the GROUP BY clause don't have to be listed in the SELECT clause. They're included in the SELECT clause only to have these groups identified in the output.□ Column aliases can't be used in the GROUP BY clause.
- Results returned from a SELECT statement that includes a GROUP BY clause are displayed in ascending order of the columns listed in the GROUP BY clause.

To have a different sort sequence, use the ORDER BY clause.When a SELECT statement includes all three clauses, the order in which they're evaluated is as follows:

- The WHERE clause □ The GROUP BY clause
- The HAVING clause In essence, the WHERE clause filters the data before grouping, and the HAVING clause filters the groups after the grouping occurs.

8.1 Group by examples

```
DELETE FROM patient;

INSERT INTO patient values (111,'john','Wei','m','11-FEB-1978',25000, 'Davis','CA');

INSERT INTO patient values (114,'billy','Bob','f','05-MAY-1985',60000,'Davis','NV');

INSERT INTO patient values (115,'dove','Grime','f','04-JUN-1960',20000,'Sacramento','CA');

INSERT INTO patient values (199,'john','Dali','m','11-FEB-1978',25000, 'Davis','CA');

INSERT INTO patient values (112,'john','Smith','m','01-MAR-1981',40000, 'Davis','CA');

INSERT INTO patient values (978,'john','Doe','m','11-FEB-1978',25000,NULL,'CA');

INSERT INTO patient values (113,'jill','Crane','m','12-APR-1999',50000,'Reno','NV');
```

111 john Wei m 11-FEB-1978 25000 Davis CA 114 billy Bob f 05-MAY-1985 60000 Davis NV 115 dove Grime f 04-JUN-1960 20000 Sacramento CA 199 john Dali m 11-FEB-1978 25000 Davis CA 112 john Smith m 01-MAR-1981 40000 Davis CA 978 john Doe m 11-FEB-1978 25000 NULL CA 113 jill Crane m 12-APR-1999 50000 Reno NV	**RESULT** 245000
`SELECT SUM(salary) FROM patient;`	
Group by city 111 john Wei m 11-FEB-1978 25000 Davis CA 114 billy Bob f 05-MAY-1985 60000 Davis NV 199 john Dali m 11-FEB-1978 25000 Davis CA 112 john Smith m 01-MAR-1981 40000 Davis CA 115 dove Grime f 04-JUN-1960 20000 Sacramento CA 978 john Doe m 11-FEB-1978 25000 NULL CA 113 jill Crane m 12-APR-1999 50000 Reno NV	**RESULT** Davis 150000 Sacramento 20000 NULL 25000 Reno 50000
`SELECT city, SUM(salary) FROM patient GROUP BY city;`	
Group by state 111 john Wei m 11-FEB-1978 25000 Davis CA 199 john Dali m 11-FEB-1978 25000 Davis CA 112 john Smith m 01-MAR-1981 40000 Davis CA 978 john Doe m 11-FEB-1978 25000 NULL CA	**RESULT** CA 135000

115 dove Grime f 04-JUN-1960 20000 Sacramento CA 114 billy Bob f 05-MAY-1985 60000 Davis NV 113 jill Crane m 12-APR-1999 50000 Reno NV	NV 110000
`SELECT state, SUM(salary) FROM patient GROUP BY state;`	
Group by gender 111 john Wei m 11-FEB-1978 25000 Davis CA 199 john Dali m 11-FEB-1978 25000 Davis CA 112 john Smith m 01-MAR-1981 40000 Davis CA 978 john Doe m 11-FEB-1978 25000 NULL CA 113 jill Crane m 12-APR-1999 50000 Reno NV 114 billy Bob f 05-MAY-1985 60000 Davis NV 115 dove Grime f 04-JUN-1960 20000 Sacramento CA	**RESULT** m 165000 f 80000
`SELECT gender, SUM(salary) FROM patient GROUP BY gender;`	
Group by city and gender 111 john Wei m 11-FEB-1978 25000 Davis CA 199 john Dali m 11-FEB-1978 25000 Davis CA 112 john Smith m 01-MAR-1981 40000 Davis CA 114 billy Bob f 05-MAY-1985 60000 Davis NV 115 dove Grime f 04-JUN-1960 20000 Sacramento CA 978 john Doe m 11-FEB-1978 25000 NULL CA	**RESULT** Davis m 90000 Davis f 60000 Sacramento f 20000 NULL m 25000 Reno m 50000

113 jill Crane m 12-APR-1999 50000 Reno NV	

```
SELECT city, gender, SUM(salary) FROM patient GROUP BY city, gender;
```

Group by state and gender	**RESULT**
111 john Wei m 11-FEB-1978 25000 Davis CA	
199 john Dali m 11-FEB-1978 25000 Davis CA	
112 john Smith m 01-MAR-1981 40000 Davis CA	
978 john Doe m 11-FEB-1978 25000 NULL CA	CA m 115000
	CA f 20000
115 dove Grime f 04-JUN-1960 20000 Sacramento CA	NV f 60000
	NV m 50000
114 billy Bob f 05-MAY-1985 60000 Davis NV	
113 jill Crane m 12-APR-1999 50000 Reno NV	

```
SELECT state, gender, SUM(salary) FROM patient GROUP BY state, gender;
```

Group by fname and city	**RESULT**
111 john Wei m 11-FEB-1978 25000 Davis CA	
199 john Dali m 11-FEB-1978 25000 Davis CA	
112 john Smith m 01-MAR-1981 40000 Davis CA	
	John Davis 90000
978 john Doe m 11-FEB-1978 25000 NULL CA	John NULL 25000
	Jill Reno 60000
113 jill Crane m 12-APR-1999 50000 Reno NV	Billy Davis 60000

114 billy Bob f 05-MAY-1985 60000 Davis NV 115 dove Grime f 04-JUN-1960 20000 Sacramento CA	Dove Sacramento 20000
SELECT fname, city, SUM(salary) FROM patient GROUP BY fname, city;	

8.2 SUM

The SUM function is used to calculate the total amount stored in a numeric field for a group of records. The syntax of the SUM function is SUM(([DISTINCT| ALL] n), where n is a column containing numeric data.

```
SELECT salary from patient;

--The result is a single number that is the summation of all the salaries.
SELECT SUM (salary) FROM patient;

--All is implied as in the above statement.
SELECT SUM (ALL salary) FROM patient;

--The result is a single number that is the summation of all the distinct salaries, which
--means it suppresses the duplicates before doing a summation.
SELECT SUM (DISTINCT salary) FROM patient;
```

```
SQL> SELECT salary from patient;

    SALARY
----------
     25000
     60000
     20000
     40000
     25000


6 rows selected.

SQL> --The result is a single number that is the summation of all the salaries
SQL> SELECT SUM (salary) FROM patient;

SUM(SALARY)
-----------
     170000

SQL> --All is implied as in the above statement
SQL> SELECT SUM (ALL salary) FROM patient;

SUM(ALLSALARY)
--------------
        170000

SQL> --The result is a single number that is the summation of all the distinct s
alaries which means
SQL> --it suppresses the duplicates before doing a summation
SQL> SELECT SUM (DISTINCT salary) FROM patient;

SUM(DISTINCTSALARY)
-------------------
             145000
```

```
--Error: how do we align the single summation number with all the cities?
SELECT city, SUM(salary) FROM patient;
```

```
SQL> --This is problematic because how do we align the single summation number w
ith all the cities
SQL> SELECT city, SUM(salary) FROM patient;
SELECT city, SUM(salary) FROM patient
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

```
--Creates a grouping for each city,  which means that it suppresses the duplicates in each group and
--and then comes up with a summation for each group order by is the last clause.
SELECT city, SUM(salary) FROM patient GROUP BY city ORDER BY 1;

--Creates a group for each of the states and gives a summation for each state.
SELECT state, SUM(salary) FROM patient GROUP BY state;

--Creates combination of fname, city catagories and provides a summation for each group.
SELECT fname,city, SUM(salary) FROM patient GROUP BY fname,city;
```

```
SQL> --Creates a grouping for each city  which means
SQL> --that it suppresses the duplicates in each group and
SQL> --and then comes up with a summation for each group
SQL> --order by is the last clause.
SQL> SELECT city, SUM(salary) FROM patient GROUP BY city ORDER BY 1;

CITY                 SUM(SALARY)
-------------------- -----------
Davis                      65000
Las Vegas                  60000
Reno
Sacramento                 20000
                           25000

SQL> --Creates a group for each of the states and gives a summation for each s
te
SQL> SELECT state, SUM(salary) FROM patient GROUP BY state;

STATE                SUM(SALARY)
-------------------- -----------
CA                        110000
NV                         60000

SQL> --Creates combination of fname, city catagories and provides a summation
r each group
SQL> SELECT fname,city, SUM(salary) FROM patient GROUP BY fname,city;

FNAME                CITY                 SUM(SALARY)
-------------------- -------------------- -----------
billy                Las Vegas                  60000
dove                 Sacramento                 20000
john                 Davis                      65000
john                                            25000
jill                 Reno
```

```
--First the where clause filters. Then it does a grouping with the data that is left over. It
--groups the different cities and then for each city group, it comes up with a summation.
SELECT city, SUM(salary) FROM patient  WHERE UPPER(city)<> 'RENO'
GROUP BY city ORDER BY 1;
```

```
SQL> SELECT city, SUM(salary) FROM patient  WHERE UPPER(city) <> 'RENO' GROUP
 city ORDER BY 1;

CITY                 SUM(SALARY)
-------------------- -----------
Davis                      65000
Las Vegas                  60000
Sacramento                 20000
```

```
--First the where clause filters. Then it does a grouping with the data that is left over. It
--groups the different cities and then for each city group, it comes up with a summation.
SELECT city, SUM(salary) FROM patient  WHERE UPPER(city) != 'RENO' or
city is NULL GROUP BY city ORDER BY 1;
```

```
SQL> --First the where clause filters. Then it does a grouping with the data tha
t is left over. It groups
SQL> --the different cities and for each city group it comes up with a summatior

SQL> SELECT city, SUM(salary) FROM patient  WHERE UPPER(city) != 'RENO' or city
is NULL GROUP BY city ORDER BY 1;

CITY                 SUM(SALARY)
-------------------- -----------
Davis                      65000
Las Vegas                  60000
Sacramento                 20000
                           25000
```

8.3 DISTINCT

The optional DISTINCT keyword instructs Oracle to include only unique numeric values in its calculation. The ALL keyword instructs Oracle to include multiple occurrences of numeric values when totaling a field. If the DISTINCT or ALL keywords aren't included when using the SUM function, Oracle assumes the ALL keyword by default and uses all the numeric values in the field when the query is executed.

```
--ALL is implied and is not needed.
SELECT SUM (ALL salary) FROM patient;

--Suppresses the duplicates and gives a summation.
SELECT SUM (DISTINCT salary) FROM patient;

--Suppresses duplicates and gives a summation for each city category.
SELECT city, SUM(DISTINCT salary) FROM patient GROUP BY city ;

--Does not suppress the duplicates for salary and gives a summation for each city category.
SELECT city, SUM(ALL salary) FROM patient GROUP BY city;
```

```
SQL> --ALL is implied and is not needed
SQL> SELECT SUM (ALL salary) FROM patient;

SUM(ALLSALARY)
--------------
        170000

SQL> --Suppresses the duplicates and gives a summation
SQL> SELECT SUM (DISTINCT salary) FROM patient;

SUM(DISTINCTSALARY)
-------------------
             145000

SQL> --suppresses duplicates and gives a summation for each city category
SQL> SELECT city, SUM(DISTINCT salary) FROM patient GROUP BY city ;

CITY                 SUM(DISTINCTSALARY)
-------------------- -------------------
                                   25000
Las Vegas                          60000
Davis                              65000
Sacramento                         20000
Reno

SQL> --Does not suppress the duplicates for salary and gives a summation for ea
h city category
SQL> SELECT city, SUM(ALL salary) FROM patient GROUP BY city;

CITY                 SUM(ALLSALARY)
-------------------- --------------
                              25000
Las Vegas                     60000
Davis                         65000
Sacramento                    20000
Reno
```

```
--Just like the above; however, there is an additional filtering with the having clause. It
-- includes only groups that have a sum greater than 25000.
SELECT city, SUM(ALL salary) FROM patient GROUP BY city HAVING sum(salary)>25000;
```

```
SQL> SELECT city, SUM(ALL salary) FROM patient GROUP BY city HAVING sum(salary)>
25000;

CITY                 SUM(ALLSALARY)
-------------------- --------------
Las Vegas                     60000
Davis                         65000

SQL>
```

8.4 AVG

The AVG function calculates the average of numeric values in a specified column. The syn-tax of the AVG function is AVG([DISTINCT| ALL] n), where n is a column containing numeric data.

```
SELECT salary FROM patient;
```

```
SQL> SELECT salary FROM patient;

    SALARY
----------
     25000
     60000
     20000
     40000
     25000

6 rows selected.
```

```
--Gives a single average for all the salaries.
SELECT AVG (salary) FROM patient;

--Same as above
SELECT AVG (ALL salary) FROM patient;

--Suppresses duplicates and then gives an average.
SELECT AVG (DISTINCT salary) FROM patient;

--Invalid: Does not know how to display the one single average salary with all the cities.
SELECT city, AVG (salary) FROM patient;

--Displays the average salary for each city category.
SELECT city, AVG (salary) FROM patient GROUP BY city ORDER BY 1;
```

```
SQL> --Gives a single average for all the salaries
SQL> SELECT AVG (salary) FROM patient;

AVG(SALARY)
-----------
      34000

SQL> --Same as above
SQL> SELECT AVG (ALL salary) FROM patient;

AVG(ALLSALARY)
--------------
         34000

SQL> --Suppresses duplicates and then gives an average
SQL> SELECT AVG (DISTINCT salary) FROM patient;

AVG(DISTINCTSALARY)
-------------------
              36250

SQL> --Invalid: Does not know how to display the one single average salary with
all the cities
SQL> SELECT city, AVG (salary) FROM patient;
SELECT city, AVG (salary) FROM patient
       *
ERROR at line 1:
ORA-00937: not a single-group group function


SQL> --Displays the average salary for each city category
SQL> SELECT city, AVG (salary) FROM patient GROUP BY city ORDER BY 1;

CITY                 AVG(SALARY)
-------------------- -----------
Davis                      32500
Las Vegas                  60000
Reno
Sacramento                 20000
                           25000
```

```
--First it filters the data based on the where clause. Then it takes the left over records and
--does a grouping for each of the cities and provides an average for each grouping.
SELECT city, AVG (salary) FROM patient  WHERE UPPER(city) <> 'RENO' GROUP BY
city ORDER BY 1;


--First it filters the data based on the where clause. Then it takes the left over records and
--does a grouping for each of the cities and provides an average for each grouping. If there is a
--city group that does not have a salary, which means that it is NULL, then it will replace it with a
--zero. There is additional filtering using the having clause after all the grouping is done.
SELECT city, AVG (nvl(salary,0)) FROM patient  WHERE UPPER(city) <> 'RENO'
GROUP BY city HAVING AVG(salary)>20000 ORDER BY 1;
```

```
SQL> --First it filters the data based on the where clause. Then it takes the le
ft over records and does a
SQL> --grouping for each of the cities and provides an average for each grouping

SQL> SELECT city, AVG (salary) FROM patient  WHERE UPPER(city) <> 'RENO' GROUP B
Y city ORDER BY 1;

CITY                 AVG(SALARY)
-------------------- -----------
Davis                      32500
Las Vegas                  60000
Sacramento                 20000

SQL> --First it filters the data based on the where clause. Then it takes the le
ft over records and does a
SQL> --grouping for each of the cities and provides an average for each grouping
. If there is a city group that
SQL> --does not have a salary which means that it is null, then it will replace
it with a zero
SQL> SELECT city, AVG (nvl(salary,0)) FROM patient  WHERE UPPER(city) <> 'RENO'
GROUP BY city HAVING AVG(salary)>20000 ORDER BY 1;

CITY                 AVG(NVL(SALARY,0))
-------------------- ------------------
Davis                             32500
Las Vegas                         60000

SQL>
```

8.5 COUNT

Depending on the argument used, the COUNT function can count the records having non- NULL values in a specified field or count the total records meeting a specific condition, including those containing NULL values. The syntax of the COUNT function is COUNT(* [DISTINCT| ALL] c), where c represents a numeric or non-numeric column.

```
SELECT fname,lname,city FROM patient;

--Counts the number of rows.
SELECT COUNT (*) FROM patient;

--Counts the number rows based on the contents of the city. If the city for a given row contains
--a NULL, then it will not be counted.
SELECT COUNT (city) FROM patient;

--Same as above
SELECT COUNT (ALL city) FROM patient;

--Invalid: Does not know how to display a single number with the six different cities.
SELECT city, COUNT (*) FROM patient;
```

```
SQL> SELECT fname,lname,city FROM patient;

FNAME                LNAME                CITY
-------------------- -------------------- --------------------
john                 Wei                  Davis
billy                Bob                  Las Vegas
dove                 Grime                Sacramento
john                 Smith                Davis
john                 Doe
jill                 Crane                Reno

6 rows selected.

SQL> --Counts the number of rows
SQL> SELECT COUNT (*) FROM patient;

  COUNT(*)
----------
         6

SQL> --Counts the number rows based on the contents of the city. If the city for
 a given row contains
SQL> --a null then it will not be counted
SQL> SELECT COUNT (city) FROM patient;

COUNT(CITY)
-----------
          5

SQL> --Same as above
SQL> SELECT COUNT (ALL city) FROM patient;

COUNT(ALLCITY)
--------------
             5

SQL> --Invalid: Does not know how to display a single number with the six differ
ent cities
SQL> SELECT city, COUNT (*) FROM patient;
SELECT city, COUNT (*) FROM patient
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

```
--Create a group for each of the different cities and do a count for each category. NULL cities are
--excluded from the count.
SELECT city, COUNT (city) FROM patient GROUP BY city;

--Same as above but the NULLs are not excluded.
SELECT city, COUNT (*) FROM patient GROUP BY city;

--After it has come up with the count per grouping, there is an additional filtering, which includes only those
--records where the count is greater than 1.
SELECT city, COUNT (*) FROM patient GROUP BY city HAVING COUNT(*) >1;
```

```
SQL> --Create a group for each of the different cities and do a count for each c
ategory. Null cities are excluded
SQL> --from the count
SQL> SELECT city, COUNT (city) FROM patient GROUP BY city;

CITY                 COUNT(CITY)
-------------------- -----------
                               0
Las Vegas                      1
Davis                          2
Sacramento                     1
Reno                           1

SQL> --Same as above but the nulls are not excluded
SQL> SELECT city, COUNT (*) FROM patient GROUP BY city;

CITY                   COUNT(*)
-------------------- ----------
                              1
Las Vegas                     1
Davis                         2
Sacramento                    1
Reno                          1

SQL> --After it has come up with the count per grouping, there is an additional
filtering which only includes
SQL> --those where the count is greater than 1.
SQL> SELECT city, COUNT (*) FROM patient GROUP BY city HAVING COUNT(*) > 1;

CITY                   COUNT(*)
-------------------- ----------
Davis                         2
```

8.6 MAX

The MAX function returns the largest value stored in the specified column. The syntax of the MAX function is MAX([DISTINCT| ALL] c), where c can represent any numeric, character, or date column.

```
SELECT salary FROM patient;

--The highest salary is displayed.
SELECT MAX (salary) FROM patient;
--same as above
SELECT MAX (ALL salary) FROM patient;
```

```
SQL> SELECT salary FROM patient;

    SALARY
----------
     25000
     60000
     20000
     40000
     25000

6 rows selected.

SQL> --The highest salary
SQL> SELECT MAX (salary) FROM patient;

MAX(SALARY)
-----------
      60000

SQL> --same as above
SQL> SELECT MAX (ALL salary) FROM patient;

MAX(ALLSALARY)
--------------
         60000
```

```
--Invalid: the highest salary is a single number and cannot be associated with all cities.
SELECT city, MAX (salary) FROM patient;

--Displays highest salary for each city.
SELECT city, MAX (salary)  FROM patient GROUP BY city;
```

```
SQL> --Invalid: the highest salary is a single number and cannot be associated w
ith all cities
SQL> SELECT city, MAX (salary) FROM patient;
SELECT city, MAX (salary) FROM patient
       *
ERROR at line 1:
ORA-00937: not a single-group group function


SQL> -- highest salary for each city
SQL> SELECT city, MAX (salary)  FROM patient GROUP BY city;

CITY                 MAX(SALARY)
-------------------- -----------
                           25000
Las Vegas                  60000
Davis                      40000
Sacramento                 20000
Reno
```

```
--Given the fname, city combination, display the the number of records and the highest salary
--for each of those combination categories.
SELECT fname, city, COUNT(*), AVG (salary), MAX(salary) FROM patient GROUP BY
fname, city;

--Same as above except that after the final result, do some additional filtering based on the count.
SELECT fname, city, COUNT(*), AVG (salary), MAX(salary) FROM patient GROUP BY
fname, city HAVING COUNT(*) > 2;
```

```
SQL> --Given the fname, city combination, display the the number of records, and
 the highest salary
SQL> --for each of those combination categories
SQL> SELECT fname, city, COUNT(*), AVG (salary), MAX(salary) FROM patient GROUP
BY fname, city;

FNAME                CITY                  COUNT(*) AVG(SALARY) MAX(SALARY)
-------------------- -------------------- ---------- ----------- -----------
billy                Las Vegas                     1       60000       60000
dove                 Sacramento                    1       20000       20000
john                 Davis                         2       32500       40000
john                                               1       25000       25000
jill                 Reno                          1

SQL> --Same as above except that after the final result do some additional filte
ring based on the count
SQL> SELECT fname, city, COUNT(*), AVG (salary), MAX(salary) FROM patient GROUP
BY fname, city HAVING COUNT(*) > 2;

no rows selected
```

8.7 MIN

In contrast to the MAX function, the MIN function returns the smallest value in a specified column. As with the MAX function, the MIN function works with any numeric, character, or date column. The syntax of the MIN function is MIN([DISTINCT| ALL] c), where c represents any character, numeric, or date column. The MIN function uses the same logic as the MAX function for numeric and character data, except it returns the smallest value rather than the largest value.

```
SELECT salary FROM patient;

--The lowest salary is displayed.
SELECT MIN (salary) FROM patient;

--Invalid: cannot display a single number with six cities.
SELECT city, MIN(salary) FROM patient;

--Display the lowest salary for each city category and display the number of records in each group.
SELECT city, MIN (salary), COUNT(*)  FROM patient GROUP BY city;
```

```
SQL> SELECT salary FROM patient;

    SALARY
----------
     25000
     60000
     20000
     40000
     25000


6 rows selected.

SQL> --The lowest salary
SQL> SELECT MIN (salary) FROM patient;

MIN(SALARY)
-----------
      20000

SQL> --Invalid: cannot display a single number with six cities
SQL> SELECT city, MIN(salary) FROM patient;
SELECT city, MIN(salary) FROM patient
       *
ERROR at line 1:
ORA-00937: not a single-group group function


SQL> --the lowest salary for each city category and display the number of record
s in each group
SQL> SELECT city, MIN (salary), COUNT(*)  FROM patient GROUP BY city;

CITY                 MIN(SALARY)   COUNT(*)
-------------------- ----------- ----------
                           25000          1
Las Vegas                  60000          1
Davis                      25000          2
Sacramento                 20000          1
Reno                                      1
```

```
--Filters with the where clause. Then given the remaining records, it groups by city and finds the
--lowest salary for each city category. Given the result set, it only includes the ones where
--there are more than two records for each group. The results are sorted by city.
SELECT city, MIN (salary)  FROM patient WHERE city IS NOT NULL GROUP BY city HAVING count(*)
>1 ORDER BY 1 DESC;
```

```
SQL> --Filters with the where clause, then given the remaining records, groups b
y city and finds the lowest
SQL> --salary for each city category. Given the result set, it only includes the
 ones where there are more than
SQL> --Two records for each group. Order by city
SQL> SELECT city, MIN (salary)  FROM patient WHERE city IS NOT NULL GROUP BY cit
y HAVING count(*) > 1 ORDER BY 1 DESC;

CITY                 MIN(SALARY)
-------------------- -----------
Davis                      25000
```

8.8 Dates and group functions

```
--Displays oldest person, youngest person, the number of records (excludes all those that have a NULL in
--DOB),  and number of records (Suppresses duplicate DOB).
SELECT min(DOB), max (DOB), count(DOB), count (DISTINCT DOB) FROM patient;

--Invalid: Cannot apply AVG to date formats. Use months_between to convert it into a number
--and then do an average.
SELECT AVG(DOB) FROM patient;

--Invalid: can do a sum on date formats.
SELECT SUM(DOB) FROM patient;
```

```
SQL> --Oldest person, youngest person, the number of records (excludes all those
 that have a null in dob),
SQL> --number of records (Suppresses duplicate dob)
SQL> SELECT min(dob), max (dob), count(dob), count (DISTINCT dob) FROM patient;

MIN(DOB)  MAX(DOB)  COUNT(DOB) COUNT(DISTINCTDOB)
--------- --------- ---------- ------------------
04-JUN-60 12-APR-99          6                  5

SQL> --Invalid: Cannot apply avg to date formats. Use months_between to convert
it into a number and then
SQL> --do an average
SQL> SELECT avg(dob) FROM patient;
SELECT avg(dob) FROM patient
           *
ERROR at line 1:
ORA-00932: inconsistent datatypes: expected NUMBER got DATE


SQL> --Invalid: can do a sum on date formats
SQL> SELECT SUM(dob) FROM patient;
SELECT SUM(dob) FROM patient
           *
ERROR at line 1:
ORA-00932: inconsistent datatypes: expected NUMBER got DATE
```

✓ *CHECK 8A*

1. Display the count of all people who make less than 10000 for each of the different personality types.
2. Display the average age and maximum salary for each personality type. Display both the average age and personality types.
3. What is wrong with the following:
    - SELECT * FROM patient WHERE salary> AVG(salary);

    - SELECT fname AS firstname, SUM (salary) summation FROM patient
      WHERE firstname='john'
      HAVING summation>10000