Objectives

After completing this lesson, you should be able to do the following:

- Identify the available group functions
- Describe the use of group functions
- Group data using the GROUP BY clause
- Include or exclude grouped rows by using the HAVING clause
What Are Group Functions?

Group functions operate on sets of rows to give one result per group.

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE
Using Group Functions

SELECT [column, ] group_function(column) 
FROM table 
[WHERE condition] 
[GROUP BY column] 
[ORDER BY column] ;

Guidelines for Using Group Functions

• DISTINCT makes the function consider only nonduplicate values; ALL makes it consider even-value including duplicates. The default is ALL and therefore does not need to be specified.

• The datatypes for the arguments may be CHAR, VARCHAR2 NUMBER, or DATE where exprs listed.

All group functions except COUNT(*) ignore null values. To substitute a value for null values, use the NVL function.

The Oracle Server implicitly sorts the result set in ascending order when using a GROUP BY clause. To override this default ordering, DESC can be used in an ORDER BY clause.
Using AVG and SUM Functions

You can use **AVG** and **SUM** for numeric data.

```
SELECT AVG(sal) , SUM(sal)
FROM emp;
```

<table>
<thead>
<tr>
<th>AVG(SAL)</th>
<th>SUM(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2073,21429</td>
<td>29025</td>
</tr>
</tbody>
</table>

**Group Functions**

You can use **AVG**, **SUM**, **MIN**, and **MAX** functions against columns that can store numeric data. The example on the slide displays the average, highest, lowest, and sum of monthly salaries for all salespeople.
Using MIN and MAX Functions

You can use MIN and MAX for any datatype.

Using MIN, MAX for numeric data types:

```
SELECT min(sal), max(sal),
FROM emp;
```

<table>
<thead>
<tr>
<th>MIN(SAL)</th>
<th>MAX(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>5000</td>
</tr>
</tbody>
</table>
Using MIN and MAX Functions

You can use MIN and MAX for date datatype.

Group Functions (continued)
You can use MAX and MIN functions for any datatype. The slide example displays the most junior and most senior employee.

```sql
SELECT MIN(hiredate) , MAX(hiredate)
FROM emp ;
```

<table>
<thead>
<tr>
<th>MIN(HIREDA)</th>
<th>MAX(HIREDA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>17/12/1980</td>
<td>12/01/1983</td>
</tr>
</tbody>
</table>
Using MIN and MAX Functions

You can use MIN and MAX for char datatype.

Group Functions (continued)
The following example displays the employee name that is first and the employee name that is the last in an alphabetized list of all employees.

SELECT MIN(ename), MAX(ename)  
FROM emp ;

<table>
<thead>
<tr>
<th>MIN(ENAME)</th>
<th>MAX(ENAME)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAMS</td>
<td>WARD</td>
</tr>
</tbody>
</table>
Using the COUNT Function

COUNT(*) returns the number of rows in a table.

```
SELECT  COUNT(*)
FROM emp;
```

<table>
<thead>
<tr>
<th>COUNT(*)</th>
<th>14</th>
</tr>
</thead>
</table>

The COUNT Function

The COUNT function has two formats:

- COUNT(*)
- COUNT(expr)

COUNT(*) returns the number of rows in a table, including duplicate rows and rows containing null values in any of the columns.

If a WHERE clause is included in the SELECT statement COUNT(*) returns the number of rows that satisfies the condition in the WHERE clause.

In contrast, COUNT(expr) returns the number of nonnull rows in the column identified by expr.
**COUNT(expr)** returns the number of non NULL rows in a table, i.e. number of rows that satisfies **expr**.

```
SELECT COUNT(deptno)
FROM emp;
```

<table>
<thead>
<tr>
<th>COUNT(DEPTNO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
</tr>
</tbody>
</table>

The COUNT Function (continued)
The slide example displays the number of employees in department 30.

```
SELECT COUNT(*)
FROM emp
WHERE deptno = 30 ;
```

<table>
<thead>
<tr>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
Using the COUNT Function

\[ \text{COUNT}(\text{expr}) \text{ returns the number of non NULL rows satisfying } \text{expr} \text{ in a table.} \]

```
SELECT COUNT(comm)
FROM emp;
```

<table>
<thead>
<tr>
<th>COUNT(COMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

The COUNT Function (continued)

COUNT(expr) returns the number of nonnull rows in the column identified by expr.
The slide example displays the number of employees in department 30 who can earn a commission. Notice that the result gives the total number of rows to be four because two employees in department 30 cannot earn a commission and contain a null value in the COMM column.

```
SELECT COUNT(comm)
FROM emp
WHERE deptno = 30;
```

<table>
<thead>
<tr>
<th>COUNT(COMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

```
SELECT COUNT(comm)
FROM emp
WHERE deptno = 20;
```

<table>
<thead>
<tr>
<th>COUNT(COMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
SELECT COUNT(*)
FROM emp
WHERE comm IS NULL;

<table>
<thead>
<tr>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
</tbody>
</table>
Using the COUNT Function

\texttt{COUNT( expr)} returns the number of rows in a table that satisfies the \texttt{expr}.

\begin{verbatim}
SELECT COUNT(DISTINCT deptno)
FROM emp;
\end{verbatim}

<table>
<thead>
<tr>
<th>COUNT(DISTINCT deptno)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Group Functions and Null Values

Group functions ignore null values in the column.

```
SELECT AVG(comm)
FROM emp;
```

```
<table>
<thead>
<tr>
<th>AVG(comm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>550</td>
</tr>
</tbody>
</table>
```

Group Functions and Null Values
All group functions except COUNT (*) ignore null values in the column. In the slide example, the average is calculated based only on the rows in the table where a valid value is stored in the COMM column. The average is calculated as total commission being paid to all employees divided by the number of employees receiving commission (4).
Using the NVL Function with Group Functions

The NVL function forces group functions to include null values.

```
SELECT AVG(NVL (comm,0))
FROM emp;
```

| AVG(NVL(comm,0)) | 157,142857 |

Group Functions and Null Values (continued)
The NVL function forces group functions to include null values. In the slide example, the average is calculated based on all rows in the table regardless of whether null values are stored in the COMM column. The average is calculated as total commission being paid to all employees divided by the total number of employees in the company (14).
Creating Groups of Data

```
SELECT deptno, AVG(sal) 
FROM emp 
GROUP BY deptno ;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>AVG(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1566.66667</td>
</tr>
<tr>
<td>20</td>
<td>2175</td>
</tr>
<tr>
<td>10</td>
<td>2916.66667</td>
</tr>
</tbody>
</table>

Groups of Data
Until now, all group functions have treated the table as one large group of information. At times, you need to divide the table of information into smaller groups. This can be done by using the GROUP BY clause.
Creating Groups of Data: GROUP BY Clause

SELECT column, group_function (column)
FROM table
[WHERE condition]
[GROUP BY group by]
[ORDER BY column];

Divide rows in a table into smaller groups by using the GROUP BY clause.

The GROUP BY Clause

You can use the GROUP BY clause to divide the rows in a table into groups.

You can then use the group functions to return summon' information for each group.

In the syntax:

*group by expression* specifies columns whose values determine the basis for grouping rows

Guidelines

If you include a group function in a SELECT clause, you cannot select individual results as well unless the individual column appears in the GROUP BY clause. You will receive an error-message if you fail to include the column list.

Using a WHERE clause, you can preexclude rows before dividing them into groups.

You must include the *columns* in the GROUP BY clause.

You cannot use the column alias in the GROUP BY clause.
Using the GROUP BY Clause

All columns in the SELECT list that are not in group functions must be in the GROUP BY clause

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>AVG(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1566,66667</td>
</tr>
<tr>
<td>20</td>
<td>2175</td>
</tr>
<tr>
<td>10</td>
<td>2916,66667</td>
</tr>
</tbody>
</table>

The GROUP BY Clause (continued)

When using the GROUP BY clause, make sure that all columns in the SELECT list that are not in the group functions are included in the GROUP BY clause. The example on the slide displays the department number and the average salary for each department. Here is how this SELECT statement, containing a GROUP BY clause, is evaluated:

The SELECT clause specifies the columns to be retrieved:
- Department number column in the EMP table
- The average of all the salaries in the group you specified in the GROUP BY clause

The FROM clause specifies the tables that the database must access: the EMP table.

The WHERE clause specifies the rows to be retrieved. Since there is no WHERE clause, by default all rows are retrieved.
Using the GROUP BY Clause

The GROUP BY column does not have to be in the SELECT list.

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
ORDER BY AVG(sal) ;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>AVG(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1566,66667</td>
</tr>
<tr>
<td>20</td>
<td>2175</td>
</tr>
<tr>
<td>10</td>
<td>2916,66667</td>
</tr>
</tbody>
</table>
Creating Groups of Data

SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno;

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>AVG(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1566,66667</td>
</tr>
<tr>
<td>20</td>
<td>2175</td>
</tr>
<tr>
<td>10</td>
<td>2916,66667</td>
</tr>
</tbody>
</table>
Using the GROUP BY Clause

All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>AVG(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1566,66667</td>
</tr>
<tr>
<td>20</td>
<td>2175</td>
</tr>
<tr>
<td>10</td>
<td>2916,66667</td>
</tr>
</tbody>
</table>
Using the GROUP BY Clause

The GROUP BY column does not have to be in the SELECT list.

SELECT AVG(sal)
FROM emp
GROUP BY deptno;

<table>
<thead>
<tr>
<th>AVG(sal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1566.66667</td>
</tr>
<tr>
<td>2175</td>
</tr>
<tr>
<td>2916.66667</td>
</tr>
</tbody>
</table>
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
ORDER BY AVG(sal) ;

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>AVG(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1566,66667</td>
</tr>
<tr>
<td>20</td>
<td>2175</td>
</tr>
<tr>
<td>10</td>
<td>2916,66667</td>
</tr>
</tbody>
</table>
Grouping by More Than One Column

```
SELECT deptno, job, sal
FROM emp
ORDER BY deptno ;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>JOB</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>MANAGER</td>
<td>2450</td>
</tr>
<tr>
<td>10</td>
<td>PRESIDENT</td>
<td>5000</td>
</tr>
<tr>
<td>10</td>
<td>CLERK</td>
<td>1300</td>
</tr>
<tr>
<td>20</td>
<td>MANAGER</td>
<td>2975</td>
</tr>
<tr>
<td>20</td>
<td>ANALYST</td>
<td>3000</td>
</tr>
<tr>
<td>20</td>
<td>CLERK</td>
<td>1100</td>
</tr>
<tr>
<td>20</td>
<td>CLERK</td>
<td>800</td>
</tr>
<tr>
<td>20</td>
<td>ANALYST</td>
<td>3000</td>
</tr>
<tr>
<td>30</td>
<td>SALESMAN</td>
<td>1250</td>
</tr>
<tr>
<td>30</td>
<td>SALESMAN</td>
<td>1500</td>
</tr>
<tr>
<td>30</td>
<td>SALESMAN</td>
<td>1600</td>
</tr>
<tr>
<td>30</td>
<td>CLERK</td>
<td>950</td>
</tr>
<tr>
<td>30</td>
<td>MANAGER</td>
<td>2850</td>
</tr>
<tr>
<td>30</td>
<td>SALESMAN</td>
<td>1250</td>
</tr>
</tbody>
</table>

14 rows selected.
Grouping by More Than One Column

```
SELECT deptno, job, SUM(sal)
FROM emp
GROUP BY deptno, job
ORDER BY deptno;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>JOB</th>
<th>SUM(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>CLERK</td>
<td>1300</td>
</tr>
<tr>
<td>10</td>
<td>MANAGER</td>
<td>2450</td>
</tr>
<tr>
<td>10</td>
<td>PRESIDENT</td>
<td>5000</td>
</tr>
<tr>
<td>20</td>
<td>ANALYST</td>
<td>6000</td>
</tr>
<tr>
<td>20</td>
<td>CLERK</td>
<td>1900</td>
</tr>
<tr>
<td>20</td>
<td>MANAGER</td>
<td>2975</td>
</tr>
<tr>
<td>30</td>
<td>CLERK</td>
<td>950</td>
</tr>
<tr>
<td>30</td>
<td>MANAGER</td>
<td>2850</td>
</tr>
<tr>
<td>30</td>
<td>SALESMAN</td>
<td>5600</td>
</tr>
</tbody>
</table>

9 rows selected.

Groups Within Groups
Sometimes there is a need to see results for groups within groups. The slide shows a report that displays the total salary being paid to each job title, within each department.
The EMP table is grouped first by department number, and within that grouping, it is grouped by job title. For example, the two clerks in department 20 are grouped together and a single result (total salary) is produced for all salespeople within the group.
Grouping by More Than One Column

```
SELECT   deptno , job, AVG(sal) , MAX(sal)
FROM emp
GROUP BY deptno, job;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>JOB</th>
<th>AVG(SAL)</th>
<th>MAX(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>CLERK</td>
<td>950</td>
<td>1100</td>
</tr>
<tr>
<td>30</td>
<td>SALESMAN</td>
<td>1400</td>
<td>1600</td>
</tr>
<tr>
<td>20</td>
<td>MANAGER</td>
<td>2975</td>
<td>2975</td>
</tr>
<tr>
<td>30</td>
<td>CLERK</td>
<td>950</td>
<td>950</td>
</tr>
<tr>
<td>10</td>
<td>PRESIDENT</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>30</td>
<td>MANAGER</td>
<td>2850</td>
<td>2850</td>
</tr>
<tr>
<td>10</td>
<td>CLERK</td>
<td>1300</td>
<td>1300</td>
</tr>
<tr>
<td>10</td>
<td>MANAGER</td>
<td>2450</td>
<td>2450</td>
</tr>
<tr>
<td>20</td>
<td>ANALYST</td>
<td>3000</td>
<td>3000</td>
</tr>
</tbody>
</table>

9 rows selected.

Groups Within Groups
Sometimes there is a need to see results for groups within groups. The slide shows a report that displays the total salary being paid to each job title, within each department. The EMP table is grouped first by department number, and within that grouping, it is grouped by job title. For example, the two clerks in department 20 are grouped together and a single result (total salary) is produced for all salespeople within the group.
Using the GROUP BY Clause on Multiple Columns

```
SELECT  deptno, job, sum(sal)
FROM     emp
GROUP BY deptno, job;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>JOB</th>
<th>SUM(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>CLERK</td>
<td>1900</td>
</tr>
<tr>
<td>30</td>
<td>SALESMAN</td>
<td>5600</td>
</tr>
<tr>
<td>20</td>
<td>MANAGER</td>
<td>2975</td>
</tr>
<tr>
<td>30</td>
<td>CLERK</td>
<td>950</td>
</tr>
<tr>
<td>10</td>
<td>PRESIDENT</td>
<td>5000</td>
</tr>
<tr>
<td>30</td>
<td>MANAGER</td>
<td>2850</td>
</tr>
<tr>
<td>10</td>
<td>CLERK</td>
<td>1300</td>
</tr>
<tr>
<td>10</td>
<td>MANAGER</td>
<td>2450</td>
</tr>
<tr>
<td>20</td>
<td>ANALYST</td>
<td>6000</td>
</tr>
</tbody>
</table>

9 rows selected.

Groups Within Groups (continued)

You can return summary results for groups and subgroups by listing more than one GROUP BY column. You can determine the default sort order of the results by the order of the columns in the GROUP BY clause. Here is how the SELECT statement on the slide, containing a GROUP BY clause, is evaluated:

- The SELECT clause specifies the column to be retrieved:
  - Department number in the EMP table
  - Job title in the EMP table
  - The sum of all the salaries in the group that you specified in the GROUP BY clause
- The FROM clause specifies the tables that the database must access: the EMP table.
- The GROUP BY clause specifies how you must group the rows:
Illegal Queries Using Group Functions

Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause.

```
SELECT deptno, COUNT(ename)
FROM emp;
```

```
SELECT deptno, COUNT(ename) *
```

ERROR at line 1:
ORA-00937: not a single-group group function

Illegal Queries Using Group Functions

Whenever you use a mixture of individual items (DEPTNO) and group functions (COUNT) in the same SELECT statement, you must include a GROUP BY clause that specifies the individual items (in this case. DEPTNO). If the GROUP BY clause is missing, then the error message "not a single-group group function" appears and an asterisk (*) points to the offending column. You can correct the error on the slide by adding the GROUP BY clause.

```
SELECT deptno, COUNT(ename)
FROM emp
GROUP BY deptno;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>COUNT(ENAME)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
illegal Queries Using Group Functions

- You cannot use the WHERE clause to restrict groups.
- You use the HAVING clause to restrict groups.

```
SELECT deptno, AVG(sal)
FROM emp
WHERE AVG(sal) > 2000
GROUP BY deptno;
```

WHERE AVG(sal) > 2000

```
ERROR at line 3:
ORA-00934: group function is not allowed here
```

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
HAVING AVG(sal) > 2000;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>AVG(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>2175</td>
</tr>
<tr>
<td>10</td>
<td>2916.66667</td>
</tr>
</tbody>
</table>
Correct form:

```sql
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
HAVING AVG(sal) > 2000;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>AVG(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>2175</td>
</tr>
<tr>
<td>10</td>
<td>2916,66667</td>
</tr>
</tbody>
</table>

Illegal Queries Using Group Functions (continued)

The WHERE clause cannot be used to restrict groups. The SELECT statement on the slide results in an error because it uses the WHERE clause to restrict the display of average salaries of those departments that have an average salary greater than $2000.

You can correct the slide error by using the HAVING clause to restrict groups.
Excluding Group Results

Restricting Group Results
In the same way that you use the WHERE clause to restrict the rows that you select, you use the HAVING clause to restrict groups. To find the maximum salary of each department, but show only the departments that have a maximum salary of more than $2900, you need to do the following:
Find the average salary for each department by grouping by department number. Restrict the groups to those departments with a maximum salary greater than $2900.

```
SELECT deptno, sal
FROM emp;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>800</td>
</tr>
<tr>
<td>30</td>
<td>1600</td>
</tr>
</tbody>
</table>

14 rows selected.

```
SELECT deptno, MAX(sal)
FROM emp
GROUP BY deptno;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>MAX(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>2850</td>
</tr>
<tr>
<td>20</td>
<td>3000</td>
</tr>
<tr>
<td>10</td>
<td>5000</td>
</tr>
</tbody>
</table>


Excluding Group Results: HAVING Clause

Use the HAVING clause to restrict groups
- Rows are grouped.
- The group function is applied.
- Groups matching the HAVING clause are displayed.

```
SELECT column, group_function
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]
[ORDER BY column];
```

The HAVING Clause

You use the HAVING clause to specify which groups are to be displayed. Therefore, you further restrict the groups on the basis of aggregate information.

In the syntax:

`group condition` restricts the groups of rows returned to those groups for which the specified condition is TRUE

The Oracle Server performs the following steps when you use the HAVING clause:

- Rows are grouped.
- The group function is applied to the group.
- The groups that match the criteria in the HAVING clause are displayed.

The HAVING clause can precede the GROUP BY clause, but it is recommended that you place the GROUP BY clause first because it is more logical. Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.
Using the HAVING Clause

Excluding Group Results

```
SELECT     deptno,   max(sal)
FROM        emp
GROUP BY    deptno
HAVING      max(sal) > 2900;
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>MAX(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>3000</td>
</tr>
<tr>
<td>10</td>
<td>5000</td>
</tr>
</tbody>
</table>

The HAVING Clause (continued)
The slide example displays department numbers and maximum salary for those departments whose maximum salary is greater than $2900.
You can use the GROUP BY clause without using a group function in the SELECT list.
If you restrict rows based on the result of a group function, you must have a GROUP BY clause as well as the HAVING clause.
The following example displays the department numbers and average salary for those departments whose maximum salary is greater than $2900:
Restricting Group Results

in the same way that you use the WHERE clause to restrict the rows that you select, you use the HAVING clause to restrict groups. To find the maximum salary of each department, but show only the departments that have a maximum salary of more than $2900, you need to do the following;

Find the average salary for each department by grouping by department number. Restrict the groups to those departments with maximum salary greater than $2900.
Using the HAVING Clause

```
SELECT job, SUM(sal) Ücretler
FROM emp
WHERE job NOT LIKE 'SALES%'
GROUP BY job
HAVING SUM(sal)>5000
ORDER BY SUM(sal);
```

<table>
<thead>
<tr>
<th>JOB</th>
<th>ÜCRETLER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYST</td>
<td>6000</td>
</tr>
<tr>
<td>MANAGER</td>
<td>8275</td>
</tr>
</tbody>
</table>

The HAVING Clause (continued)
The slide example displays the job title and total monthly salary for each job title with a total payroll exceeding $5000. The example excludes salespeople and sorts the list by the total monthly salary.
**Nesting Group Functions**

Display the maximum average salary.

```sql
SELECT max(avg(sal))
FROM emp
GROUP BY deptno;
```

<table>
<thead>
<tr>
<th>MAX(AVG(SAL))</th>
<th>2916,66667</th>
</tr>
</thead>
</table>

**Nesting Group Functions**

Group functions can be nested to a depth of two. The slide example displays the maximum average salary.
Summary

`SELECT column, group_function (column)`
`FROM table`
`[WHERE condition]`
`[GROUP BY group_by_expression]`
`[HAVING group_condition]`
`[ORDER BY column];`

Order of evaluation of the clauses:

- WHERE clause
- GROUP BY clause
- HAVING clause

Summary
Seven group functions are available in SQL.

- AVG
- COUNT
- MAX
- MIN
- SUM
- STDDEV
- VARIANCE
Excluding Group Results: HAVING Clause

Use the HAVING clause to restrict groups
- Rows are grouped.
- The group function is applied.
- Groups matching the HAVING clause are displayed.

```sql
SELECT column, group_function
FROM table
WHERE condition
GROUP BY group_by_expression
HAVING group_condition
ORDER BY column;
```

The HAVING Clause
You use the HAVING clause to specify to which groups are to be displayed. Therefore, you further restrict the groups on the basis of aggregate information in the syntax.

`group condition` restricts the groups of rows returned to those groups for which the specified condition is TRUE.

The Oracle Server performs the following steps when you use the HAVING clause:
- Rows are grouped.
- The group function is applied to the group. The groups that match the criteria in the HAVING clause are displayed.
- The HAVING clause can precede the GROUP BY clause, but it is recommended that you place the GROUP BY clause first because it is more logical. Groups are formed and group functions are
Practice 5

1. Groups functions work across many rows to produce one result per group.  TRUE / FALSE

2. Group functions include NULLs in calculations.  TRUE / FALSE

3. The WHERE clause restricts rows prior to inclusion in a group calculations.  TRUE / FALSE

4. Display the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number. Save your SQL statement in a file called p5q4.sql.

```
SELECT MAX(sal) Maximum, MIN(sal) Minimum, SUM(sal) SUM,
ROUND(AVG(sal),0) Average
FROM emp;
```

```
<table>
<thead>
<tr>
<th>MAXIMUM</th>
<th>MINIMUM</th>
<th>SUM</th>
<th>AVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>800</td>
<td>29025</td>
<td>2073</td>
</tr>
</tbody>
</table>
```

5.
8.
Exercises

Practice 5 (continued)

If you have time, complete the following exercises.

9 Display the manager number and the salary of the lowest paid employee for that manager. Exclude any one whose manager is not known. Exclude any groups where the minimum salary is less than $1000. Sort the output in descending order of salary.

```
SELECT  mgr, MIN(sal)
FROM emp
WHERE mgr IS NOT NULL
GROUP BY mgr
HAVING MIN(sal)  > 1000
ORDER BY MIN(sal) DESC;
```

<table>
<thead>
<tr>
<th>MGR</th>
<th>MIN(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7566</td>
<td>3000</td>
</tr>
<tr>
<td>7839</td>
<td>2450</td>
</tr>
<tr>
<td>7782</td>
<td>1300</td>
</tr>
<tr>
<td>7788</td>
<td>1100</td>
</tr>
</tbody>
</table>

4 rows returned in 0,00 seconds
10. Write a query to display the department name, location name, number of employees, and the average salary for all employees in that department. Label the columns dname, loc, Number of People, and Salary, respectively. Round the average salary to two decimal places.

```
Select  d.dname, d.loc, COUNT(e.empno) COUNT ÜçüSayısı, ROUND(AVG(e.sal),2) OrtalamaÜcret
FROM dept d, emp e
WHERE d.deptno = e.deptno
GROUP BY d.dname, d.loc;
```

<table>
<thead>
<tr>
<th>DNAME</th>
<th>LOC</th>
<th>İÇÇİSAYIŞI</th>
<th>ORTALAMAÜCRET</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESEARCH</td>
<td>DALLAS</td>
<td>5</td>
<td>2175</td>
</tr>
<tr>
<td>SALES</td>
<td>CHICAGO</td>
<td>6</td>
<td>1566.67</td>
</tr>
<tr>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
<td>3</td>
<td>2916.67</td>
</tr>
</tbody>
</table>

3 rows returned in 0.00 seconds
11. Create a query that will display the total number of employees and the total number of employees who were hired in 1980, 1981, 1982, and 1983. Give appropriate column headings.

Solution 11a

```sql
SELECT TO_CHAR(hiredate, 'YY') "İşe Giriş Yılı" ,
COUNT(*) İşeGirenSayısı
FROM emp
GROUP BY TO_CHAR(hiredate, 'YY')
ORDER BY TO_CHAR(hiredate, 'YY');
```

<table>
<thead>
<tr>
<th>Işe Giriş Yılı</th>
<th>İşeGirenSayısı</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>81</td>
<td>10</td>
</tr>
<tr>
<td>82</td>
<td>2</td>
</tr>
<tr>
<td>83</td>
<td>1</td>
</tr>
</tbody>
</table>

4 rows returned in 0.00 seconds
Solution 11b

SELECT COUNT(a.ename) "Toplam" , COUNT(b.ename) "1980" ,
COUNT(c.ename) "1981" , COUNT(d.ename) "1982" ,
COUNT(e.ename) "1983"
FROM emp a,
    ( SELECT ename FROM emp WHERE TO_CHAR(hiredate, 'YY') = '80' ) b,
    ( SELECT ename FROM emp WHERE TO_CHAR(hiredate, 'YY') = '81' ) c,
    ( SELECT ename FROM emp WHERE TO_CHAR(hiredate, 'YY') = '82' ) d,
    ( SELECT ename FROM emp WHERE TO_CHAR(hiredate, 'YY') = '83' ) e
WHERE
a.ename = b.ename(+) AND a.ename = c.ename(+) AND a.ename = d.ename(+)
AND a.ename = e.ename(+);

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>14</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>