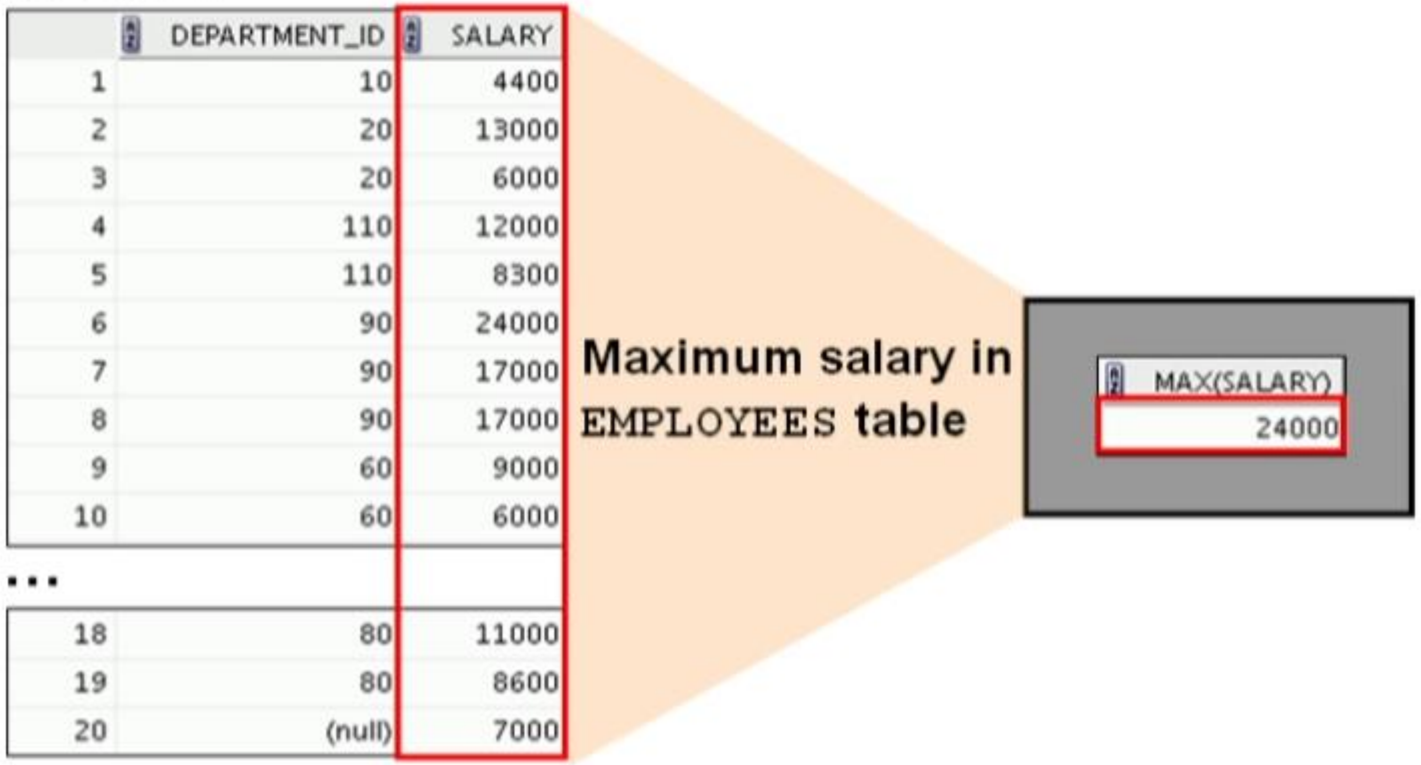
Reporting Aggregated Data Using the Group Functions

What Are Group Functions?

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

EMPLOYEES

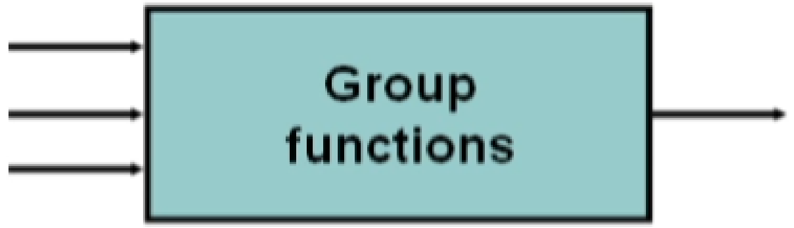


	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
…		
18	80	11000
19	80	8600
20	(null)	7000

MAX(SALARY)
24000

Types of Group Functions

- `AVG`
- `COUNT`
- `MAX`
- `MIN`
- `STDDEV`
- `SUM`
- `VARIANCE`



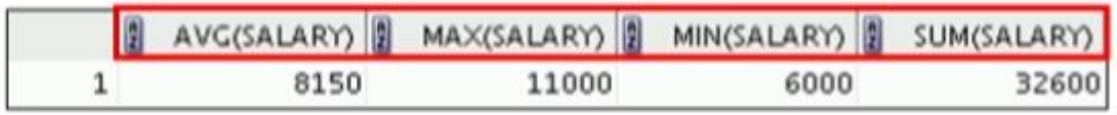
Group Functions: Syntax

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

Using the AVG and SUM Functions

You can use AVG and SUM for numeric data.

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

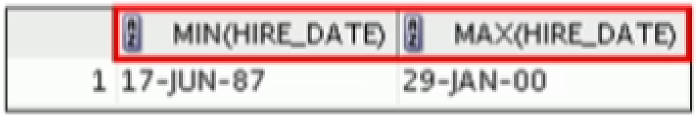


```
     AVG(SALARY)  MAX(SALARY)  MIN(SALARY)  SUM(SALARY)
1           8150        11000         6000        32600
```

Using the MIN and MAX Functions

You can use MIN and MAX for numeric, character, and date data types.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM        employees;
```



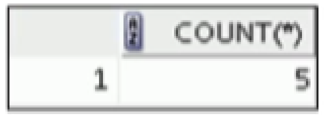
	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	17-JUN-87	29-JAN-00

Using the COUNT Function

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

1

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

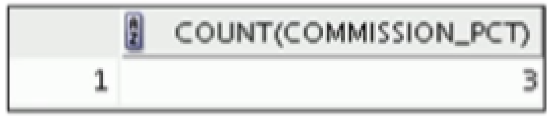


	COUNT(*)
1	5

`COUNT(expr)` returns the number of rows with non-null values for *expr*:

2

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 80;
```

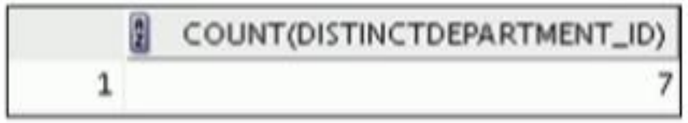


	COUNT(COMMISSION_PCT)
1	3

Using the DISTINCT Keyword

- COUNT(DISTINCT *expr*) returns the number of distinct non-null values of *expr*.
- To display the number of distinct department values in the EMPLOYEES table:

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```



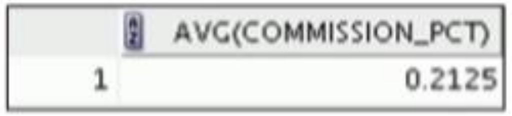
```
     COUNT(DISTINCTDEPARTMENT_ID)
1                               7
```

Group Functions and Null Values

Group functions ignore null values in the column:

1

```
SELECT AVG(commission_pct)
FROM   employees;
```

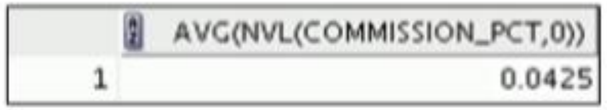


```
     AVG(COMMISSION_PCT)
1                 0.2125
```

The NVL function forces group functions to include null values:

2

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

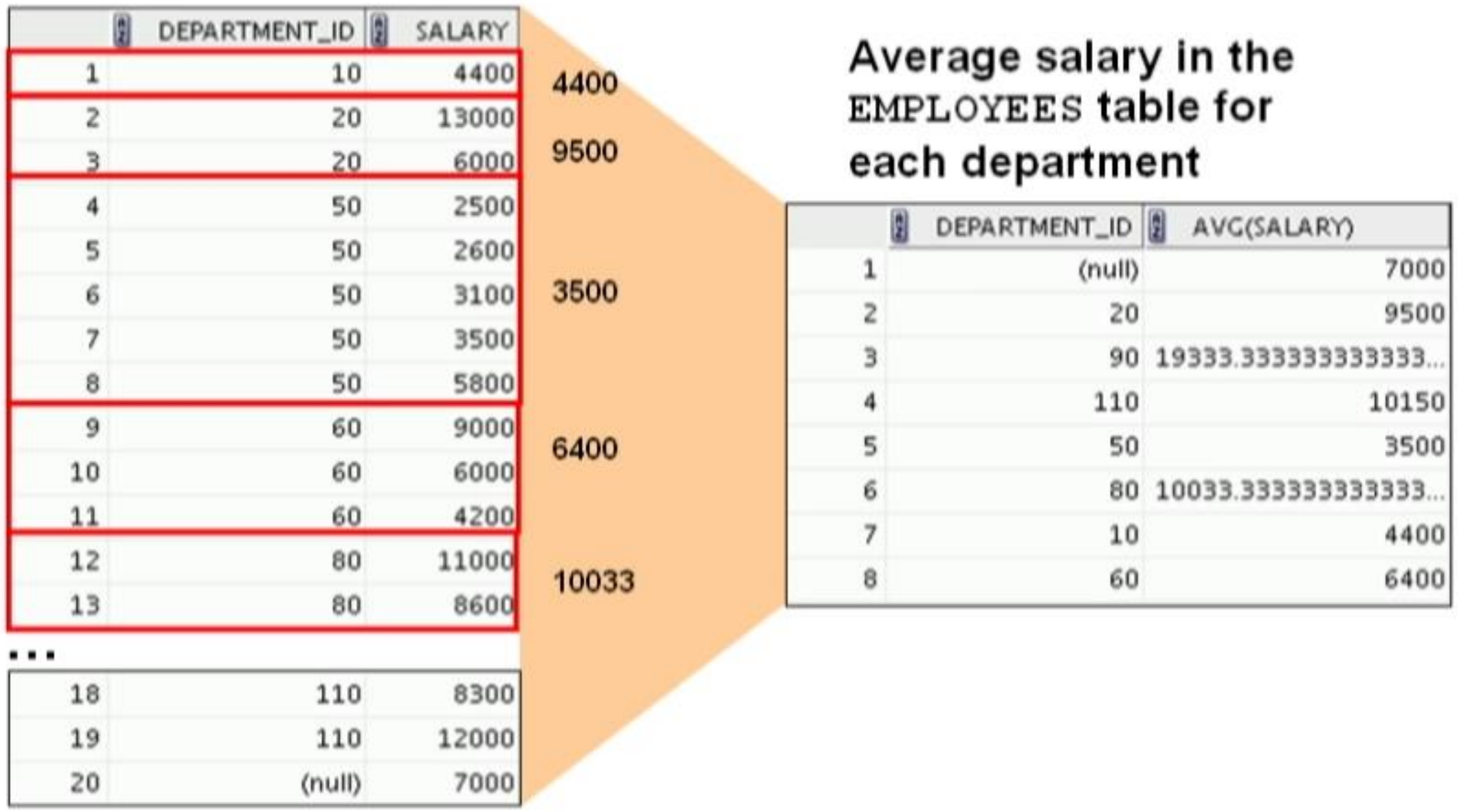


```
     AVG(NVL(COMMISSION_PCT,0))
1                        0.0425
```

Group data by using the GROUP BY clause

Creating Groups of Data

EMPLOYEES



	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	80	8600

...

18	110	8300
19	110	12000
20	(null)	7000

Average salary in the EMPLOYEES table for each department

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	20	9500
3	90	19333.3333333333333...
4	110	10150
5	50	3500
6	80	10033.3333333333333...
7	10	4400
8	60	6400

Creating Groups of Data: GROUP BY Clause Syntax

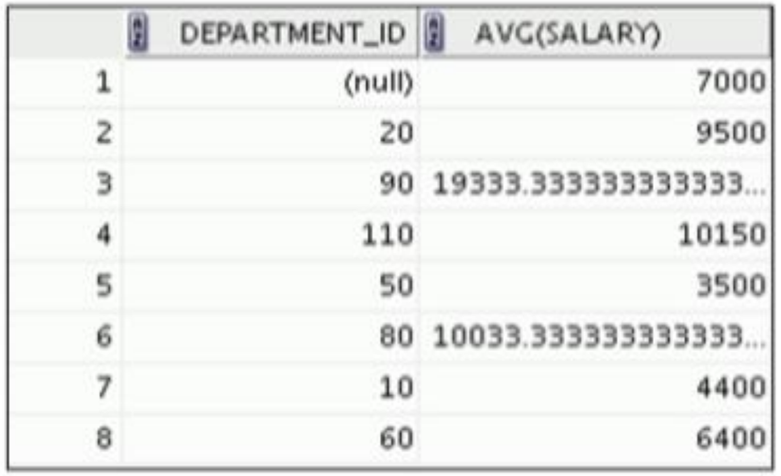
You can divide rows in a table into smaller groups by using the GROUP BY clause.

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

Using the GROUP BY Clause

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

```
SELECT   department_id, AVG(salary)
FROM     employees
GROUP BY department_id ;
```

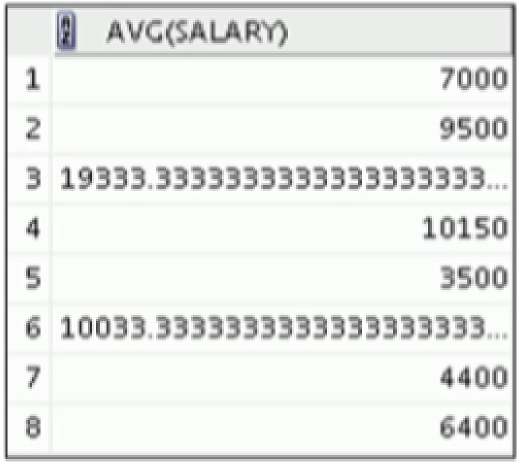


	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	20	9500
3	90	19333.333333333333...
4	110	10150
5	50	3500
6	80	10033.333333333333...
7	10	4400
8	60	6400

Using the GROUP BY Clause

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

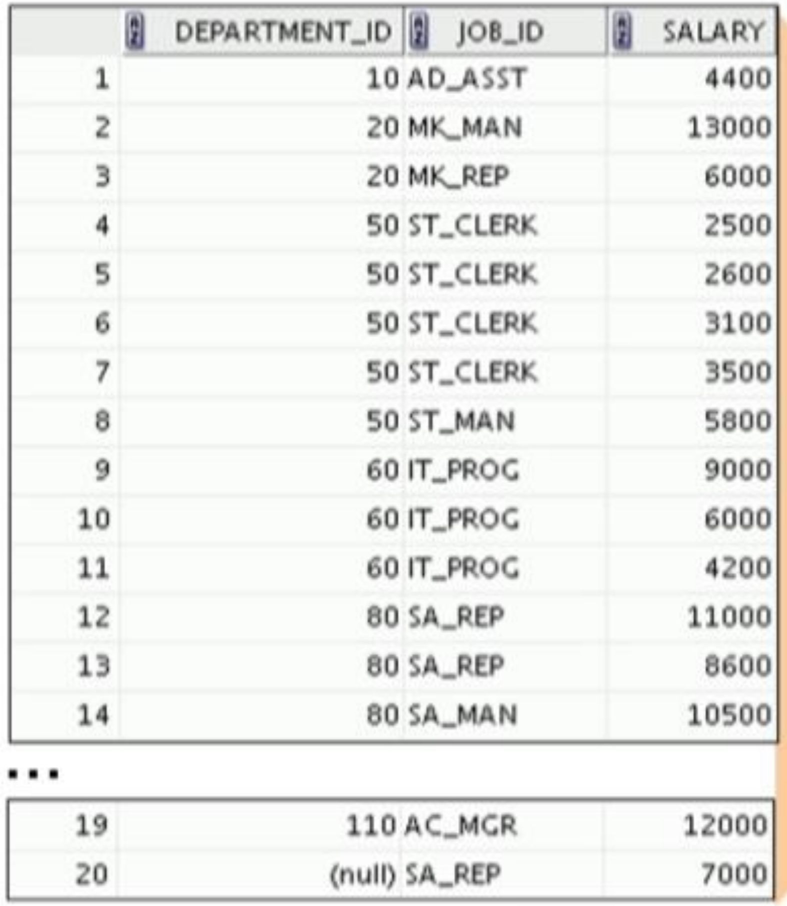
```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id ;
```



```
     AVG(SALARY)
1                      7000
2                      9500
3 19333.3333333333333333333...
4                     10150
5                      3500
6 10033.3333333333333333333...
7                      4400
8                      6400
```

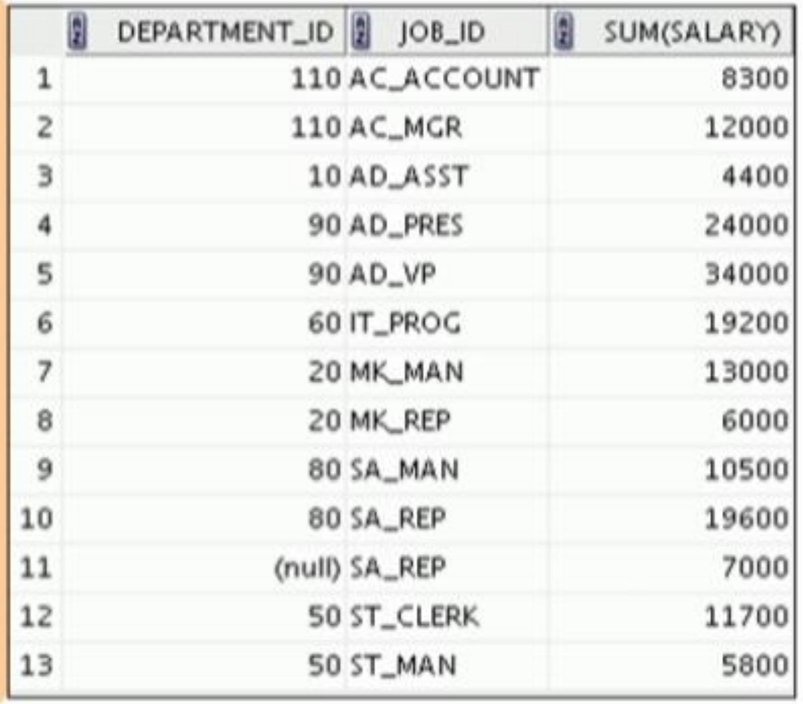
Grouping by More Than One Column

EMPLOYEES



	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	2500
5	50	ST_CLERK	2600
6	50	ST_CLERK	3100
7	50	ST_CLERK	3500
8	50	ST_MAN	5800
9	60	IT_PROG	9000
10	60	IT_PROG	6000
11	60	IT_PROG	4200
12	80	SA_REP	11000
13	80	SA_REP	8600
14	80	SA_MAN	10500
...			
19	110	AC_MGR	12000
20	(null)	SA_REP	7000

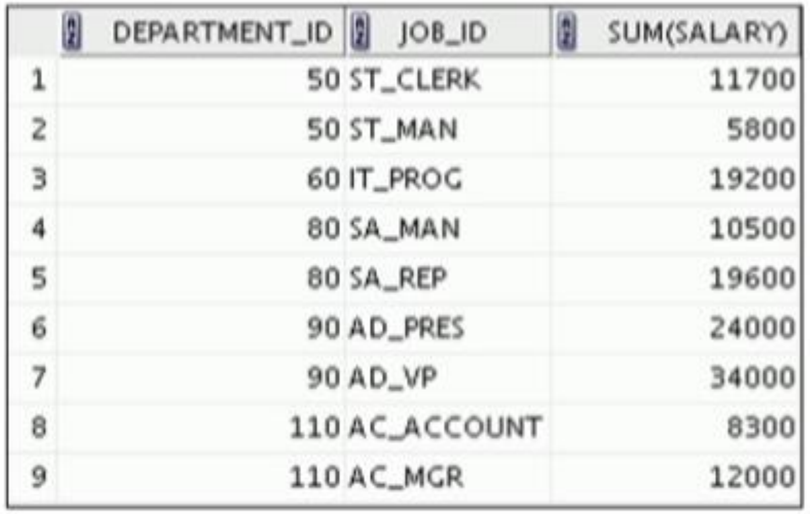
Add the salaries in the `EMPLOYEES` table for each job, grouped by department.



	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	110	AC_ACCOUNT	8300
2	110	AC_MGR	12000
3	10	AD_ASST	4400
4	90	AD_PRES	24000
5	90	AD_VP	34000
6	60	IT_PROG	19200
7	20	MK_MAN	13000
8	20	MK_REP	6000
9	80	SA_MAN	10500
10	80	SA_REP	19600
11	(null)	SA_REP	7000
12	50	ST_CLERK	11700
13	50	ST_MAN	5800

Using the GROUP BY Clause on Multiple Columns

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE     department_id > 40
GROUP BY department_id, job_id
ORDER BY department_id;
```



	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12000

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 department_id, COUNT(last_name)
FROM   employees;
```

```
ORA-00937: not a single-group group function
00937. 00000 -  "not a single-group group function"
```

A GROUP BY clause must be added to count the last names for each department_id.

```
SELECT department_id, job_id, COUNT(last_name)
FROM   employees
GROUP BY department_id;
```

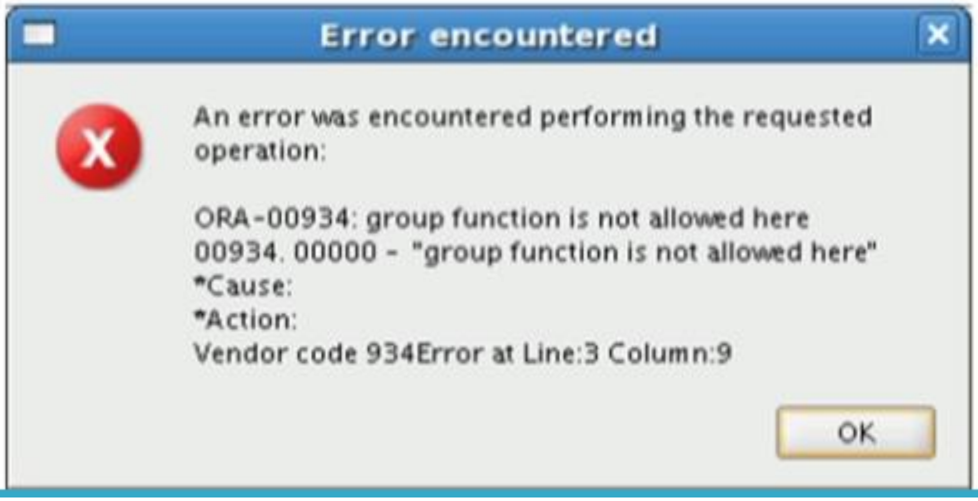
```
ORA-00979: not a GROUP BY expression
00979. 00000 -  "not a GROUP BY expression"
```

Either add job_id in the GROUP BY or remove the job_id column from the SELECT list.

Illegal Queries Using Group Functions

- You cannot use the `WHERE` clause to restrict groups.
- You use the `HAVING` clause to restrict groups.
- You cannot use group functions in the `WHERE` clause.

```
SELECT   department_id, AVG(salary)
FROM     employees
WHERE    AVG(salary) > 8000
GROUP BY department_id;
```



An error was encountered performing the requested operation:

```
ORA-00934: group function is not allowed here
00934. 00000 - "group function is not allowed here"
*Cause:
*Action:
Vendor code 934Error at Line:3 Column:9
```

Cannot use the `WHERE` clause to restrict groups

Restricting Group Results with the `HAVING` Clause

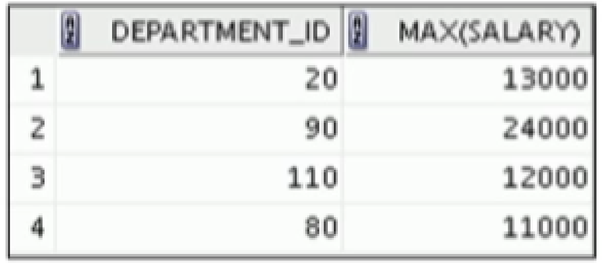
When you use the `HAVING` clause, the Oracle server restricts groups as follows:

1. Rows are grouped.
2. The group function is applied.
3. 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];
```

Using the HAVING Clause

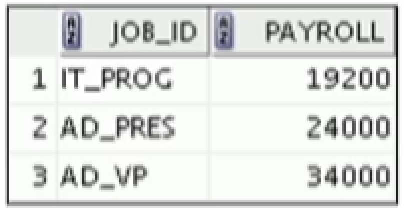
```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000 ;
```



	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

Using the HAVING Clause

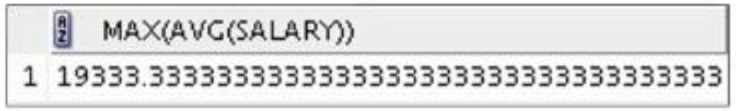
```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```



	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

Nesting Group Functions in Oracle

```
SELECT    MAX(AVG(salary))
FROM      employees
GROUP BY department_id;
```



```
   MAX(AVG(SALARY))
1  19333.3333333333333333333333333333333
```

Summary

In this section, you should have learned how to:

- Use the group functions `COUNT`, `MAX`, `MIN`, `SUM`, and `AVG`
- Write queries that use the `GROUP BY` clause
- Write queries that use the `HAVING` clause

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