Oracle Database 11g

SQL Fundamentals – Lab 2

SQL Functions

Single-Row Functions

Character Functions

These functions convert case for character strings.

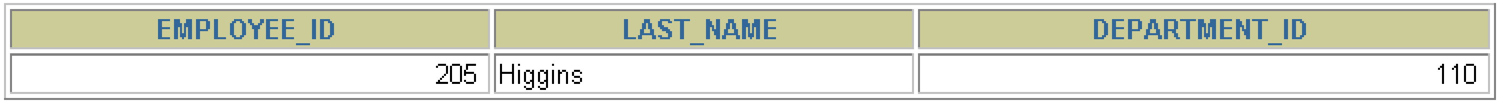
Function	Result
`LOWER('SQL Course')`	`sql course`
`UPPER('SQL Course')`	`SQL COURSE`
`INITCAP('SQL Course')`	`Sql Course`

Using Case Manipulation Functions

Display the employee number, name, and department number for employee Higgins:

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  last_name = 'higgins';
no rows selected
```

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'higgins';
```

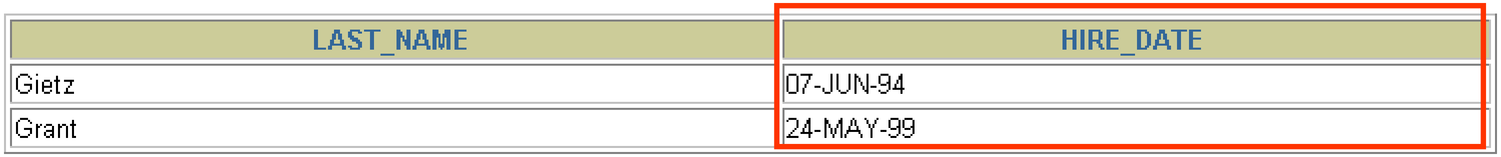


EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
205	Higgins	110

Working with Dates

- The default date display format is DD-MON-RR.

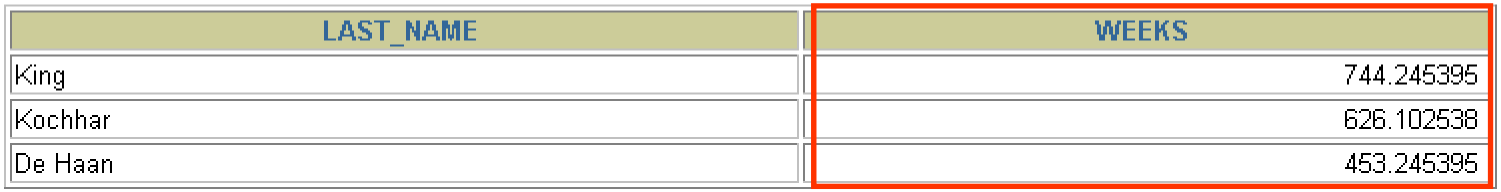
```
SELECT last_name, hire_date
FROM   employees
WHERE  last_name like 'G%';
```



LAST_NAME	HIRE_DATE
Gietz	07-JUN-94
Grant	24-MAY-99

Using Arithmetic Operators with Dates

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM   employees
WHERE  department_id = 90;
```

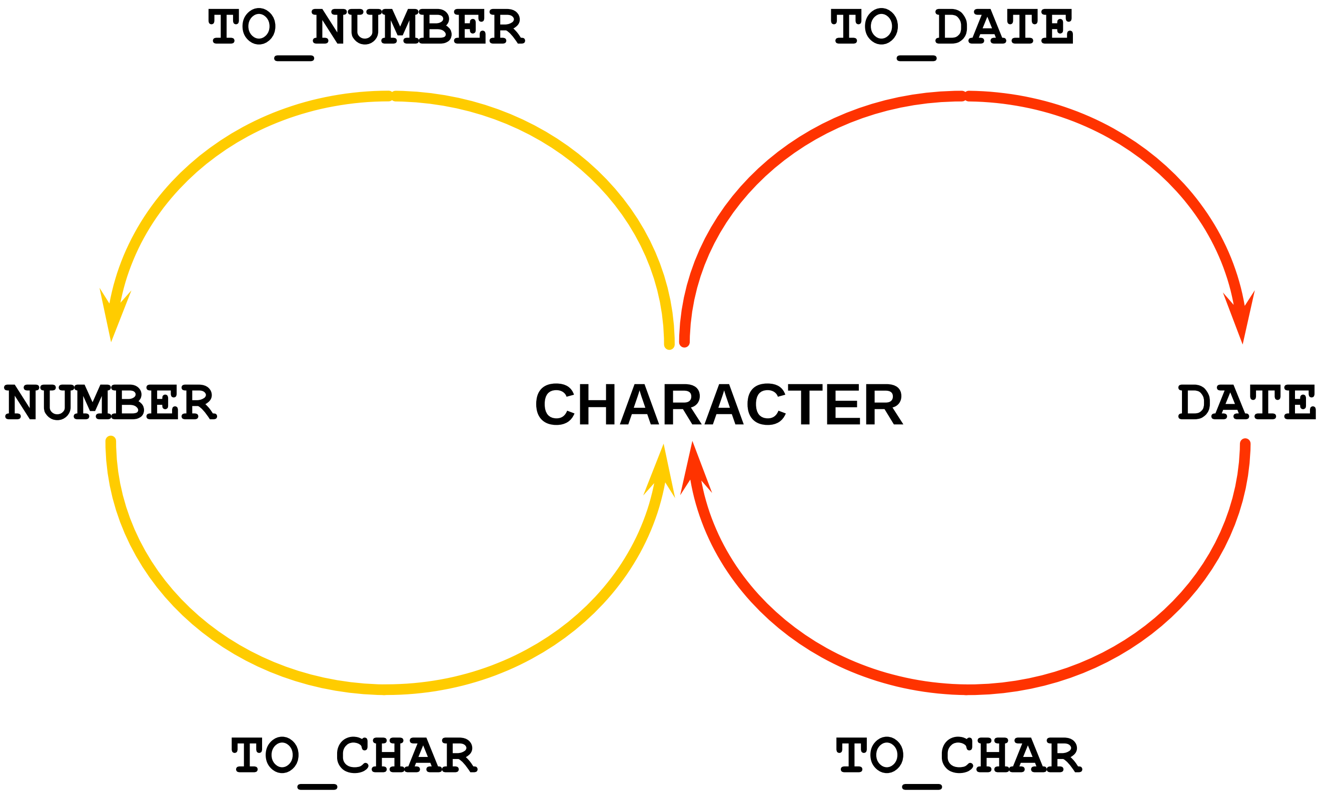


LAST_NAME	WEEKS
King	744.245395
Kochhar	626.102538
De Haan	453.245395

SYSDATE is a function that returns current Date and Time

Subtract two dates to find the number of days between those dates

Data Type Conversion



TO_NUMBER

TO_DATE

NUMBER

CHARACTER

DATE

TO_CHAR

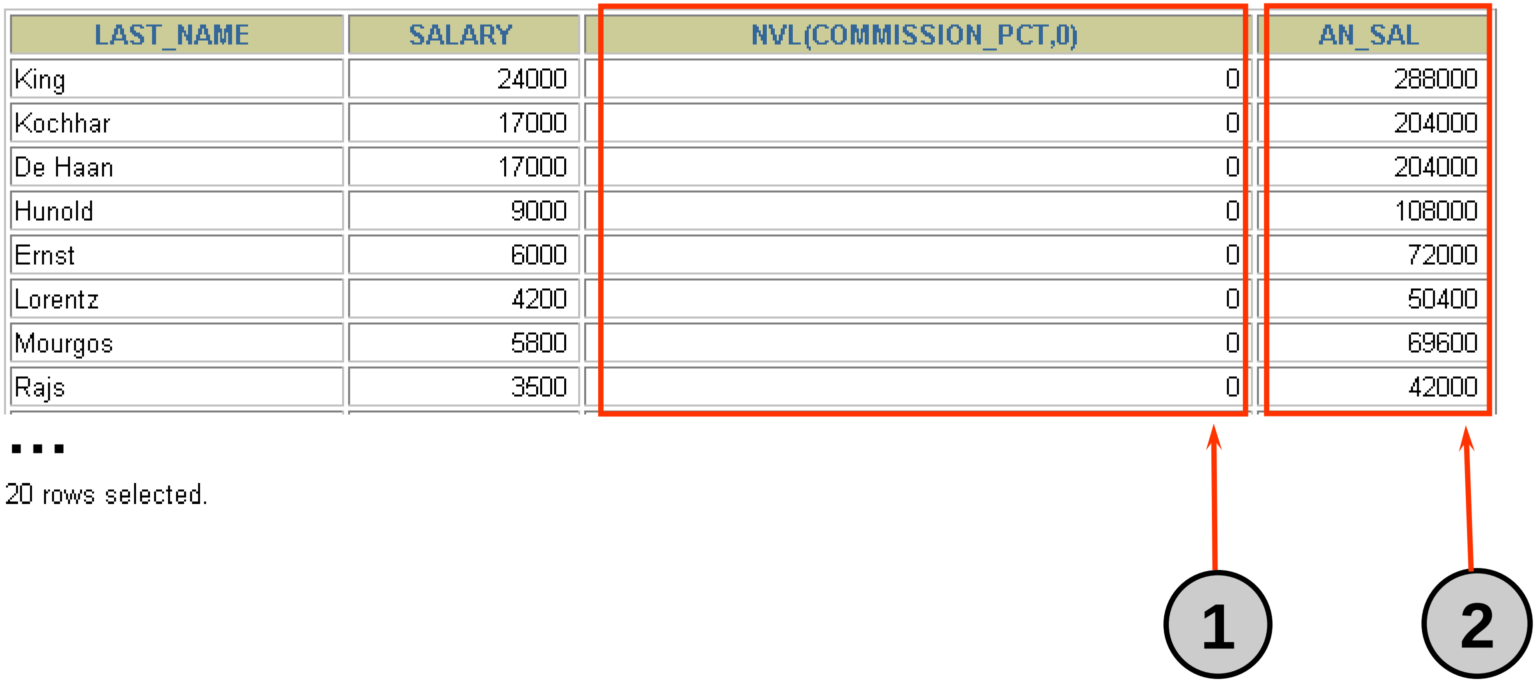
TO_CHAR

General Functions: `NVL` Function

- Converts a null to an actual value.
- Data types that can be used are date, character, and number.
- Data types must match:
  - `NVL (commission_pct, 0)`
  - `NVL (hire_date, '01-JAN-97')`
  - `NVL (job_id, 'No Job Yet')`

Using the NVL Function

```
SELECT last_name, salary, NVL(commission_pct, 0),
   (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```



LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
King	24000	0	288000
Kochhar	17000	0	204000
De Haan	17000	0	204000
Hunold	9000	0	108000
Ernst	6000	0	72000
Lorentz	4200	0	50400
Mourgos	5800	0	69600
Rajs	3500	0	42000

...

20 rows selected.

Summarizing Data Using Group Functions

Group Functions

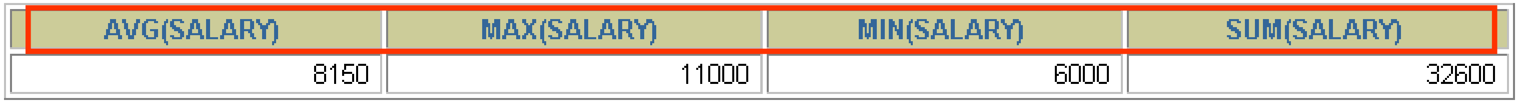
- Types of Group Functions include:
  - AVG – COUNT – MAX – MIN – SUM
- Group functions operate on set of values to return ONE value.
- Group Functions Syntax

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

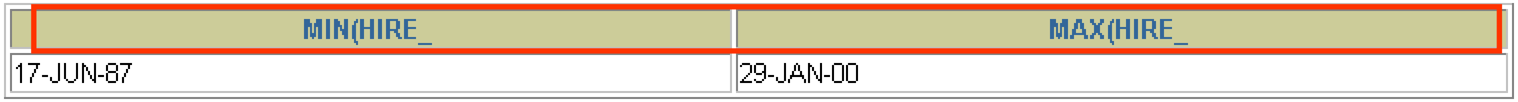


AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8150	11000	6000	32600

Using the **MIN** and **MAX** Functions

You can use **MIN** and **MAX** for any data type.

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

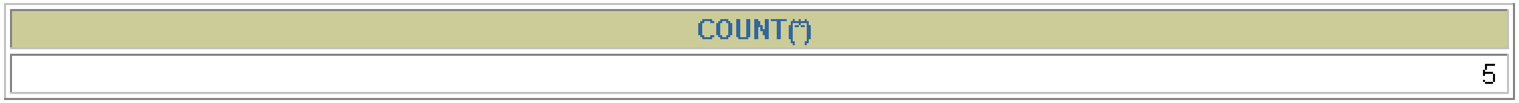


MIN(HIRE_	MAX(HIRE_
17-JUN-87	29-JAN-00

Using the COUNT Function

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

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



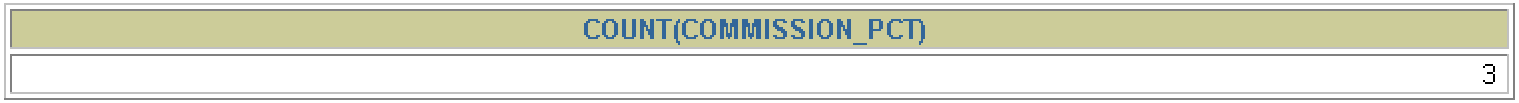
COUNT(*)
5

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

Using the COUNT Function

- COUNT(*column*) returns the number of rows with non-null values for the *column*.

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

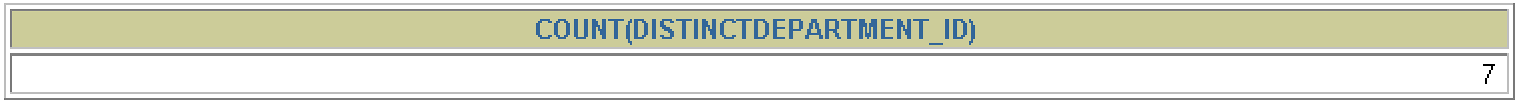


```
COUNT(COMMISSION_PCT)
                    3
```

Using the `DISTINCT` Keyword

- `COUNT(DISTINCT` *`column`*`)` returns the number of distinct non-null values of the *`column`*.
- Display the number of distinct department values in the `EMPLOYEES` table.

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

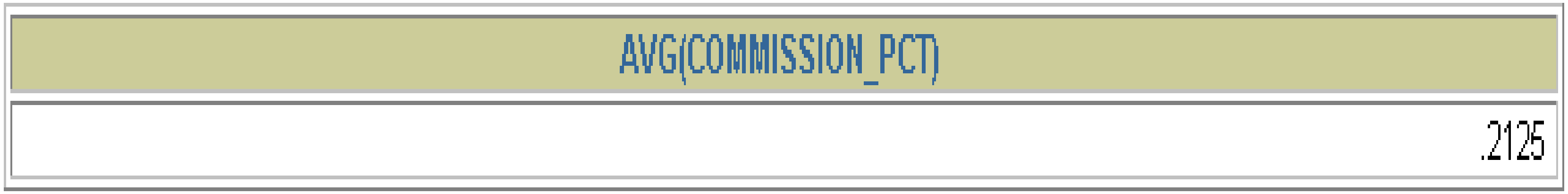


COUNT(DISTINCTDEPARTMENT_ID)
7

Group Functions and Null Values

- Group functions ignore null values in the column.

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



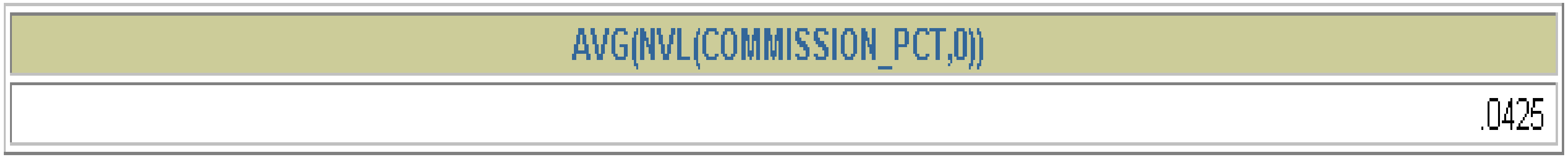
```
AVG(COMMISSION_PCT)
              .2125
```

- The average is calculated as the total commission paid divided by the number of employees receiving a commission (4).

Using the **NVL** Function with Group Functions

- The **NVL** function forces group functions to include null values.

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



AVG(NVL(COMMISSION_PCT,0))
.0425

- The average is calculated as the total commission paid to all employees divided by the total number of employees in the company (20).

Group Functions

- **Display maximum salary of all employees.**

```
Select   MAX (Salary)
From     Employees;
```

- Display maximum salary of employees in department 20.

```
Select   MAX (Salary)
From     Employees
Where  department_id= 20;
```

Grouping

- **What if we need to display max salary of employees in each department?**

We need to repeat the last select statement x times where x is the number of departments in the system.

OR Use Group By Clause

Creating Groups of Data: The GROUP BY Clause Syntax

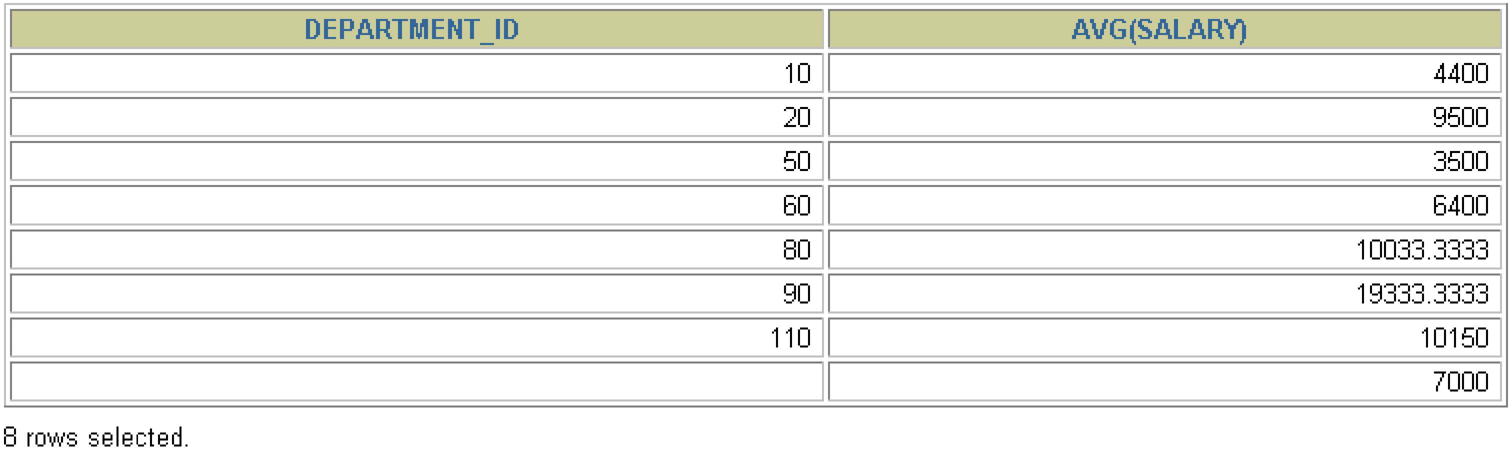
```
SELECT        column, group_function(column)
FROM          table
[WHERE        condition]
[GROUP BY     group_by_ column ]
[ORDER BY     column];
```

- Divide rows in a table into smaller groups by using the GROUP BY clause.
- If the group-by column contains null values, a group will be created for them.

Using the GROUP BY Clause

- Display the average salary for each department

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



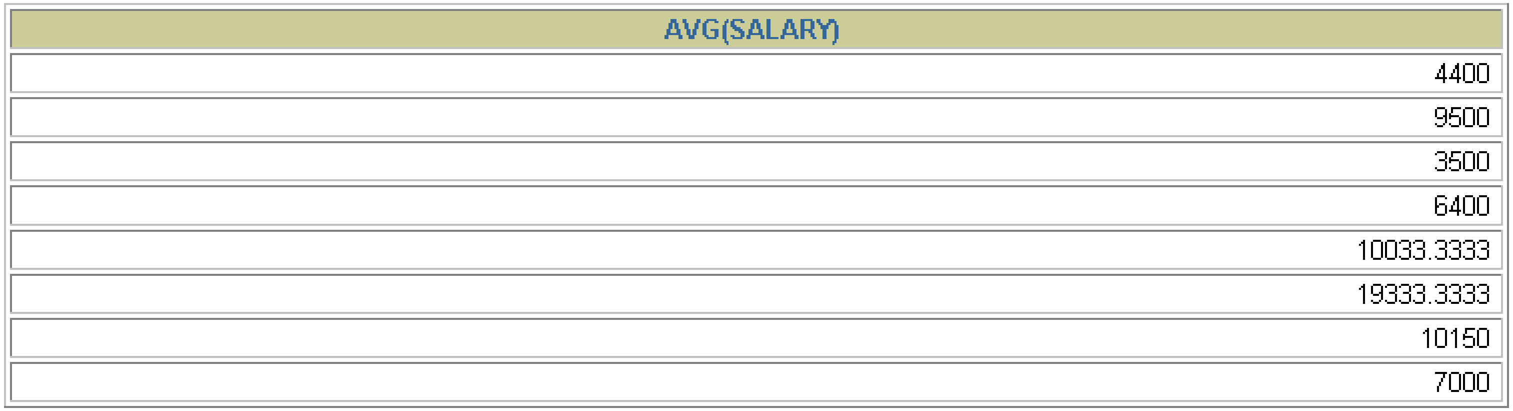
DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

8 rows selected.

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)
4400
9500
3500
6400
10033.3333
19333.3333
10150
7000

Is the query objective clear in this case ??

Illegal Queries
Using Group Functions

- Any column 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;
```

```
SELECT department_id, COUNT(last_name)
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

Column missing in the `GROUP  BY` clause

SQL Statement Execution

1. Table is identified due to FROM clause.
2. Rows are selected due to the WHERE condition.
3. Rows are grouped due to the GROUP BY clause.
4. The GROUP FUNCTION is applied to each group.
5. OREDER BY clause sorts results.

SQL Statement Execution: Example

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

- The **FROM** clause specifies the tables that the database must access: the EMPLOYEES table.
- The **WHERE** clause specifies the rows to be retrieved. Since there is no WHERE clause, all rows are retrieved by default.
- The **GROUP BY** clause specifies how the rows should be grouped. The rows are being grouped by department number.
- The **AVG** function applied to the salary column will calculate the average salary for each department.
- The **SELECT** clause displays department number and average salary for each department.

Illegal Queries
Using Group Functions

- You cannot use the `WHERE` 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;
```

```
WHERE  AVG(salary) > 8000
       *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

Cannot use the `WHERE` clause to restrict groups

Having Clause

Excluding Group Results: The HAVING Clause

- **To restrict the group results, that is display only groups that satisfy a specific condition, we use HAVING clause.**

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

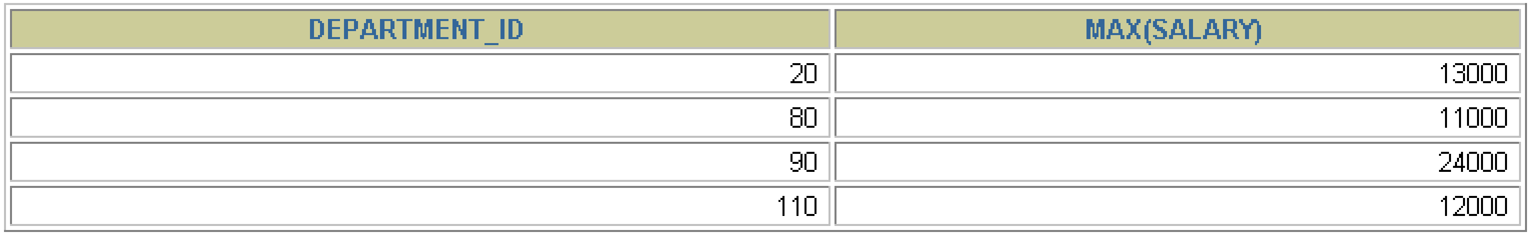
Excluding Group Results: The HAVING Clause

Use the HAVING clause to restrict groups:

1. Table is identified due to FROM clause.
2. Rows are selected due to the WHERE condition.
3. Rows are grouped due to the GROUP BY clause.
4. The group function is applied to each group.
5. Groups matching HAVING clause are returned.
6. The ORDER BY clause sorts results.

Using the HAVING Clause

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



DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

General Syntax

```
SELECT        [DISTINCT]  { * | column  [alias],  …}
FROM           table
[WHERE         condition (s)]
[GROUP BY     group-by column]
[HAVING       group condition]
[ORDER BY     {column | alias}  [ASC|DESC]];
```

Subqueries

Subquery Syntax

```
SELECT   select_list
FROM     table
WHERE    column operator
                    (SELECT       select_list
                     FROM         table);
```

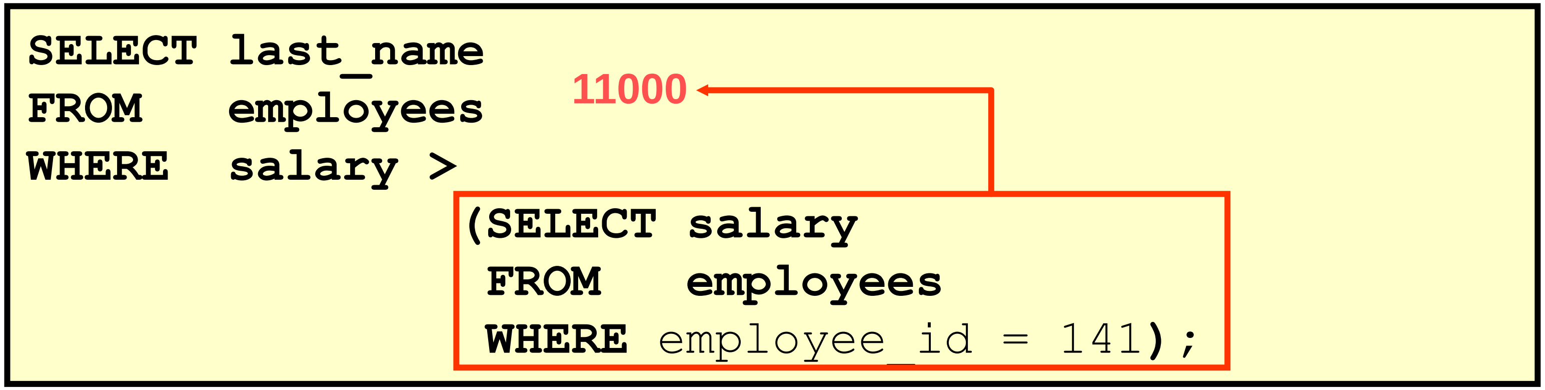
- The subquery (inner query) executes once before the main query (outer query).
- The result of the subquery is used by the main query.
- You can place the subquery in a number of SQL clauses, including the following:
  - `WHERE` clause
  - `HAVING` clause

Subquery

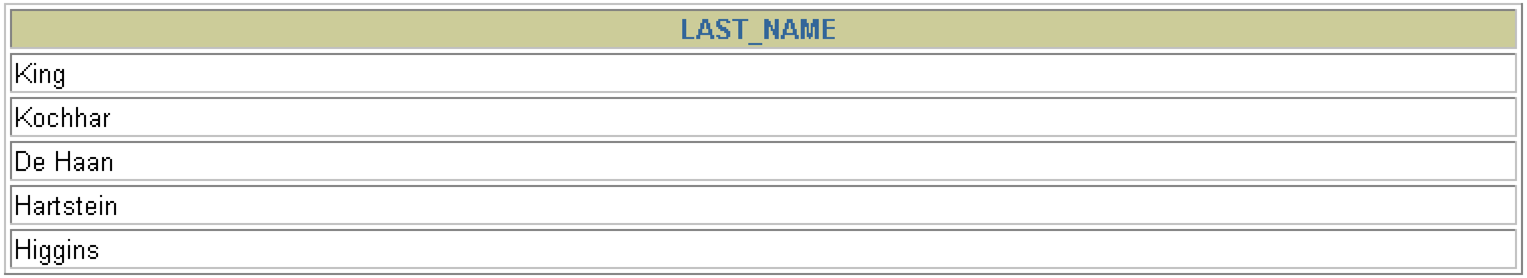
- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- The ORDER BY clause cannot be used in the subquery.
- They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself, or data from another table.

Using a Subquery

Who gets a higher salary than employee number 141 ?



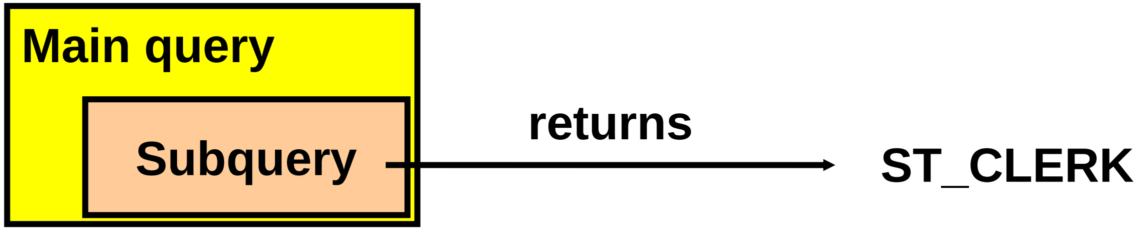
```
SELECT last_name
FROM   employees
WHERE  salary >
               (SELECT salary
                FROM   employees
                WHERE  employee_id = 141);
```



LAST_NAME
King
Kochhar
De Haan
Hartstein
Higgins

Types of Subqueries

– Single-row subquery



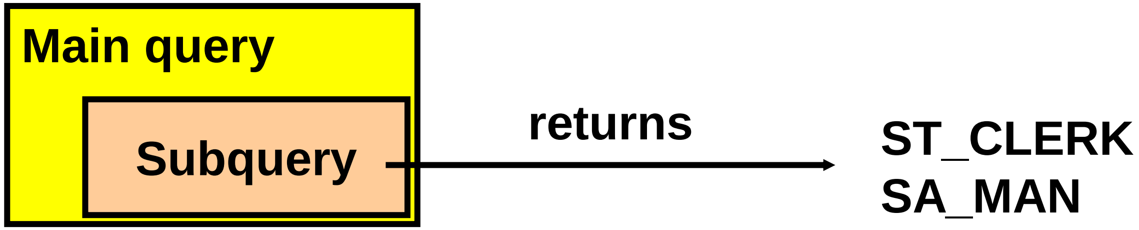
Main query

Subquery

returns

ST_CLERK

– Multiple-row subquery



Main query

Subquery

returns

ST_CLERK
SA_MAN

Single-row subqueries:

Queries that return only one row from inner `SELECT` statement

Multiple-row subqueries:

Queries that return more than one row from inner `SELECT` statement

Using Subqueries

- Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.

Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Multiple-Row Subqueries

- Returns more than one row
- Use multiple-row comparison operators, such as IN.

Single-Row Subqueries

- Display the employees whose job ID is the same as that of employee 141:

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
              (SELECT job_id
               FROM   employees
               WHERE  employee_id = 141);
```

Single-Row Subqueries

- Display information about the employees who work in the Sales department:

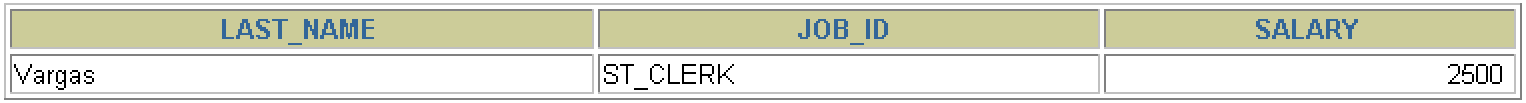
```
SELECT last_name, job_id
FROM   employees
WHERE  departmentid =
              (SELECT departmentid
               FROM   departments
               WHERE  departmentName=
                  ‘Sales’);
```

Using Group Functions in a Subquery

Display the employee last name, job ID, and salary of all employees whose salary is equal to the minimum salary

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees);
```

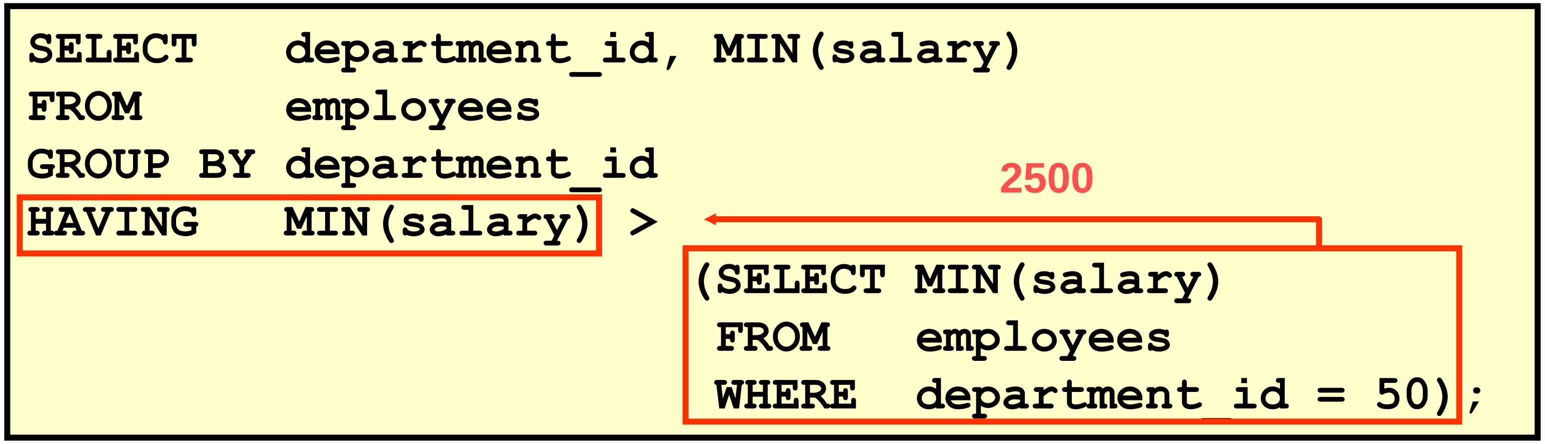
2500



LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

The HAVING Clause with Subqueries

- The Oracle server executes subqueries first.
- The Oracle server returns results into the HAVING clause of the main query.
- Display all the departments that have a minimum salary greater than that of department 50.



```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                       (SELECT MIN(salary)
                        FROM   employees
                        WHERE  department_id = 50);
```

What Is Wrong with This Statement?

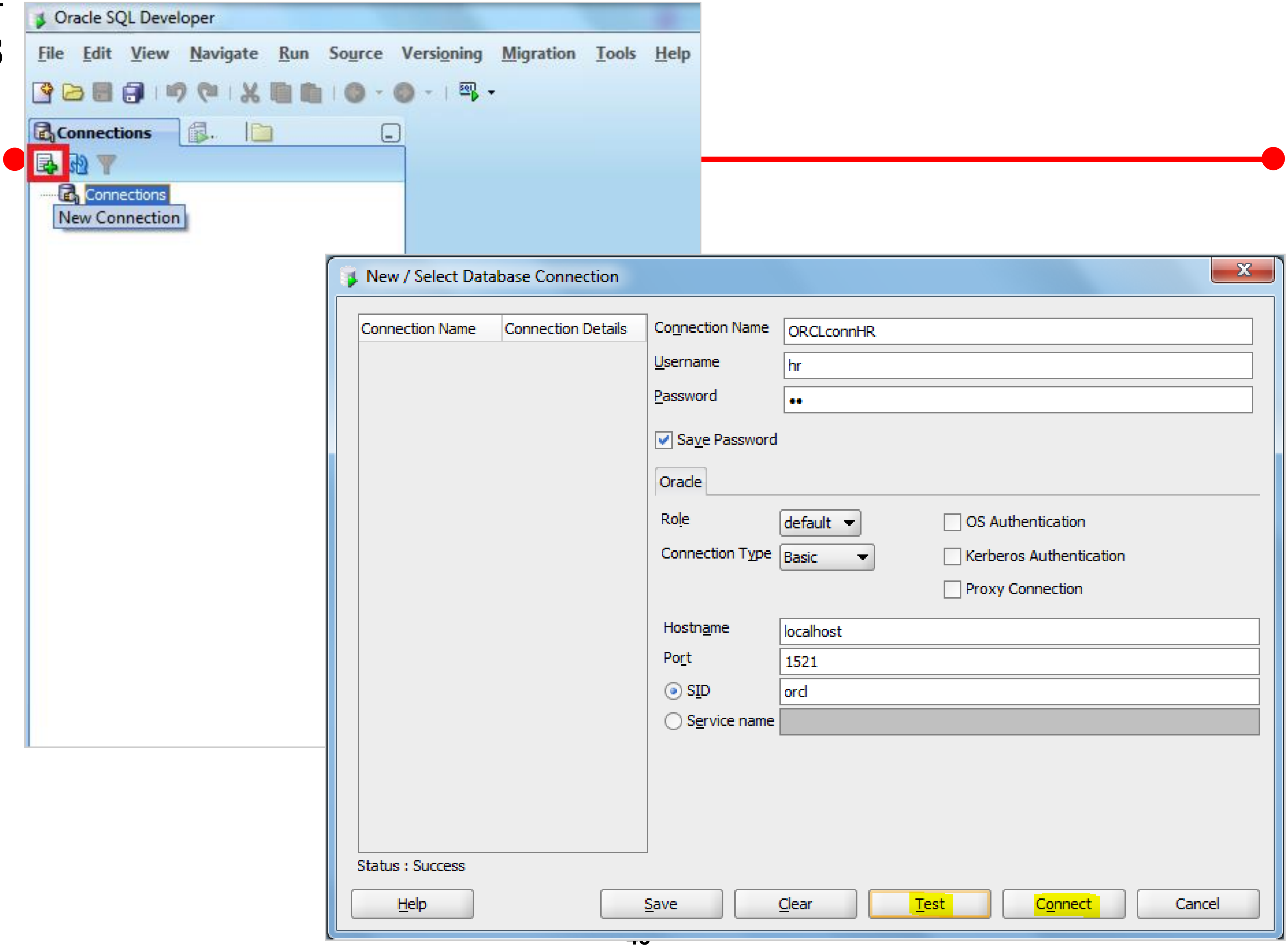
```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                 (SELECT   MIN(salary)
                  FROM     employees
                  GROUP BY department_id);
```

```
ERROR at line 4:
ORA-01427: single-row subquery returns more than
one row
```

Single-row operator with multiple-row subquery

How to correct this error ?? Change the = operator to `IN`

Practice
SQL Developer



Connection Name: ORCLconnHR

Username: hr

Password: ••

☑ Save Password

Role: default

Connection Type: Basic

□ OS Authentication

□ Kerberos Authentication

□ Proxy Connection

Hostname: localhost

Port: 1521

● SID: orcl

○ Service name:

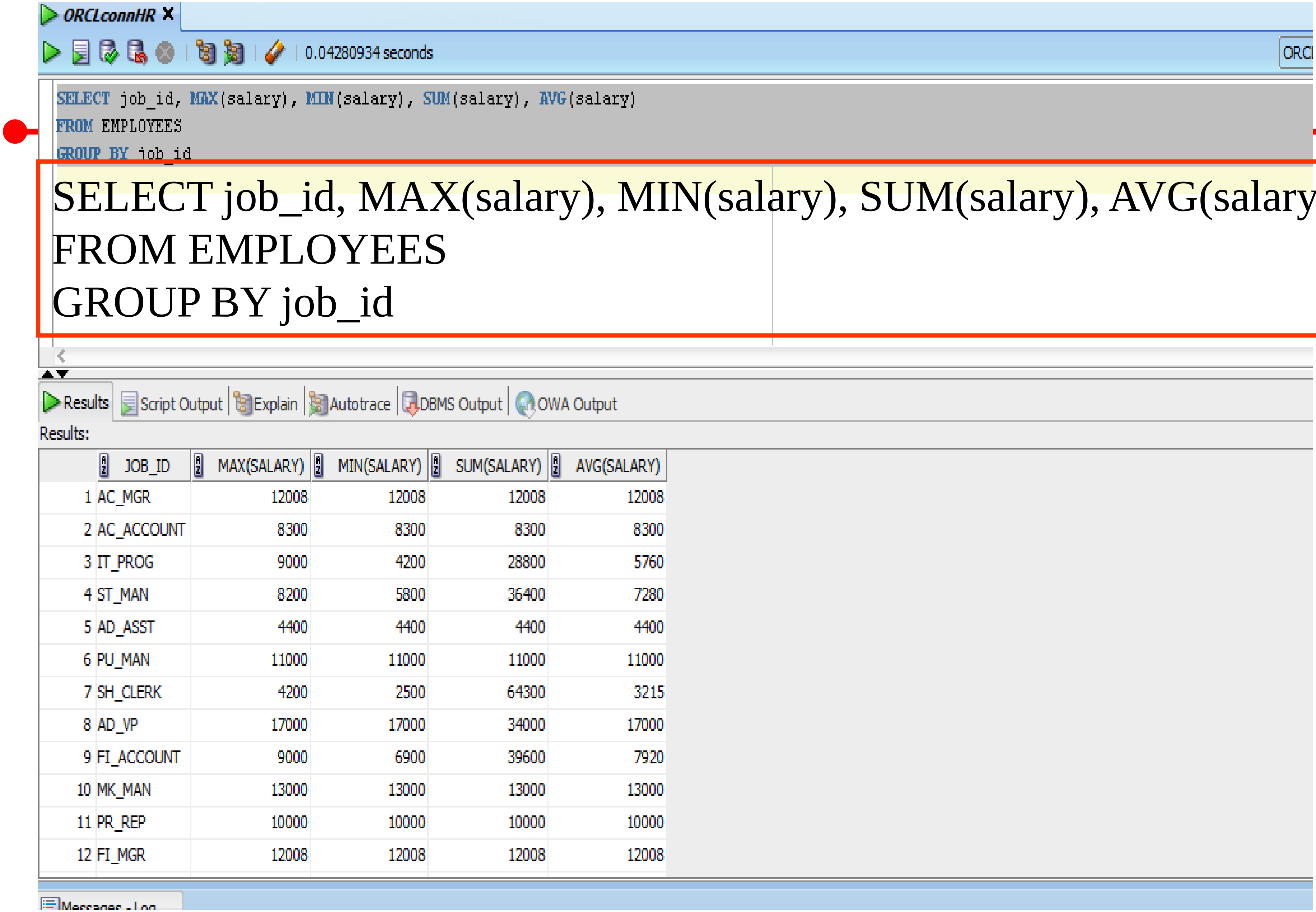
Status : Success

Writing SQL Statements

- SQL statements are not case sensitive.
- SQL statements can be on one or more lines.
- Keywords cannot be abbreviated.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.

Practice

- Display the minimum, maximum, sum, and average salary for each job type.



```
SELECT job_id, MAX(salary), MIN(salary), SUM(salary), AVG(salary)
FROM EMPLOYEES
GROUP BY job_id
```

```
SELECT job_id, MAX(salary), MIN(salary), SUM(salary), AVG(salary)
FROM EMPLOYEES
GROUP BY job_id
```

Results:

	JOB_ID	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)	AVG(SALARY)
1	AC_MGR	12008	12008	12008	12008
2	AC_ACCOUNT	8300	8300	8300	8300
3	IT_PROG	9000	4200	28800	5760
4	ST_MAN	8200	5800	36400	7280
5	AD_ASST	4400	4400	4400	4400
6	PU_MAN	11000	11000	11000	11000
7	SH_CLERK	4200	2500	64300	3215
8	AD_VP	17000	17000	34000	17000
9	FI_ACCOUNT	9000	6900	39600	7920
10	MK_MAN	13000	13000	13000	13000
11	PR_REP	10000	10000	10000	10000
12	FI_MGR	12008	12008	12008	12008

Thank You