# Functions and Aggregations

CS 341 Database Systems Lab

## One User – One Schema

*A schema is a collection of database objects.*
*A schema is owned by a database user*

## Sqlplus / as sysdba

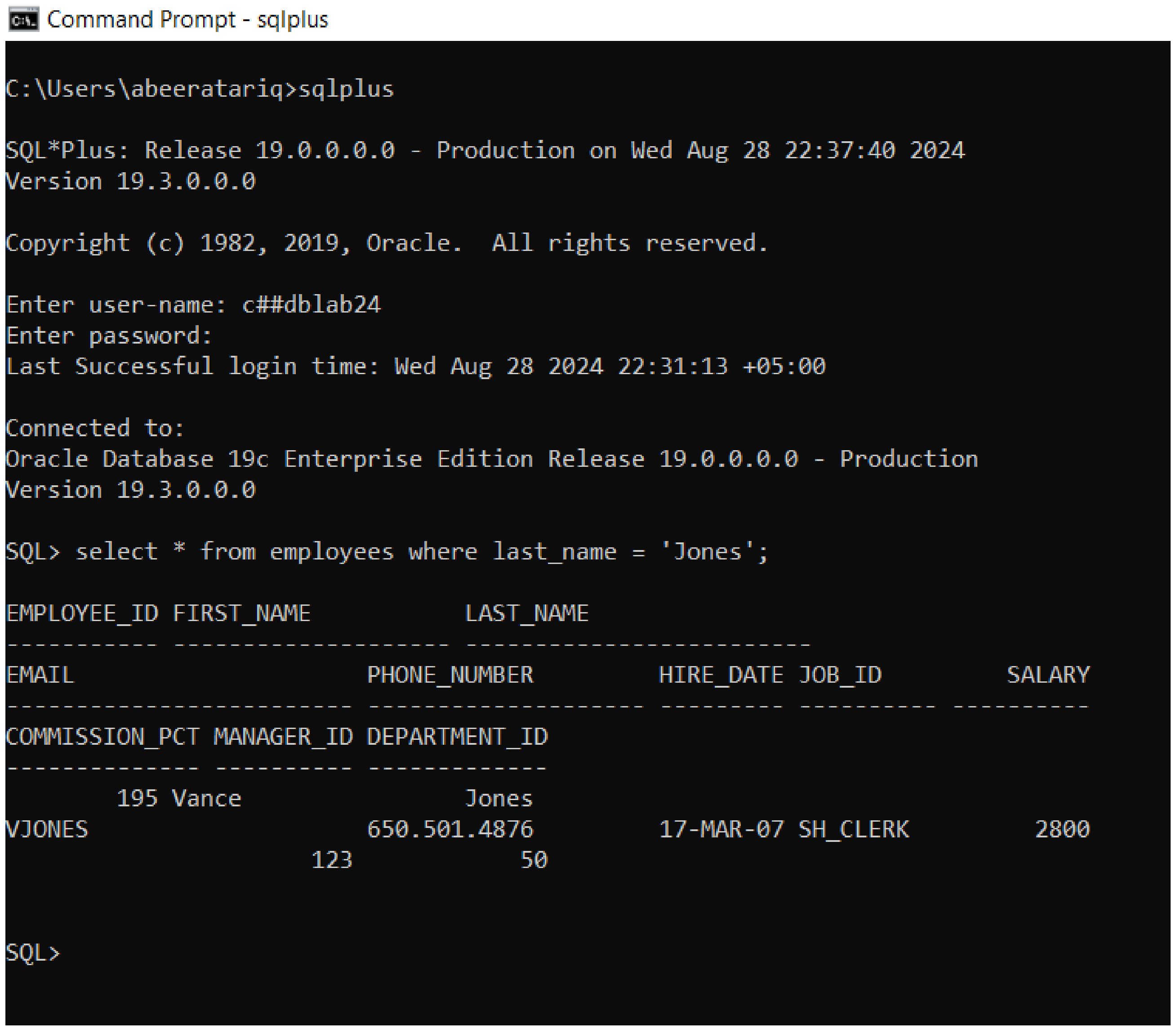
*We are now connected as the admin user of the database which has privileges to create more users.*

## Setting your user

- `SQL> CREATE USER` <u>`(any user name with prefix c##)`</u> `IDENTIFIED BY` <u>`(any password);`</u>
- e.g. create user c##myuser identified by 123; *you may create a user with your name or specific to the schema*
- `SQL> GRANT UNLIMITED TABLESPACE TO C##MYUSER;`
- `SQL> GRANT CONNECT, RESOURCE, DBA TO C##MYUSER;`

# Running Queries via Command Line

Similarly, you can use the SQLPlus Command line



```
C:\Users\abeeratariq>sqlplus

SQL*Plus: Release 19.0.0.0.0 - Production on Wed Aug 28 22:37:40 2024
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.

Enter user-name: c##dblab24
Enter password:
Last Successful login time: Wed Aug 28 2024 22:31:13 +05:00

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

SQL> select * from employees where last_name = 'Jones';

EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- -------------------------
EMAIL                     PHONE_NUMBER         HIRE_DATE JOB_ID         SALARY
------------------------- -------------------- --------- ---------- ----------
COMMISSION_PCT MANAGER_ID DEPARTMENT_ID
-------------- ---------- -------------
        195 Vance                Jones
VJONES                    650.501.4876         17-MAR-07 SH_CLERK         2800
                      123            50


SQL>
```

## Recap

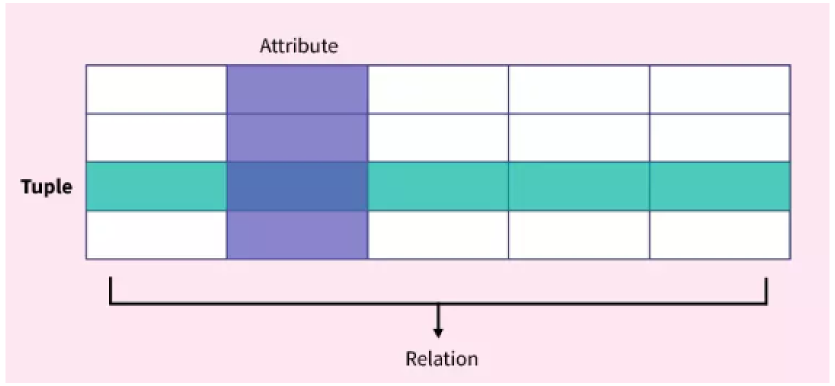
## Syntax details

- SQL commands are **case insensitive -** SELECT = Select, Product = product
- Values are not, ‘Seattle’ not equal to ‘seattle’
- Use single quotes for constants : ‘abc’ - best practice (versus “abc” with mixed support)
- To say “don’t know the value”/ missing values we use **NULL** E.g., Student GPA in 1st quarter = NULL, not zero
- Free-form language – no special indentation is required but consistent formatting style is recommended for easy maintenance of SQL queries.

## SELECT

**SELECT c1, c2 FROM t;**
Query data in columns c1, c2 from a table

**SELECT * FROM t;**
Query all rows and columns from a table



**Projection** is the operation of producing an output table with tuples that have a subset of their prior attributes

## WHERE

**Selection** is the operation of filtering a relation's tuples on some condition

```
SELECT c1, c2 FROM t
WHERE condition;
```

Query data and filter rows with a condition

- Comparison Operators →

| **Operator** | **Description** |
|---|---|
| **=** | Equal |
| **>** | Greater than |
| **<** | Less than |
| **>=** | Greater than or equal |
| **<=** | Less than or equal |
| **<>** | Not equal. **Note:** In some versions of SQL this operator may be written as != |
| **BETWEEN** | Between a certain range |
| **LIKE** | Search for a pattern |
| **IN** | To specify multiple possible values for a column |

## AND, OR, NOT

- Used in WHERE clause
- The AND operator : all the conditions are TRUE.
- The OR operator: if any of the conditions is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.
- Take care of brackets when defining multiple conditions

## IN operator

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```

Query rows in a list

- The `IN` operator allows you to specify multiple values in a `WHERE` clause.
- The `IN` operator is a shorthand for multiple `OR` conditions.

## Distinct

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```

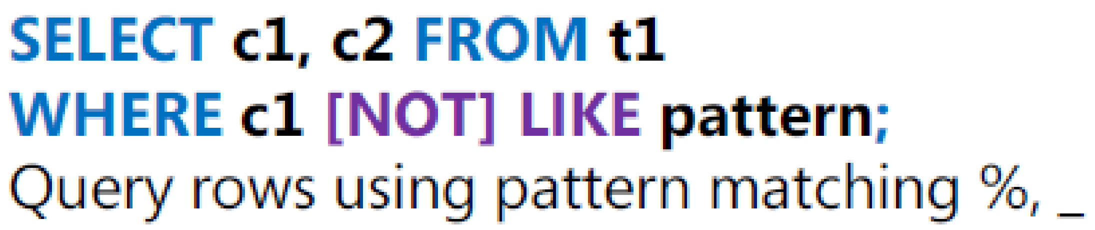
Query distinct rows from a table

## Between

```
SELECT c1, c2 FROM t
WHERE  c1 BETWEEN low AND high;
```
Query rows between two values

- Selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.
- Similar to querying using a combination of >= and <=

# LIKE (Wildcards)



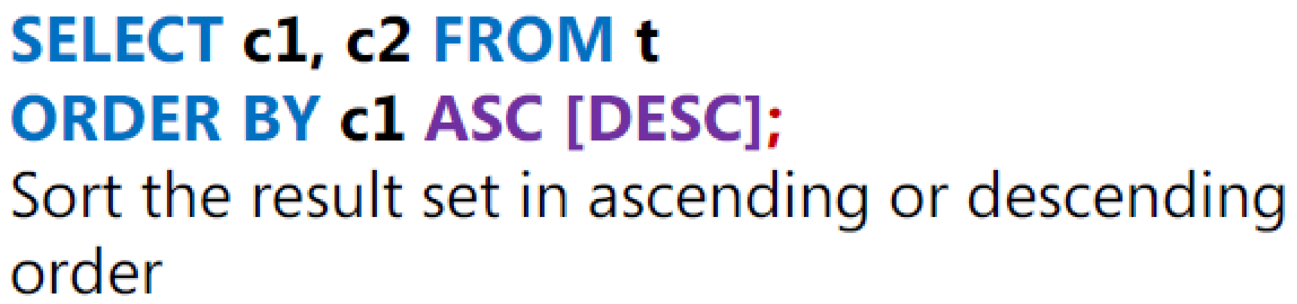
```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```

Query rows using pattern matching %, _

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

| LIKE Operator | Description |
|---|---|
| WHERE CustomerName LIKE 'a%' | Finds any values that start with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that end with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE CustomerName LIKE '_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE 'a_%' | Finds any values that start with "a" and are at least 2 characters in length |
| WHERE CustomerName LIKE 'a__%' | Finds any values that start with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

## Order by



```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```

Sort the result set in ascending or descending order

- Default ascending order
- DESC – descending
- ASC - ascending

## Aliases

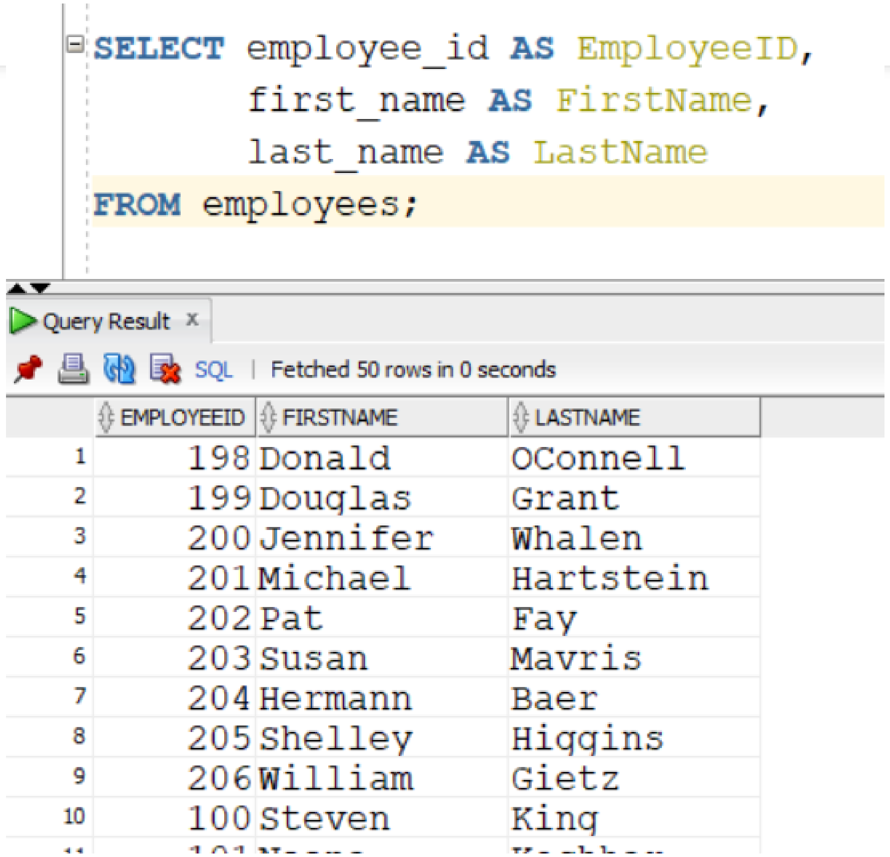
### Column Name

```
SELECT column_name AS alias_name
FROM table_name;
```

### Table Name

-
  ```
  SELECT column_name(s)
  FROM table_name AS alias_name;
  ```

# Example of Column Aliases



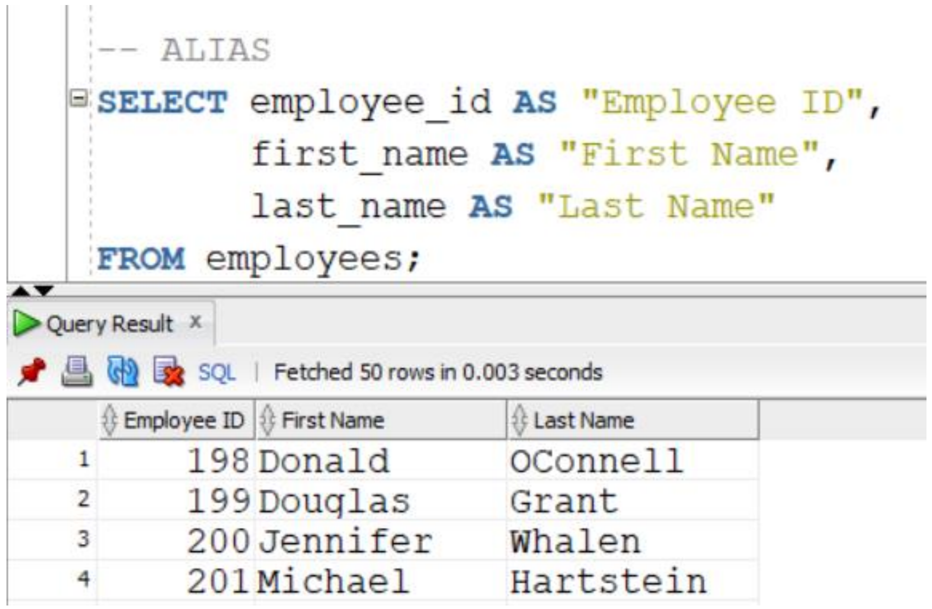
```sql
SELECT employee_id AS EmployeeID,
       first_name AS FirstName,
       last_name AS LastName
FROM employees;
```

Query Result ×

SQL | Fetched 50 rows in 0 seconds

```
   EMPLOYEEID FIRSTNAME  LASTNAME
1         198 Donald     OConnell
2         199 Douglas    Grant
3         200 Jennifer   Whalen
4         201 Michael    Hartstein
5         202 Pat        Fay
6         203 Susan      Mavris
7         204 Hermann    Baer
8         205 Shelley    Higgins
9         206 William    Gietz
10        100 Steven     King
```

- No spaces in the alias
- If you need spaces then enclose in double quotes



```sql
-- ALIAS
SELECT employee_id AS "Employee ID",
       first_name AS "First Name",
       last_name AS "Last Name"
FROM employees;
```

Query Result ×

SQL | Fetched 50 rows in 0.003 seconds

```
   Employee ID First Name Last Name
1          198 Donald     OConnell
2          199 Douglas    Grant
3          200 Jennifer   Whalen
4          201 Michael    Hartstein
```

## Aggregation

Lab02A

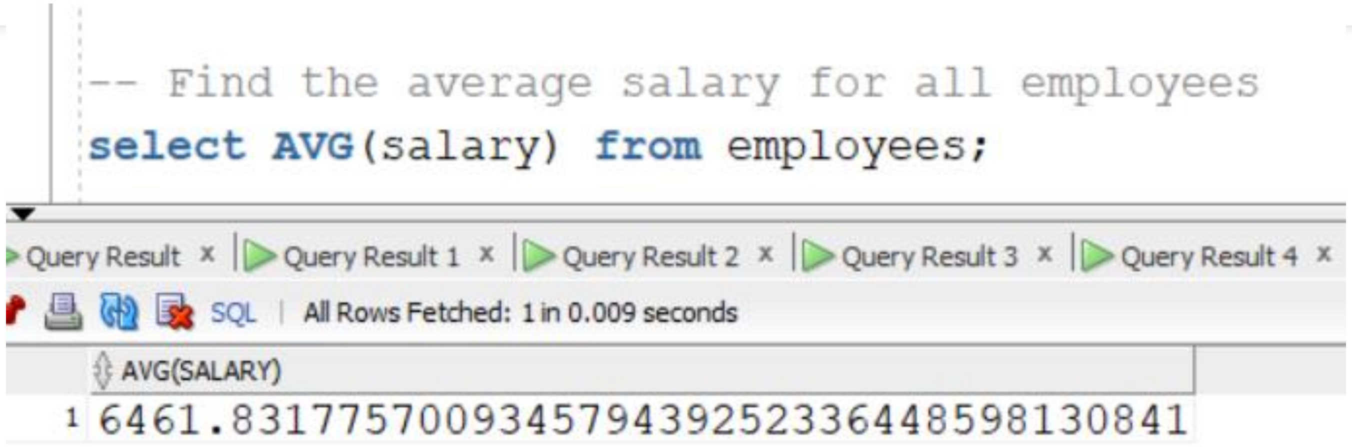
## Aggregate Functions

- **AVG()** – returns the average of a set.
- **COUNT()** – returns the number of items in a set.
- **MAX()** – returns the maximum value in a set.
- **MIN()** – returns the minimum value in a set
- **SUM()** – returns the sum of all or distinct values in a set

SQL aggregate functions ignore null values.

If using COUNT(*), it counts all rows including those with NULLs. However, COUNT(column_name) ignores NULLs in that column.

# Example

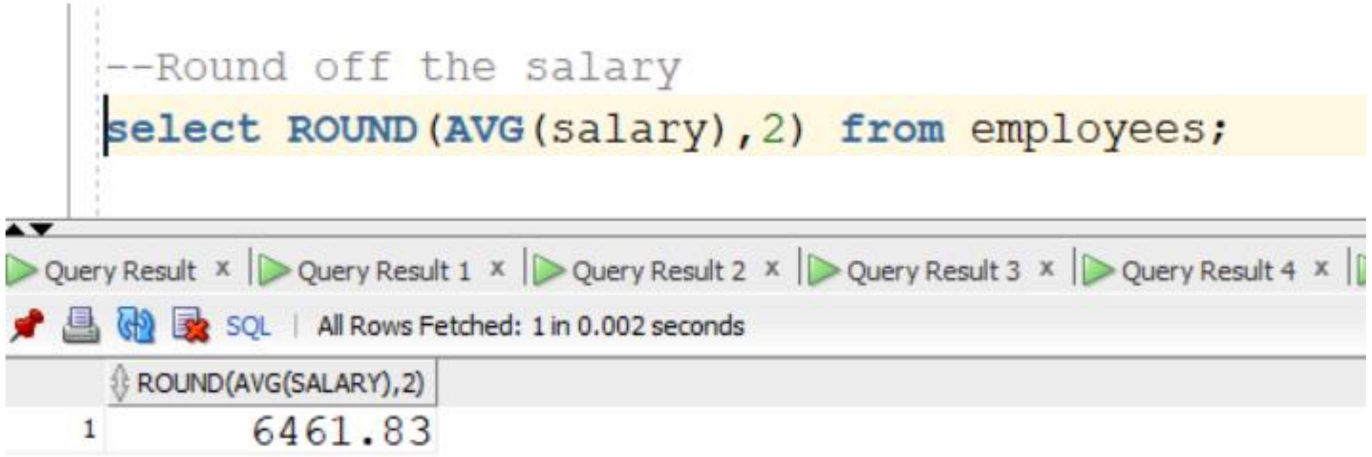


```
-- Find the average salary for all employees
select AVG(salary) from employees;
```

Query Result × | Query Result 1 × | Query Result 2 × | Query Result 3 × | Query Result 4 ×

SQL | All Rows Fetched: 1 in 0.009 seconds

```
   AVG(SALARY)
1  6461.831775700934579439252336448598130841
```



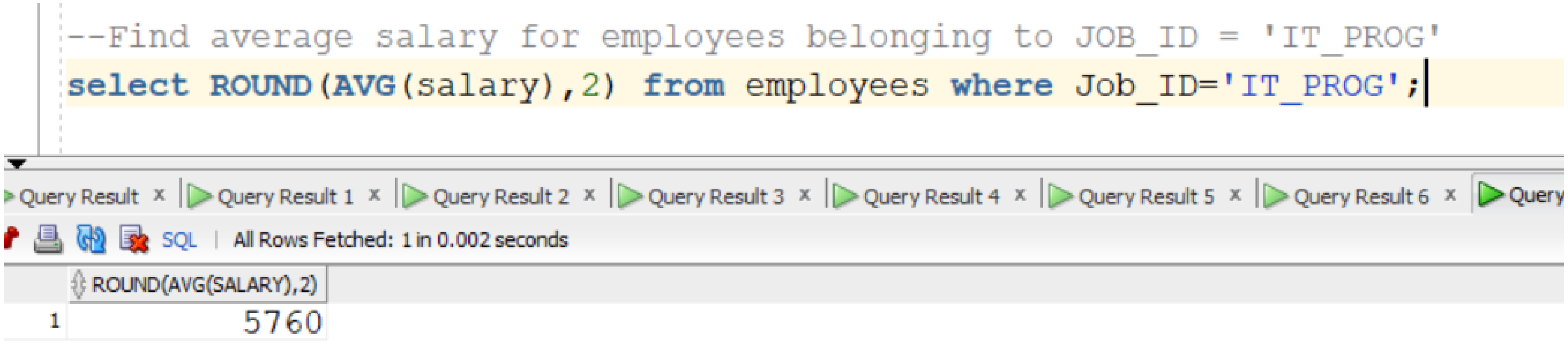
```
--Round off the salary
select ROUND(AVG(salary),2) from employees;
```

Query Result × | Query Result 1 × | Query Result 2 × | Query Result 3 × | Query Result 4 ×

SQL | All Rows Fetched: 1 in 0.002 seconds

```
   ROUND(AVG(SALARY),2)
1               6461.83
```

## Avg of specific Job_id

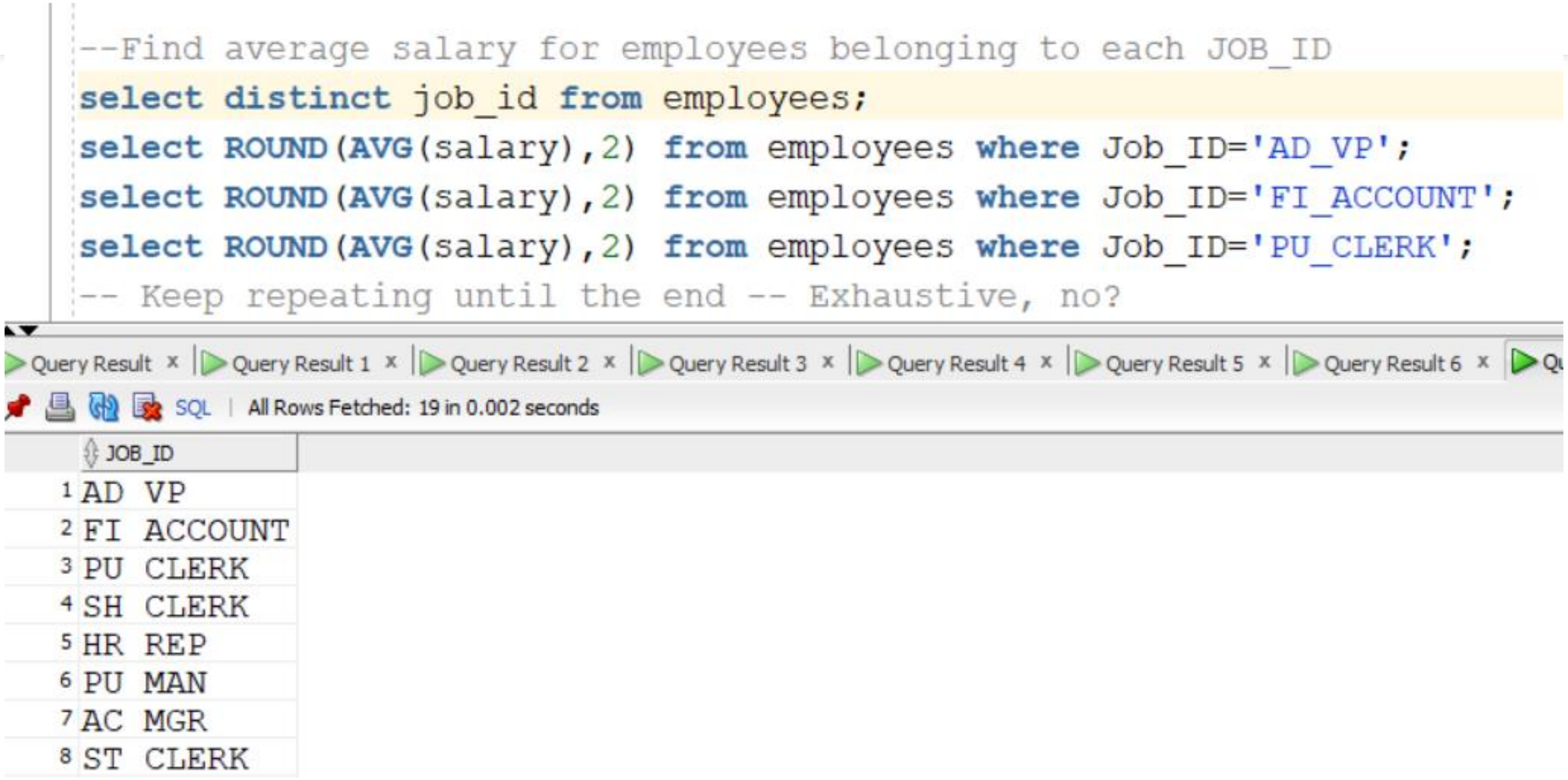


```
--Find average salary for employees belonging to JOB_ID = 'IT_PROG'
select ROUND(AVG(salary),2) from employees where Job_ID='IT_PROG';
```

```
All Rows Fetched: 1 in 0.002 seconds

   ROUND(AVG(SALARY),2)
1                  5760
```

## Avg of all Job_ids

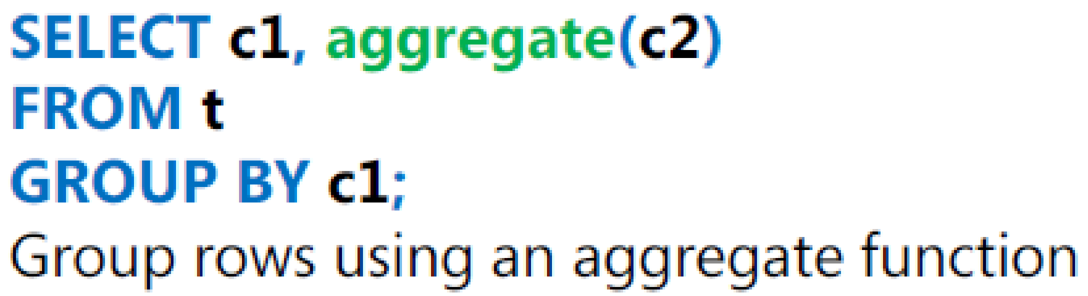


```
--Find average salary for employees belonging to each JOB_ID
select distinct job_id from employees;
select ROUND(AVG(salary),2) from employees where Job_ID='AD_VP';
select ROUND(AVG(salary),2) from employees where Job_ID='FI_ACCOUNT';
select ROUND(AVG(salary),2) from employees where Job_ID='PU_CLERK';
-- Keep repeating until the end -- Exhaustive, no?
```

```
Query Result × | Query Result 1 × | Query Result 2 × | Query Result 3 × | Query Result 4 × | Query Result 5 × | Query Result 6 × | Qu
SQL | All Rows Fetched: 19 in 0.002 seconds

  JOB_ID
1 AD_VP
2 FI_ACCOUNT
3 PU_CLERK
4 SH_CLERK
5 HR_REP
6 PU_MAN
7 AC_MGR
8 ST_CLERK
```

## Group by

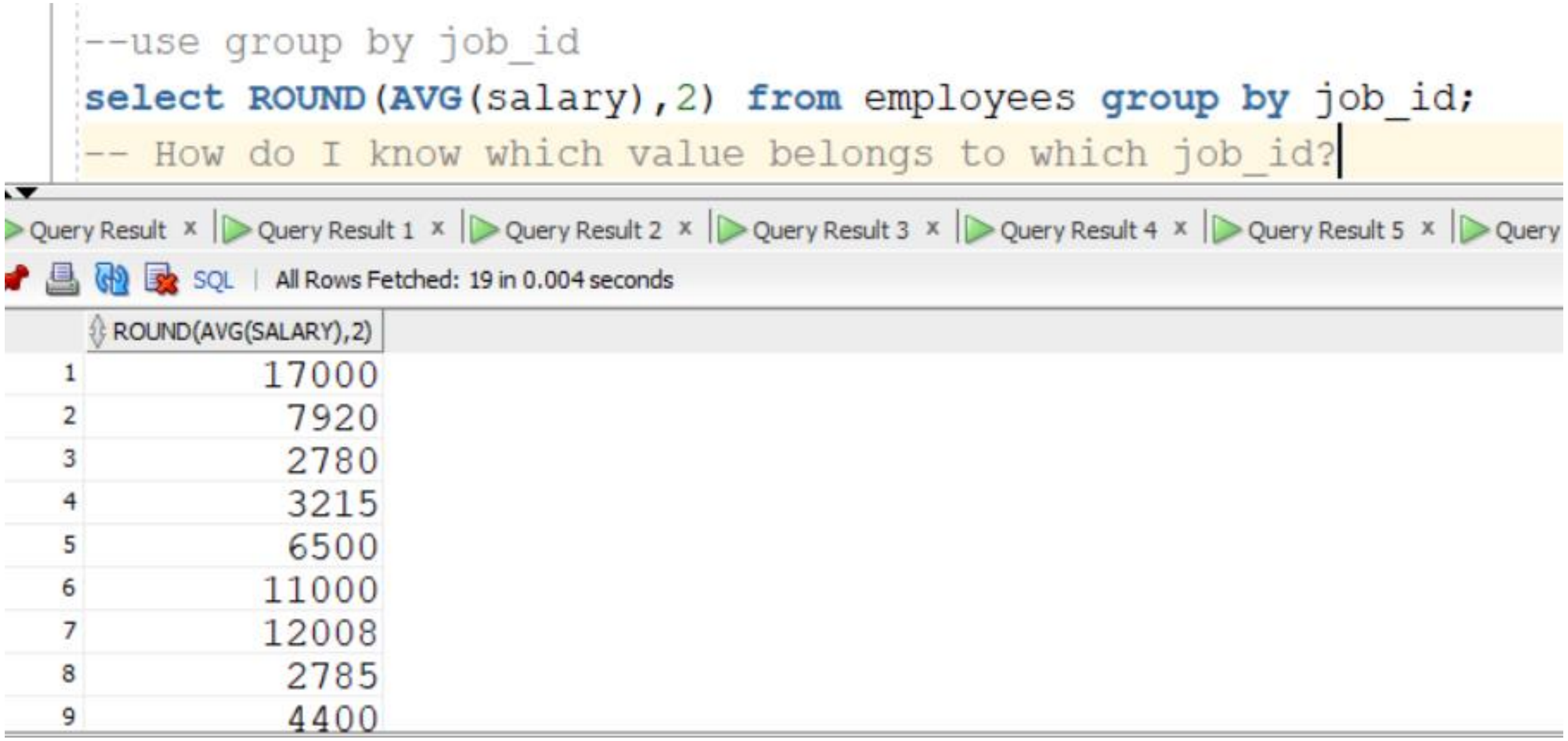


```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```

Group rows using an aggregate function

- Allows you to group rows based on values of one or more columns. It returns one row for each group.

## Group by Job_id

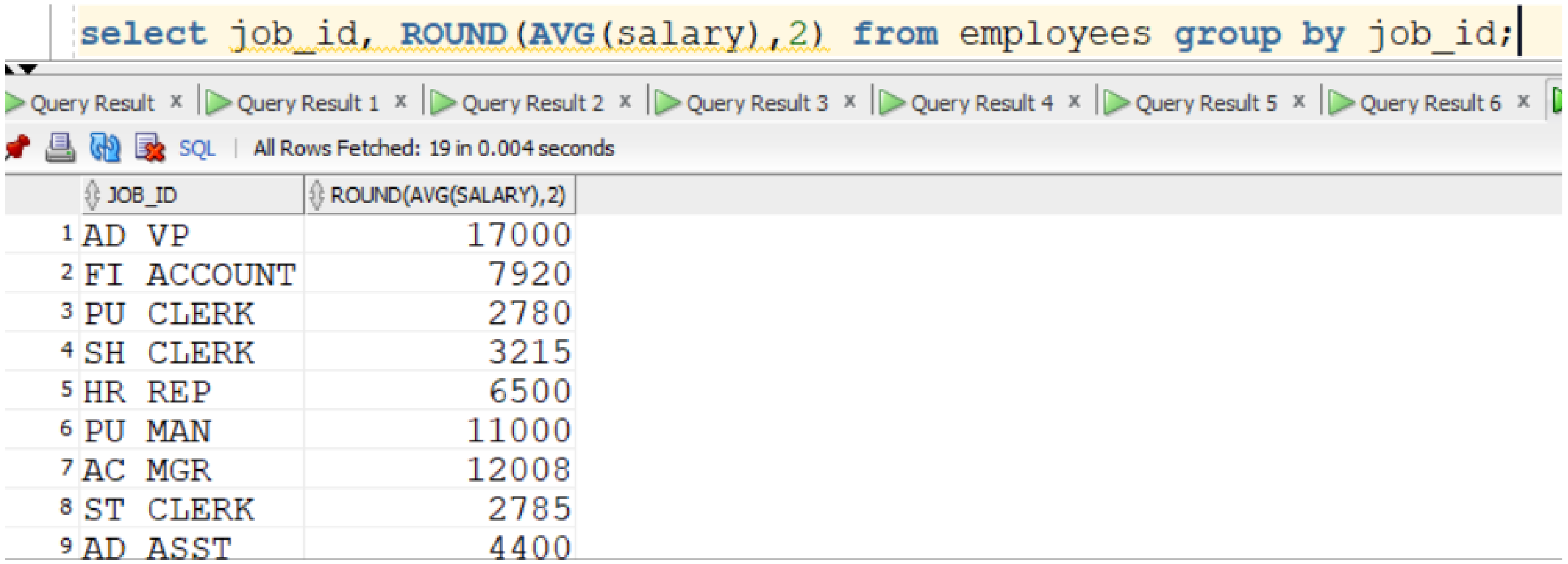


```
--use group by job_id
select ROUND(AVG(salary),2) from employees group by job_id;
-- How do I know which value belongs to which job_id?
```

```
All Rows Fetched: 19 in 0.004 seconds

   ROUND(AVG(SALARY),2)
1                 17000
2                  7920
3                  2780
4                  3215
5                  6500
6                 11000
7                 12008
8                  2785
9                  4400
```

## Project Job_id in the results

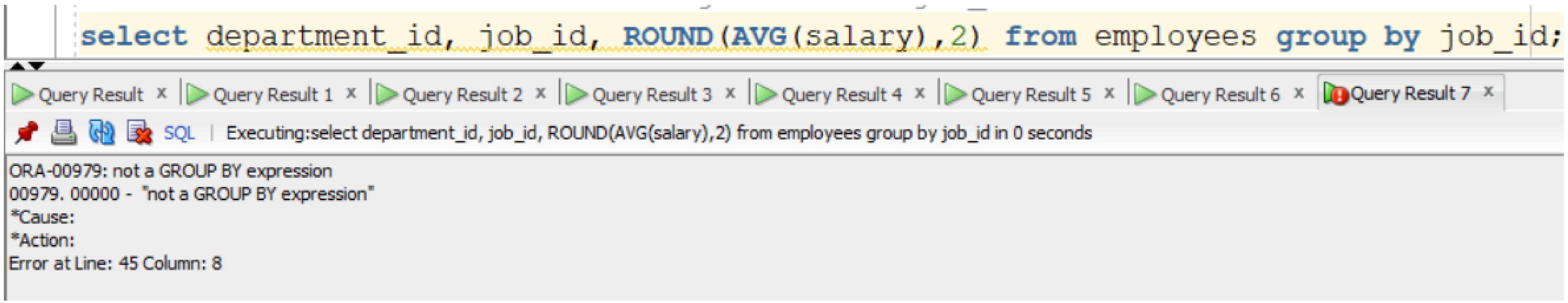


```
select job_id, ROUND(AVG(salary),2) from employees group by job_id;
```

All Rows Fetched: 19 in 0.004 seconds

| | JOB_ID | ROUND(AVG(SALARY),2) |
|---|---|---|
| 1 | AD VP | 17000 |
| 2 | FI ACCOUNT | 7920 |
| 3 | PU CLERK | 2780 |
| 4 | SH CLERK | 3215 |
| 5 | HR REP | 6500 |
| 6 | PU MAN | 11000 |
| 7 | AC MGR | 12008 |
| 8 | ST CLERK | 2785 |
| 9 | AD ASST | 4400 |

## Adding more columns in SELECT



```
select department_id, job_id, ROUND(AVG(salary),2) from employees group by job_id;
```

Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query Result 5 | Query Result 6 | Query Result 7

SQL | Executing:select department_id, job_id, ROUND(AVG(salary),2) from employees group by job_id in 0 seconds

```
ORA-00979: not a GROUP BY expression
00979. 00000 -  "not a GROUP BY expression"
*Cause:
*Action:
Error at Line: 45 Column: 8
```

# Group by - Example

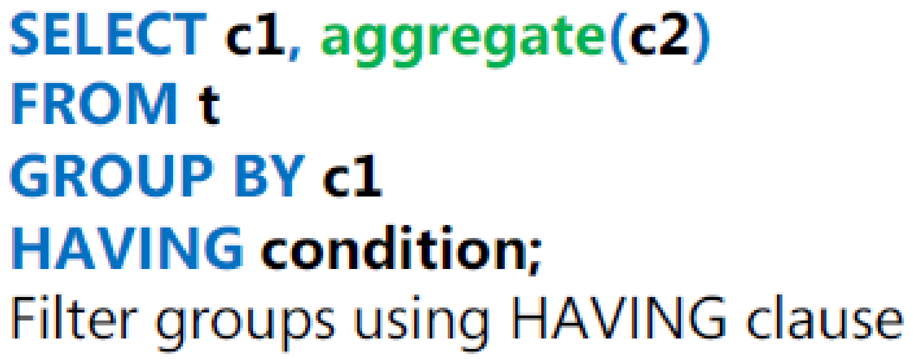
| StudentID | Name | Section | Marks |
|---|---|---|---|
| 1 | Alpha | A | 60 |
| 2 | Bravo | B | 88 |
| 3 | Charlie | A | 70 |
| 4 | Danny | B | 25 |
| 5 | Eric | B | 50 |

- Find how many students are there in **each section**

| Section | Count(*) |
|---|---|
| A | 2 |
| B | 3 |

- SELECT * FROM Students GROUP BY Section ✗
- SELECT Count(*) FROM Students GROUP BY Section
- SELECT Section, Count(*) FROM Students GROUP BY Section
- SELECT Name, Section, Count(*) FROM Students GROUP BY Section ✗

## Having



```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```

Filter groups using HAVING clause

- Specify a condition for groups, you use the HAVING clause.

The WHERE clause applies the condition to individual rows **before the rows are summarized into groups** by the GROUP BY clause. However, the HAVING clause applies the condition to the groups **after the rows are grouped into groups.**

## Having - Example

| StudentID | Name | Section | Marks |
|---|---|---|---|
| 1 | Alpha | A | 60 |
| 2 | Bravo | B | 88 |
| 3 | Charlie | A | 70 |
| 4 | Danny | B | 25 |
| 5 | Eric | B | 50 |

- SELECT Section, Count(*) As NumberOfStudents
  FROM Students
  GROUP BY Section

| Section | NumberOfStudents |
|---|---|
| A | 2 |
| B | 3 |

- SELECT Section, Count(*) As NumberOfStudents FROM Students
  GROUP BY Section
  HAVING Count(*) < 3

OR

- SELECT Section, Count(*) As NumberOfStudents FROM Students
  GROUP BY Section
  HAVING NumberOfStudents < 3

| Section | NumberOfStudents |
|---|---|
| A | 2 |

# Functions

Lab02B

## Numeric Functions

| **Functions** | **Description** |
|---|---|
| **FLOOR** | Returns the largest whole number equal to or less than a specified number. |
| **MOD** | Returns the modulus of a number. |
| **POWER** | Returns m_value raised to the n_value power |
| **REMAINDER** | Returns the remainder after one numeric expression is divided by another. |
| **ROUND** | Rounds the number based on the standard rounding rules (e.g., 2.5 rounds to 3, while 2.4 rounds to 2). |
| **SQRT** | Computes the square root of an expression. |
| **TRUNC** | Truncates a number to a specified number of decimal places. |

## String Functions

## Upper, Lower, Initcap

- UPPER(expression) → All to uppercase
- LOWER(expression) → All to lowercase
- INITCAP(expression) → Capitalize first character of each word and rest to lower case.

## CONCAT, ||

- Concatenate 2 strings and return the combined string.
- Equivalent to using **||** operator. (Oracle)
- Other DBMS accept **+** operator for string concatenation

## TRIM, LTRIM, RTRIM

Remove the space character or other specified characters from the

- TRIM (trim_source, [set]) → **start or end**
- LTRIM (trim_source, [set]) →**left** end
- RTRIM (trim_source, [set]) → **right** end

of a string.

By default, the set will be a single space.

## LPAD, RPAD

- LPAD (source_string, target_length [,pad_string]) → Return a string that is **left-padded** with the specified characters to a certain length.
- RPAD (source_string, target_length [,pad_string]) → Return a string that is **right-padded** with the specified characters to a certain length.
- Default pad_string is a single space

## REPLACE

Replaces all occurrences of a specified substring in a string with another.

- REPLACE (expression, string_pattern, string_replacement)

## SUBSTR

Extract a substring from a string.

- SUBSTR (str, start_position [, substring_length, [, occurrence ]])

## INSTR

Search for a substring and return the location of the substring in a string

- INSTR(string , substring [, start_position [, occurrence]])

## LENGTH

Return the number of characters (or length) of a specified string

- LENGTH(string_expression);

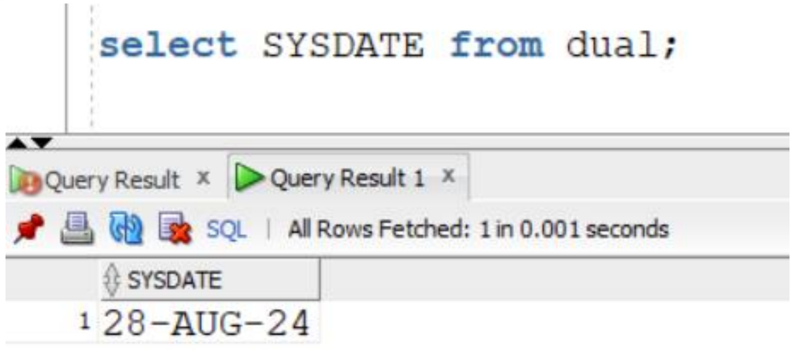
## DUAL TABLE

- The DUAL table is owned by the user SYS and can be accessed by users.
- Contains one column, DUMMY, and one row with the value X.
- Useful when you want to return a value once only – for instance, the value of a constant, pseudo-column, or expression that is not derived from a table with user data.

## Date Functions

- **SYSDATE** - Return the current system date and time of the operating system where the Oracle Database resides
- **ADD_MONTHS** - Add a number of months (n) to a date and return the same day which is n of months away.
- **MONTHS_BETWEEN** - Return the number of months between two dates.

## Example

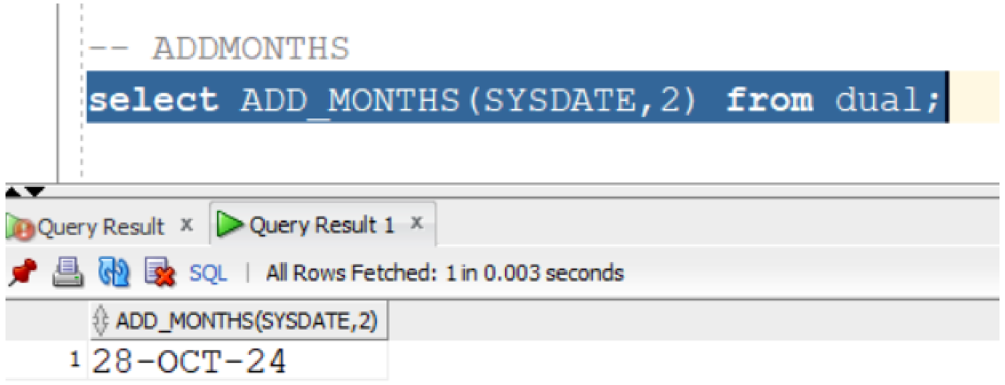


```
select SYSDATE from dual;
```

Query Result × | Query Result 1 ×

SQL | All Rows Fetched: 1 in 0.001 seconds

```
   SYSDATE
1  28-AUG-24
```

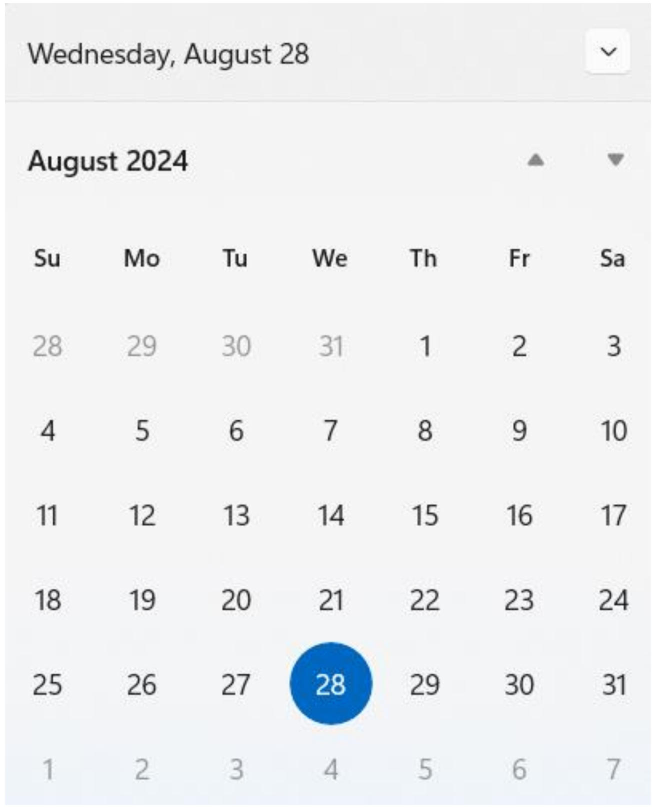


```
-- ADDMONTHS
select ADD_MONTHS(SYSDATE,2) from dual;
```

Query Result × | Query Result 1 ×

SQL | All Rows Fetched: 1 in 0.003 seconds

```
   ADD_MONTHS(SYSDATE,2)
1  28-OCT-24
```

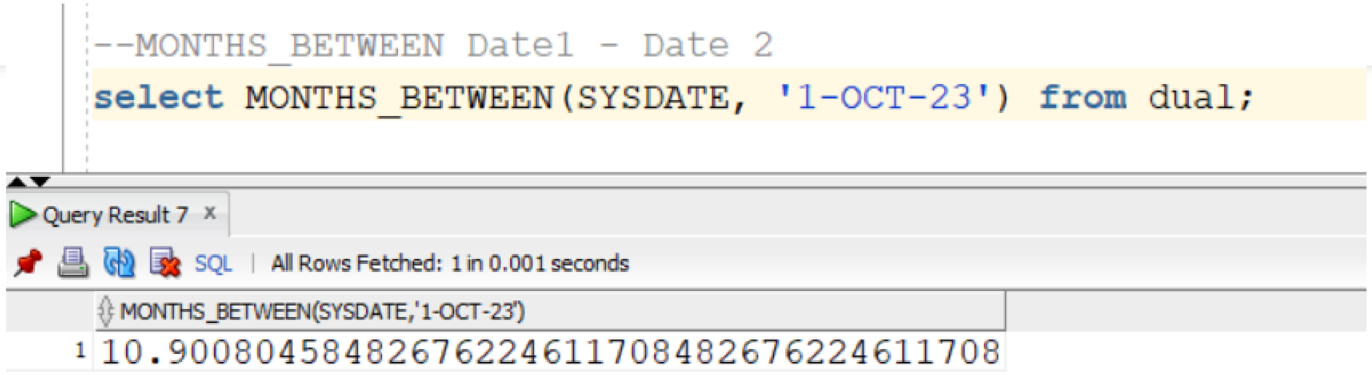


Wednesday, August 28

August 2024

| Su | Mo | Tu | We | Th | Fr | Sa |
|---|---|---|---|---|---|---|
| 28 | 29 | 30 | 31 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

## Example

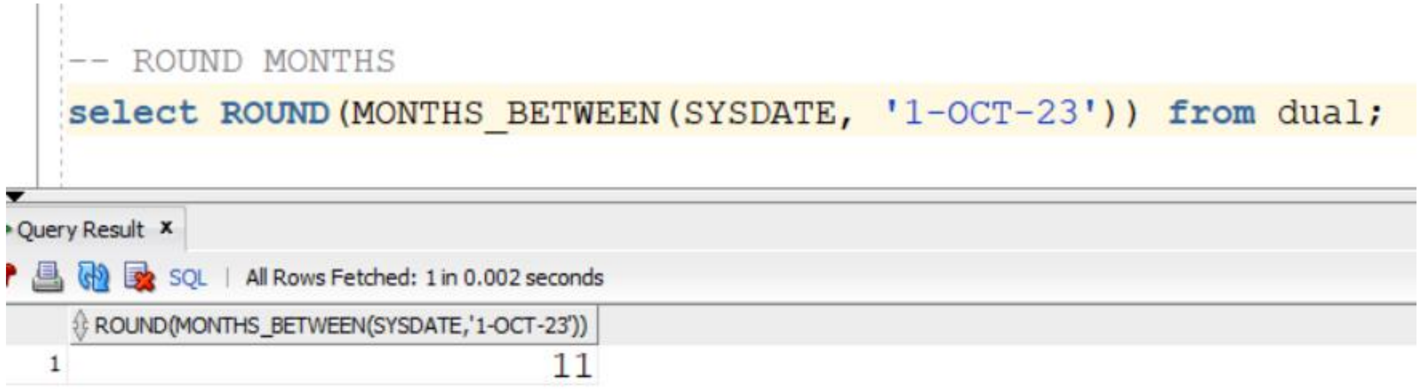


```
--MONTHS_BETWEEN Date1 - Date 2
select MONTHS_BETWEEN(SYSDATE, '1-OCT-23') from dual;
```

Query Result 7

All Rows Fetched: 1 in 0.001 seconds

```
  MONTHS_BETWEEN(SYSDATE,'1-OCT-23')
1 10.90080458482676224611708482676224611708
```



```
-- ROUND MONTHS
select ROUND(MONTHS_BETWEEN(SYSDATE, '1-OCT-23')) from dual;
```

Query Result

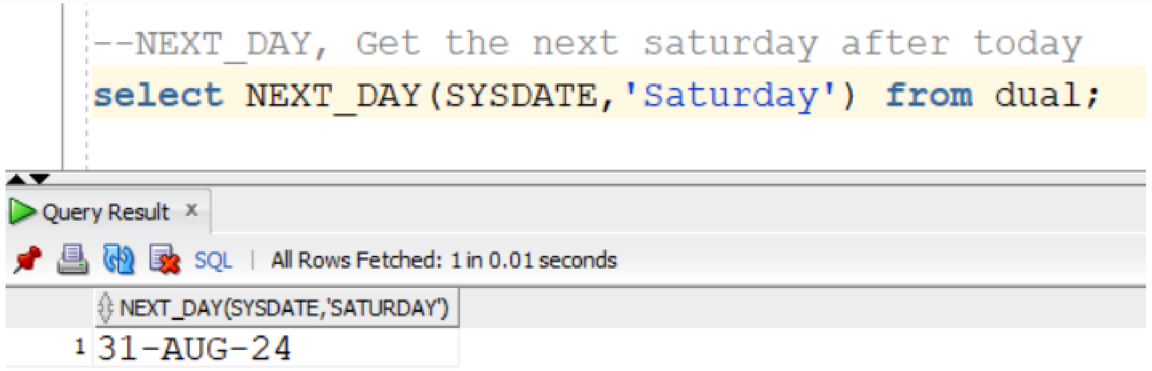
All Rows Fetched: 1 in 0.002 seconds

```
  ROUND(MONTHS_BETWEEN(SYSDATE,'1-OCT-23'))
1                                        11
```

## Date Functions

- **NEXT_DAY** - Get the first weekday that is later than a specified date.
- **LAST_DAY** - Gets the last day of the month of a specified date.

## Example

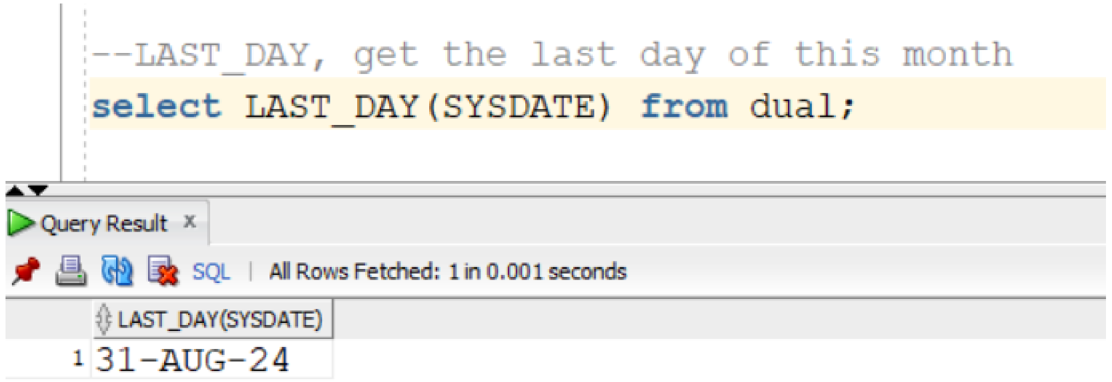


```
--NEXT_DAY, Get the next saturday after today
select NEXT_DAY(SYSDATE,'Saturday') from dual;
```

Query Result ×

SQL | All Rows Fetched: 1 in 0.01 seconds

```
  NEXT_DAY(SYSDATE,'SATURDAY')
1 31-AUG-24
```



```
--LAST_DAY, get the last day of this month
select LAST_DAY(SYSDATE) from dual;
```

Query Result ×

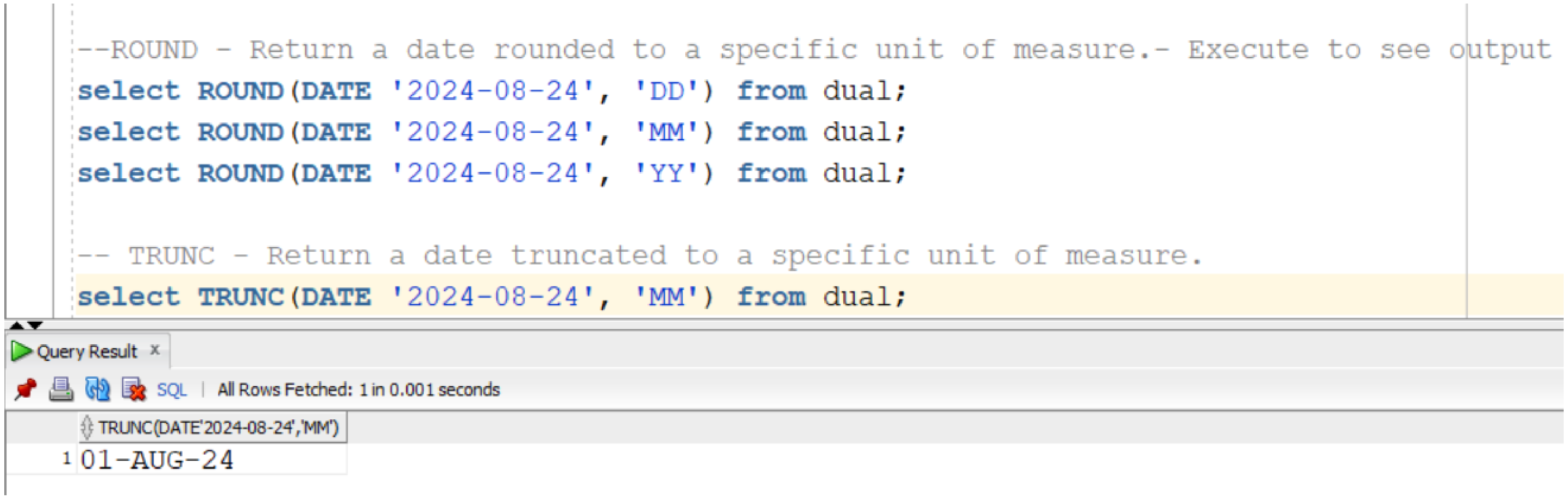
SQL | All Rows Fetched: 1 in 0.001 seconds

```
  LAST_DAY(SYSDATE)
1 31-AUG-24
```

## Date Functions

- **ROUND** - Return a date rounded to a specific unit of measure.
- **TRUNC** - Return a date truncated to a specific unit of measure.

## Example



```
--ROUND - Return a date rounded to a specific unit of measure.- Execute to see output
select ROUND(DATE '2024-08-24', 'DD') from dual;
select ROUND(DATE '2024-08-24', 'MM') from dual;
select ROUND(DATE '2024-08-24', 'YY') from dual;

-- TRUNC - Return a date truncated to a specific unit of measure.
select TRUNC(DATE '2024-08-24', 'MM') from dual;
```

Query Result

SQL | All Rows Fetched: 1 in 0.001 seconds

```
  TRUNC(DATE'2024-08-24','MM')
1 01-AUG-24
```