Islamic University Of Gaza

Faculty of Engineering

Computer Engineering Department

Database Lab (ECOM 4113)

Lab 4

SQL

Functions

(Single-Row, Aggregate)

Eng. Ibraheem Lubbad

Part one: Single-Row Functions:

Operators in that they manipulate data items and return a result

- **Character Functions**

  - **Case-conversion functions:**

    LOWER, UPPER, INITCAP

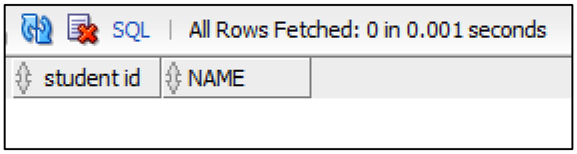
Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

Example: Find the id and student name for all student whose name contains the begin with “s”.

Use Conversion function

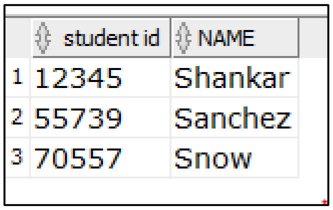
```
select ID " student id" ,  NAME
from student
where name LIKE 's%';
```

If you don’t care about capital later will not get any records



```
SQL | All Rows Fetched: 0 in 0.001 seconds
student id  NAME
```

```
select ID " student id" ,  NAME
from student
where lower(name) LIKE 's%';
```



	student id	NAME
1	12345	Shankar
2	55739	Sanchez
3	70557	Snow

- **Character-manipulation functions:**

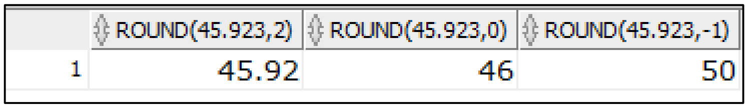
CONCAT, SUBSTR, LENGTH, INSTR, (LPAD | RPAD), TRIM, REPLACE

Function	**Result**
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR ('HelloWorld',’o’)	5
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
REPLACE ('JACK and JUE','J','BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

- **Number Functions**:
  - ROUND: Rounds value to a specified decimal
  - TRUNC: Truncates value to a specified decimal
  - MOD: Returns remainder of division

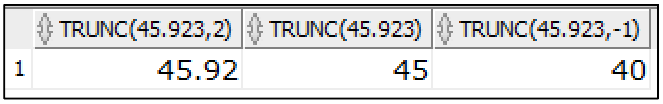
Function	**Result**
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
MOD(1600, 300)	100

```
SELECT ROUND(45.923,2),ROUND(45.923,0),ROUND(45.923,-1)
FROM DUAL
```



```
   ROUND(45.923,2)  ROUND(45.923,0)  ROUND(45.923,-1)
1            45.92               46                50
```

```
SELECT TRUNC(45.923,2),TRUNC(45.923,0),TRUNC(45.923,-1)
FROM DUAL
```



```
   TRUNC(45.923,2)  TRUNC(45.923)  TRUNC(45.923,-1)
1            45.92             45                40
```

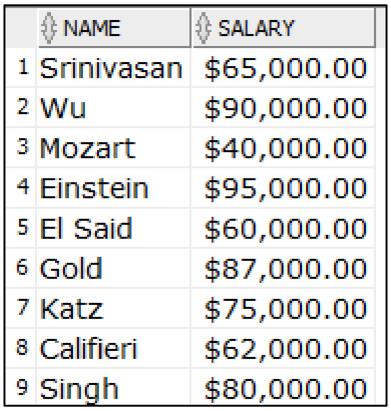
- **TO_CHAR Function with Numbers:**

These are some of the format elements that you can use with the TO_CHAR function to display a number value as a character

Element	Result
9	Represents a number
0	Forces a zero to be displayed
$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a comma as a thousand indicator

Example: Display Salary as the following pattern $##,###.00

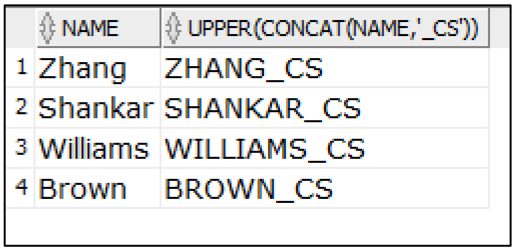
```
SELECT NAME, TO_CHAR(SALARY, '$99,999.00') SALARY
FROM INSTRUCTOR;
```



	NAME	SALARY
1	Srinivasan	$65,000.00
2	Wu	$90,000.00
3	Mozart	$40,000.00
4	Einstein	$95,000.00
5	El Said	$60,000.00
6	Gold	$87,000.00
7	Katz	$75,000.00
8	Califieri	$62,000.00
9	Singh	$80,000.00

- **Nesting Functions:**
  - Single-row functions can be nested to any level.
  - Nested functions are evaluated from the deepest level to the least deep level

```
SELECT NAME,  UPPER(CONCAT(NAME,'_CS'))
FROM STUDENT
where DEPT_NAME='Comp. Sci.’;
```



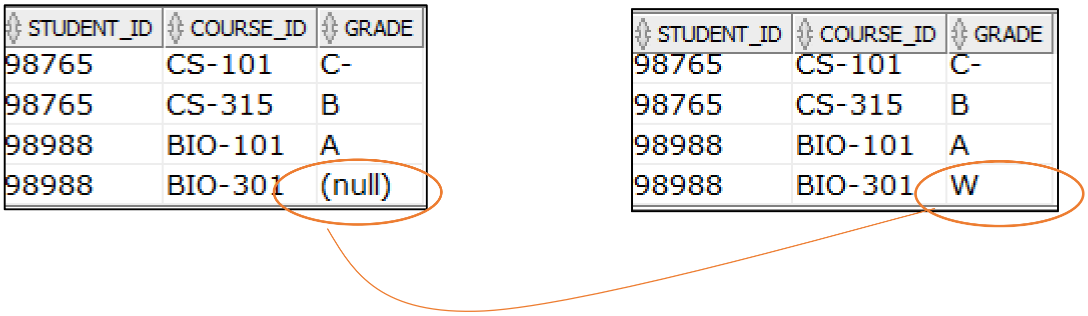
	NAME	UPPER(CONCAT(NAME,'_CS'))
1	Zhang	ZHANG_CS
2	Shankar	SHANKAR_CS
3	Williams	WILLIAMS_CS
4	Brown	BROWN_CS

General Functions:

The following functions work with any data type and pertain to using nulls:

NVL (expr1, expr2): Converts a null value to an actual value.

```
SELECT ID "STUDENT_ID",COURSE_ID, NVL(GRADE ,'W' ) "GRADE"
FROM TAKES;
```

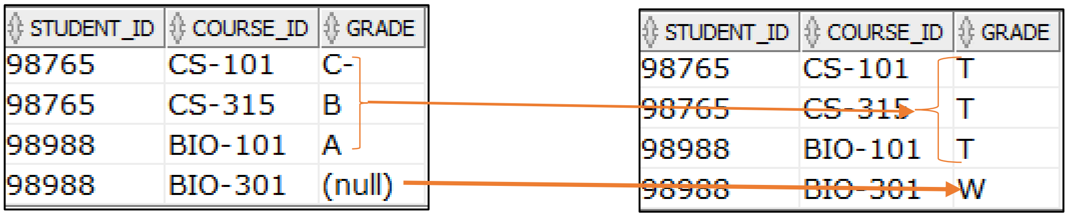


STUDENT_ID	COURSE_ID	GRADE
98765	CS-101	C-
98765	CS-315	B
98988	BIO-101	A
98988	BIO-301	(null)

STUDENT_ID	COURSE_ID	GRADE
98765	CS-101	C-
98765	CS-315	B
98988	BIO-101	A
98988	BIO-301	W

NVL2 (expr1, expr2, expr3): Converts a null value or not null to an actual value.

```
SELECT ID "STUDENT_ID",COURSE_ID,NVL2(GRADE ,'T', 'W' )”GRADE"
FROM TAKES;
```



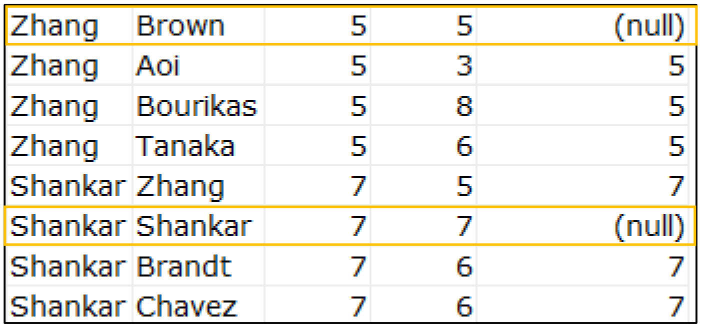
STUDENT_ID	COURSE_ID	GRADE
98765	CS-101	C-
98765	CS-315	B
98988	BIO-101	A
98988	BIO-301	(null)

STUDENT_ID	COURSE_ID	GRADE
98765	CS-101	T
98765	CS-315	T
98988	BIO-101	T
98988	BIO-301	W

NULLIF (expr1, expr2): compares expr1 and expr2. If expr1 and expr2 are equal, the NULLIF function returns NULL. Otherwise, it returns expr1.

```
SELECT S1.NAME, S2.NAME , LENGTH(S1.NAME) AS NAME1,
LENGTH(S2.NAME) AS NAME2,
NULLIF(LENGTH(S1.NAME),LENGTH(S2.NAME))

FROM STUDENT S1, STUDENT S2;
```



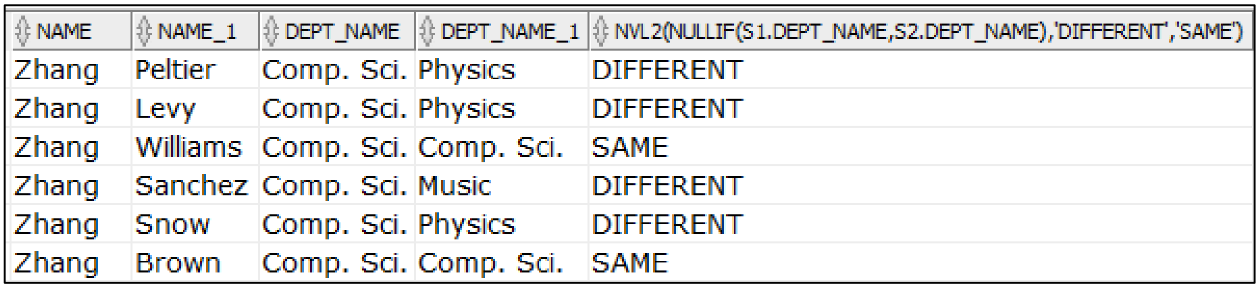
Zhang	Brown	5	5	(null)
Zhang	Aoi	5	3	5
Zhang	Bourikas	5	8	5
Zhang	Tanaka	5	6	5
Shankar	Zhang	7	5	7
Shankar	Shankar	7	7	(null)
Shankar	Brandt	7	6	7
Shankar	Chavez	7	6	7

Example: Use NULLIF function to check if two student study in the same department or not.

```
SELECT S1.NAME, S2.NAME, S1.DEPT_NAME, S2.DEPT_NAME,
NVL2(NULLIF(S1.DEPT_NAME,  S2.DEPT_NAME),'DIFFERENT','SAME')

FROM STUDENT S1, STUDENT S2

WHERE S1.ID <> S2.ID ;
```



NAME	NAME_1	DEPT_NAME	DEPT_NAME_1	NVL2(NULLIF(S1.DEPT_NAME,S2.DEPT_NAME),'DIFFERENT','SAME')
Zhang	Peltier	Comp. Sci.	Physics	DIFFERENT
Zhang	Levy	Comp. Sci.	Physics	DIFFERENT
Zhang	Williams	Comp. Sci.	Comp. Sci.	SAME
Zhang	Sanchez	Comp. Sci.	Music	DIFFERENT
Zhang	Snow	Comp. Sci.	Physics	DIFFERENT
Zhang	Brown	Comp. Sci.	Comp. Sci.	SAME

Part Two: Aggregate Functions:

Unlike single-row functions, group functions operate on sets of rows to give one result per group. These sets may comprise the entire table or the table split into groups.
The group function is placed after the SELECT keyword. You may have multiple group functions separated by commas.

Syntax of Group Functions

```
SELECT group-function (column), ...
FROM table-name
[WHERE condition]
[ORDER BY column];
```

Guidelines for using the group functions:

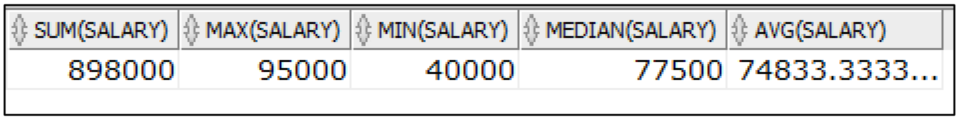
- **DISTINCT** and **UNIQU** makes the function consider only no duplicate values
- **ALL** makes it consider every value, including duplicates. The default is **ALL** and therefore does not need to be specified.
- The data types for the functions with an expr argument may be CHAR, VARCHAR2, NUMBER, or DATE.
- All group functions ignore null values. To substitute a value for null values, use the NVL or NVL2 functions.

Types of Group Functions:

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

```
SELECT SUM(SALARY),MAX(SALARY),MIN(SALARY),MEDIAN(SALARY),AVG(SALARY)

FROM INSTRUCTOR;
```



SUM(SALARY)	MAX(SALARY)	MIN(SALARY)	MEDIAN(SALARY)	AVG(SALARY)
898000	95000	40000	77500	74833.3333...

Note: The AVG, SUM, VARIANCE, and STDDEV functions can be used only with numeric data types.

- **Count Function:**

Syntax of Group Functions

```
SELECT group-function (column), ...
FROM table-name
[WHERE condition]
[ORDER BY column];
```

The COUNT function has three formats:

- **COUNT(*):**

Returns the number of rows in a table that satisfy the criteria(where condition) of the SELECT statement, including all the rows of the table even if there is any NULL value , Returns 0 if there were no matching row.

- **COUNT (expr)** :

Returns the number of non-null values that are in the column identified by expr.

- **COUNT (DISTINCT expr)** :

Returns the number of unique, non-null values that are in the column identified by expr.

Example: Find the number of instructor in department Computer Science.

```
SELECT COUNT(*)

FROM INSTRUCTOR
```

Using the DISTINCT Keyword

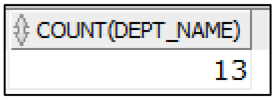
Use the DISTINCT keyword to suppress the counting of any duplicate values in a column.

Example: Find the number of department, which students belong to it.

Incorrect

```
SELECT COUNT(DEPT_NAME)

FROM STUDENT;
```



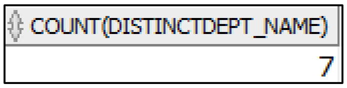
```
COUNT(DEPT_NAME)
              13
```

This is incorect way to count department , since there is duplicate.

Correct way

```
SELECT COUNT(DISTINCT DEPT_NAME)

FROM STUDENT;
```



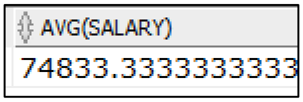
```
COUNT(DISTINCTDEPT_NAME)
                       7
```

Group Functions and Null Values:

All group functions ignore null values in the column. However, the NVL function forces group functions to include null values by using actual value to null value.

Example: the average is calculated based on only those rows in the table in which a valid value is stored in the salary column. The average is calculated as the total salary that is paid to all instructor divided by the number of instructors receiving a salary

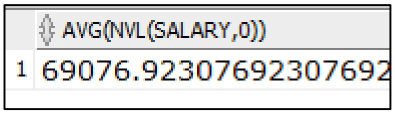
```
SELECT AVG(SALARY )
FROM INSTRUCTOR;
```



AVG(SALARY)
74833.3333333333

By using **NVL** function, The average is calculated based on **all** rows in the table, regardless of whether null values are stored

```
SELECT AVG(NVL(SALARY,0) )
FROM INSTRUCTOR;
```



	AVG(NVL(SALARY,0))
1	69076.92307692307692

➢ Grouping of Data:

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 summary information for each group.

```
SELECT column, group_function(column)
FROM table-name
[WHERE condition]
[GROUP BY columns]
[ORDER BY column-names || aliases || column-numbers];
```

You should be aware of some notes you deal with group functions:

- 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.
- Using a WHERE clause, you can exclude rows before dividing them into groups.
- You cannot use a column alias in the GROUP BY clause.

➢ Using the GROUP BY Clause

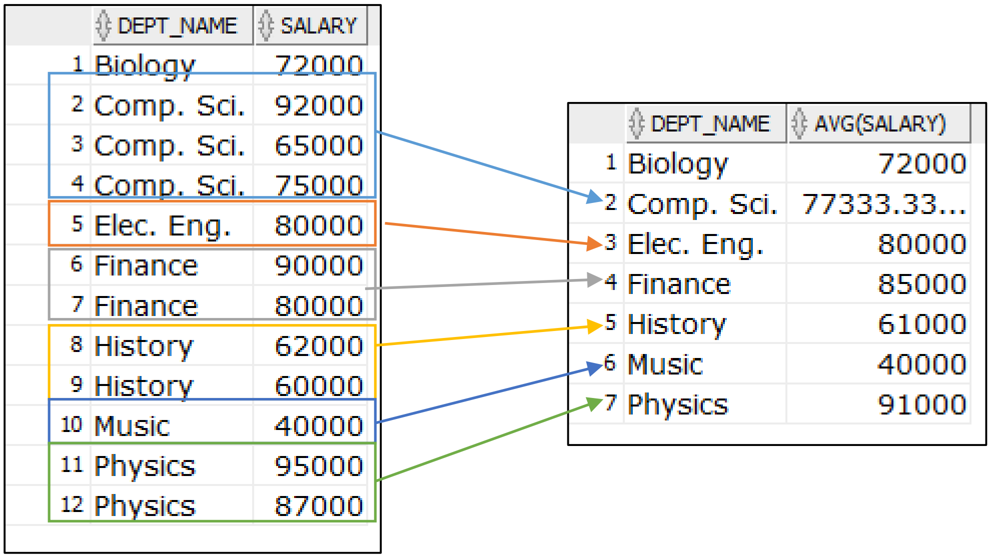
When using the **GROUP BY** clause, make sure that all columns in the SELECT list that are not group functions are included in the GROUP BY clause

Example: Find the average salary in each department.

```
SELECT DEPT_NAME, AVG (SALARY) AS AVG_SALARY

FROM INSTRUCTOR

GROUP BY DEPT_NAME;
```



	DEPT_NAME	SALARY
1	Biology	72000
2	Comp. Sci.	92000
3	Comp. Sci.	65000
4	Comp. Sci.	75000
5	Elec. Eng.	80000
6	Finance	90000
7	Finance	80000
8	History	62000
9	History	60000
10	Music	40000
11	Physics	95000
12	Physics	87000

	DEPT_NAME	AVG(SALARY)
1	Biology	72000
2	Comp. Sci.	77333.33...
3	Elec. Eng.	80000
4	Finance	85000
5	History	61000
6	Music	40000
7	Physics	91000

. You can also use the group function in the **ORDER BY** clause:

```
SELECT DEPT_NAME, AVG (SALARY) AS AVG_SALARY

FROM INSTRUCTOR

GROUP BY DEPT_NAME

ORDER BY AVG_SALARY DESC;
```

Apply group by on multiple columns:

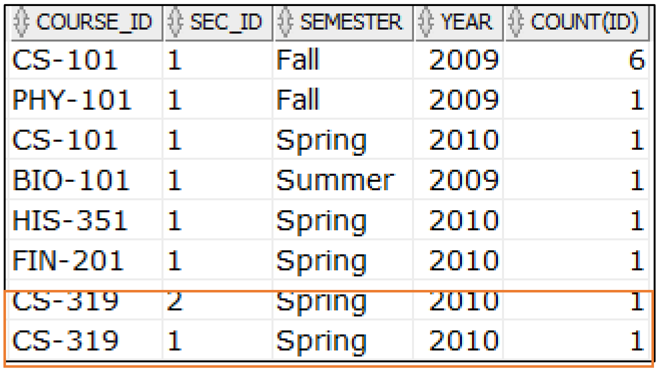
When add more one column that’s mean put the same values for all columns participant in grouping in the one group.

Example: Find the total number of students enrolled in each course section.

```
SELECT COURSE_ID ,SEC_ID,SEMESTER,YEAR,count(id)

FROM TAKES

GROUP BY COURSE_ID, SEC_ID, SEMESTER, YEAR
```



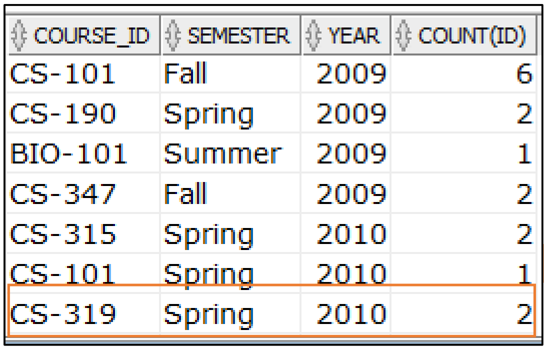
COURSE_ID	SEC_ID	SEMESTER	YEAR	COUNT(ID)
CS-101	1	Fall	2009	6
PHY-101	1	Fall	2009	1
CS-101	1	Spring	2010	1
BIO-101	1	Summer	2009	1
HIS-351	1	Spring	2010	1
FIN-201	1	Spring	2010	1
CS-319	2	Spring	2010	1
CS-319	1	Spring	2010	1

Note: To express about each course section we need add course_id ,sec_id, semester and year in grouping ,but ,if we need count number of student enrolled in course which taken in every semester, regardless about section number, we need add just course_id, semester and year in grouping.

```
SELECT COURSE_ID ,SEC_ID,SEMESTER,YEAR,count(id)

FROM TAKES

GROUP BY COURSE_ID, SEC_ID, SEMESTER, YEAR
```



COURSE_ID	SEMESTER	YEAR	COUNT(ID)
CS-101	Fall	2009	6
CS-190	Spring	2009	2
BIO-101	Summer	2009	1
CS-347	Fall	2009	2
CS-315	Spring	2010	2
CS-101	Spring	2010	1
CS-319	Spring	2010	2

Note: That any attribute that is not present in the group by clause must appear only inside an aggregate function, otherwise the query is treated as **erroneous**.

```
SELECT DEPT NAME, ID, AVG (SALARY)
FROM INSTRUCTOR
GROUP BY DEPT NAME;
```

➢ Restricting Group Results with the HAVING Clause:

You use the **HAVING** clause to apply condition to groups rather than to rows Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.

Example: Find the average salary of instructors in those departments where the average salary is more than $50,000

```
SELECT DEPT_NAME, AVG (SALARY) AS AVG_SALARY

FROM INSTRUCTOR

WHERE AVG (SALARY) > 50000

GROUP BY DEPT_NAME
```

Note that you **cannot** use group functions in the WHERE clause: Instead, use them in HAVING clause

```
SELECT DEPT_NAME, AVG (SALARY) AS AVG_SALARY

FROM INSTRUCTOR

GROUP BY DEPT_NAME

HAVING AVG (SALARY) > 50000;
```

Subqueries with grouping:

Example: Find the departments that have the highest average salary:

```
SELECT DEPT_NAME

FROM INSTRUCTOR

GROUP BY DEPT_NAME

HAVING AVG (SALARY) >= ALL (  SELECT AVG (SALARY)

                               FROM INSTRUCTOR

                               GROUP BY DEPT_NAME  );
```

Example: Find the average instructors' salaries of those departments where the average salary is greater than $50,000." We wrote this query in above, We can now rewrite this query, without using the having clause, by using a subquery in the from clause, as follows:

```
SELECT DEPT_NAME, AVG_SALARY

FROM (SELECT DEPT_NAME,AVG(SALARY) AVG_SALARY

     FROM INSTRUCTOR

     GROUP BY DEPT_NAME

     )

WHERE AVG_SALARY >= 50000
```

Subquery return new table with new columns, we can use it to apply our condition

Note: Select attributes, its new table columns

END