

# CSE 302

# Database Management Systems

## Sessional

**DATE: 27-04-2021**

**Sanjida Nasreen Tumpa**  
**Afia Anjum**  
**Lecturer**  
**Dept of CSE, MIST**



# GROUP FUNCTIONS

# Group Function

- Also known as "Multiple-Row Functions".
- They operates on set of rows to give one result per group.
- These set may be the whole table or the table split into groups.
- These are similar to the "aggregate functions" or "Group By" functions in Access

# Group/ Aggregate Functions

- SUM
- AVG
- COUNT
- MIN
- MAX

# Group Functions

- **GROUP BY clause**
  - To identify groups of records to be processed
- **ORDER BY clause**
  - To sort the records
- **HAVING clause**
  - To restrict the groups displayed

```
SELECT * | column1, column2, ...  
FROM tableName  
WHERE Condition  
GROUP BY column1, column2, ...  
ORDER BY cloumn1, column2,...  
HAVING group condition
```

# Group Functions

# SUM function

- Calculates the total amount in a numeric field for a group of records.
  - $\text{SUM}(n)$  - where  $n$  is a numeric column
  - $\text{SUM}(\text{ALL } n)$  - the same as above
  - $\text{SUM}(\text{DISTINCT } n) / \text{SUM}(\text{UNIQUE } n)$  - returns only the unique numeric values

# SUM function

- *Display total salary of all employees.*

Emp_name	Salary
A	200
B	300
C	200

```
SELECT SUM(Salary) "Total Salary"  
FROM Employee;
```

Total Salary
700



# SUM function

- *Display total UNIQUE salary of all employees.*

Emp_name	Salary
A	200
B	300
C	200

```
SELECT SUM(DISTINCT Salary)
"Total Salary" FROM Employee;
```

Total Salary
500

# SUM function

- *Display total salary of the employees of Dhaka city.*

Emp_name	Salary	Emp_city
A	200	Dhaka
B	300	Dhaka
C	200	Khulna
D	500	Dhaka

```
SELECT SUM(Salary) "Total Salary of Dhaka Emp"  
FROM Employee  
WHERE Employee_city='Dhaka' ;
```

Total Salary of Dhaka Emp
1000

# AVG function

- AVG(column containing numeric data)
- AVG(DISTINCT [column containing numeric data] )
  - **DISTINCT** keyword returns only unique values

# AVG function

- *Display average salary of all employees.*

Emp_name	Salary	Emp_city
A	200	Dhaka
B	300	Dhaka
C	200	Khulna
D	500	Dhaka

```
SELECT AVG(Salary) "Average Salary"  
FROM Employee;
```

Average Salary
300

# MAX and MIN function

- Returns the largest and smallest values in a specified column.
- MAX(ALL c) or MIN(ALL c)
  - where c is any numeric, character, or date field
- MAX(c) or MIN(c)
  - the same result as above
- MAX(DISTINCT c) or MIN(DISTINCT c)
  - returns the highest or lowest distinct value

# MAX and MIN function

- *Display the maximum salary of employees.*

Emp_name	Salary	Emp_city
A	200	Dhaka
B	300	Dhaka
C	200	Khulna
D	500	Dhaka

```
SELECT MAX(Salary) "Highest Salary" FROM  
Employee;
```

Highest Salary
500

# MAX and MIN function

- *Display the minimum DOB of employees.*

Emp_name	Salary	Emp_dob
A	200	1/25/1999
B	300	2/21/1965
C	200	6/26/1996
D	500	6/21/1964

```
SELECT MIN(EMPLOYEE_DOB)
FROM EMPLOYEE;
```

MIN(EMPLOYEE_DOB)
6/21/1964

# COUNT function

- Counts the records that have **non-NULL** values
- Counts the total records meeting **a specific condition**



# COUNT function

- *Display the count of cities only.*

Emp_name	Salary	Emp_city
A	200	Dhaka
B	300	
C	200	Khulna
D	500	Dhaka

```
SELECT COUNT(EMP_CITY)
FROM EMPLOYEE;
```

COUNT(EMP_CITY)
3

# Group functions and NULL values

- All Group functions except **COUNT(\*)** ignore null values in the column.
- Including the NULL values
  - **COUNT(\*)** counts all the records, even NULLS
  - Whenever NULL values may affect the **COUNT** the function, use an **\*** as the argument, rather than a column name.

# COUNT Function - NULL Values

Emp_name	Salary	Emp_city
A	200	Dhaka
B	300	
C	200	Khulna
D	500	Dhaka

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

Count(*)
4

# GROUP BY Clause

# GROUP BY Clause

- Divides the table of information into smaller groups.

SELECT .....

FROM .....

GROUP BY column1, column2,... ;

# GROUP BY Clause

- *Display the Average Salary of all employees according to their City.*

Emp_name	Salary	Emp_city
A	400	Dhaka
B	300	Dhaka
C	100	Khulna
D	500	Dhaka
E	100	Khulna
F	600	Rajshahi

# GROUP BY Clause

- *Display the Average Salary of all employees according to their City.*

```
SELECT Employee_city, Avg(Salary)
FROM Employee
GROUP BY Employee_city;
```

- The query execution goes like this:
  - The records in the Employee table are grouped by City
  - The average Salary for each City is calculated.

# GROUP BY Clause

Emp_name	Salary	Emp_city
A	400	Dhaka
B	300	Dhaka
C	100	Khulna
D	500	Dhaka
E	100	Khulna
F	600	Rajshahi

Emp_n ame	Salary	Emp_ci ty
A	400	Dhaka
B	300	
D	500	

Emp_n ame	Salary	Emp_ci ty
C	100	Khulna
E	100	

Emp_na me	Salary	Emp_cit y
F	600	Rajshahi



# GROUP BY Clause

EMPLOYEE_CITY	AVG(SALARY)
Dhaka	400
Khulna	100
Rajshahi	600

# GROUP BY Clause

- *Display the Sum of All Salary of the Same City according to their account type.*

Emp_name	Salary	Emp_city	Acc_type
A	400	Dhaka	Current
B	300	Dhaka	Savings
C	100	Khulna	Current
D	500	Dhaka	Savings
E	100	Khulna	Current
F	600	Rajshahi	Savings

# GROUP BY Clause

- *Display the Sum of All Salary of the Same City according to their account type.*

```
SELECT Emp_city, SUM(Salary), Acc_type  
FROM Employee GROUP BY Emp_city,  
Acc_type;
```

- The GROUP BY first groups the results by emp\_city
- Then groups the Account type within each employee City group.
- Then the SUM function calculates the salary total.

Emp_name	Salary	Emp_city	Account Type
A	400	Dhaka	Current
B	300	Dhaka	Savings
C	100	Khulna	Current
D	500	Dhaka	Savings
E	100	Khulna	Current
F	600	Rajshahi	Savings

Emp_name	Salary	Emp_city	Acc_type
A	400	Dhaka	Current
B	300		Savings
D	500		Savings

Emp_name	Salary	Emp_city	Acc_type
C	100	Khulna	Current
E	100		Current

Emp_name	Salary	Emp_city	Acc_type
F	600	Rajshahi	Savings

Emp_name	Salary	Emp_city	Account Type
A	400	Dhaka	Current
B	300	Dhaka	Savings
C	100	Khulna	Current
D	500	Dhaka	Savings
E	100	Khulna	Current
F	600	Rajshahi	Savings

Emp_name	Salary	Emp_city	Acc_type
A	400	Dhaka	Current
B	300		Savings
D	500		Savings

Emp_name	Salary	Emp_city	Acc_type
C	100	Khulna	Current
E	100		Current

Emp_name	Salary	Emp_city	Acc_type
F	600	Rajshahi	Savings

Emp_name	Salary	Emp_city	Acc_type
A	400	Dhaka	Current
B	300		Savings
D	500		Savings

Emp_name	Salary	Emp_city	Acc_type
C	100	Khulna	Current
E	100		Current

Emp_name	Salary	Emp_city	Acc_type
F	600	Rajshahi	Savings

### Final Output

Emp_city	Sum(Salary)	Account Type
Dhaka	400	Current
Dhaka	800	Savings
Khulna	200	Current
Rajshahi	600	Savings

# ORDER BY Clause

# ORDER BY Clause

- *Display the Average Salary of all employees according to their City and order the result by average salary.*

```
SELECT Employee_city, Avg(Salary)
FROM Employee
GROUP BY Employee_city
ORDER BY Avg(Salary);
```



# ORDER BY Clause

Emp_name	Salary	Emp_city
A	400	Dhaka
B	300	Dhaka
C	100	Khulna
D	500	Dhaka
E	100	Khulna
F	600	Rajshahi

Emp_n ame	Salary	Emp_ci ty
A	400	Dhaka
B	300	
D	500	

Emp_n ame	Salary	Emp_ci ty
C	100	Khulna
E	100	

Emp_na me	Salary	Emp_cit y
F	600	Rajshahi

# ORDER BY Clause

EMPLOYEE_CITY	AVG(SALARY)
Khulna	200
Dhaka	400
Rajshahi	600

# HAVING Clause

# HAVING Clause

- To further restrict groups returned by a query (Specifies which groups will be returned)
- Use a HAVING clause instead of a WHERE clause **when group functions are involved.**

HAVING(condition)

# HAVING Clause

- *Display the Sum of All Salary of the Same City according to their account type with total Salary > 200.*

```
SELECT Emp_city, SUM(Salary), Acc_type  
FROM Employee GROUP BY Emp_city,  
Acc_type WHERE SUM(Balance) > 200;
```

**ERROR!!!!!!!!!!!!!!**

# HAVING Clause

- *Display the Sum of All Salary of the Same City according to their account type with total Salary > 200.*

```
SELECT Emp_city, SUM(Salary), Acc_type  
FROM Employee GROUP BY Emp_city,  
Acc_type HAVING SUM(Salary) > 200;
```

Emp_name	Salary	Emp_city	Account Type
A	400	Dhaka	Current
B	300	Dhaka	Savings
C	100	Khulna	Current
D	500	Dhaka	Savings
E	100	Khulna	Current
F	600	Rajshahi	Savings

Emp_name	Salary	Emp_city	Acc_type
A	400	Dhaka	Current
B	300		Savings
D	500		Savings

Emp_name	Salary	Emp_city	Acc_type
C	100	Khulna	Current
E	100		Current

Emp_name	Salary	Emp_city	Acc_type
F	600	Rajshahi	Savings

Emp_name	Salary	Emp_city	Account Type
A	400	Dhaka	Current
B	300	Dhaka	Savings
C	100	Khulna	Current
D	500	Dhaka	Savings
E	100	Khulna	Current
F	600	Rajshahi	Savings

Emp_name	Salary	Emp_city	Acc_type
A	400	Dhaka	Current
B	300		Savings
D	500		Savings

Emp_name	Salary	Emp_city	Acc_type
C	100	Khulna	Current
E	100		Current

Emp_name	Salary	Emp_city	Acc_type
F	600	Rajshahi	Savings



Emp_name	Salary	Emp_city	Acc_type
A	400	Dhaka	Current
B	300		Savings
D	500		Savings

Emp_name	Salary	Emp_city	Acc_type
C	100	Khulna	Current
E	100		Current

Emp_name	Salary	Emp_city	Acc_type
F	600	Rajshahi	Savings

### Final Output

Emp_city	Sum(Salary)	Account Type
Dhaka	400	Current
Dhaka	800	Savings
Khulna	200	Current
Rajshahi	600	Savings

Emp_city	Sum(Salary )	Account Type
Dhaka	400	Current
Dhaka	800	Savings
Rajshahi	600	Savings

# WHERE and HAVING

- Both can be used in the same query.

*Display the employee city and average salary of employees **Whose name start with A** by grouping them according to their city with avg salary > 500*

```
SELECT Emp_city, AVG(Salary)
FROM Employee WHERE Emp_name like 'A%' GROUP BY Emp_city
HAVING Avg(Salary)>500;
```

# WHERE and HAVING

Emp_name	Salary	Emp_city
A	400	Dhaka
B	300	Dhaka
C	100	Khulna
AA	500	Dhaka
E	100	Khulna
AF	600	Rajshahi



Emp_name	Salary	Emp_city
A	400	Dhaka
AA	500	Dhaka
AF	600	Rajshahi

# WHERE and HAVING

Emp_name	Salary	Emp_city
A	400	Dhaka
AA	500	Dhaka
AF	600	Rajshahi



Emp_n ame	Salary	Emp_ci ty
A	400	Dhaka
AA	500	

Emp_na me	Salary	Emp_cit y
AF	600	Rajshahi

# WHERE and HAVING

EMPLOYEE_CITY	AVG(SALARY)
Dhaka	450
Rajshahi	600



EMPLOYEE_CITY	AVG(SALARY)
Rajshahi	600

# Nesting Group Functions

- Group Functions can be nested to a **depth of two.**

```
SELECT Max(Avg(Salary))  
FROM Employee  
GROUP BY Employee_city;
```

# Some general rules

- For using a mixture of individual items(`Employee_city`) and group functions (`AVG`) in the same `SELECT` statement, you must include a `GROUP BY` Clause that specifies the individual items.
- You can't use `WHERE` Clause to restrict groups.
- You have to use the `HAVING` Clause to restrict groups.

# Practice Problems for Group Functions



- **Write a query to display the number of customer with the same city.**
- **Display the Manager Number and the Salary of the lowest paid employee for that manager.**
- **Display the Manager Number and the difference between the highest and the lowest Salary of the employee for that manager.**
- **Display the minimum, maximum, sum and average salary for each group of employee having the same city.**



# **FUNCTIONS**

# **CASE Based Functions**

# CASE CONVERSION FUNCTIONS

- To convert letters to lower or upper case
- Most database administrators rarely need to use character functions
- Application developers frequently include them to create user-friendly database interfaces
- In Oracle, the comparisons of data are case-sensitive.

# CASE CONVERSION FUNCTIONS

Emp_name	Emp_city
A	Dhaka
B	Dhaka
C	Khulna
D	Dhaka

```
SELECT Emp_name FROM Employee  
WHERE Emp_city = "DHAKA";
```

- Executing it **No rows will be returned.**
  - **Why?**
    - The employee city we're looking for is stored in as "**Dhaka**". But the search key has been entered in upper case as "**DHAKA**".

# CASE CONVERSION FUNCTIONS

- 2 Functions
  - **LOWER** – Converts character strings to lower-case
  - **UPPER** - Converts character strings to upper-case

# LOWER

```
SELECT Emp_name  
FROM Employee  
WHERE LOWER(Emp_city) = "dhaka";
```

Emp_name	Emp_city
A	Dhaka
B	Dhaka
C	Khulna
D	Dhaka



Emp_name	Emp_city
A	dhaka
B	dhaka
C	khulna
D	dhaka

Output:

Emp_name
A
B
D

# UPPER

Emp_name	Emp_city
A	Dhaka
B	Dhaka
C	Khulna
D	Dhaka

SELECT **UPPER**(Emp\_city)  
FROM Employee;

Emp_city
DHAKA
DHAKA
KHULNA
DHAKA



# INITCAP

- To convert character strings to mixed case, with each word beginning with a capital letter.

```
SELECT INITCAP (Emp_name)  
“EMPLOYEE NAME AS INITCAP” FROM  
Employee
```

Emp_name
Afia <b>a</b> njum
Anika Binte Islam
<b>s</b> harifa Rania
<b>u</b> mme <b>h</b> abiba



Emp_name
Afia <b>A</b> njum
Anika Binte Islam
<b>S</b> harifa Rania
<b>U</b> mme <b>h</b> abiba



# **CHARACTER MANIPULATION FUNCTIONS**

# SUBSTR

- Used to return a substring, or portion of a string

SUBSTR(character string or column name, beginning character position, length of string to be returned)

- SELECT branch\_name,  
**SUBSTR(branch\_name,1,3)** FROM branch;

BRANCH_NAME	SUBSTR
Downtown	Dow
Redwood	Red
Perryridge	Per
Mianus	Mia
Brighton	Bri

# SUBSTR

- SELECT branch\_name,  
**SUBSTR(branch\_name,4,2)** FROM  
branch;

BRANCH_NAME	SUBSTR
Downtown	
Redwood	
Perryridge	
Mianus	
Brighton	

# SUBSTR

- `SELECT branch_name,  
SUBSTR(branch_name,4,2) FROM  
branch;`

BRANCH_NAME	SUBSTR
Downtown	nt
Redwood	wo
Perryridge	ry
Mianus	nu
Brighton	gh

# LENGTH

LENGTH(character string)

```
SELECT branch_name, LENGTH(branch_name)
FROM branch;
```

BRANCH_NAME	LENGTH(BRANCH_NAME)
Downtown	8
Redwood	7
Perryridge	10
Mianus	6

# LPAD

LPAD(string to be padded, length of string after padding, symbol used to pad)

```
SELECT branch_name, LPAD(branch_name,12,'*')  
FROM branch;
```

BRANCH_NAME	LPAD(BRANCH_NAME,12,'*')
Downtown	****Downtown
Mianus	*****Mianus
Round Hill	**Round Hill
Pownal	*****Pownal

# RPAD

RPAD(string to be padded, length of string after padding, symbol used to pad)

```
SELECT branch_name, RPAD(branch_name,12,'*')  
FROM branch;
```

BRANCH_NAME	RPAD(BRANCH_NAME,12,'*')
Downtown	Downtown****
Perryridge	Perryridge**
Mianus	Mianus*****
Round Hill	Round Hill**
Pownal	Pownal*****



# LTRIM

- Removes a specific string of characters from the left side of the data

LTRIM(data, specific string to be removed from the left side of the data)

```
SELECT LTRIM(cust_id,'C') FROM  
customer;
```

CUST_ID
C000000000001
CC000000000002
CCC000000000003
CCCC000000000004



LTRIM(CUST_ID,'C')
000000000001
000000000002
000000000003
000000000004

# RTRIM

- Removes a specific string of characters from the Right side of the data

RTRIM(data, specific string to be removed from the right side of the data)

```
SELECT RTRIM(cust_id,'C') FROM  
customer;
```

CUST_ID
00000000001C
00000000002CC
00000000003CCC
00000000004CCCC



RTRIM(CUST_ID,'C')
00000000001
00000000002
00000000003
00000000004

# REPLACE

- Similar to "search and replace" in some application programs

REPLACE(column, string to be found, string replacement)

SELECT REPLACE(cust\_id, 'C000', 'Cust') FROM customer;

CUST_ID		REPLACE(CUST_ID,'C000','CUST')
C00000000001		Cust00000001
C00000000002		Cust00000002
C00000000003		Cust00000003
C00000000004		Cust00000004

# CONCAT

- Concatenates the data from two columns
- Combines only two items (columns or string literals)

CONCAT(column or string, column or string)

```
SELECT cust_name, CONCAT('Customer  
Number: ', cust_id) "Customer ID" FROM  
customer;
```

- To concatenate more than two items, you must nest a CONCAT function inside another CONCAT function

**TRY YOURSELF**

# CONCAT

```
SELECT cust_name, CONCAT('Customer  
Number: ', cust_id) "Customer ID" FROM  
customer;
```

CUST_NAME	Customer ID
Jones	Customer Number: C000000000001
Smith	Customer Number: C000000000002
Hayes	Customer Number: C000000000003
Curry	Customer Number: C000000000004
Lindsay	Customer Number: C000000000005



# NUMERIC FUNCTIONS

# ROUND

- To round numeric fields to the stated precision
  - If position is a positive number, it refers to the right side of the decimal point.
  - If position is a negative number, function rounds to the left side of the decimal point.

ROUND(numeric field to be rounded, position of the digit to which the data should be rounded)

# ROUND

↓  
SELECT ROUND(3162.845, 1) AS ROUNDED  
FROM dual;

•3162.8

↓  
SELECT ROUND(3162.8451297, 5) AS ROUNDED  
FROM dual;

•3162.84512



3162.8451**3**

**Final Answer!!**



# ROUND

↓  
SELECT **ROUND(3162.845, -2)** AS ROUNDED  
FROM dual;

•3162 → 3100 → **3200** Final Answer!!

↓  
SELECT **ROUND(1234, -2)** AS ROUNDED  
FROM dual;

•1234 → **1200** Final Answer!!

↓  
SELECT **ROUND(5232.85, -3)** from dual

•5232 → **5000** Final Answer!!

# TRUNC

- To truncate a numeric value to a specific position
  - If position is a positive number, it refers to the right side of the decimal point.
  - If position is a negative number, function rounds to the left side of the decimal point.

TRUNC (numeric field, position of the digit from which the data should be removed)

# TRUNC

```
SELECT TRUNC(15.79,1) "Truncate" FROM  
DUAL;
```

- 15.7

```
SELECT TRUNC(123456.76,-4) "Truncate" FROM  
DUAL;
```

- 123456



120000

**Final Answer!!**



# DATE FUNCTIONS

# Difference between Two dates in Days

```
SELECT Emp_id, Emp_dob, Emp_startdate,  
       Emp_startdate – Emp_dob  
FROM Employee;
```

E_ID	E_DOB	E_STARTDATE	E_S_DATE - E_DOB
E000002	01/22/1958	01/22/1978	7305
E000010	04/21/1956	04/21/1986	10957

## Difference between Two dates (Weeks)

- **The delay between the two dates in weeks:**

```
SELECT Employee_id, Employee_dob,  
Employee_startdate,  
(Employee_startdate – Employee_dob)/7  
"DELAY IN WEEKS"  
FROM Employee;
```

# Difference between Two dates

EMP_ID	EMP_DOB	EMP_S_DATE	DELAY IN WEEKS
E000002	01/22/1958	01/22/1978	1043.57142
E000010	04/21/1956	04/21/1986	1565.285714

# MONTHS\_BETWEEN

- Determines the number of months between two dates

MONTHS\_BETWEEN(later date, earlier date)

```
SELECT Employee_id,  
MONTHS_BETWEEN (Employee_startdate,  
Employee_dob) “No. of Months”  
FROM Employee;
```



# ADD\_MONTHS

ADD\_MONTHS(beginning date, number of months to add to the date)

```
SELECT Employee_id, Employee_startdate,  
ADD_MONTHS (Employee_startdate, 60)  
FROM Employee;
```

# ADD\_MONTHS

EMP_ID	EMP_STARTDATE	ADD_MONTHS(EMP_STARTDATE,60)
E0000002	01/22/1978	01/22/1983
E0000003	02/23/1982	02/23/1987

# NEXT\_DAY

- Determines the next occurrence of a specific day of the week after a given date – **Output is a DATE**

NEXT\_DAY(starting date, day of week to be identified)

```
SELECT Employee_id,  
       NEXT_DAY(Employee_startdate, 'MONDAY')  
       "First Monday After Joining"  
FROM Employee;
```

# NEXT\_DAY

```
SELECT next_day('3/31/2020','TUESDAY') from dual
```

<code>next_day('3/31/2020','TUESDAY')</code>
04/07/2020

# The Nesting of Functions

- A function is used as an argument inside another function
- Rules
  - One must include all arguments for each function.
  - For every open parenthesis, there must be a corresponding closed parenthesis.
  - The inner function is resolved first, then the outer function.

# The Nesting of Functions

To determine the **Number of months** between the Employee\_startdate and Employee\_dob, we use the MONTHS\_BETWEEN function.

```
SELECT      Employee_id,      MONTHS_BETWEEN
(Employee_startdate, Employee_dob) "Delay in
Months" FROM Employee;
```

# The Nesting of Functions

To suppress the decimal places generated by the Months\_Between function, we can use the result of the Months\_Between function as an input to the TRUNC function.

```
SELECT Employee_id, TRUNC(MONTHS_BETWEEN  
(Employee_startdate, Employee_dob),0) "Delay in  
Months" FROM Employee;
```



**THE END**