

SQL 3: DML 2

SCSD2523 Database
Semester 1 2024/2025

Learning Objective

- At the end of the topic, students are able to:
 - Write DML statements that uses the followings:
 - **ORDER BY**
 - **COUNT, SUM, AVG, MIN, MAX**
 - **GROUP BY**
 - **HAVING**
 - Subqueries

Using ORDER BY

- Purpose: To sort results in ascending (default) ASC or descending DESC order.
- Example:
 - List employee salaries in all department, sort by department number in ascending order and by salary in descending order.

```
SELECT department_id,  
employee_id, salary  
FROM employees  
ORDER BY department_id,  
salary DESC;
```

Ascending
by default

Descending
order for each
department

DEPTNO	EMPNO	SAL
10	7839	5000
10	7782	2450
10	7934	1300
20	7788	3000
20	7902	3000
20	7566	2975
20	7876	1100
20	7369	800
30	7698	2850
30	7499	1600
30	7844	1500
30	7654	1250
30	7521	1250
30	7900	950

14 rows selected.

Using ORDER BY

- Example:
 - Display a sorted list of employee salaries in ascending order with salary column header as EMP_SALARY

```
SELECT department_id,  
employee_id, salary emp_salary  
FROM employees  
ORDER BY emp_salary;
```

```
SQL> select deptno, empno, sal emp_salary  
2 from emp  
3 order by emp_salary;
```

DEPTNO	EMPNO	EMP_SALARY
20	7369	800
30	7900	950
20	7876	1100
30	7521	1250
30	7654	1250
10	7934	1300
30	7844	1500
30	7499	1600
10	7782	2450
30	7698	2850
20	7566	2975

DEPTNO	EMPNO	EMP_SALARY
20	7788	3000
20	7902	3000
10	7839	5000

14 rows selected.

This will **rename** display of column name in **output ONLY**

Numeric Functions

- **ROUND**: Rounds value to specified decimal
- **TRUNC**: Truncates value to specified decimal
- **MOD**: Returns remainder of division (modulus)

FUNCTION	RESULT
ROUND(45.926, 2) <ul style="list-style-type: none">• 1st parameter: the value• 2nd parameter: number of decimal places to round	45.93
TRUNC(45.926, 2) <ul style="list-style-type: none">• 1st parameter: the value• 2nd parameter: number of decimal places to truncate	45.92
MOD(1600, 300)	100


Using **TRUNC** (Oracle) / **TRUNCATE** (MySQL)

- Example of using **TRUNC/TRUNCATE**:



```
SELECT TRUNC(commission_pct, 1)
FROM employees
WHERE commission_pct
IS NOT NULL;
```

NOTE: Use **TRUNCATE** in MySQL

With **TRUNC**

Result Grid		 Filter Rows:
	<code>truncate(commission_pct,1)</code>	
▶	0.4	
	0.3	
	0.3	
	0.3	
	0.2	
	0.3	
	0.2	
	0.2	
	0.2	
	0.2	
	0.1	

Normal data

Result Grid   Filter Rows:

	commission_pct
▶	0.40
	0.30
	0.30
	0.30
	0.20
	0.30
	0.25
	0.25
	0.20
	0.20
	0.15

Display normally without truncate:

```
SELECT commission_pct
FROM employees
WHERE commission_pct
IS NOT NULL;
```

Using ROUND

- Example of using **ROUND**:

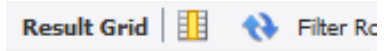
```
SELECT ROUND(commission_pct, 1)
FROM employees
WHERE commission_pct
IS NOT NULL;
```

NOTE: Use **TRUNCATE** in MySQL

Display normally without ROUND:


```
SELECT commission_pct
FROM employees
WHERE commission_pct
IS NOT NULL;
```

With **ROUND**



round(commission_pct, 1)
0.4
0.3
0.3
0.3
0.2
0.3
0.3
0.3
0.2
0.2

Normal data



commission_pct
0.40
0.30
0.30
0.30
0.20
0.30
0.25
0.25
0.20
0.20

Compare **ROUND**, **TRUNC** and normal

- Execute the SQL statement below to see the difference

```
SELECT TRUNC(commission_pct, 1), ROUND(commission_pct, 1),  
commission_pct  
FROM employees  
WHERE commission_pct IS NOT NULL;
```

NOTE: Use **TRUNCATE in MySQL**

	TRUNCATE(commission_pct, 1)	ROUND(commission_pct, 1)	commission_pct
▶	0.4	0.4	0.40
	0.3	0.3	0.30
	0.3	0.3	0.30
	0.3	0.3	0.30
	0.2	0.2	0.20
	0.3	0.3	0.30
	0.2	0.3	0.25
	0.2	0.3	0.25
	0.2	0.2	0.20
	0.2	0.2	0.20

Aggregate Functions

- Sometimes referred to as Group Functions
- Can be used only in the SELECT and HAVING clause
- Operate on sets of rows to give one result per group
 - For numeric data:
 - **AVG** - returns average of values in a column **
 - **SUM** - returns the sum of values in a column **
 - For numeric, character & date data types
 - **MAX** - returns the largest value of values in a column **
 - **MIN** - returns the smallest value of values in a column **
 - **COUNT** - returns number of rows

*** - Ignoring NULL values*

Substitute NULL Values

- The following functions work with any data type and pertain to using NULLs
 - **NVL**(expr1, expr2)
 - Only works in Oracle!
 - For MySQL, use **IFNULL**(expr1, expr2)
 - **NVL2**(expr1, expr2, expr3)
 - Only works in Oracle!
 - For MySQL can use **IF**(expr1, expr2, expr3)
 - **NULLIF**(expr1, expr2)
 - Works for both Oracle and MySQL
 - **COALESCE**(expr1, expr2, ... , exprn)
 - Works for both Oracle and MySQL

Substitute **NULL** Values

Function	Description
NVL	Converts a null value to an actual value
NVL2	If <code>expr1</code> is not null, NVL2 returns <code>expr2</code> . If <code>expr1</code> is null, NVL2 returns <code>expr3</code> . The argument <code>expr1</code> can have any data type
NULLIF	Compares two expressions and returns null if they are equal; Returns the first expression if they are not equal
COALESCE	Returns the first non-null expression in the expression list

Using **NVL** (Oracle) / **IFNULL** (MySQL)

- Converts a NULL value to an actual value.
- Syntax:
 - **NVL** (expr1, expr2) / **IFNULL**(expr1, expr2)
 - expr1 is the source value or expression that may contain NULL
 - expr2 is the target value for converting the null.
- You can use the **NVL** function with any data type, but the return value is always the **same as data type in expr1**

Using **NVL** (Oracle) / **IFNULL** (MySQL)

- Data types that can be used are date, character, and number
- Data types must match
 - **NVL**(commission_pct, 0) / **IFNULL** (commission_pct, 0)
 - **NVL**(hire_date, '01-JAN-15') / **IFNULL** (hire_date, '01-JAN-15')
 - **NVL**(job_id, 'No Job Yet') / **IFNULL**(job_id, 'No Job Yet')

Data Type	Conversion Example
NUMBER	<code>NVL (number_column, 9)</code>
DATE	<code>NVL (date_column, '01-JAN-15')</code>
CHAR or VARCHAR2	<code>NVL (character_column, 'Unavailable')</code>

Using the NVL function

- Calculate the annual salary of employees. We need to multiply the monthly salary by 12 and then add the commission percentage to the result

```
SELECT last_name, salary, commission_pct,
       (salary*12) + (salary * 12 * commission_pct) ANN_SAL
FROM employees;
```

1

2

- Notice that the annual salary is calculated for only employees who earn a commission.
- If any column value expression is null, the result is null.
- We must convert the null value to a number before applying the arithmetic operator.

R	LAST_NAME	R	SALARY	R	COMMISSION_PCT	R	AN_SAL
1	Whalen		4400		(null)		(null)
2	Hartstein		13000		(null)		(null)

1

2

...

R	LAST_NAME	R	SALARY	R	COMMISSION_PCT	R	AN_SAL
15	Matos		2600		(null)		(null)
16	Vargas		2500		(null)		(null)
17	Zlotkey		10500		0.2		151200
18	Abel		11000		0.3		171600
19	Taylor		8600		0.2		123840
20	Grant		7000		0.15		96600

Using the NVL function

NOTE: Use **IFNULL** in MySQL

- Calculate the annual salary of employees. We need to multiply the monthly salary by 12 and then add the commission percentage to the result

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

- Notice that the annual salary is calculated for only employees who earn a commission.
- If any column value expression is null, the result is null.
- We must convert the null value to a number before applying the arithmetic operator.

	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
1	Whalen	4400	0	52800
2	Hartstein	13000	0	156000

...				
	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
15	Matos	2600	0	31200
16	Vargas	2500	0	30000
17	Zlotkey	10500	0.2	151200
18	Abel	11000	0.3	171600
19	Taylor	8600	0.2	123840
20	Grant	7000	0.15	96600

Using **NVL2** (Oracle) / **IF** (MySQL)

- The **NVL2** function examines the first expression
- If the first expression is not null, the **NVL2** function returns the second expression
- If the first expression is null, the third expression is returned
- Syntax:
 - **NVL2** (**expr1**, **expr2**, **expr3**) / **IF** (**expr1**, **expr2**, **expr3**)
 - **expr1** is the source value or expression that may contain a null
 - **expr2** is the value returned if **expr1** is not null
 - **expr3** is the value returned if **expr1** is null

NOTE: Use **IF** in MySQL

Using NVL2 (Oracle) / IF (MySQL)

- The commission_pct column is examined.
 - If a value is detected, the text literal value of salary+commission_pct is returned

NOTE: Use **IF** in MySQL

- If the commission_pct column contains a null value, the text literal value of salary is returned

```
SELECT last_name, salary, commission_pct,  
NVL2(commission_pct, 'SAL+COMM', 'SAL') income  
FROM employees  
WHERE department_id IN (50, 80);
```

1

2

Note:

- The argument **expr1** can have any data type.
- The argument **expr2** and **expr3** can have any data types except **LONG**

	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	Taylor	8600	0.2	SAL+COMM

1

2

Aggregate Functions

- **COUNT(column)**
 - Returns the number of rows with non-null values for the column
- Example:
 - List the total number of employees who receive commissions

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

```
SQL> select count(commission_pct)
      2  from employees;
```

```
COUNT(COMMISSION_PCT)
```

```
-----
```

```
4
```

Aggregate Functions

- **COUNT(*)**
 - Return number of records (rows) in the table
 - Includes duplicate rows and rows with NULL values
- Example:
 - List the total number of employees in department number 10.

```
SELECT COUNT(*) AS "No of Employees"  
FROM employees  
WHERE department_id = 10;
```

```
SQL> select count(*) as "No of Employees"  
2   from employees  
3   where department_id = 10;
```

```
No of Employees  
-----  
1
```

Aggregate Functions

- Example:
 - List the average, sum, minimum and maximum salary for employees in department number 80.
 - Department number 80 has 34 employees

```
SELECT AVG(salary), MIN(salary), MAX(salary), SUM(salary)
FROM employees
WHERE department_id=80;
```

AVG (SALARY)	MIN (SALARY)	MAX (SALARY)	SUM (SALARY)
8955.88235	6100	14000	304500

Creating Groups of Data – **GROUP BY**

- You can divide rows in a table into smaller groups by using the **GROUP BY** clause.
- All columns in the **SELECT** list that are not in group functions must be in the **GROUP BY** clause.

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

[] = OPTIONAL

Creating Groups of Data – GROUP BY

	employee_id	first_name	last_name	department_id	job_id
▶	100	Steven	King	90	AD_PRES
	101	Neena	Kochhar	90	AD_VP
	102	Lex	De Haan	90	AD_VP
	103	Alexander	Hunold	60	IT_PROG
	104	Bruce	Ernst	60	IT_PROG
	105	David	Austin	60	IT_PROG
	106	Valli	Pataballa	60	IT_PROG
	107	Diana	Lorentz	60	IT_PROG
	108	Nancy	Greenberg	100	FI_MGR
	109	Daniel	Faviet	100	FI_ACCOUNT
	110	John	Chen	100	FI_ACCOUNT
	111	Ismael	Sciarra	100	FI_ACCOUNT
	112	Jose Manuel	Urman	100	FI_ACCOUNT
	113	Luis	Popp	100	FI_ACCOUNT
	114	Den	Raphaely	30	PU_MAN
	115	Alexander	Khoo	30	PU_CLERK
	116	Shelli	Baida	30	PU_CLERK
	117	Sigal	Tobias	30	PU_CLERK
	118	Guy	Himuro	30	PU_CLERK
	119	Karen	Colmenares	30	PU_CLERK
	120	Matthew	Weiss	50	ST_MAN
	121	Adam	Fripp	50	ST_MAN

What if:

- List the department_id and number of staffs in each department.

Creating Groups of Data – **GROUP BY**

- Issue:

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

```
SQL> SELECT department_id, COUNT(last_name)
      2  FROM employees;
SELECT department_id, COUNT(last_name)
      *
```

ERROR at line 1:

ORA-00937: not a single-group group function

SOLUTION:

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

Creating Groups of Data – GROUP BY

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

- Whenever you use a mixture of
 - individual items
(**DEPARTMENT_ID**) and
 - group functions (**COUNT**)
 - In the same **SELECT** statement,
 - You must include a **GROUP BY** clause that specifies the individual item (in this case, **DEPARTMENT_ID**)

	department_id	count(last_name)
▶	90	3
	60	5
	100	6
	30	6
	50	2

Creating Groups of Data – GROUP BY

	employee_id	first_name	last_name	department_id	job_id
▶	100	Steven	King	90	AD_PRES
	101	Neena	Kochhar	90	AD_VP
	102	Lex	De Haan	90	AD_VP
	103	Alexander	Hunold	60	IT_PROG
	104	Bruce	Ernst	60	IT_PROG
	105	David	Austin	60	IT_PROG
	106	Valli	Pataballa	60	IT_PROG
	107	Diana	Lorentz	60	IT_PROG
	108	Nancy	Greenberg	100	FI_MGR
	109	Daniel	Faviet	100	FI_ACCOUNT
	110	John	Chen	100	FI_ACCOUNT
	111	Ismael	Sciarra	100	FI_ACCOUNT
	112	Jose Manuel	Urman	100	FI_ACCOUNT
	113	Luis	Popp	100	FI_ACCOUNT
	114	Den	Raphaely	30	PU_MAN
	115	Alexander	Khoo	30	PU_CLERK
	116	Shelli	Baida	30	PU_CLERK
	117	Sigal	Tobias	30	PU_CLERK
	118	Guy	Himuro	30	PU_CLERK
	119	Karen	Colmenares	30	PU_CLERK
	120	Matthew	Weiss	50	ST_MAN
	121	Adam	Fripp	50	ST_MAN

What if:

- List the department_id, job_id and number of staffs based on the job_id in each department.

Creating Groups of Data – GROUP BY

- Issue:

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

```
SQL> SELECT department_id, job_id, COUNT(last_name)
      2 FROM employees
      3 GROUP BY department_id;
SELECT department_id, job_id, COUNT(last_name)
                        *
```

ERROR at line 1:

ORA-00979: not a GROUP BY expression

SOLUTION:

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

Creating Groups of Data – GROUP BY

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

- Any column or expression in the **SELECT** list that is not an aggregate function must be in the **GROUP BY** clause
- job_id is neither in the **GROUP BY** clause nor is it being used by a group function, so there is a **“not a GROUP BY expression”** error
- We can correct this error by adding job_id in the **GROUP BY** clause

	department_id	job_id	count(last_name)
▶	90	AD_PRES	1
	90	AD_VP	2
	60	IT_PROG	5
	100	FI_MGR	1
	100	FI_ACCOUNT	5
	30	PU_MAN	1
	30	PU_CLERK	5
	50	ST_MAN	2

```
SQL> SELECT department_id, job_id, COUNT(last_name)
2 FROM employees
3 GROUP BY department_id, job_id;
```

DEPARTMENT_ID	JOB_ID	COUNT(LAST_NAME)
110	AC_ACCOUNT	1
90	AD_VP	2
50	ST_CLERK	4
80	SA_REP	2
110	AC_MGR	1
50	ST_MAN	1
80	SA_MAN	1
20	MK_MAN	1
90	AD_PRES	1
60	IT_PROG	3
	SA_REP	1

DEPARTMENT_ID	JOB_ID	COUNT(LAST_NAME)
10	AD_ASST	1
20	MK_REP	1

13 rows selected.

Creating Groups of Data – GROUP BY

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

JOB_ID	PAYROLL
IT_PROG	19200
AD_PRES	24000
AD_VP	34000

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

JOB_ID	PAYROLL
IT_PROG	19200
AD_PRES	24000
AD_VP	34000

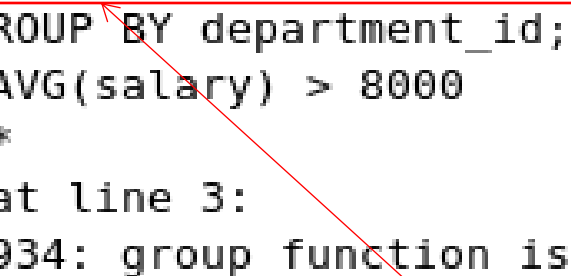
- List the job ID and total monthly salary for each job that has a payroll exceeding \$13,000
- It excludes sales representatives
- It sorts the list by the total monthly salary

Illegal Queries Using Group Functions

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

```
SQL> SELECT department_id, AVG(salary)
2   FROM employees
3   WHERE AVG(salary) > 8000
4   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

Restricting Group Results with the **HAVING** Clause

- When you use the **HAVING** clause, the Oracle server restricts groups as follows:
 - Rows are grouped.
 - The group function is applied.
 - Groups matching the **HAVING** clause are displayed.

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

[] = OPTIONAL

Restricting Group Results with the **HAVING** Clause

Restricting Group Results with the **HAVING** Clause

- List the average salary for every department which has an average salary of \$8000

```
SQL> SELECT department_id, AVG(salary)
2    FROM employees
3    GROUP BY department_id
4    HAVING AVG(salary) > 8000;
```

DEPARTMENT_ID	AVG(SALARY)
20	9500
90	19333.3333
110	10150
80	10033.3333

Restricting Group Results with the **HAVING** Clause

- List the average salary for every department which has an average salary of \$8000

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

department_id	avg(salary)
NULL	7000.000000
10	4400.000000
20	9500.000000
30	4150.000000
40	6500.000000
50	3475.555556
60	5760.000000
70	10000.000000
80	8955.882353
90	19333.333333
100	8600.000000
110	10150.000000

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

department_id	avg(salary)
20	9500.000000
70	10000.000000
80	8955.882353
90	19333.333333
100	8600.000000
110	10150.000000

Summary

- **ORDER BY** : To sort display in either ascending or descending order
- Number functions : **ROUND, TRUNC, MOD**
- Aggregate functions : **COUNT, MAX, MIN, AVG**
- Grouping data: **GROUP BY**
- Group functions
- **HAVING** : use with conditions for restricting group results

Simple Hand Written Exercise

Given the following relation schemas, construct the SQL statement for following tasks:

```
CUSTOMER (customer_id, store_id, first_name, last_name, email, address, active)
RENTAL (rental_id, rental_date, inventory_id, customer_id, return_date, staff_id)
INVENTORY (inventory_id, film_id, store_id)
STORE (store_id, manager_staff_id, location)
STAFF (staff_id, first_name, last_name, address, email, store_id, salary)
PAYMENT (payment_id, customer_id, staff_id, rental_id, amount, payment_date)
FILM (film_id, title, description, rental_duration, rental_rate)
```

Schemas used in DML Question in Final Exam 2016/2017

- 1) The SQL query below returns an error. Revise the SQL query.

```
SELECT film_id, title AVG(rental_rate)
FROM FILM
WHERE AVG(rental_rate)<50
GROUP BY film_id;
```

- 2) List stores which has more than 100 inactive customer (the value in “active” attribute is either YES or NO)