Chapter 4
Structure Query Language

SQL - Single row and Group Functions

Department: Computer
Course: DBMS
Faculty: Sana Shaikh

Learning Objectives:

- To explore the single-row functions and aggregate functions available in SQL.
- To apply various functions within the query statement.
- Practice Queries using Single-row functions, aggregate functions (COUNT, SUM, AVG, MAX and MIN), GROUP BY, and HAVING clause.

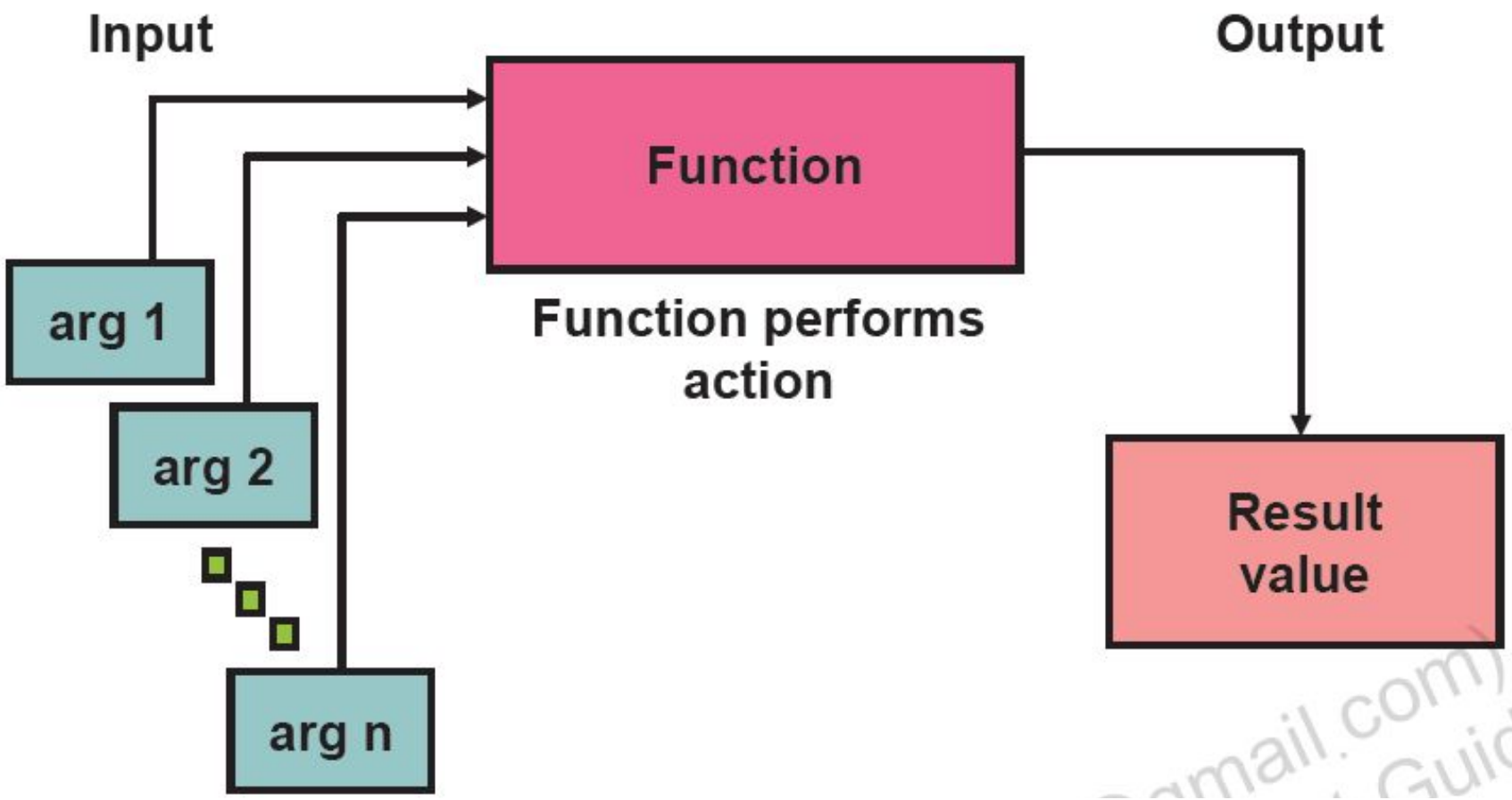
Learning Outcomes:

The student should be able to:

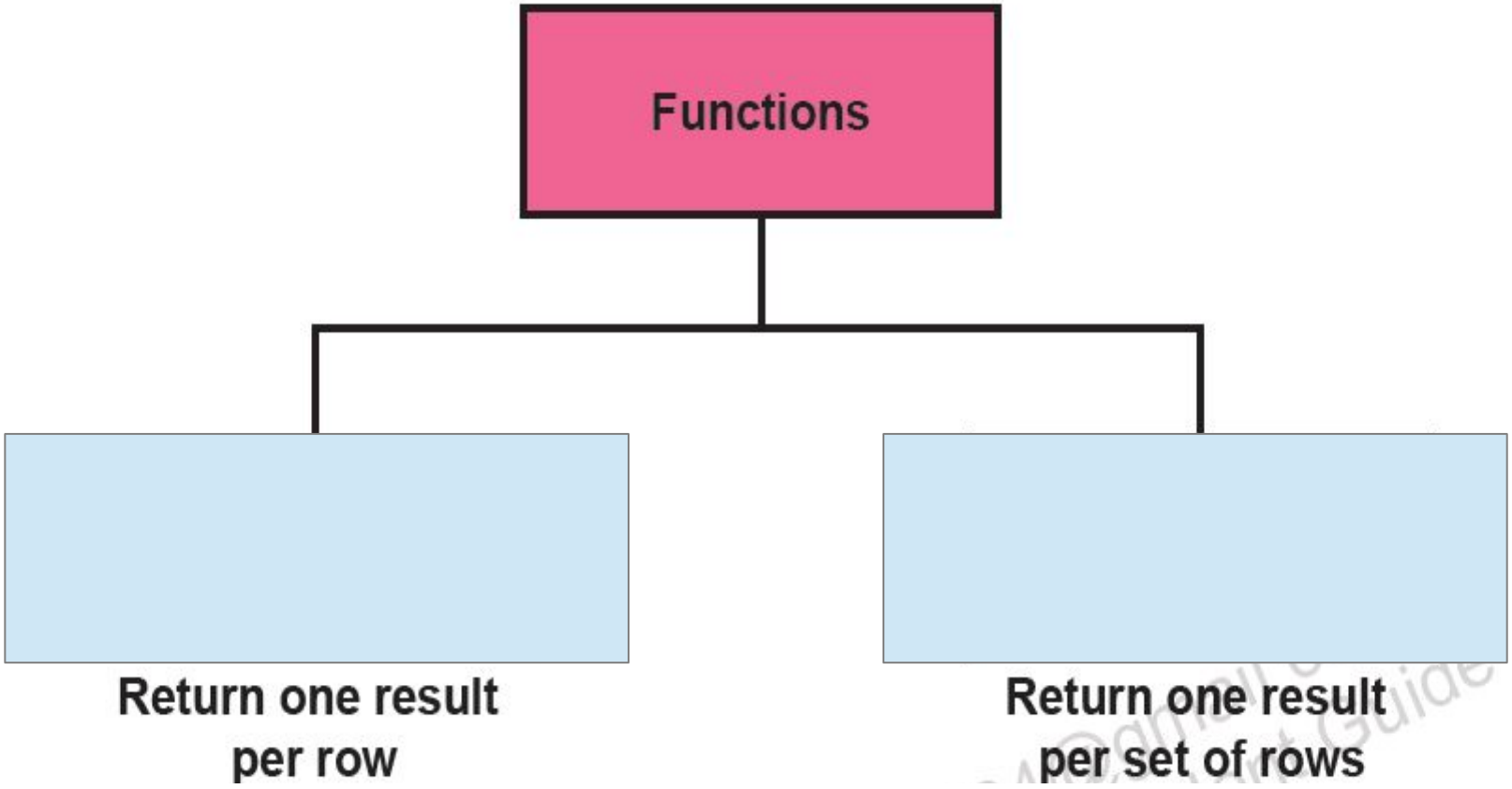
- Implement Single row functions in SQL statements
- Implement Aggregate functions in SQL statements
- Implement Order by, Group by and Having clause in SQL statements
- Explain and use SQL functions to manipulate dates, strings, and other data.
- Describe various types of functions available in SQL
- Use character, number, and date functions in SELECT statements
- Describe the use of conversion functions

Using Single-Row Functions to customize Output

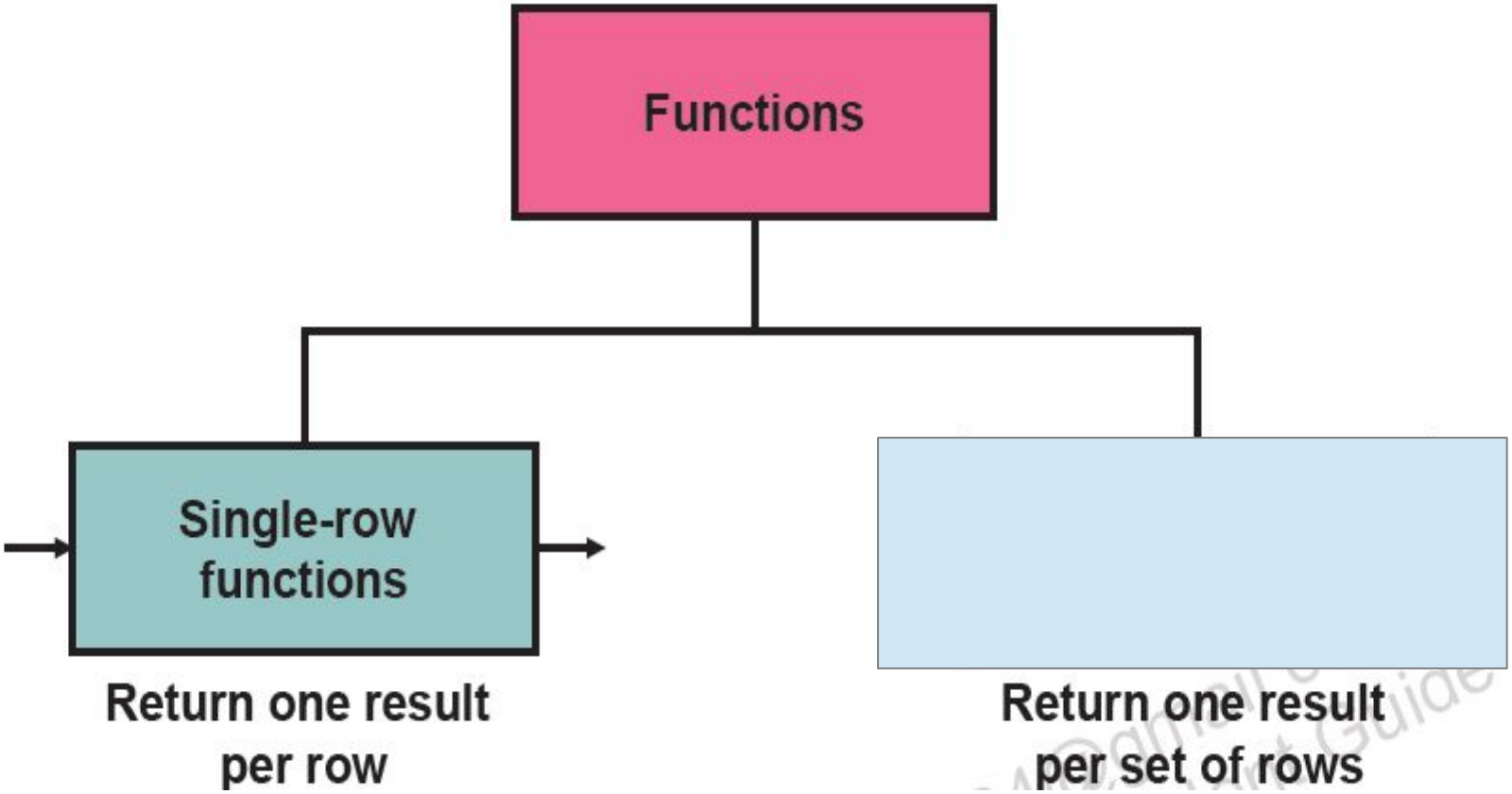
SQL Functions



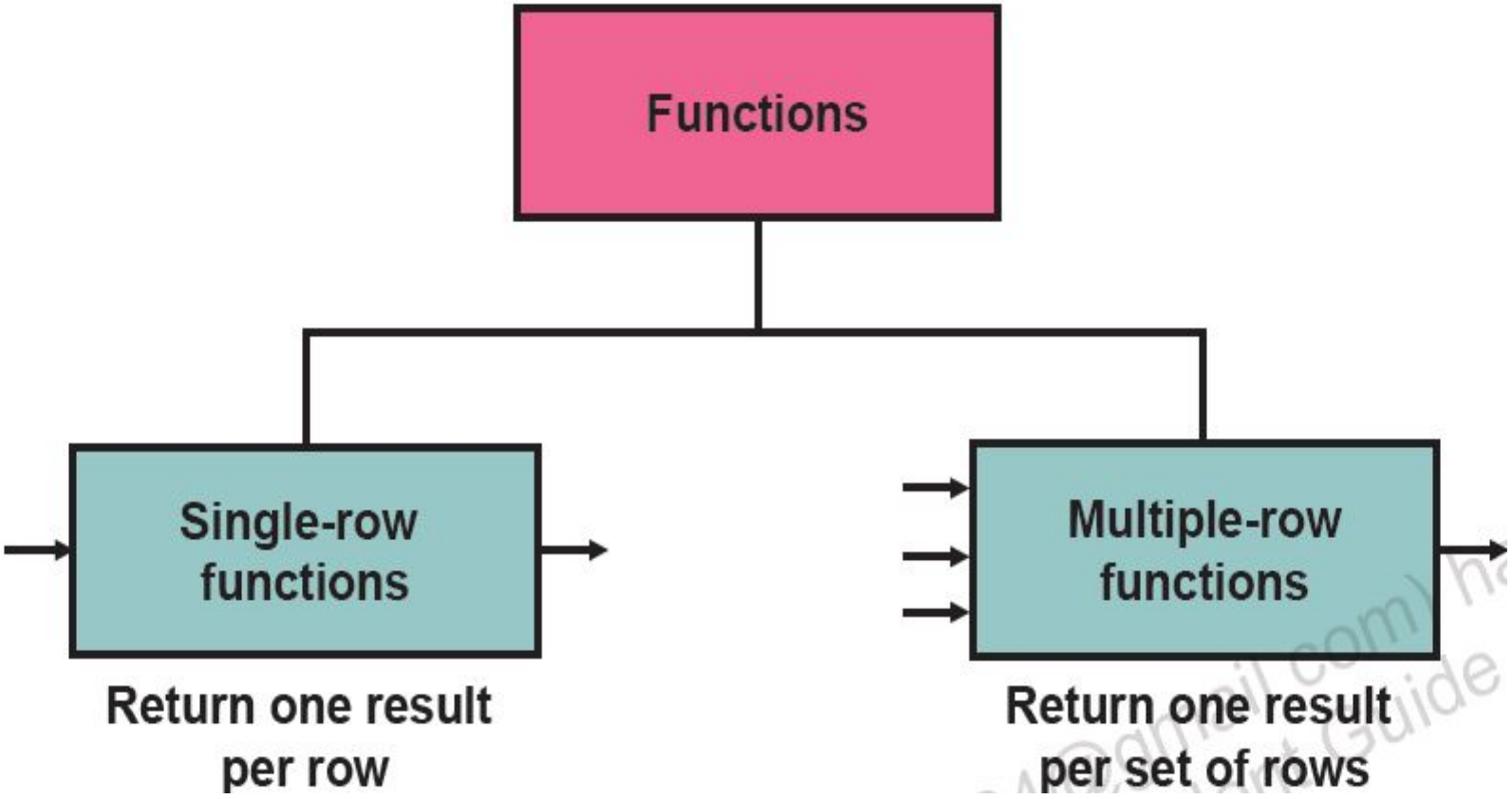
Two Types of SQL Functions



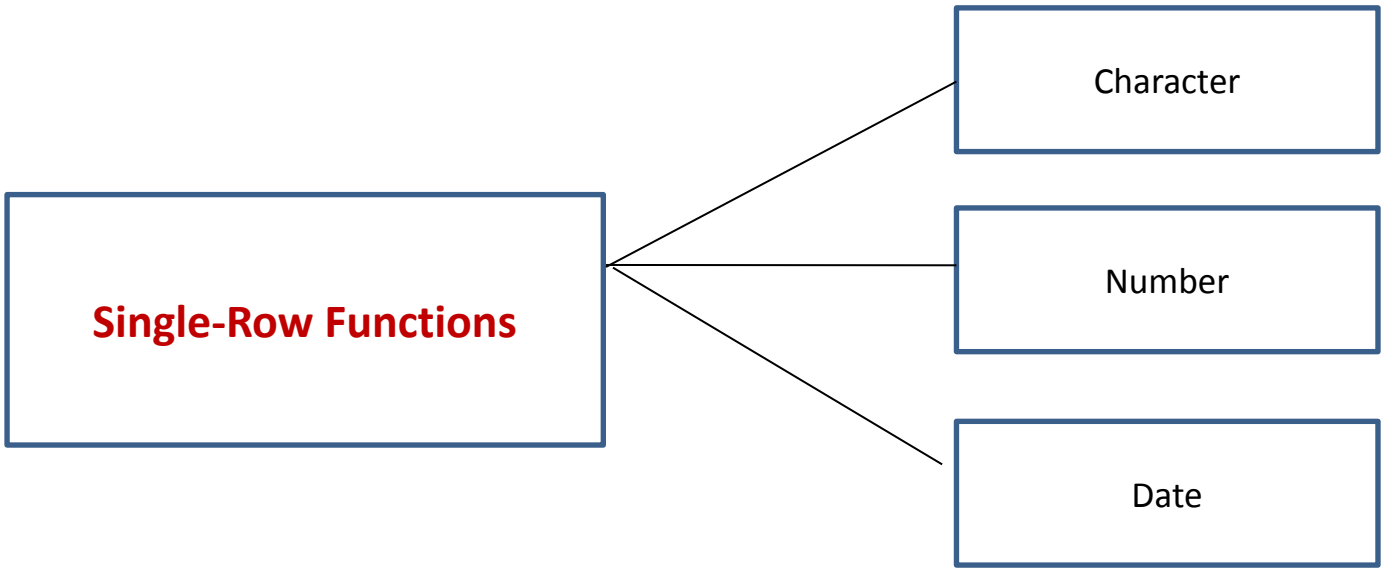
Two Types of SQL Functions



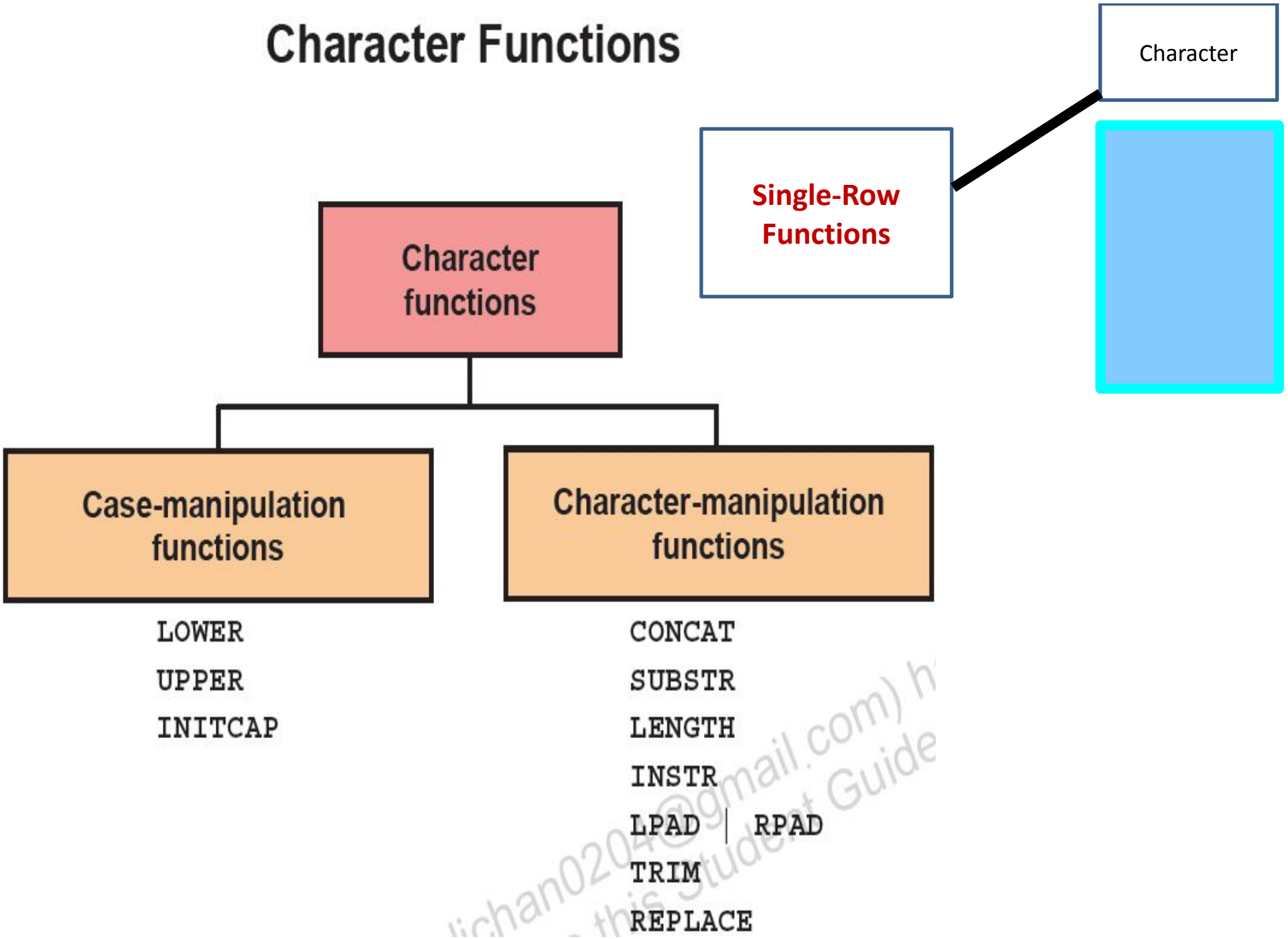
Two Types of SQL Functions



Single-Row Functions



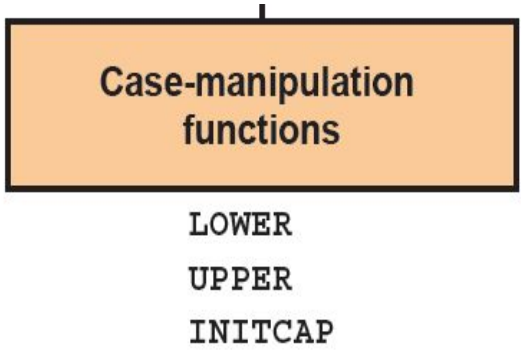
Character Functions



Case-Manipulation 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

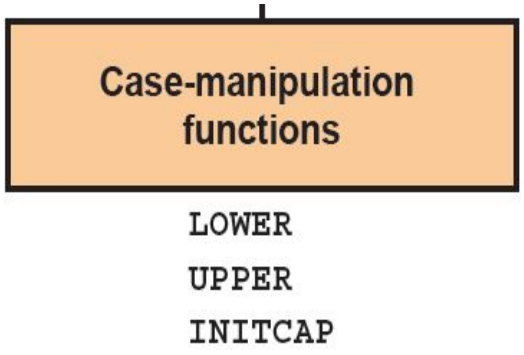


Display the employee number, name, and department number for employee Higgins:

Case-Manipulation 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



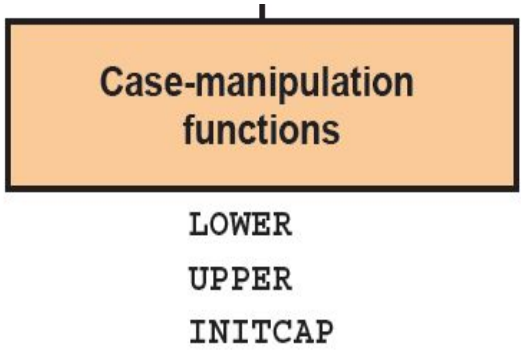
Display the employee number, name, and department number for employee Higgins:

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

Case-Manipulation 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



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
```

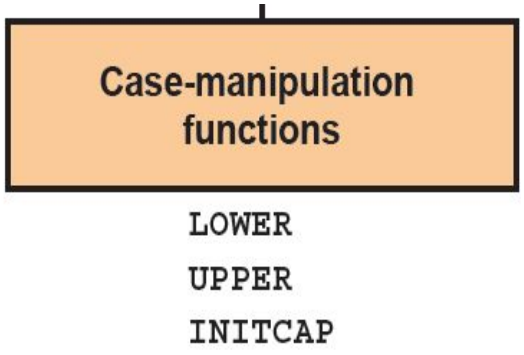
```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'higgins';
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205	Higgins	110

Case-Manipulation 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



NOTE:

- Actually, there is no single function in MySQL to capitalize only first letter of the string.
- We need to use nesting of functions and for this case, we can use UPPER() and LOWER() with SUBSTRING() and CONCAT() functions.

Character-Manipulation Functions

Character-manipulation functions

- CONCAT
- SUBSTR
- LENGTH
- INSTR
- LPAD | RPAD
- TRIM
- REPLACE

These functions manipulate character strings:

Function	Result
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
REPLACE ('JACK and JUE','J','BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

Write a query to display employee_id, first_name & last_name together give alias as NAME, length of last_name and check 'a' exist in last_name or not,if yes at which position(alias contains 'a'?) for those employees whose job_id contains REP which starts from 4th position.

Character-Manipulation Functions

Character-manipulation functions

- CONCAT
- SUBSTR
- LENGTH
- INSTR
- LPAD | RPAD
- TRIM
- REPLACE

These functions manipulate character strings:

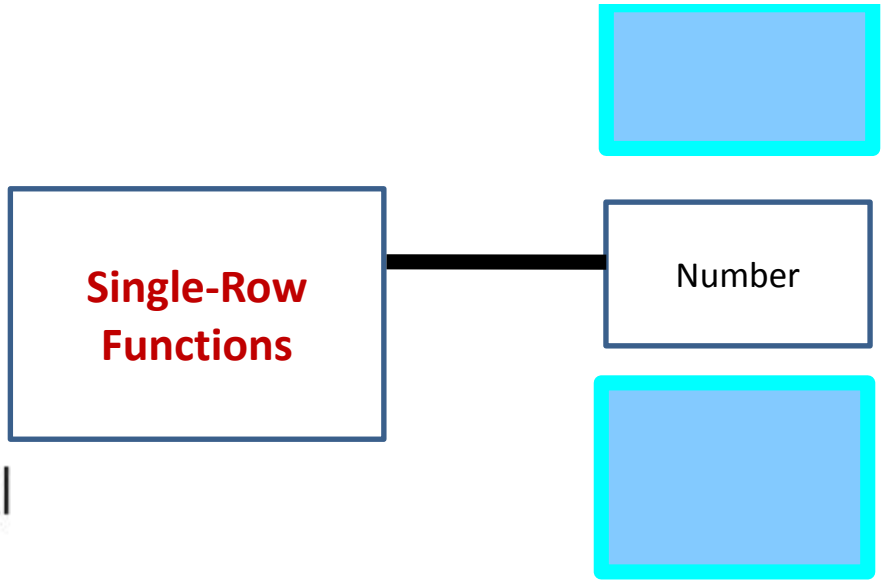
Function	Result
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
REPLACE ('JACK and JUE','J','BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

Write a query to display employee_id, first_name & last_name together give alias as NAME, length of last_name and check 'a' exist in last_name or not,if yes at which position(alias contains 'a'?) for those employees whose job_id contains REP which starts from 4th position.

```
SELECT  employee_id, CONCAT(first_name, last_name) NAME,
        job_id, LENGTH (last_name),
        INSTR(last_name, 'a') "Contains 'a'?"
FROM    employees
WHERE   SUBSTR(job_id, 4) = 'REP';
```

	EMPLOYEE_ID	NAME	JOB_ID	LENGTH(LAST_NAME)	Contains 'a'?
1	202	PatFay	MK_REP	3	2
2	174	EllenAbel	SA_REP	4	0
3	176	JonathonTaylor	SA_REP	6	2
4	178	KimberelyGrant	SA_REP	5	3

Number Functions



- ROUND: Rounds value to specified decimal
- TRUNC: Truncates value to specified decimal
- MOD: Returns remainder of division

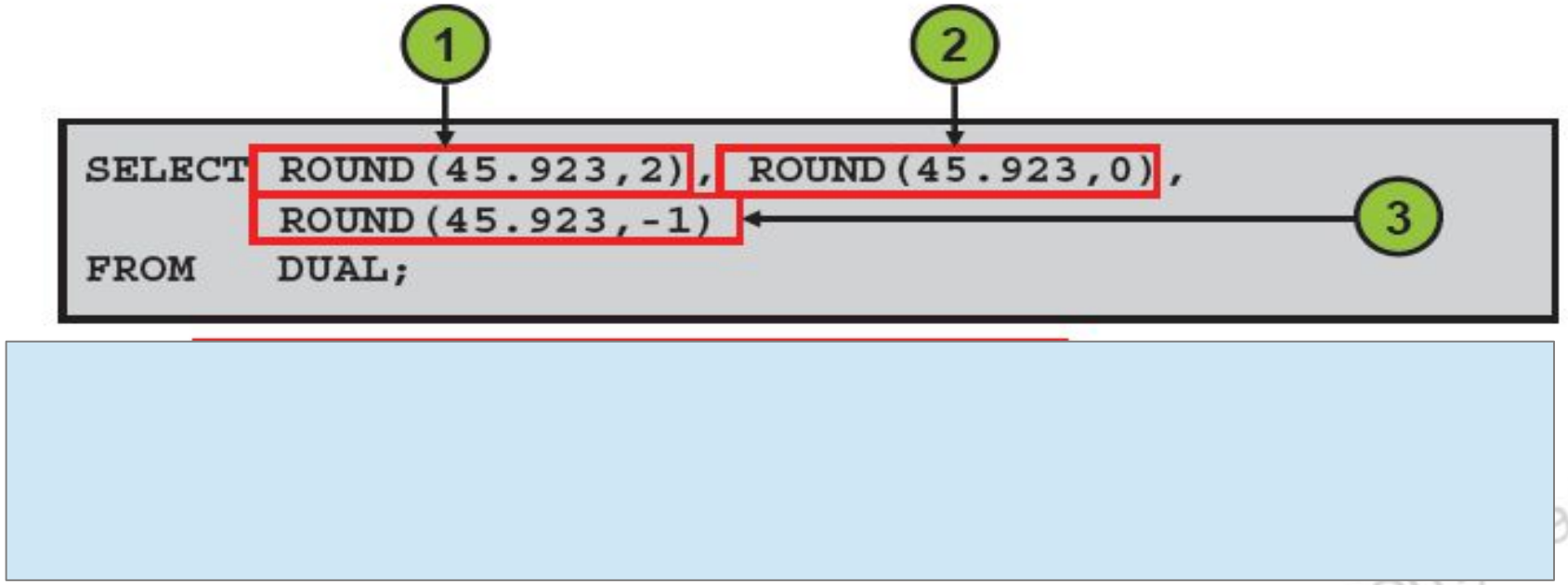
Function	Result
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
MOD(1600, 300)	100

ABS(), CEIL(), FLOOR(), …etc.

ROUND Function

The ROUND function rounds the column, expression, or value to n decimal places. If the second argument is 0 or is missing, the value is rounded to zero decimal places. If the second argument is 2, the value is rounded to two decimal places. Conversely, if the second argument is -2, the value is rounded to two decimal places to the left.

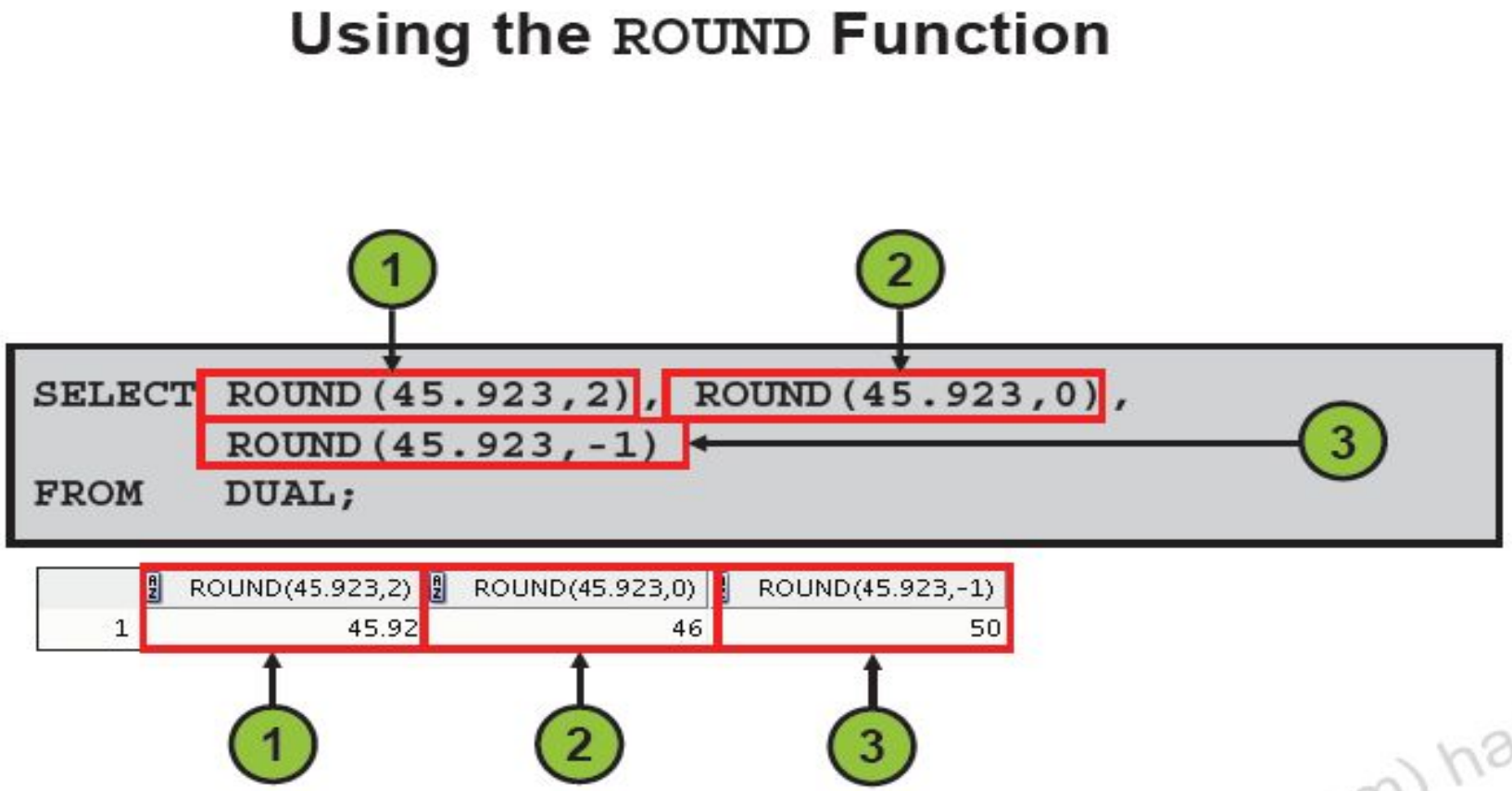
Using the ROUND Function



DUAL is a dummy table that you can use to view results from functions and calculations.

ROUND Function

The ROUND function rounds the column, expression, or value to n decimal places. If the second argument is 0 or is missing, the value is rounded to zero decimal places. If the second argument is 2, the value is rounded to two decimal places. Conversely, if the second argument is -2, the value is rounded to two decimal places to the left.



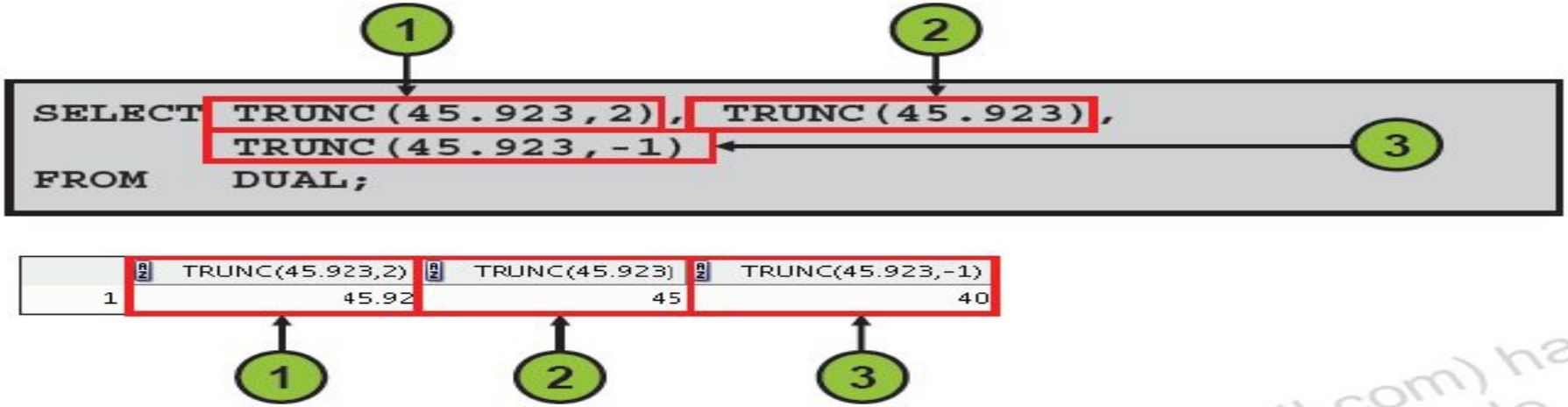
TRUNC/ TRUNCATE Function

The TRUNC function truncates the column, expression, or value to n decimal places. The TRUNC function works with arguments similar to those of the ROUND function. If the second argument is 0 or is missing, the value is truncated to zero decimal places. If the second argument is 2, the value is truncated to two decimal places.

MOD Function

The MOD function finds the remainder of value1 divided by value2.

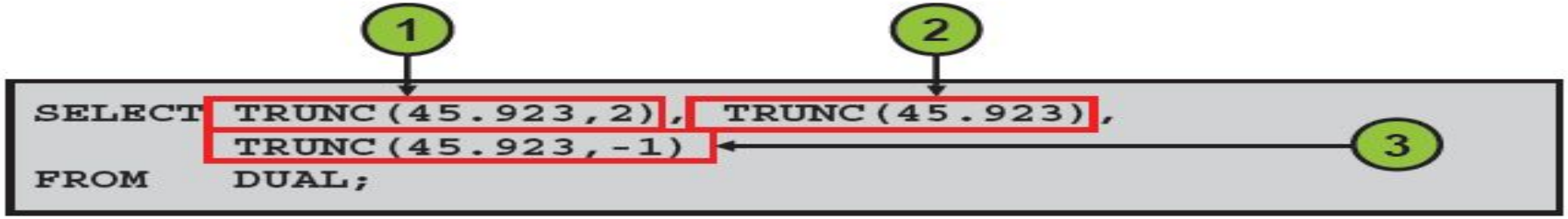
Using the TRUNC Function



Using the MOD Function

For all employees with job title of Sales Representative, calculate the remainder of the salary after it is divided by 5,000.

Using the TRUNC Function



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

1 2 3

Using the MOD Function

For all employees with job title of Sales Representative, calculate the remainder of the salary after it is divided by 5,000.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM   employees
WHERE  job_id = 'SA_REP';
```

	LAST_NAME	SALARY	MOD(SALARY,5000)
1	Abel	11000	1000
2	Taylor	8600	3600
3	Grant	7000	2000

Working with Dates

SYSDATE is a function that returns:

- Date
- Time

Example

Display the current date using the DUAL table.

```
SELECT  SYSDATE
FROM    DUAL;
```

	SYSDATE
1	06-NOV-08

In MySQL,
Curdate()
CurTime()

Arithmetic with Dates

- Add or subtract a number to or from a date for a resultant date value.
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24.

Arithmetic with Dates

- Add or subtract a number to or from a date for a resultant date value.
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24.

You can perform the following operations:

Operation	Result	Description
date + number	Date	Adds a number of days to a date
date – number	Date	Subtracts a number of days from a date
date – date	Number of days	Subtracts one date from another
date + number/24	Date	Adds a number of hours to a date

Using Date Functions

Function	Result
MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

In MySQL Date functions,

```
SELECT DATEDIFF("2017-06-25", "2017-06-15");
SELECT DATE_ADD("2017-06-15", INTERVAL 10 DAY);
SELECT DATE_FORMAT("2017-06-15", "%Y");
SELECT DATE_SUB("2017-06-15", INTERVAL 10 DAY);
SELECT MAKEDATE(2017, 3);
```

Day, DayName, DayofWeek, DayofYear etc..

Using Arithmetic Operators with Dates

The example in the slide displays the last name and the number of weeks employed for all employees in department 90.

Using Arithmetic Operators with Dates

The example in the slide displays the last name and the number of weeks employed for all employees in department 90. It subtracts the date on which the employee was hired from the current date (SYSDATE) and divides the result by 7 to calculate the number of weeks that a worker has been employed.

Using Arithmetic Operators with Dates

The example in the slide displays the last name and the number of weeks employed for all employees in department 90. It subtracts the date on which the employee was hired from the current date (SYSDATE) and divides the result by 7 to calculate the number of weeks that a worker has been employed.

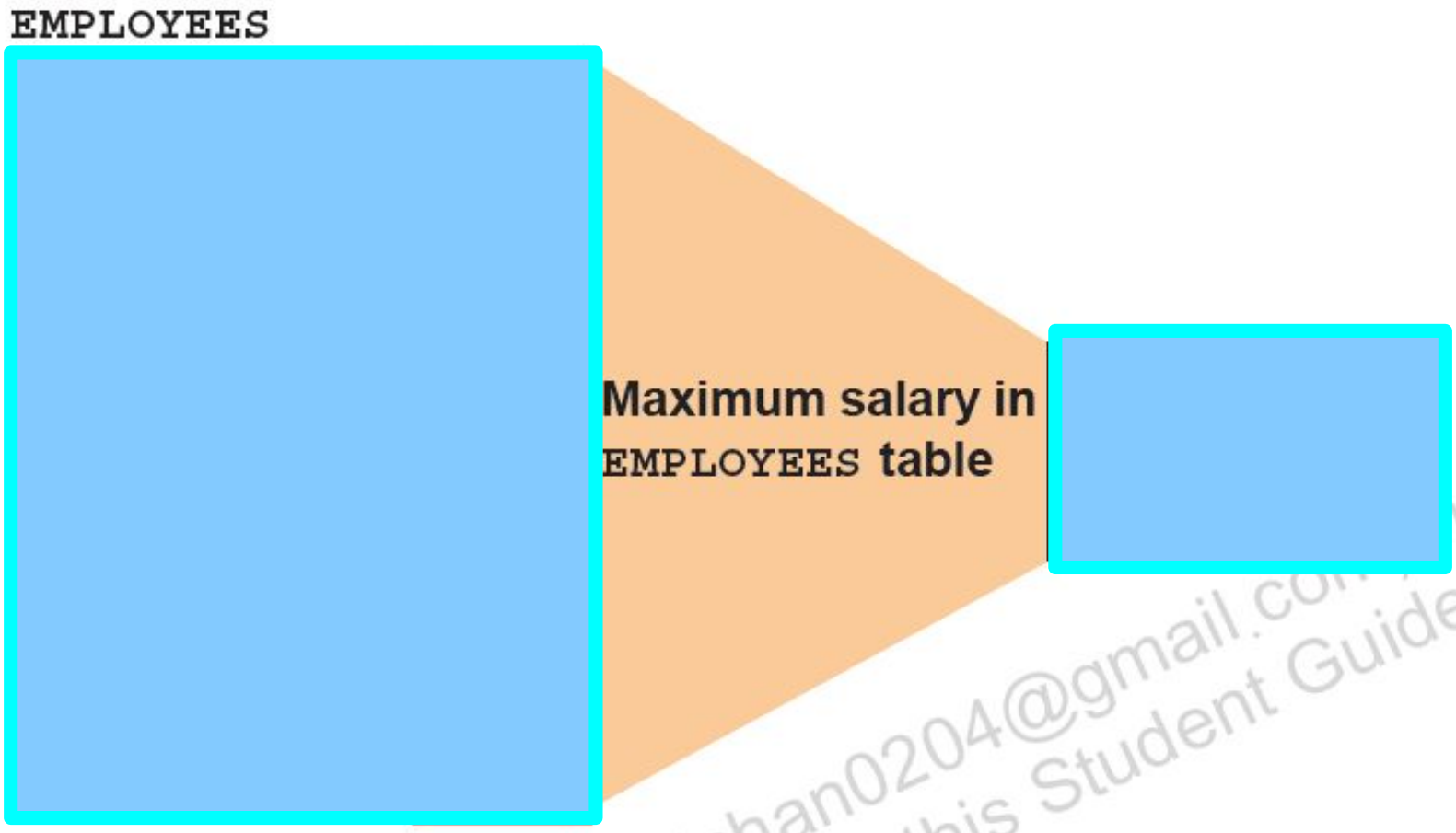
```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM   employees
WHERE  department_id = 90;
```

	LAST_NAME	WEEKS
1	King	1116.148574735449735449735449735449735449735
2	Kochhar	998.005717592592592592592592592592592592593
3	De Haan	825.148574735449735449735449735449735449735

Reporting Aggregated Data using the Group Functions

What Are Group Functions?

Group functions operate on sets of rows to give one result per group.

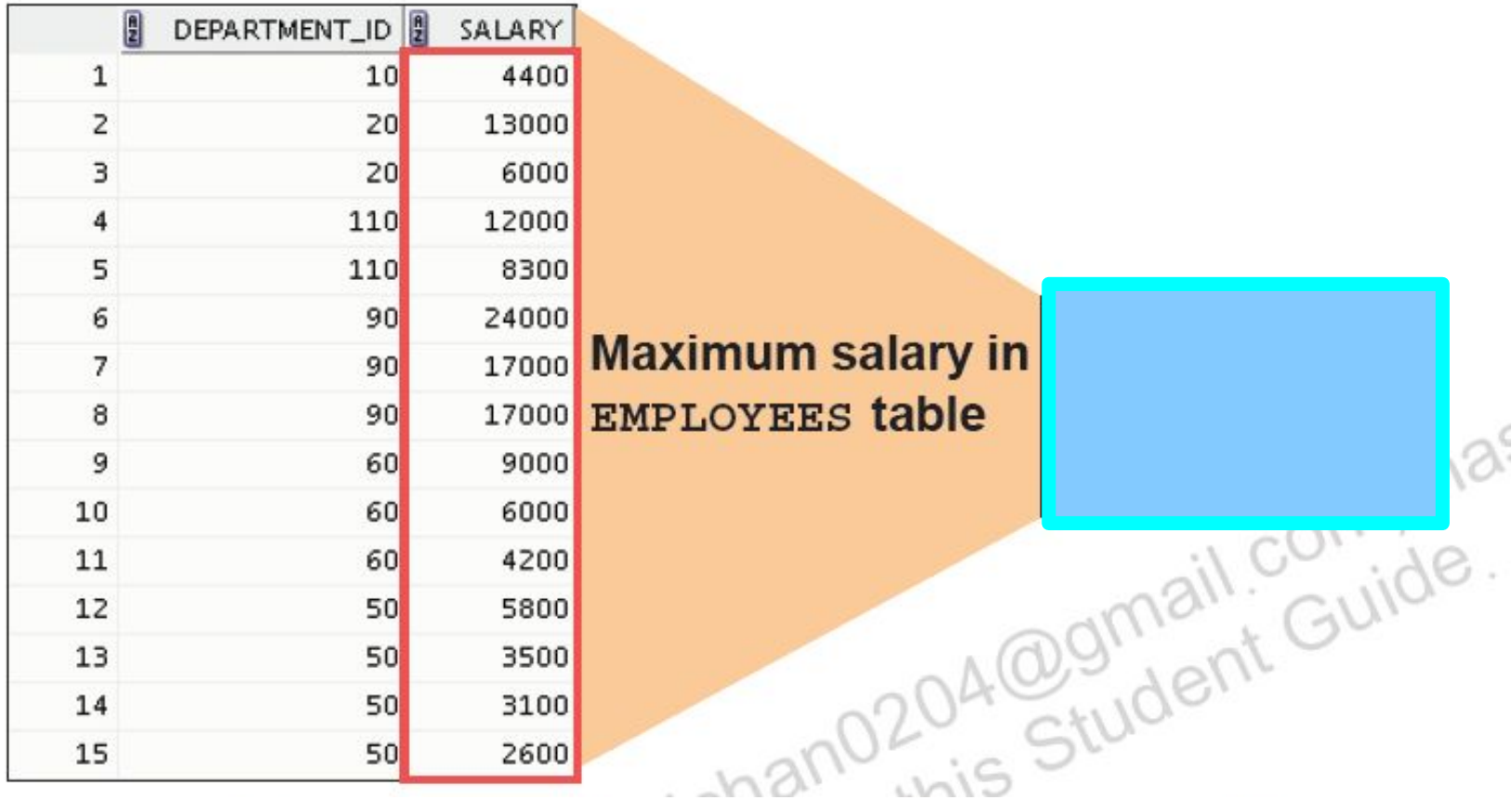


What Are Group Functions?

Group functions operate on sets of rows to give one result per group.

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
11	60	4200
12	50	5800
13	50	3500
14	50	3100
15	50	2600



What Are Group Functions?

Group functions operate on sets of rows to give one result per group.

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
11	60	4200
12	50	5800
13	50	3500
14	50	3100
15	50	2600

Maximum salary in EMPLOYEES table

	MAX(SALARY)
1	24000

Types of Group Functions

1. Avg
2. Min
3. Max
4. Sum
5. Count

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)
1	8150	11000	6000	32600

Using the MIN and MAX Functions

You can use MIN and MAX for numeric, character, and date data types.

```
SELECT MIN(hire date), MAX(hire date)
FROM       employees;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	17-JUN-87	29-JAN-00

Using the COUNT Function

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

1
```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(*)
1	5

COUNT(expr) returns the number of rows with non-null values for expr:

2
```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 80;
```

	COUNT(COMMISSION_PCT)
1	3

Using the DISTINCT Keyword

- COUNT(DISTINCT *expr*) returns the number of distinct non-null values of *expr*.
- To display the number of distinct department values in the EMPLOYEES table:

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

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

has

Group Functions and Null Values

Group functions ignore null values in the column:

1
```
SELECT AVG(commission_pct)
FROM   employees;
```

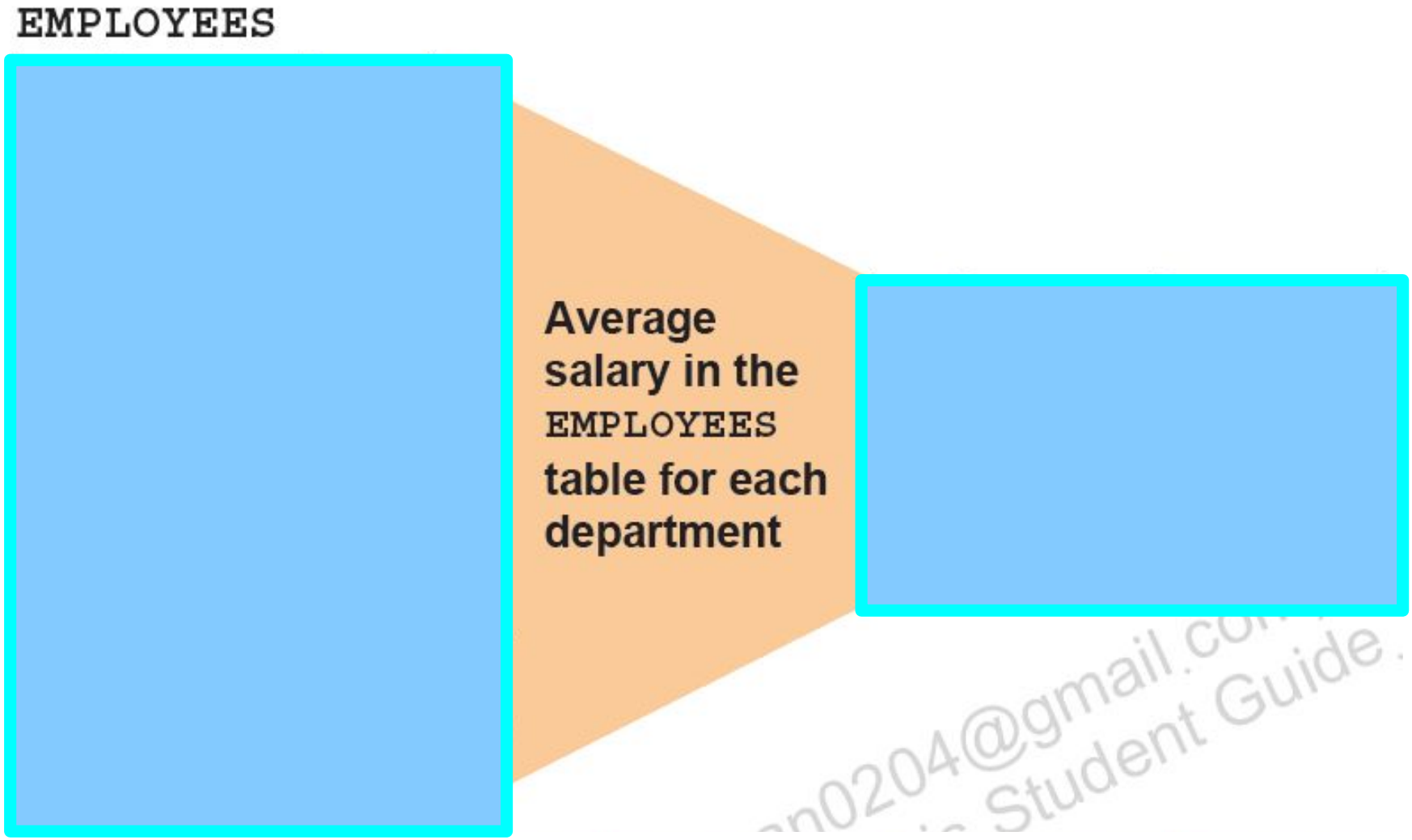
	AVG(COMMISSION_PCT)
1	0.2125

The NVL function forces group functions to include null values:

2
```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

Creating Groups of Data



Creating Groups of Data

EMPLOYEES

	DEPARTMENT_ID	SALARY	
1	10	4400	4400
2	20	13000	9500
3	20	6000	
4	50	2500	
5	50	2600	
6	50	3100	3500
7	50	3500	
8	50	5800	
9	60	9000	
10	60	6000	6400
11	60	4200	
12	80	11000	
13	80	8600	10033
14	80	10500	
15	90	17000	
16	90	24000	19333
17	90	17000	
18	110	8300	10150
19	110	12000	
20	(null)	7000	7000

Average salary in the EMPLOYEES table for each department

Creating Groups of Data

EMPLOYEES

	DEPARTMENT_ID	SALARY	
1	10	4400	4400
2	20	13000	9500
3	20	6000	
4	50	2500	
5	50	2600	
6	50	3100	3500
7	50	3500	
8	50	5800	
9	60	9000	
10	60	6000	6400
11	60	4200	
12	80	11000	
13	80	8600	10033
14	80	10500	
15	90	17000	
16	90	24000	19333
17	90	17000	
18	110	8300	10150
19	110	12000	
20	(null)	7000	7000

Average salary in the EMPLOYEES table for each department

	DEPARTMENT_ID	AVG(SALARY)
1	10	4400
2	20	9500
3	50	3500
4	60	6400
5	80	10033.333333333333...
6	90	19333.333333333333...
7	110	10150
8	(null)	7000

Creating Groups of Data: GROUP BY Clause Syntax

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

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

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	20	9500
3	90	19333.333333333333...
4	110	10150
5	50	3500
6	80	10033.333333333333...
7	10	4400
8	60	6400

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)
1	7000
2	9500
3	19333.333333333333333333333333333333333
4	10150
5	3500
6	10033.333333333333333333333333333333333
7	4400
8	6400

Using the GROUP BY Clause on Multiple Columns

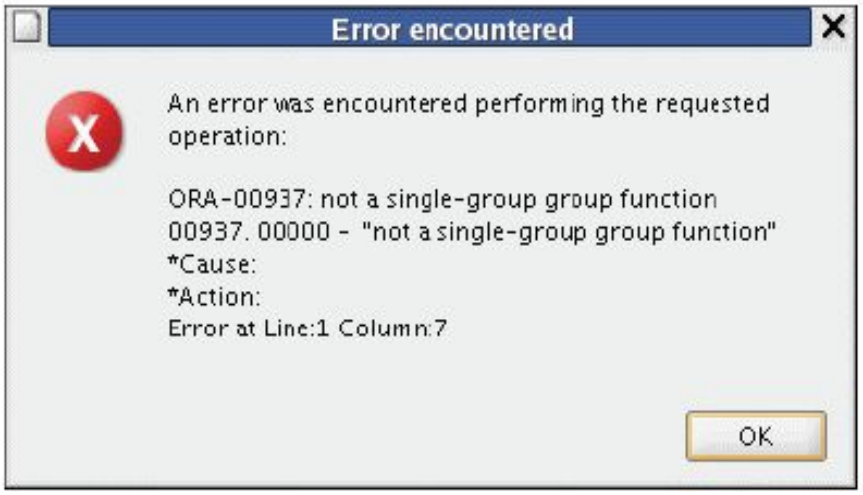
```
SELECT    department_id dept_id, job_id, SUM(salary)
FROM      employees
GROUP BY department_id, job_id ;
```

	DEPT_ID	JOB_ID	SUM(SALARY)
1	110	AC_ACCOUNT	8300
2	90	AD_VP	34000
3	50	ST_CLERK	11700
4	80	SA_REP	19600
5	110	AC_MGR	12000
6	50	ST_MAN	5800
7	80	SA_MAN	10500
8	20	MK_MAN	13000
9	90	AD_PRES	24000
10	60	IT_PROG	19200
11	(null)	SA_REP	7000
12	10	AD_ASST	4400
13	20	MK_REP	6000

Illegal Queries Using Group Functions

Any column or expression in the `SELECT` list that is not an aggregate function must be in the `GROUP BY` clause:

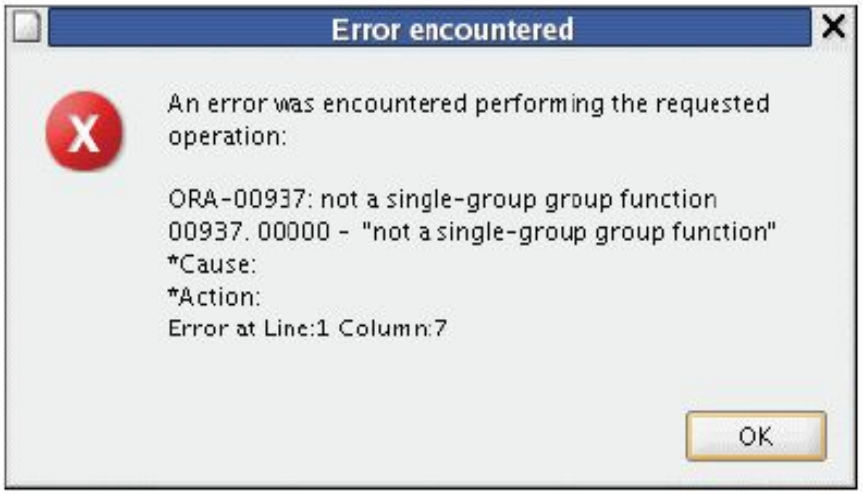
```
SELECT department_id, COUNT(last_name)
FROM   employees;
```



Illegal Queries Using Group Functions

Any column or expression 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;
```

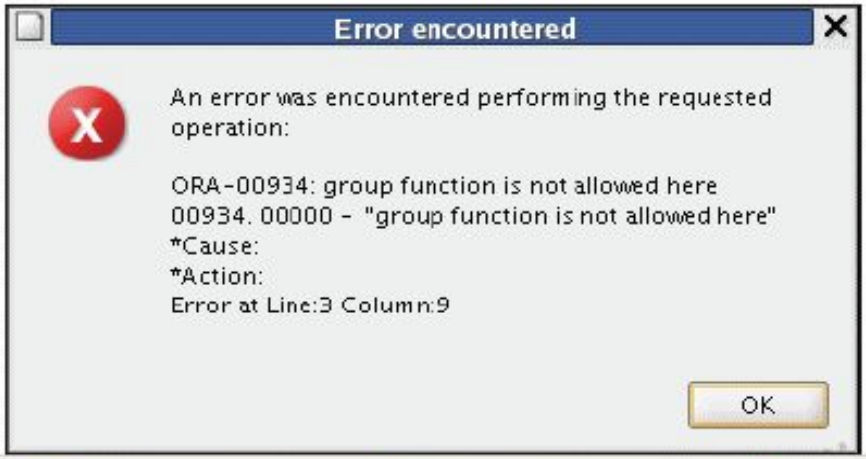


Column missing in the GROUP BY clause

Illegal Queries Using Group Functions

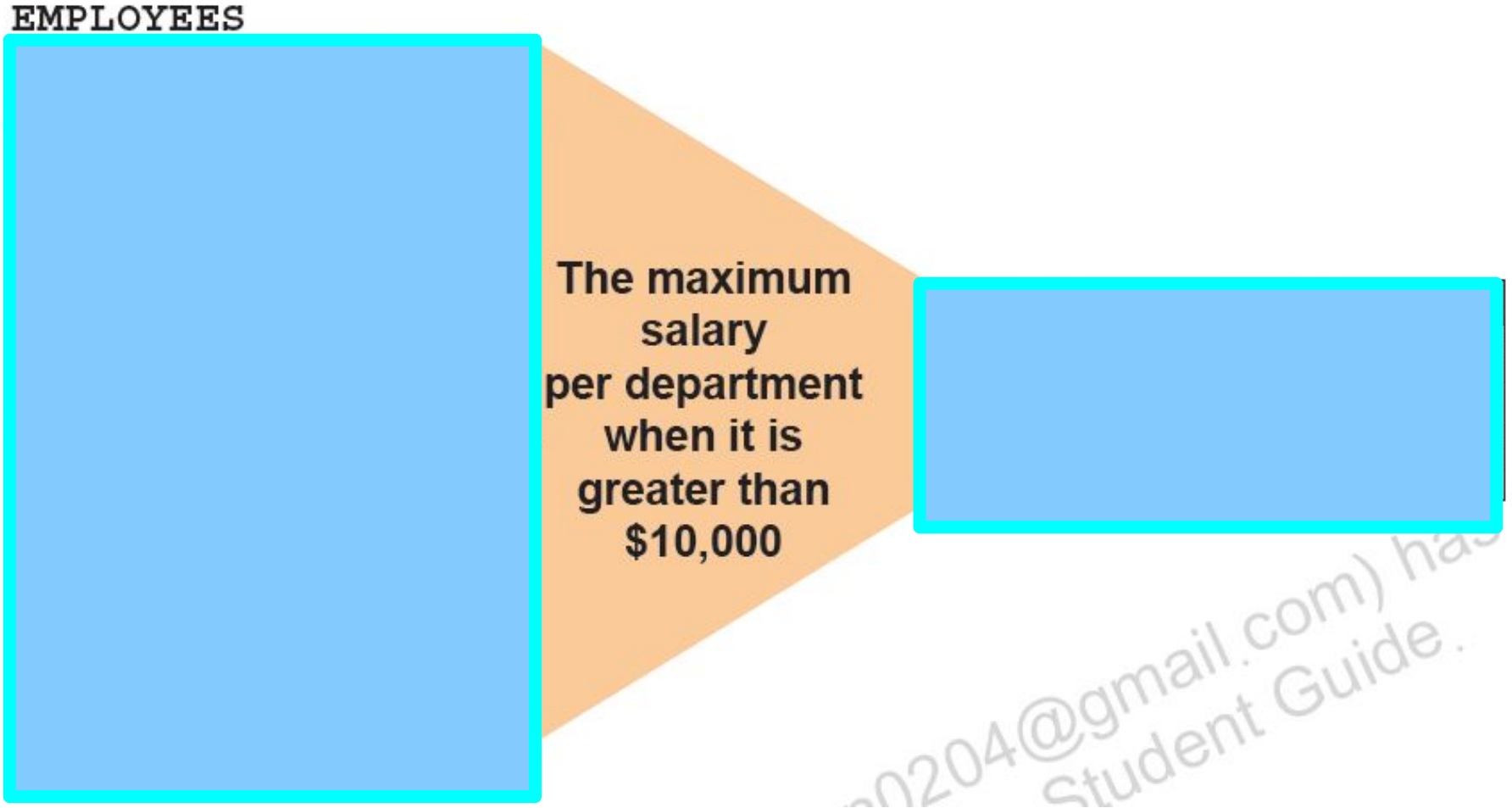
- You cannot use the `WHERE` clause to restrict groups.
- You use the `HAVING` clause to restrict groups.
- You cannot use group functions in the `WHERE` clause.

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



Cannot use the `WHERE` clause to restrict groups

Restricting Group Results



Restricting Group Results

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
11	60	4200
12	50	5800
13	50	3500
14	50	3100
15	50	2600

...

The maximum salary per department when it is greater than $10,000

Restricting Group Results

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
11	60	4200
12	50	5800
13	50	3500
14	50	3100
15	50	2600

...

The maximum salary per department when it is greater than $10,000

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	80	11000
3	90	24000
4	110	12000

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

The maximum salary per department when it is greater than $10,000

Using the HAVING Clause

Using the HAVING Clause

The maximum salary per department when it is greater than $10,000

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

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

Using the HAVING Clause (continued)

The slide example displays the job ID and total monthly salary for each job that has a total payroll exceeding $13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

Using the HAVING Clause (continued)

The slide example displays the job ID and total monthly salary for each job that has a total payroll exceeding $13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

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

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

Nesting Group Functions

Display the maximum average salary:

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

	MAX(AVG(SALARY))
1	19333.333333333333333333333333333333333333

Nesting Group Functions

Group functions can be nested to a depth of two. The slide example displays the maximum average salary.