

# 3

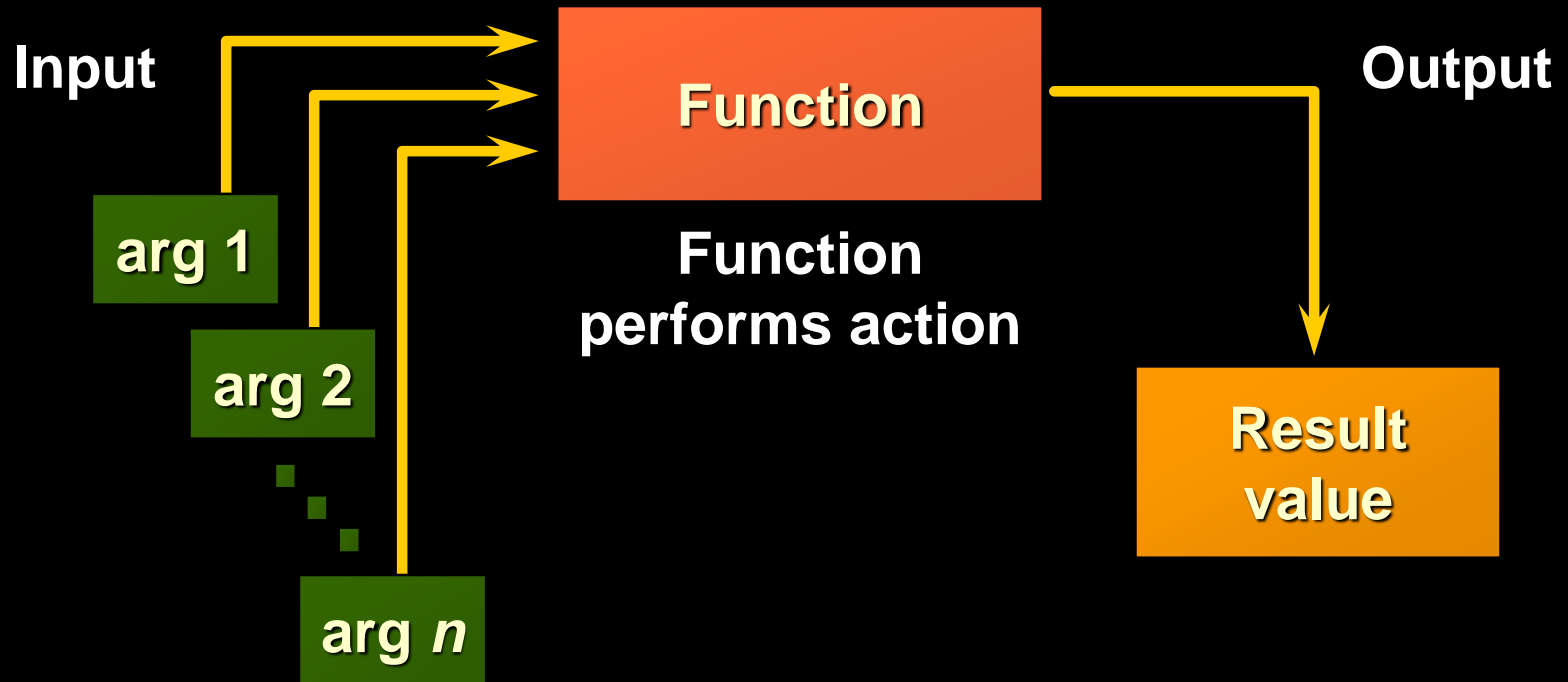
## Single-Row Functions

# Objectives

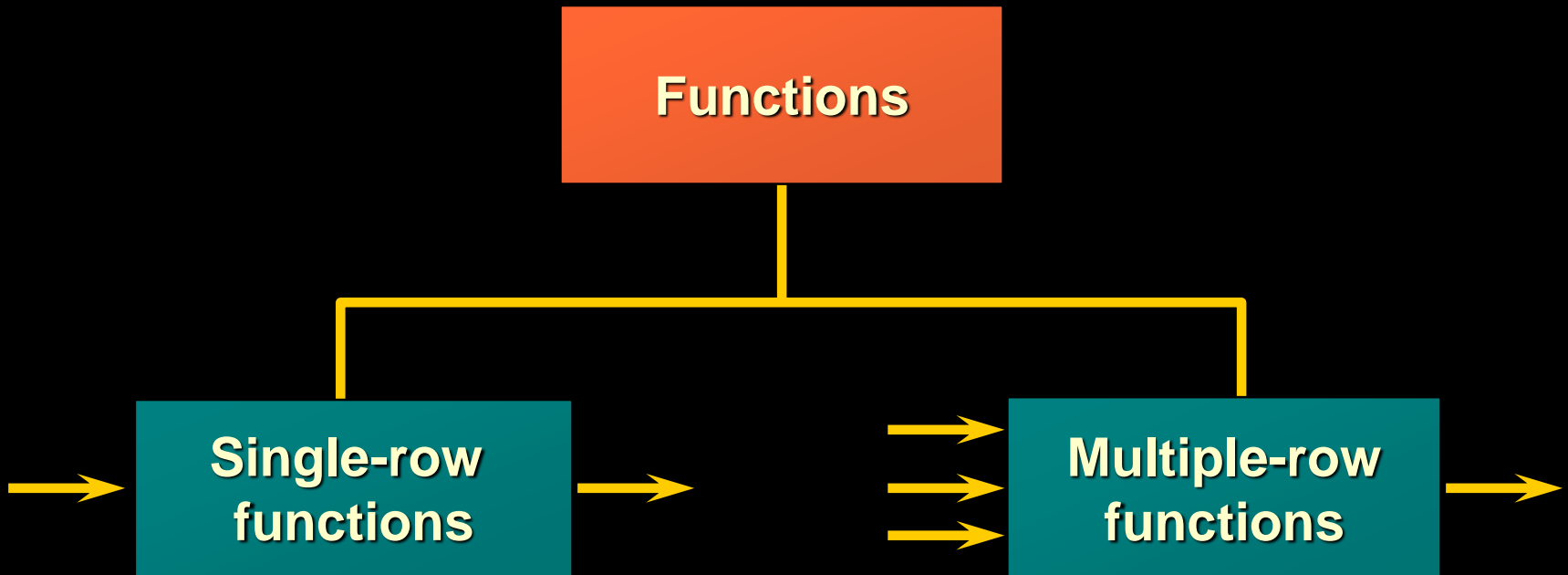
**After completing this lesson, you should be able to do the following:**

- **Describe various types of functions available in SQL**
- **Use character, number, and date functions in `SELECT` statements**
- **Describe the use of conversion functions**

# SQL Functions



# Two Types of SQL Functions



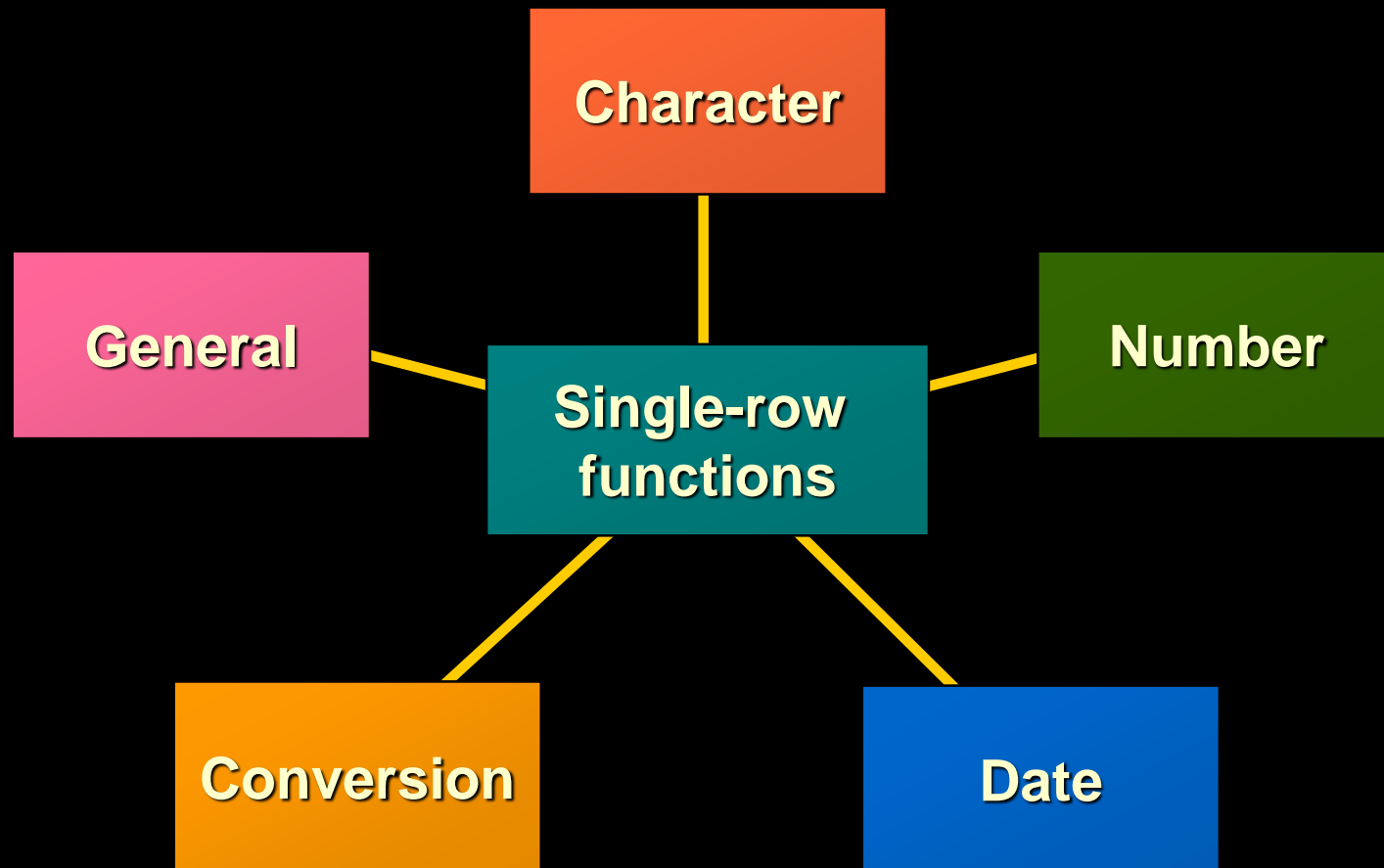
# Single-Row Functions

## Single row functions:

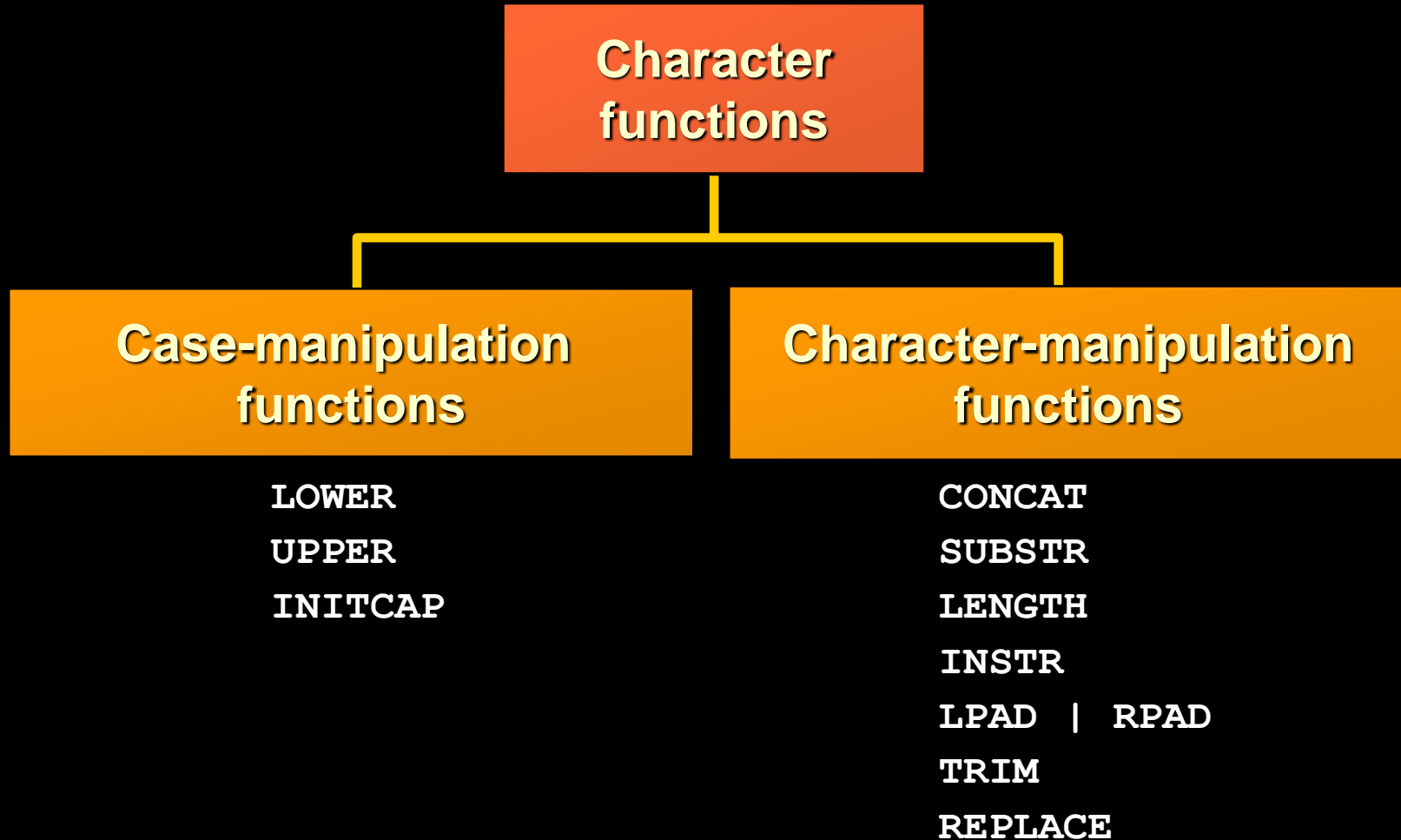
- Manipulate data items
- Accept arguments and return one value
- Act on each row returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments which can be a column or an expression

```
function_name [(arg1, arg2, ...)]
```

# Single-Row Functions



# Character Functions



# Case Manipulation Functions

These functions convert case for character strings.

Function	Result
<code>LOWER('SQL Course')</code>	sql course
<code>UPPER('SQL Course')</code>	SQL COURSE
<code>INITCAP('SQL Course')</code>	Sql Course



# Using Case Manipulation Functions

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
205	Higgins	110

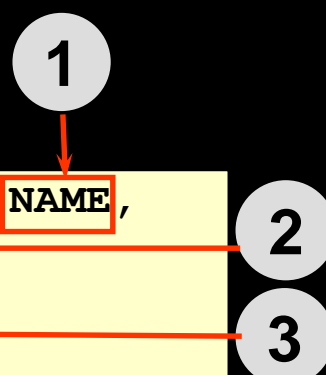
# Character-Manipulation Functions

These functions manipulate character strings:

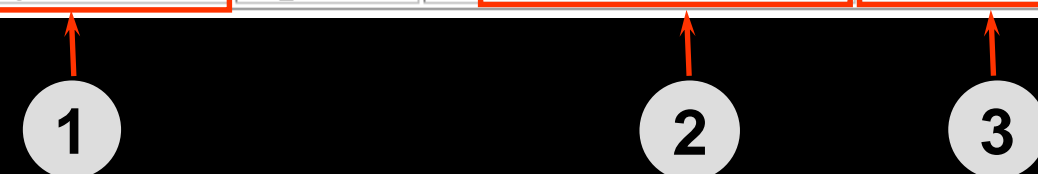
Function	Result
<code>CONCAT('Hello', 'World')</code>	HelloWorld
<code>SUBSTR('HelloWorld',1,5)</code>	Hello
<code>LENGTH('HelloWorld')</code>	10
<code>INSTR('HelloWorld', 'W')</code>	6
<code>LPAD(salary,10,'*')</code>	*****24000
<code>RPAD(salary, 10, '*')</code>	24000*****
<code>TRIM('H' FROM 'HelloWorld')</code>	elloWorld

# Using the Character-Manipulation Functions

```
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'?
174	EllenAbel	SA_REP	4	0
176	JonathonTaylor	SA_REP	6	2
178	KimberelyGrant	SA_REP	5	3
202	PatFay	MK_REP	3	2



# Number Functions


- **ROUND:** Rounds value to specified decimal

`ROUND (45.926, 2)`  `45.93`

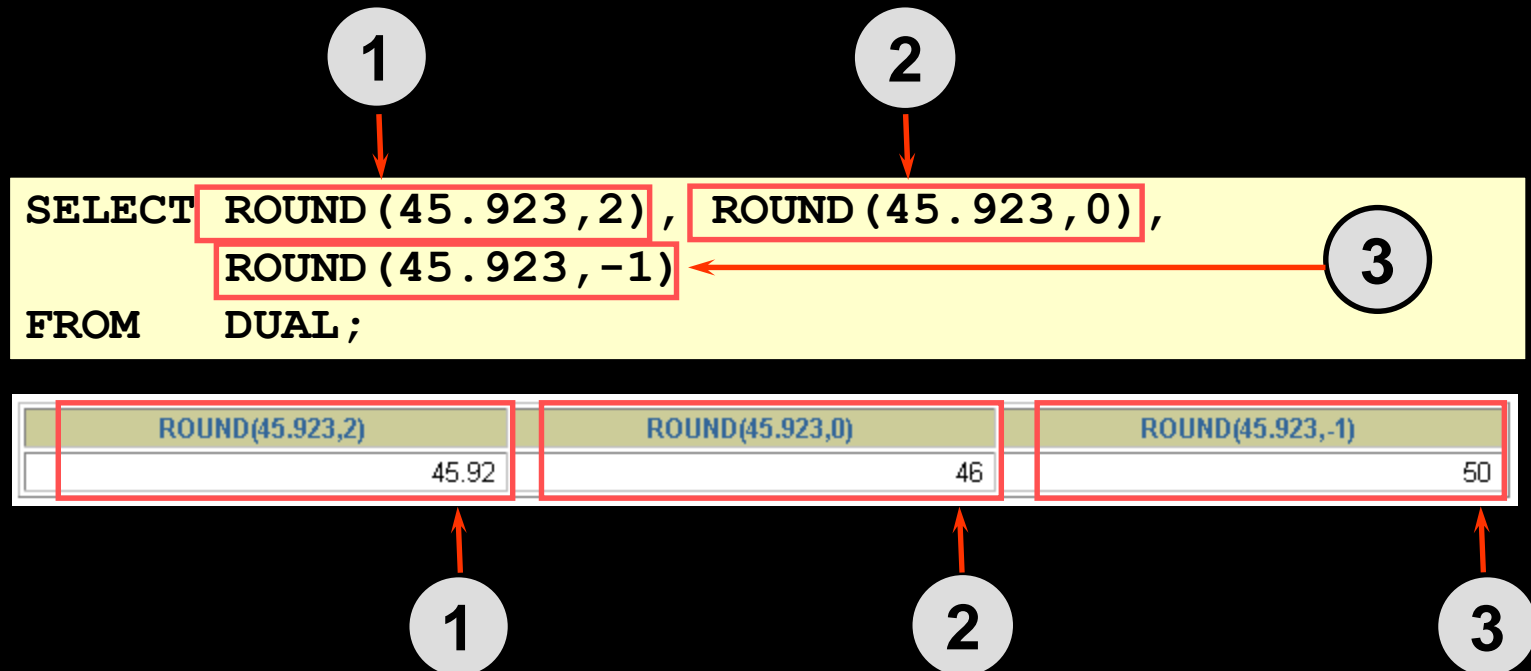
- **TRUNC:** Truncates value to specified decimal

`TRUNC (45.926, 2)`  `45.92`

- **MOD:** Returns remainder of division

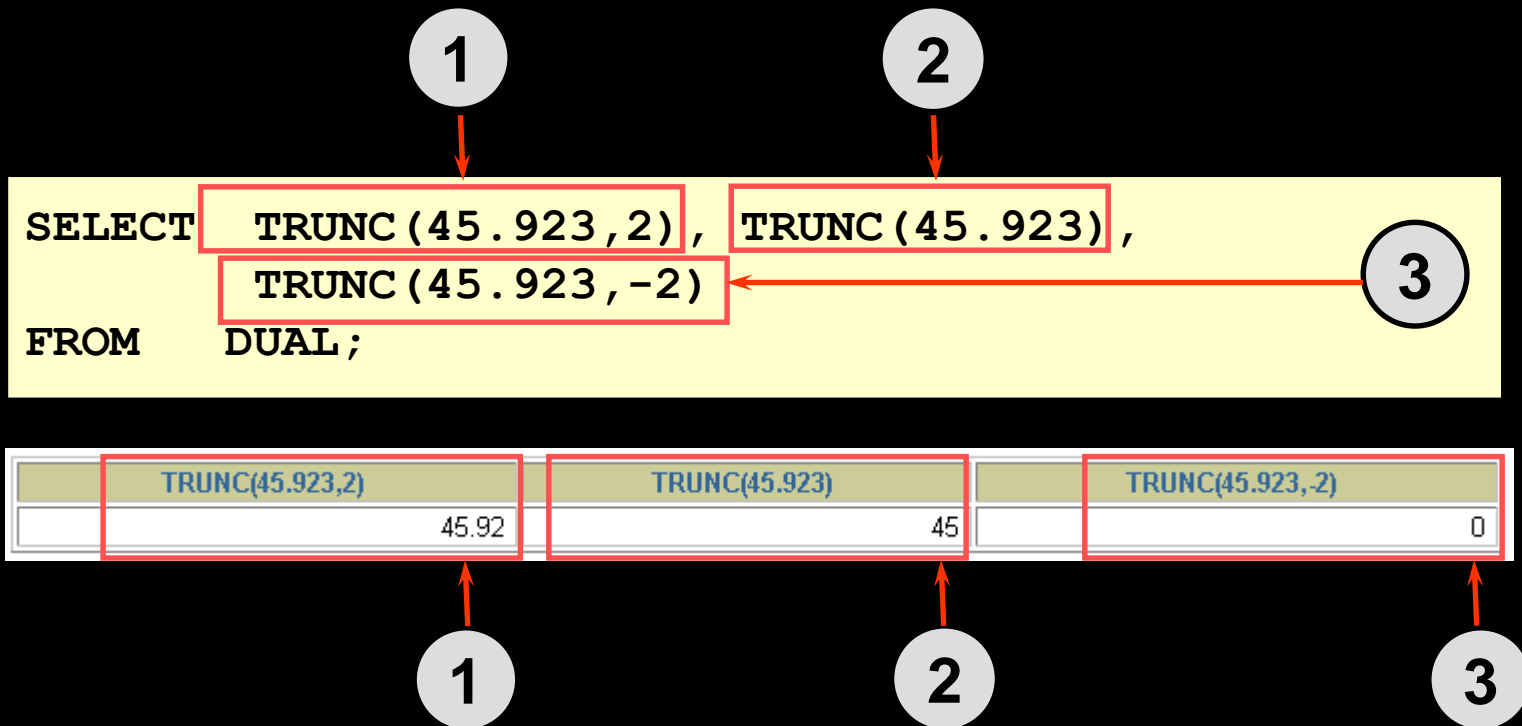
`MOD (1600, 300)`  `100`

# Using the ROUND Function



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

# Using the TRUNC Function



# Using the MOD Function

Calculate the remainder of a salary after it is divided by 5000 for all employees whose job title is sales representative.

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

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

# Working with Dates

- Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, seconds.
- The default date display format is DD-MON-RR.
  - Allows you to store 21st century dates in the 20th century by specifying only the last two digits of the year.
  - Allows you to store 20th century dates in the 21st century in the same way.

```
SELECT last_name, hire_date
FROM   employees
WHERE  last_name like 'G%';
```

LAST_NAME	HIRE_DATE
Gietz	07-JUN-94
Grant	24-MAY-99



# Working with Dates

**SYSDATE** is a function that returns:

- **Date**
- **Time**

# 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.**

# Using Arithmetic Operators with Dates

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

LAST_NAME	WEEKS
King	744.245395
Kochhar	626.102538
De Haan	453.245395

# Date Functions

Function	Description
<b>MONTHS_BETWEEN</b>	Number of months between two dates
<b>ADD_MONTHS</b>	Add calendar months to date
<b>NEXT_DAY</b>	Next day of the date specified
<b>LAST_DAY</b>	Last day of the month
<b>ROUND</b>	Round date
<b>TRUNC</b>	Truncate date

# Using Date Functions

- `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'

# Using Date Functions

**Assume SYSDATE = '25-JUL-95':**

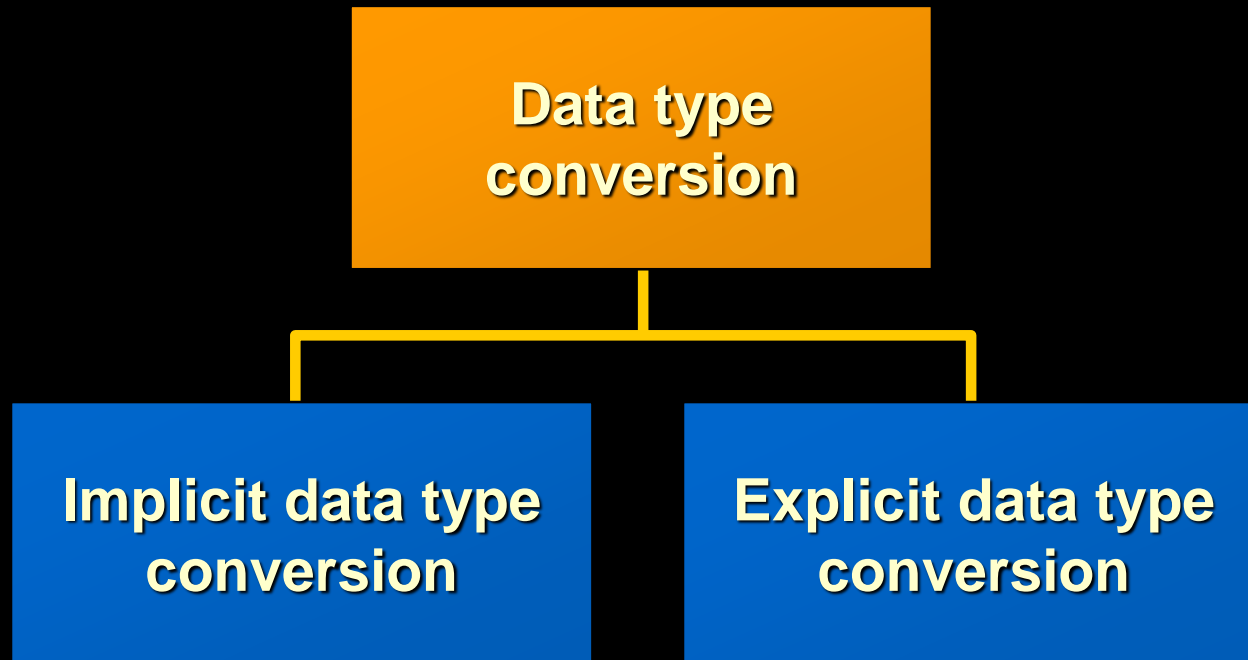
- **ROUND (SYSDATE, 'MONTH') → 01-AUG-95**
- **ROUND (SYSDATE, 'YEAR') → 01-JAN-96**
- **TRUNC (SYSDATE, 'MONTH') → 01-JUL-95**
- **TRUNC (SYSDATE, 'YEAR') → 01-JAN-95**

# Practice 3, Part One: Overview

**This practice covers the following topics:**

- **Writing a query that displays the current date**
- **Creating queries that require the use of numeric, character, and date functions**
- **Performing calculations of years and months of service for an employee**

# Conversion Functions





# Implicit Data Type Conversion

For assignments, the Oracle server can automatically convert the following:

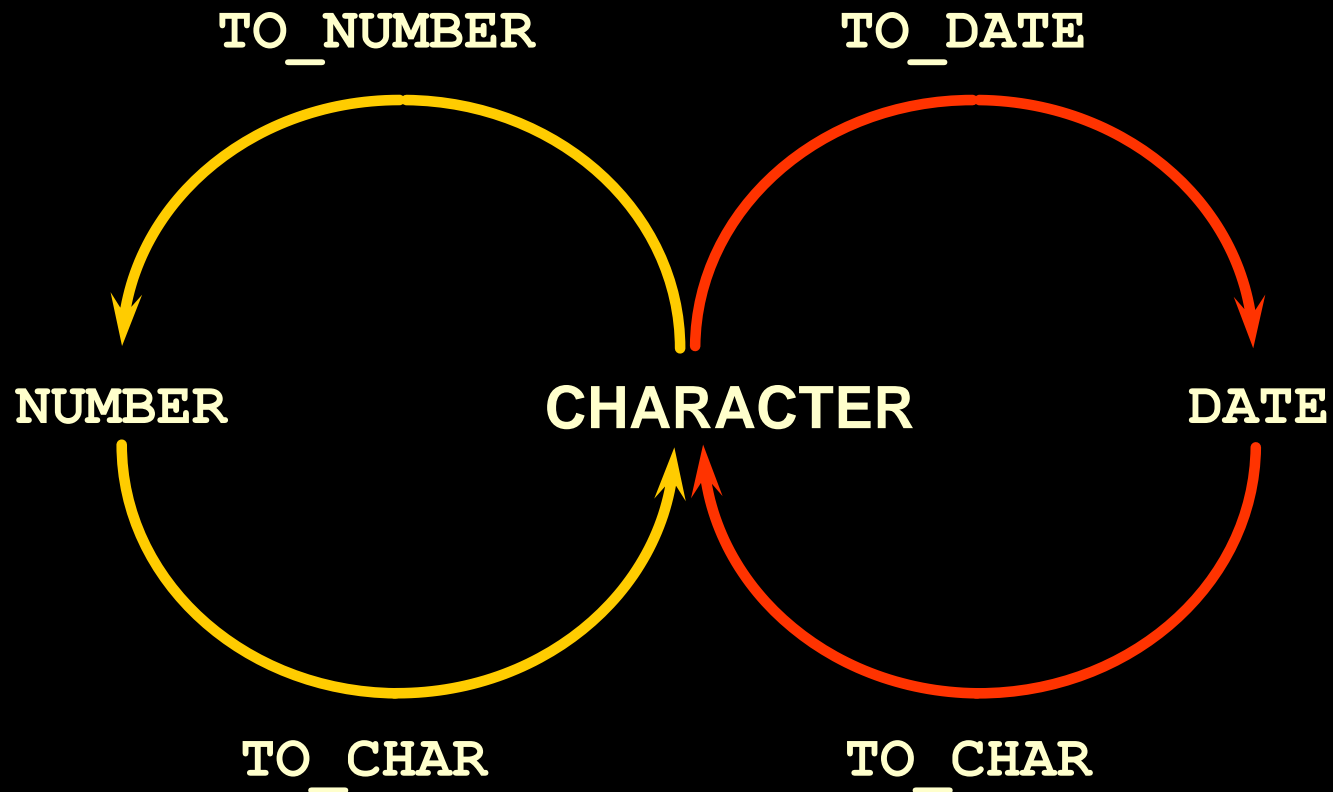
From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

# Implicit Data Type Conversion

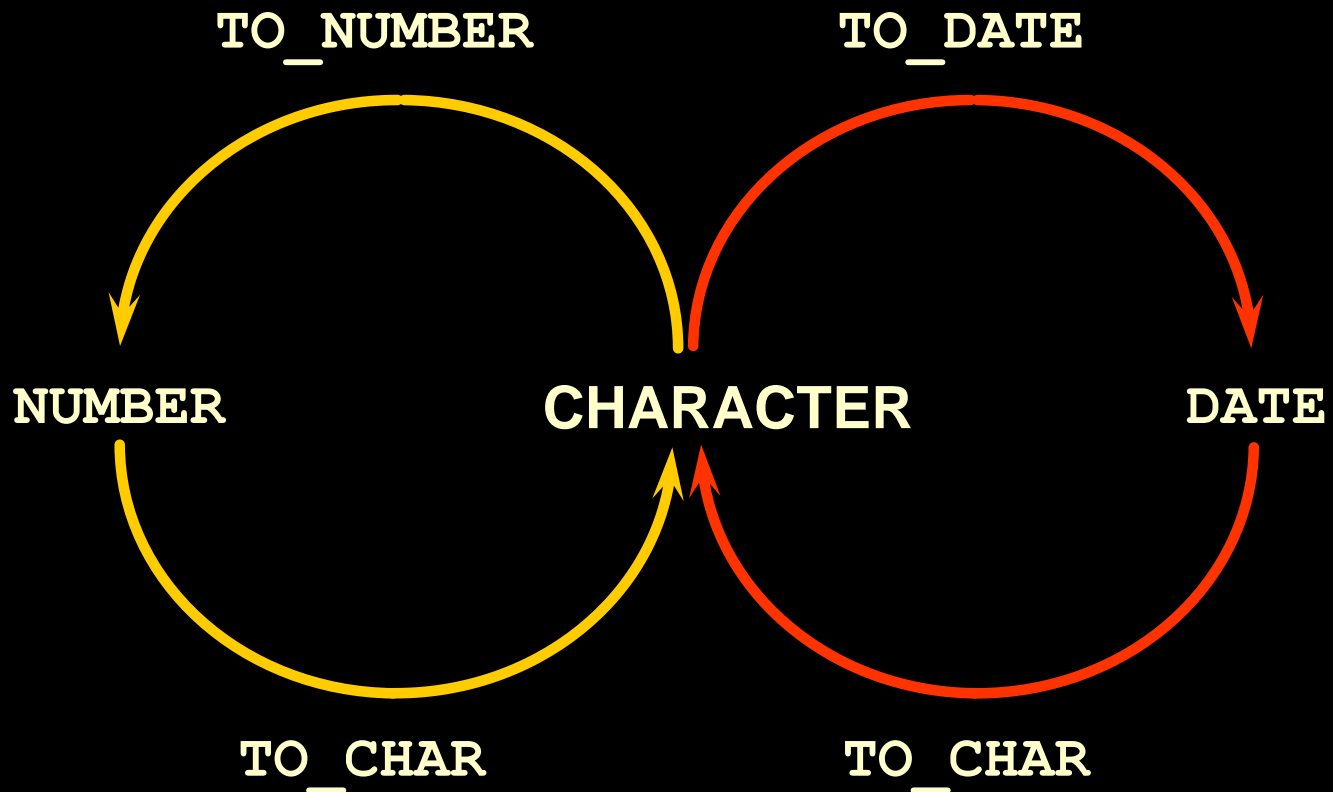
**For expression evaluation, the Oracle Server can automatically convert the following:**

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

# Explicit Data Type Conversion



# Explicit Data Type Conversion



# Using the TO\_CHAR Function with Dates

```
TO_CHAR(date, 'format_model') 
```

## The format model:

- Must be enclosed in single quotation marks and is case sensitive
- Can include any valid date format element
- Has an *fm* element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

# Elements of the Date Format Model

<b>YYYY</b>	<b>Full year in numbers</b>
<b>YEAR</b>	<b>Year spelled out</b>
<b>MM</b>	<b>Two-digit value for month</b>
<b>MONTH</b>	<b>Full name of the month</b>
<b>MON</b>	<b>Three-letter abbreviation of the month</b>
<b>DY</b>	<b>Three-letter abbreviation of the day of the week</b>
<b>DAY</b>	<b>Full name of the day of the week</b>
<b>DD</b>	<b>Numeric day of the month</b>

# Elements of the Date Format Model

- Time elements format the time portion of the date.

HH24:MI:SS AM	15:45:32 PM
---------------	-------------

- Add character strings by enclosing them in double quotation marks.

DD "of" MONTH	12 of OCTOBER
---------------	---------------

- Number suffixes spell out numbers.

ddspth	fourteenth
--------	------------

# Using the TO\_CHAR Function with Dates

```
SELECT last_name,  
       TO_CHAR(hire_date, 'fmDD Month YYYY')  
       AS HIREDATE  
FROM   employees;
```

LAST_NAME	HIREDATE
King	17 June 1987
Kochhar	21 September 1989
De Haan	13 January 1993
Hunold	3 January 1990
Ernst	21 May 1991
Lorentz	7 February 1999
Mourgos	16 November 1999

...

20 rows selected.



# Using the TO\_CHAR Function with Numbers

```
TO_CHAR(number, 'format_model') 
```

**These are some of the format elements you can use with the TO\_CHAR function to display a number value as a character:**

<b>9</b>	<b>Represents a number</b>
<b>0</b>	<b>Forces a zero to be displayed</b>
<b>\$</b>	<b>Places a floating dollar sign</b>
<b>L</b>	<b>Uses the floating local currency symbol</b>
<b>.</b>	<b>Prints a decimal point</b>
<b>,</b>	<b>Prints a thousand indicator</b>

# Using the TO\_CHAR Function with Numbers

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY  
FROM   employees  
WHERE  last_name = 'Ernst';
```

SALARY
\$6,000.00

# Using the TO\_NUMBER and TO\_DATE Functions

- Convert a character string to a number format using the TO\_NUMBER function:

```
TO_NUMBER(char[, 'format_model'])
```

- Convert a character string to a date format using the TO\_DATE function:

```
TO_DATE(char[, 'format_model'])
```

- These functions have an **fx** modifier. This modifier specifies the exact matching for the character argument and date format model of a TO\_DATE function

# Using the TO\_NUMBER and TO\_DATE Functions

- Convert a character string to a number format using the TO\_NUMBER function:

```
TO_NUMBER(char[, 'format_model'])
```

- Convert a character string to a date format using the TO\_DATE function:

```
TO_DATE(char[, 'format_model'])
```

- These functions have an **fx** modifier. This modifier specifies the exact matching for the character argument and date format model of a TO\_DATE function

# RR Date Format

Current Year	Specified Date	RR Format	YY Format
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		If the specified two-digit year is:	
		0–49	50–99
If two digits of the current year are:	0–49	The return date is in the current century	The return date is in the century before the current one
	50–99	The return date is in the century after the current one	The return date is in the current century

# Example of RR Date Format

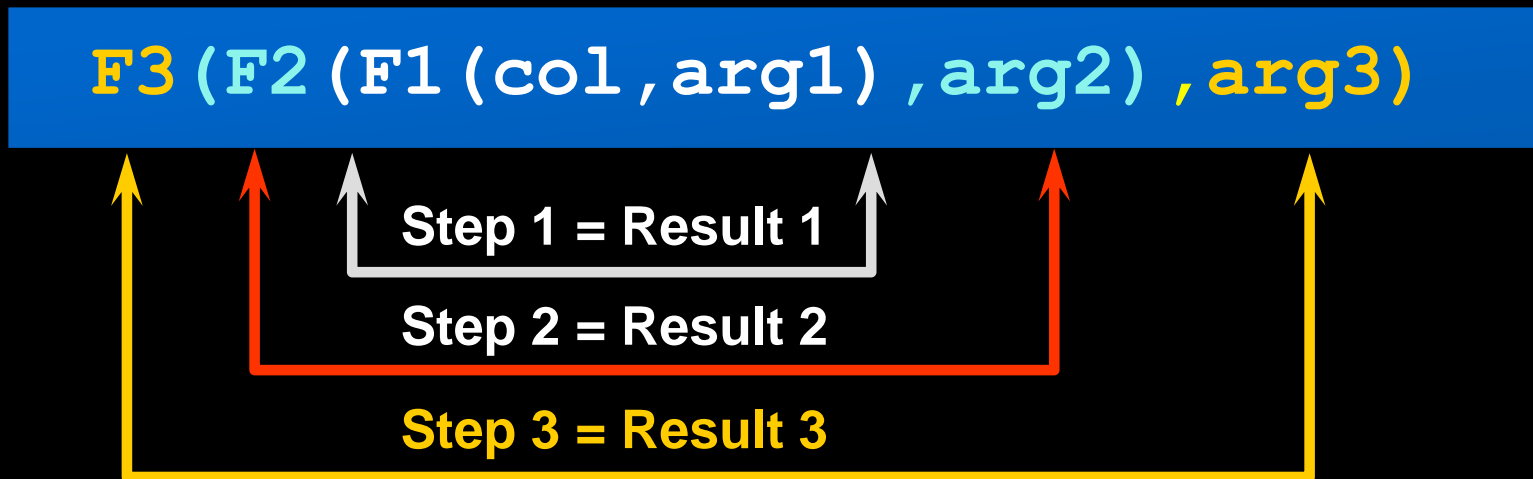
To find employees hired prior to 1990, use the RR format, which produces the same results whether the command is run in 1999 or now:

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')  
FROM employees  
WHERE hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

LAST_NAME	TO_CHAR(HIR
King	17-Jun-1987
Kochhar	21-Sep-1989
Whalen	17-Sep-1987

# Nesting Functions

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



# Nesting Functions

```
SELECT last_name,  
       NVL(TO_CHAR(manager_id), 'No Manager')  
FROM   employees  
WHERE  manager_id IS NULL;
```

LAST_NAME	NVL(TO_CHAR(MANAGER_ID), 'NOMANAGER')
King	No Manager



# General Functions

These functions work with any data type and pertain to using nulls.

- `NVL (expr1, expr2)`
- `NVL2 (expr1, expr2, expr3)`
- `NULLIF (expr1, expr2)`
- `COALESCE (expr1, expr2, ..., exprn)`

# NVL Function

**Converts a null to an actual value.**

- **Data types that can be used are date, character, and number.**
- **Data types must match:**
  - `NVL(commission_pct,0)`
  - `NVL(hire_date,'01-JAN-97')`
  - `NVL(job_id,'No Job Yet')`

# Using the NVL Function

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

Diagram annotations: A red box highlights the `NVL(commission_pct, 0)` expression, with a red arrow pointing to a circled '1'. Another red box highlights the entire calculation `(salary*12) + (salary*12*NVL(commission_pct, 0))`, with a red arrow pointing to a circled '2'.

LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
King	24000	0	288000
Kochhar	17000	0	204000
De Haan	17000	0	204000
Hunold	9000	0	108000
Ernst	6000	0	72000
Lorentz	4200	0	50400
Mourgos	5800	0	69600
Rajs	3500	0	42000

...

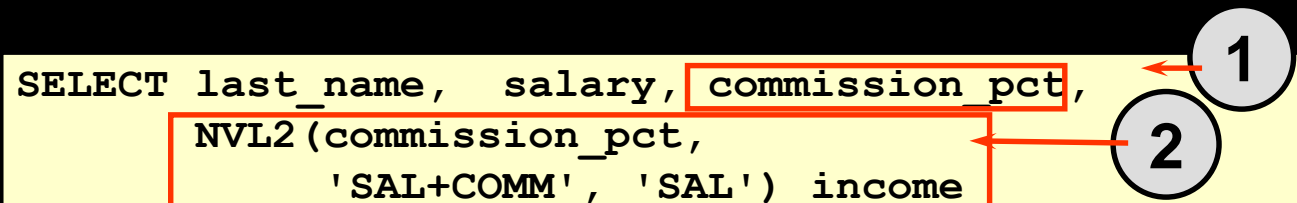
20 rows selected.

1

2

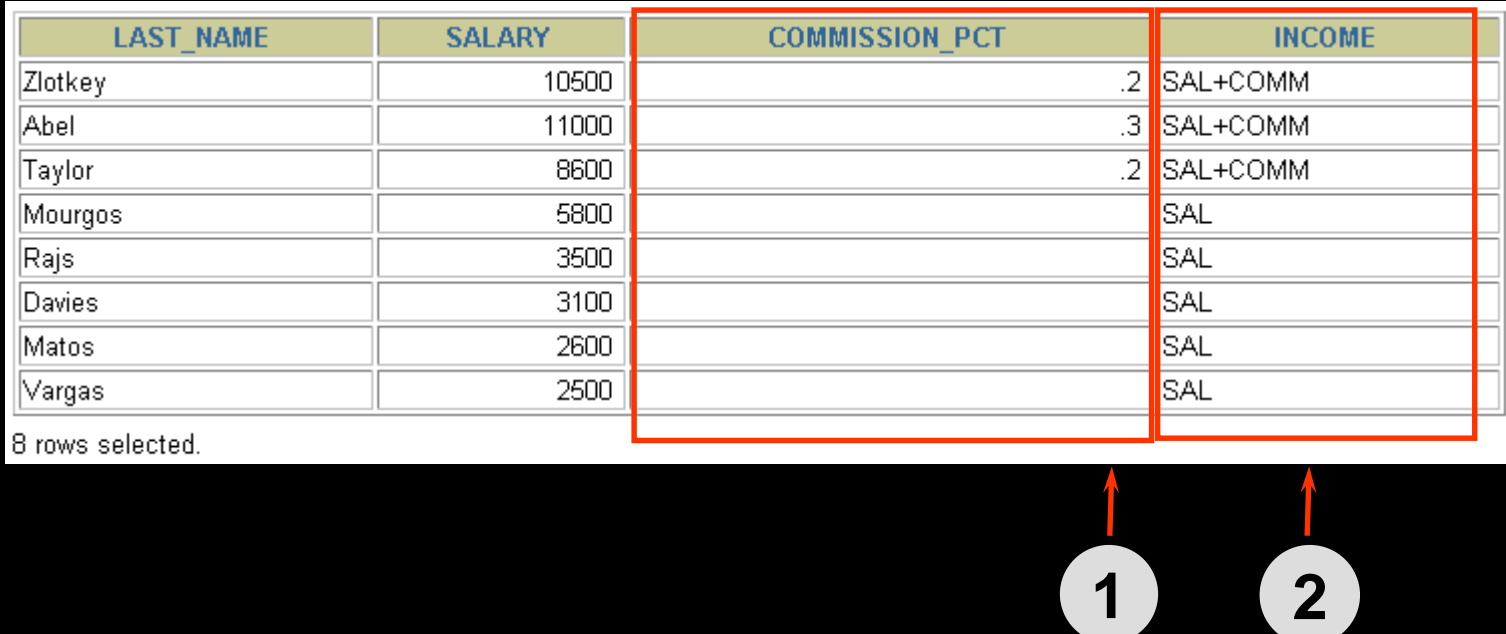
# Using the NVL2 Function

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



LAST_NAME	SALARY	COMMISSION_PCT	INCOME
Zlotkey	10500	.2	SAL+COMM
Abel	11000	.3	SAL+COMM
Taylor	8600	.2	SAL+COMM
Mourgos	5800		SAL
Rajs	3500		SAL
Davies	3100		SAL
Matos	2600		SAL
Vargas	2500		SAL

8 rows selected.



# Using the NULLIF Function

**1**

```
SELECT first_name, LENGTH(first_name) "expr1",  
       last_name,  LENGTH(last_name)  "expr2",  
       NULLIF(LENGTH(first_name), LENGTH(last_name)) result  
FROM   employees;
```

**2**

**3**

FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
Steven	6	King	4	6
Neena	5	Kochhar	7	5
Lex	3	De Haan	7	3
Alexander	9	Hunold	6	9
Bruce	5	Ernst	5	
Diana	5	Lorentz	7	5
Kevin	5	Mourgos	7	5
Trenna	6	Rajs	4	6
Curtis	6	Davies	6	

...

20 rows selected.

**1**

**2**

**3**

# Using the COALESCE Function

- The advantage of the COALESCE function over the NVL function is that the COALESCE function can take multiple alternate values.
- If the first expression is not null, it returns that expression; otherwise, it does a COALESCE of the remaining expressions.

# Using the COALESCE Function

```
SELECT    last_name,  
          COALESCE (commission_pct, salary, 10) comm  
FROM      employees  
ORDER BY  commission_pct;
```

LAST_NAME	COMM
Grant	.15
Zlotkey	.2
Taylor	.2
Abel	.3
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000

■ ■ ■

20 rows selected.

# Conditional Expressions

- Provide the use of IF-THEN-ELSE logic within a SQL statement
- Use two methods:
  - CASE expression
  - DECODE function



# The CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
CASE expr WHEN comparison_expr1 THEN return_expr1  
      [WHEN comparison_expr2 THEN return_expr2  
      WHEN comparison_exprn THEN return_exprn  
      ELSE else_expr]  
END
```

# Using the CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
SELECT last_name, job_id, salary,  
       CASE job_id WHEN 'IT_PROG' THEN 1.10*salary  
                  WHEN 'ST_CLERK' THEN 1.15*salary  
                  WHEN 'SA_REP' THEN 1.20*salary  
       ELSE salary END "REVISED_SALARY"  
FROM employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

# The DECODE Function

Facilitates conditional inquiries by doing the work of a CASE or IF-THEN-ELSE statement:

```
DECODE(col|expression, search1, result1  
      [, search2, result2, ...,]  
      [, default])
```

# Using the DECODE Function

```
SELECT last name, job id, salary,  
       DECODE(job_id, 'IT_PROG', 1.10*salary,  
                 'ST_CLERK', 1.15*salary,  
                 'SA_REP', 1.20*salary,  
                 salary)  
       REVISED_SALARY  
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

# Using the DECODE Function

Display the applicable tax rate for each employee in department 80.

```
SELECT last name, salary,  
       DECODE (TRUNC (salary/2000, 0),  
               0, 0.00,  
               1, 0.09,  
               2, 0.20,  
               3, 0.30,  
               4, 0.40,  
               5, 0.42,  
               6, 0.44,  
               0.45) TAX_RATE  
FROM   employees  
WHERE  department_id = 80;
```

# Summary

**In this lesson, you should have learned how to:**

- **Perform calculations on data using functions**
- **Modify individual data items using functions**
- **Manipulate output for groups of rows using functions**
- **Alter date formats for display using functions**
- **Convert column data types using functions**
- **Use NVL functions**
- **Use IF-THEN-ELSE logic**

# Practice 3, Part Two: Overview

**This practice covers the following topics:**

- **Creating queries that require the use of numeric, character, and date functions**
- **Using concatenation with functions**
- **Writing case-insensitive queries to test the usefulness of character functions**
- **Performing calculations of years and months of service for an employee**
- **Determining the review date for an employee**





