# Oracle 11g - SQL

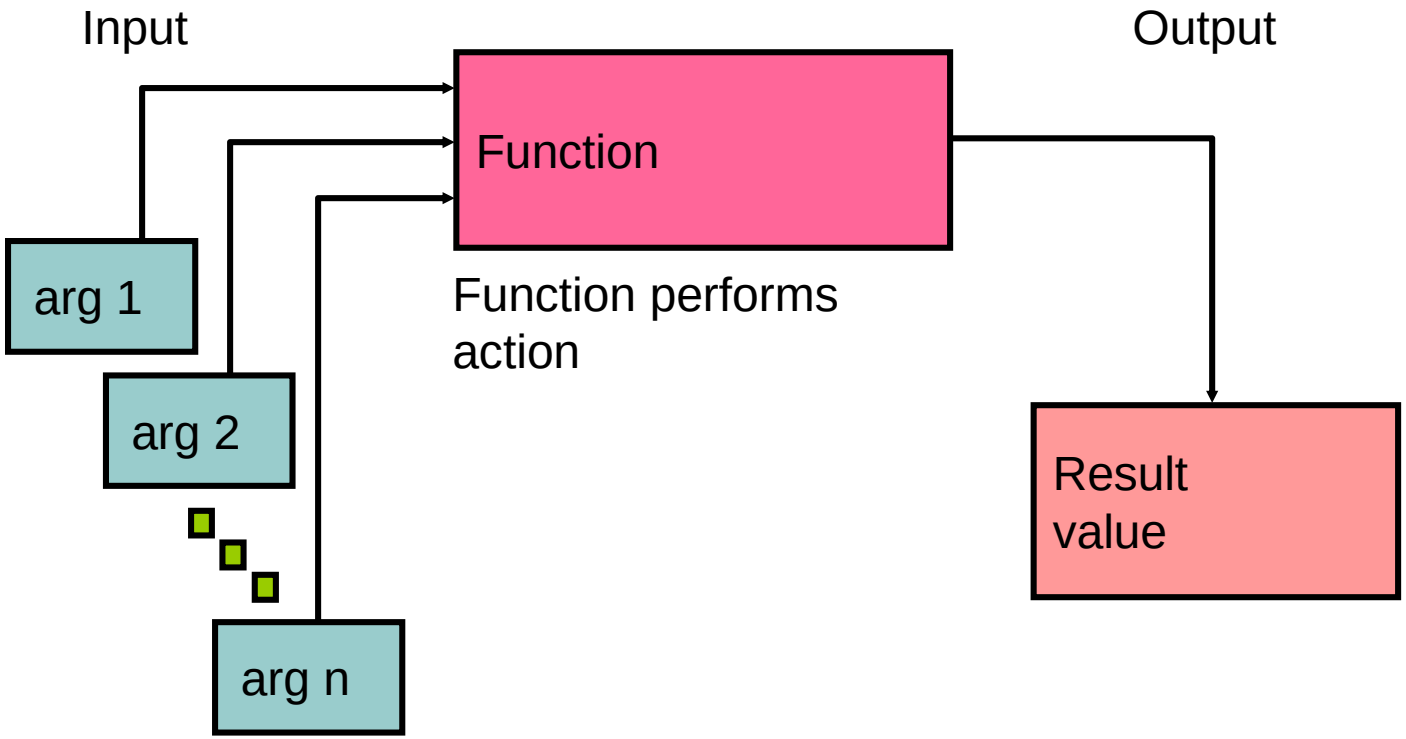
## Using Single-Row Functions to Customize Output

# Objectives

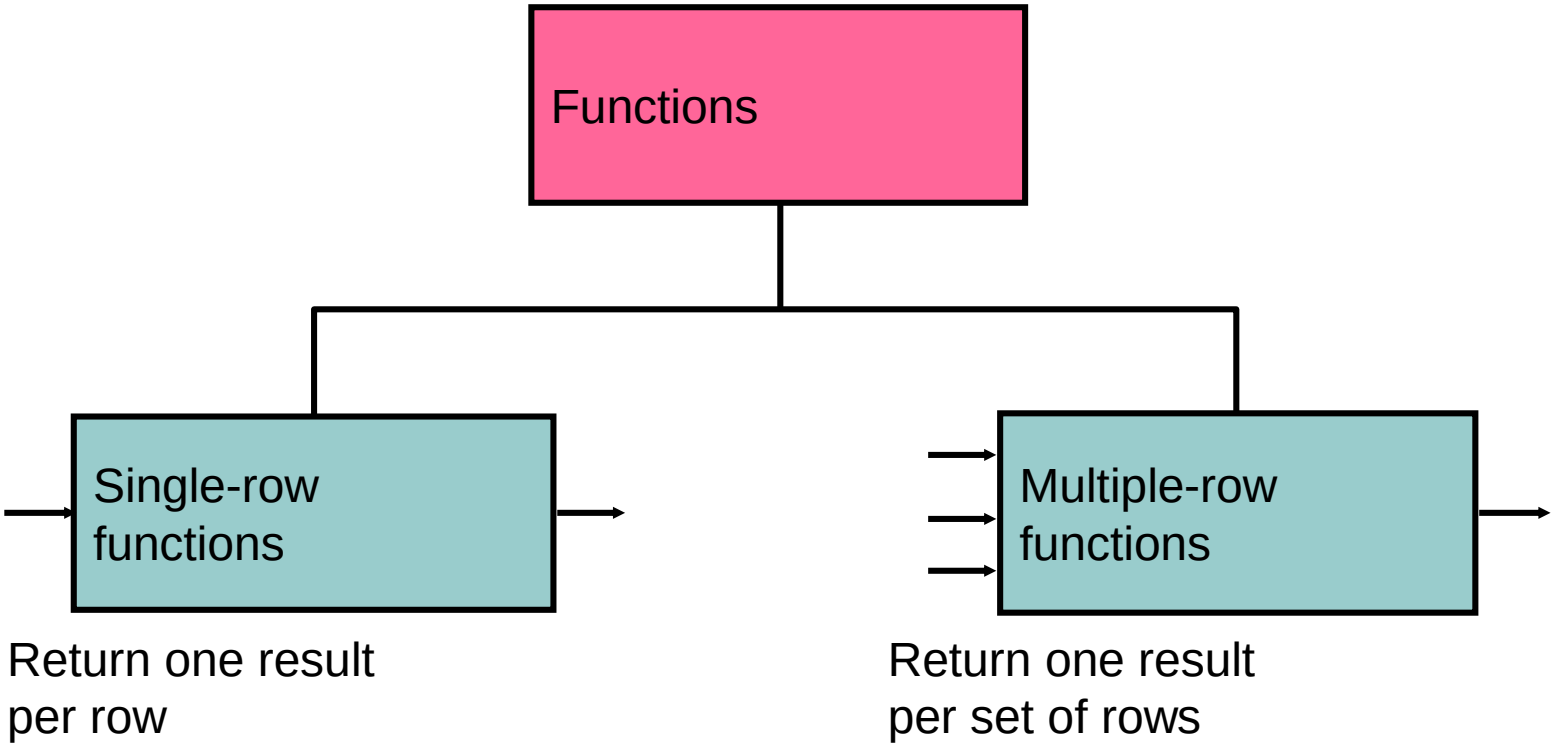
After completing this lesson, you should be able to do the following:

- Describe various types of functions that are 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



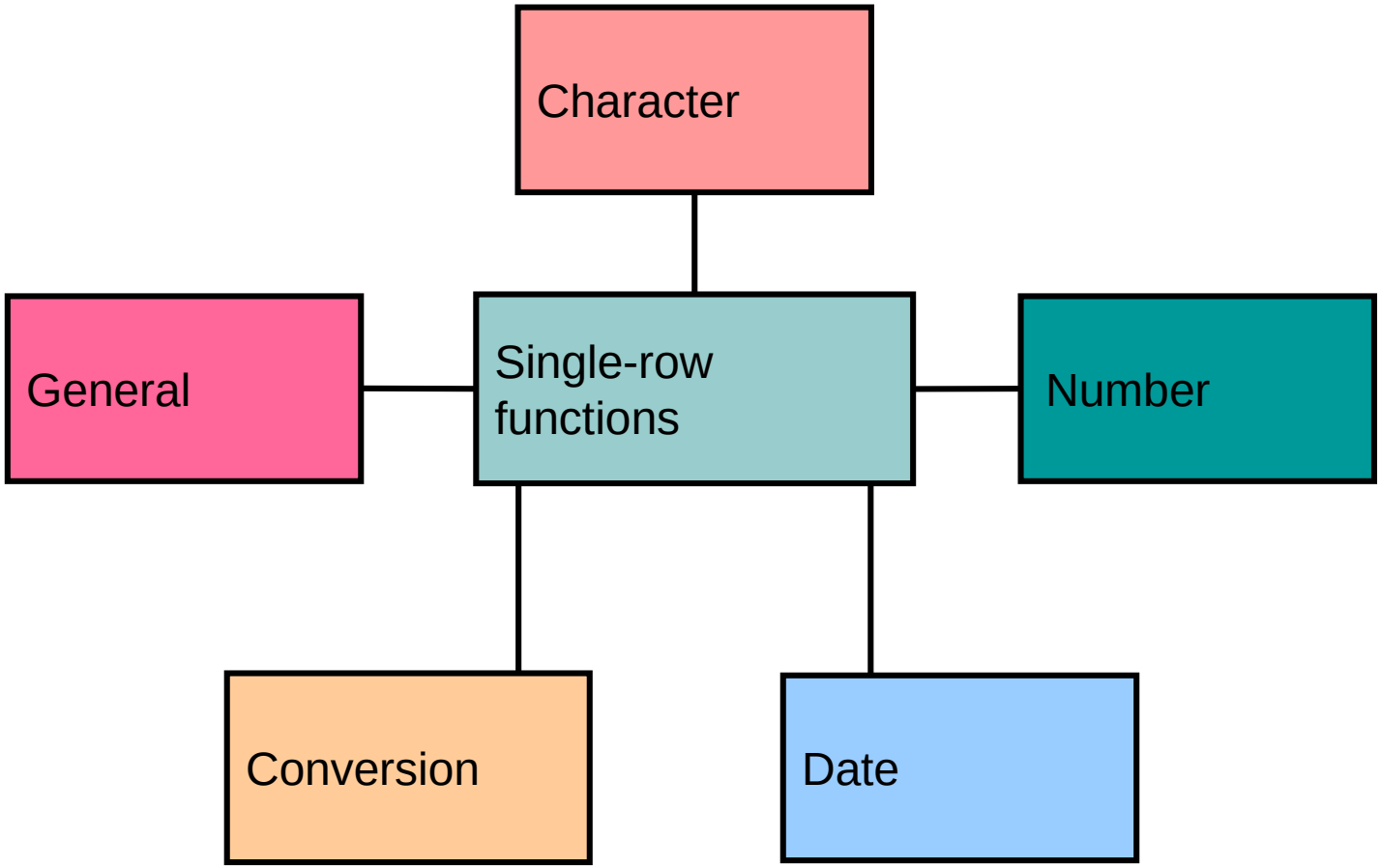
# Single-Row Functions

Single-row functions:

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

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

# Single-Row Functions



# Character Functions

Character functions

- Case-manipulation functions
  - LOWER
  - UPPER
  - INITCAP
- Character-manipulation functions
  - CONCAT
  - SUBSTR
  - LENGTH
  - INSTR
  - LPAD | RPAD
  - TRIM
  - REPLACE

# 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` |

# 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 |
|---|---|---|---|
| 1 | 205 | Higgins | 110 |

# Character-Manipulation Functions

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 |

# 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';
```

1 2 3

| | 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 |

1 2 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 |

# Using the ROUND Function

```
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 |

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

# Using the TRUNC Function

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

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

# 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

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

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '01-FEB-88';
```

| | LAST_NAME | HIRE_DATE |
|---|---|---|
| 1 | Whalen | 17-SEP-87 |
| 2 | King | 17-JUN-87 |

# Working with Dates

`SYSDATE` is a function that returns:

- Date
- Time

# Arithmetic with Dates

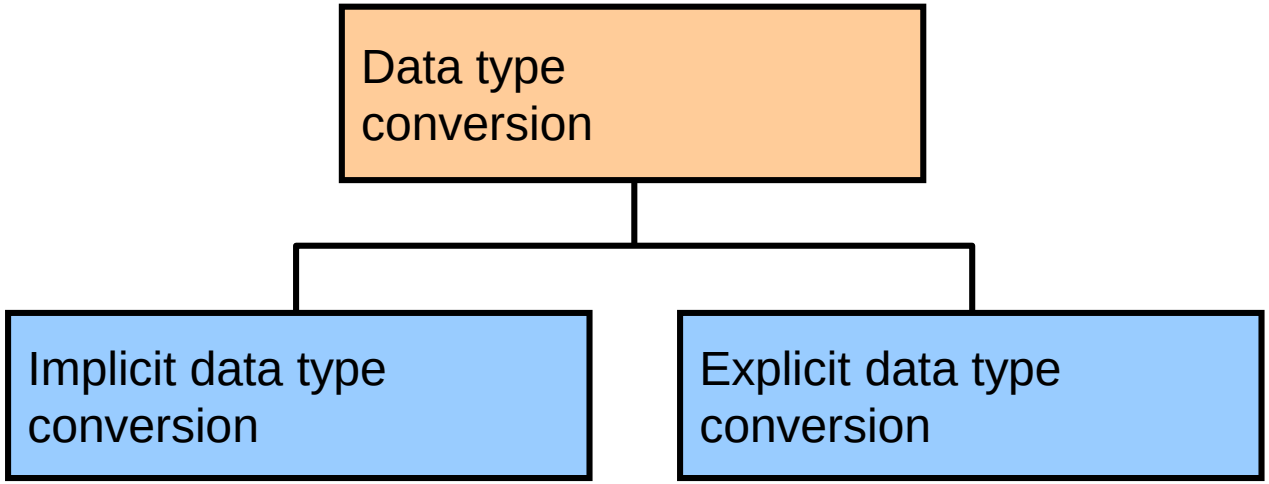
- Add or subtract a number to or from a date for a resultant date value.
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24.

# Using Arithmetic Operators with Dates

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

| | LAST_NAME | WEEKS |
|---|---|---|
| 1 | King | 1116.1485747354497354497354497354497354 5 |
| 2 | Kochhar | 998.00571759259259259259259259259259259 3 |
| 3 | De Haan | 825.1485747354497354497354497354497354 5 |

# Conversion Functions

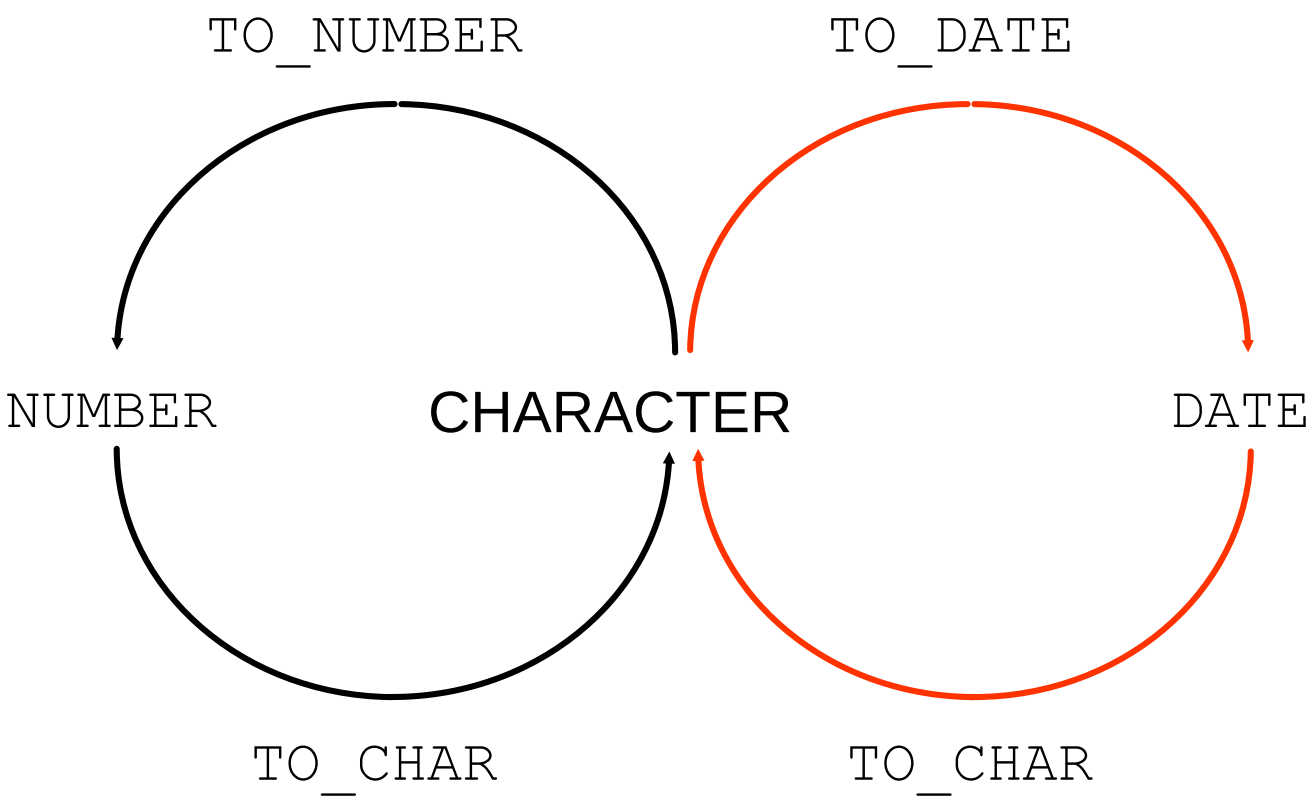


# Implicit Data Type Conversion

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

| From | To |
|---|---|
| VARCHAR2 or CHAR | NUMBER |
| VARCHAR2 or CHAR | DATE |
| NUMBER | VARCHAR2 |
| DATE | VARCHAR2 |

# Explicit Data Type Conversion



# Using the TO_CHAR Function with Dates

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

The format model:

- Must be enclosed by single quotation marks
- 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

| Element | Result |
|---|---|
| YYYY | Full year in numbers |
| YEAR | Year spelled out (in English) |
| MM | Two-digit value for month |
| MONTH | Full name of the month |
| MON | Three-letter abbreviation of the month |
| DY | Three-letter abbreviation of the day of the week |
| DAY | Full name of the day of the week |
| DD | Numeric day of the month |

# Elements of the Date Format Model

- Time elements format the time portion of the

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

- Add character strings by enclosing them in

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

| | |
|---|---|
| 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 |
|---|---|---|
| 1 | Whalen | 17 September 1987 |
| 2 | Hartstein | 17 February 1996 |
| 3 | Fay | 17 August 1997 |
| 4 | Higgins | 7 June 1994 |
| 5 | Gietz | 7 June 1994 |

...

| | | |
|---|---|---|
| 19 | Taylor | 24 March 1998 |
| 20 | Grant | 24 May 1999 |

# Using the TO_CHAR Function with Numbers

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

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 thousands indicator |

# Using the TO_CHAR Function with Numbers

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

| | SALARY |
|---|---|
| 1 | $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.

# 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 |

# RR Date Format: Example

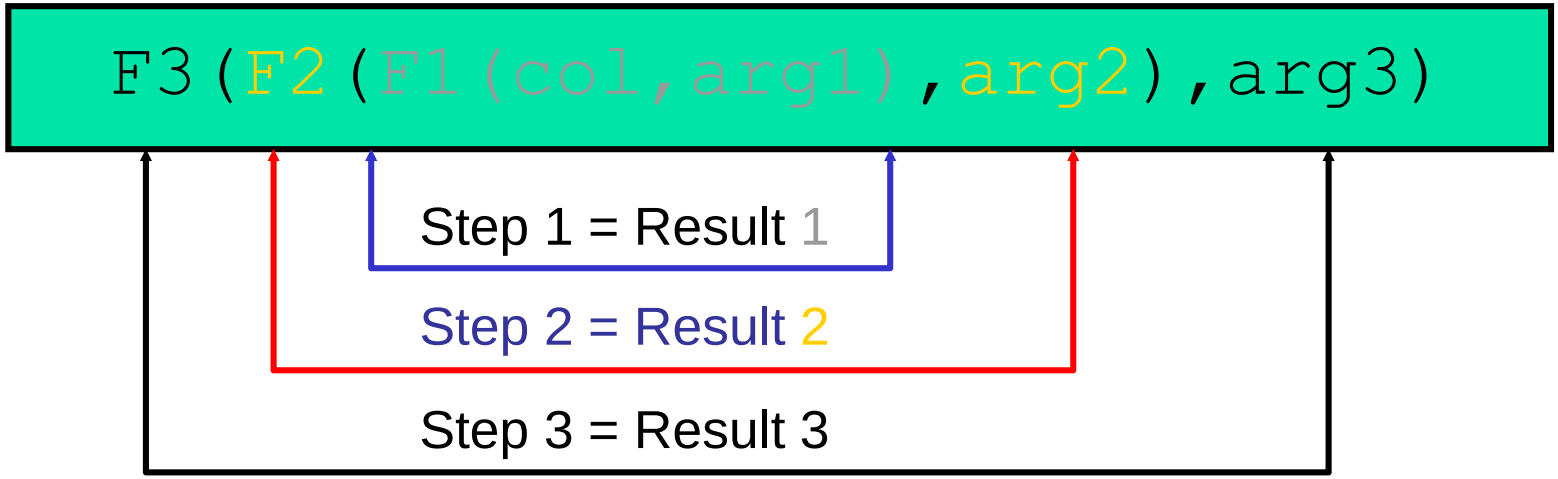
To find employees hired before 1990, use the RR date format, which produces the same results whether the command is run in 1999 or now:

```
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(HIRE_DATE,'DD-MON-YYYY') |
|---|---|---|
| 1 | Whalen | 17-Sep-1987 |
| 2 | King | 17-Jun-1987 |
| 3 | Kochhar | 21-Sep-1989 |

# Nesting Functions

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



# Nesting Functions

```
SELECT last name,
  UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))
FROM   employees
WHERE  department_id = 60;
```

| | LAST_NAME | UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US')) |
|---|---|---|
| 1 | Hunold | HUNOLD_US |
| 2 | Ernst | ERNST_US |
| 3 | Lorentz | LORENTZ_US |

# General Functions

The following 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 value 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;
```

1

2

| | LAST_NAME | SALARY | NVL(COMMISSION_PCT,0) | AN_SAL |
|---|---|---|---|---|
| 1 | Whalen | 4400 | 0 | 52800 |
| 2 | Hartstein | 13000 | 0 | 156000 |
| 3 | Fay | 6000 | 0 | 72000 |
| 4 | Higgins | 12000 | 0 | 144000 |
| 5 | Gietz | 8300 | 0 | 99600 |
| 6 | King | 24000 | 0 | 288000 |
| 7 | Kochhar | 17000 | 0 | 204000 |
| 8 | De Haan | 17000 | 0 | 204000 |

...

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);
```

1

2

| | 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

# Using the NULLIF Function

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

| | FIRST_NAME | expr1 | LAST_NAME | expr2 | RESULT |
|---|---|---|---|---|---|
| 1 | Ellen | 5 | Abel | 4 | 5 |
| 2 | Curtis | 6 | Davies | 6 | (null) |
| 3 | Lex | 3 | De Haan | 7 | 3 |
| 4 | Bruce | 5 | Ernst | 5 | (null) |
| 5 | Pat | 3 | Fay | 3 | (null) |
| 6 | William | 7 | Gietz | 5 | 7 |
| 7 | Kimberely | 9 | Grant | 5 | 9 |
| 8 | Michael | 7 | Hartstein | 9 | 7 |

...

# 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, the COALESCE function returns that expression; otherwise, it does a COALESCE of the remaining expressions.

# Using the COALESCE Function

```
SELECT last_name,
       COALESCE(manager_id,commission_pct, -1) comm
FROM   employees
ORDER BY commission_pct;
```

| | LAST_NAME | COMM |
|---|---|---|
| 1 | Grant | 149 |
| 2 | Taylor | 149 |
| 3 | Zlotkey | 100 |
| 4 | Abel | 149 |
| 5 | King | -1 |
| 6 | Kochhar | 100 |
| 7 | De Haan | 100 |
| 8 | Hunold | 102 |

...

# Conditional Expressions

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

# 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 |
|---|---|---|---|---|
| ... | | | | |
| 11 | Lorentz | IT_PROG | 4200 | 4620 |
| 12 | Mourgos | ST_MAN | 5800 | 5800 |
| 13 | Rajs | ST_CLERK | 3500 | 4025 |
| 14 | Davies | ST_CLERK | 3100 | 3565 |
| 15 | Matos | ST_CLERK | 2600 | 2990 |
| 16 | Vargas | ST_CLERK | 2500 | 2875 |
| 17 | Zlotkey | SA_MAN | 10500 | 10500 |
| 18 | Abel | SA_REP | 11000 | 13200 |
| 19 | Taylor | SA_REP | 8600 | 10320 |
| 20 | Grant | SA_REP | 7000 | 8400 |

# DECODE Function

Facilitates conditional inquiries by doing the work of a CASE expression or an 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 |
|---|---|---|---|---|
| ... | | | | |
| 11 | Lorentz | IT_PROG | 4200 | 4620 |
| 12 | Mourgos | ST_MAN | 5800 | 5800 |
| 13 | Rajs | ST_CLERK | 3500 | 4025 |
| 14 | Davies | ST_CLERK | 3100 | 3565 |
| 15 | Matos | ST_CLERK | 2600 | 2990 |
| 16 | Vargas | ST_CLERK | 2500 | 2875 |
| 17 | Zlotkey | SA_MAN | 10500 | 10500 |
| 18 | Abel | SA_REP | 11000 | 13200 |
| 19 | Taylor | SA_REP | 8600 | 10320 |
| 20 | Grant | SA_REP | 7000 | 8400 |

# 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;
```