# Oracle Database 12c: SQL Workshop I

**Student Guide - Volume I**



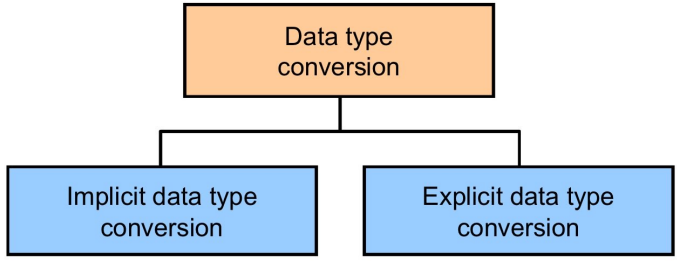
D80190GC10
Edition 1.0
August 2013
D83122

ORACLE

5

# Using Conversion Functions and Conditional Expressions

# Conversion Functions








In addition to Oracle data types, columns of tables in an Oracle Database can be defined by using the American National Standards Institute (ANSI), DB2, and SQL/DS data types. However, the Oracle server internally converts such data types to Oracle data types.

In some cases, the Oracle server receives data of one data type where it expects data of a different data type. When this happens, the Oracle server can automatically convert the data to the expected data type. This data type conversion can be done *implicitly* by the Oracle server or *explicitly* by the user.

Implicit data type conversions work according to the rules explained in the following slides.

Explicit data type conversions are performed by using the conversion functions. Conversion functions convert a value from one data type to another. Generally, the form of the function names follows the convention *data type* `TO` *data type*. The first data type is the input data type and the second data type is the output.

**Note:** Although implicit data type conversion is available, it is recommended that you do the explicit data type conversion to ensure the reliability of your SQL statements.

# Implicit Data Type Conversion

In expressions, the Oracle server can automatically convert the following:

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



Oracle server can automatically perform data type conversion in an expression. For example, the expression `hire_date > '01-JAN-90'` results in the implicit conversion from the string `'01-JAN-90'` to a date. Therefore, a `VARCHAR2` or `CHAR` value can be implicitly converted to a number or date data type in an expression.

**Note:** `CHAR` to `NUMBER` conversions succeed only if the character string represents a valid number.

# Implicit Data Type Conversion

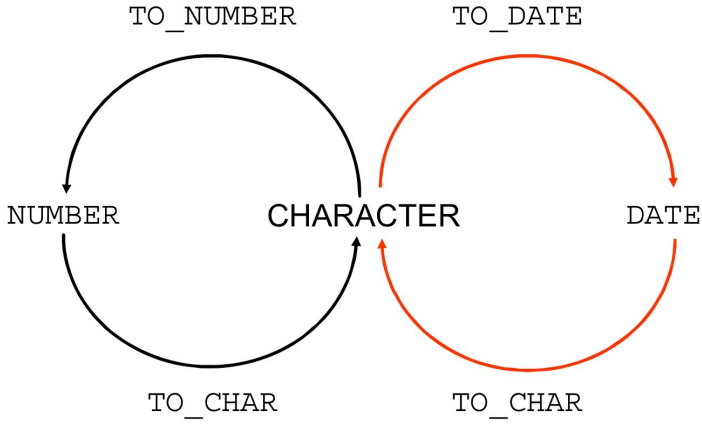
For expression evaluation, the Oracle server can automatically convert the following:

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



In general, the Oracle server uses the rule for expressions when a data type conversion is needed. For example, the expression `grade = 2` results in the implicit conversion of the number `2` to the string "2" because grade is a `CHAR(2)` column.

# Explicit Data Type Conversion






SQL provides three functions to convert a value from one data type to another:

| Function | Purpose |
|---|---|
| `TO_CHAR(`*`number`*`\|`*`date`*` [, `*`fmt`*` [, `*`nlsparams`*`] ] )` | Converts a number or date value to a `VARCHAR2` character string with the format model *`fmt`*<br>**Number conversion:** The `nlsparams` parameter specifies the following characters, which are returned by number format elements:<br>• Decimal character<br>• Group separator<br>• Local currency symbol<br>• International currency symbol<br>If `nlsparams` or any other parameter is omitted, this function uses the default parameter values for the session. |

# Using the TO_CHAR Function with Dates

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

The format model:

- Must be enclosed with 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





TO_CHAR converts a datetime data type to a value of VARCHAR2 data type in the format specified by the *format_model*. A format model is a character literal that describes the format of datetime stored in a character string. For example, the datetime format model for the string '11-Nov-2000' is 'DD-Mon-YYYY'. You can use the TO_CHAR function to convert a date from its default format to the one that you specify.

**Guidelines**

- The format model must be enclosed with single quotation marks and is case-sensitive.
- The format model can include any valid date format element. But be sure to separate the date value from the format model with a comma.
- The names of days and months in the output are automatically padded with blanks.
- To remove padded blanks or to suppress leading zeros, use the fill mode *fm* element.

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM   employees
WHERE  last_name = 'Higgins';
```

| | EMPLOYEE_ID | MONTH_HIRED |
|---|---|---|
| 1 | 205 | 06/02 |

# Elements of the Date Format Model

| Element | Result |
|---|---|
| YYYY | Full year in numbers |
| YEAR | Year spelled out (in English) |
| MM | Two-digit value for the 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 date:

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

- Add character strings by enclosing them with double quotation marks:

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

- Number suffixes spell out numbers:

| | |
|---|---|
| `ddspth` | `fourteenth` |



Use the formats that are listed in the following tables to display time information and literals, and to change numerals to spelled numbers.

| Element | Description |
|---|---|
| AM or PM | Meridian indicator |
| A.M. or P.M. | Meridian indicator with periods |
| HH or HH12 | 12 hour format |
| HH24 | 24 hour format |
| MI | Minute (0–59) |
| SS | Second (0–59) |
| SSSSS | Seconds past midnight (0–86399) |

# 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 | King | 17 June 2003 |
| 2 | Kochhar | 21 September 2005 |
| 3 | De Haan | 13 January 2001 |
| 4 | Hunold | 3 January 2006 |
| 5 | Ernst | 21 May 2007 |
| 6 | Lorentz | 7 February 2007 |
| 7 | Mourgos | 16 November 2007 |
| 8 | Rajs | 17 October 2003 |

...

ORACLE



The SQL statement in the slide displays the last names and hire dates for all the employees. The hire date appears as 17 June 2003.

**Example**

Modify the example in the slide to display the dates in a format that appears as “Seventeenth of June 2003 12:00:00 AM.”

```
SELECT  last_name,
 TO_CHAR(hire_date,
       'fmDdspth "of" Month YYYY fmHH:MI:SS AM')
 HIREDATE
FROM    employees;
```

| | LAST_NAME | HIREDATE |
|---|---|---|
| 1 | King | Seventeenth of June 2003 12:00:00 AM |
| 2 | Kochhar | Twenty-First of September 2005 12:00:00 AM |

...

Notice that the month follows the format model specified; in other words, the first letter is capitalized and the rest are in lowercase.

# 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 a thousands indicator |

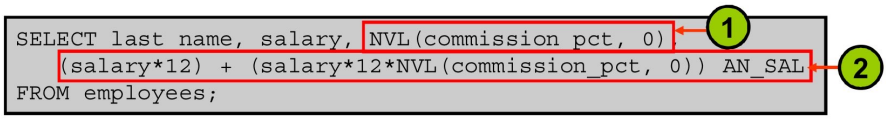
ORACLE



When working with number values, such as character strings, you should convert those numbers to the character data type using the TO_CHAR function, which translates a value of NUMBER data type to VARCHAR2 data type. This technique is especially useful with concatenation.

# Using the NVL Function

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




| | LAST_NAME | SALARY | NVL(COMMISSION_PCT,0) | AN_SAL |
|---|---|---|---|---|
| 1 | King | 24000 | 0 | 288000 |
| 2 | Kochhar | 17000 | 0 | 204000 |
| 3 | De Haan | 17000 | 0 | 204000 |
| 4 | Hunold | 9000 | 0 | 108000 |
| 5 | Ernst | 6000 | 0 | 72000 |
| 6 | Lorentz | 4200 | 0 | 50400 |
| 7 | Mourgos | 5800 | 0 | 69600 |
| 8 | Rajs | 3500 | 0 | 42000 |
| 9 | Davies | 3100 | 0 | 37200 |
| 10 | Matos | 2600 | 0 | 31200 |

...




To calculate the annual compensation of all employees, you 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) AN_SAL
FROM   employees;
```

| | LAST_NAME | SALARY | COMMISSION_PCT | AN_SAL |
|---|---|---|---|---|
| 1 | King | 24000 | (null) | (null) |
| 2 | Kochhar | 17000 | (null) | (null) |

...

| | | | | |
|---|---|---|---|---|
| 14 | Taylor | 8600 | 0.2 | 123840 |
| 15 | Grant | 7000 | 0.15 | 96600 |
| 16 | Whalen | 4400 | (null) | (null) |
| 17 | Hartstein | 13000 | (null) | (null) |
| 18 | Fay | 6000 | (null) | (null) |
| 19 | Higgins | 12008 | (null) | (null) |
| 20 | Gietz | 8300 | (null) | (null) |

Notice that the annual compensation is calculated for only those employees who earn a commission. If any column value in an expression is null, the result is null. To calculate values for all employees, you must convert the null value to a number before applying the arithmetic operator. In the example in the slide, the NVL function is used to convert null values to zero.

# 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 |
|---|---|---|---|---|
| 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 |



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)
```

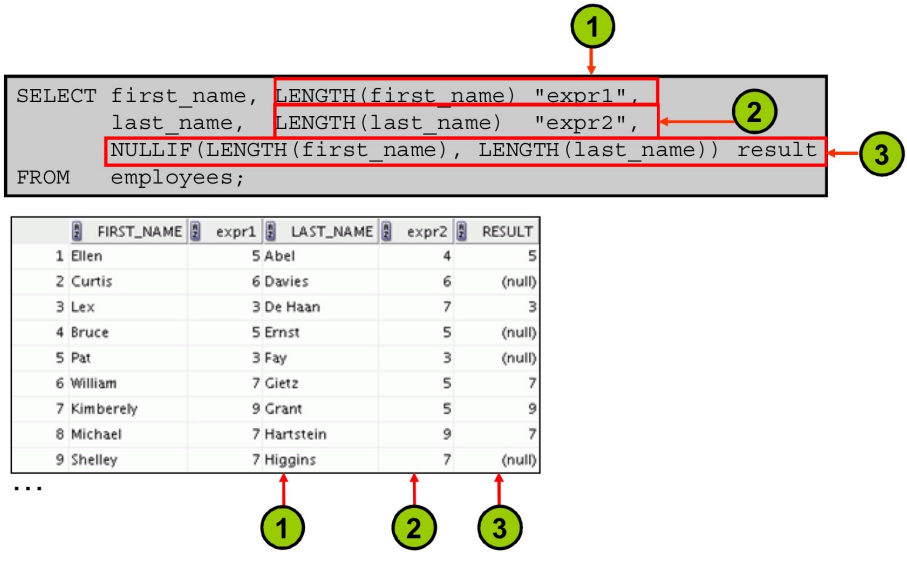
In the syntax:

- *expr1* is the source value or expression that may contain a null
- *expr2* is the value that is returned if *expr1* is not null
- *expr3* is the value that is returned if *expr1* is null

In the example shown in the slide, the COMMISSION_PCT column is examined. If a value is detected, the text literal value of SAL+COMM is returned. If the COMMISSION_PCT column contains a null value, the text literal value of SAL is returned.

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

# Using the NULLIF Function






The NULLIF function compares two expressions.

**Syntax**

```
NULLIF (expr1, expr2)
```

In the syntax:

- NULLIF compares *expr1* and *expr2*. If they are equal, the function returns null. If they are not, the function returns *expr1*. However, you cannot specify the literal NULL for *expr1*.

In the example shown in the slide, the length of the first name in the EMPLOYEES table is compared to the length of the last name in the EMPLOYEES table. When the lengths of the names are equal, a null value is displayed. When the lengths of the names are not equal, the length of the first name is displayed.

# Using the COALESCE Function

```
SELECT last_name, employee_id,
COALESCE(TO_CHAR(commission_pct),TO_CHAR(manager_id),
         'No commission and no manager')
FROM employees;
```

| | LAST_NAME | EMPLOYEE_ID | COALESCE(TO_CHAR(COMMISSION_PCT),TO_CHAR(MANAGER_ID),'NOCOMMISSIONANDNOMANAGER') |
|---|---|---|---|
| 1 | King | 100 | No commission and no manager |
| 2 | Kochhar | 101 | 100 |
| 3 | De Haan | 102 | 100 |
| 4 | Hunold | 103 | 102 |

...

| | | | |
|---|---|---|---|
| 13 | Abel | 174 | .3 |
| 14 | Taylor | 176 | .2 |
| 15 | Grant | 178 | .15 |
| 16 | Whalen | 200 | 101 |
| 17 | Hartstein | 201 | 100 |
| 18 | Fay | 202 | 201 |
| 19 | Higgins | 205 | 101 |
| 20 | Gietz | 206 | 205 |

ORACLE



In the example shown in the slide, if the `manager_id` value is not null, it is displayed. If the `manager_id` value is null, the `commission_pct` is displayed. If the `manager_id` and `commission_pct` values are null, “No commission and no manager” is displayed. Note that `TO_CHAR` function is applied so that all expressions are of the same data type.

**Example**

For the employees who do not get any commission, your organization wants to give a salary increment of $2,000 and for employees who get commission, the query should compute the new salary that is equal to the existing salary added to the commission amount.

```
SELECT last_name, salary, commission_pct,
 COALESCE((salary+(commission_pct*salary)), salary+2000)"New
  Salary"
FROM   employees;
```

**Note:** Examine the output. For employees who do not get any commission, the New Salary column shows the salary incremented by $2,000 and for employees who get commission, the New Salary column shows the computed commission amount added to the salary.

| | LAST_NAME | SALARY | COMMISSION_PCT | New Salary |
|---|---|---|---|---|
| 1 | King | 24000 | (null) | 26000 |
| 2 | Kochhar | 17000 | (null) | 19000 |
| 3 | De Haan | 17000 | (null) | 19000 |
| 4 | Hunold | 9000 | (null) | 11000 |
| 5 | Ernst | 6000 | (null) | 8000 |
| 6 | Lorentz | 4200 | (null) | 6200 |
| 7 | Mourgos | 5800 | (null) | 7800 |
| 8 | Rajs | 3500 | (null) | 5500 |
| 9 | Davies | 3100 | (null) | 5100 |
| 10 | Matos | 2600 | (null) | 4600 |
| 11 | Vargas | 2500 | (null) | 4500 |
| 12 | Zlotkey | 10500 | 0.2 | 12600 |
| 13 | Abel | 11000 | 0.3 | 14300 |
| 14 | Taylor | 8600 | 0.2 | 10320 |
| 15 | Grant | 7000 | 0.15 | 8050 |
| 16 | Whalen | 4400 | (null) | 6400 |
| 17 | Hartstein | 13000 | (null) | 15000 |
| 18 | Fay | 6000 | (null) | 8000 |
| 19 | Higgins | 12008 | (null) | 14008 |
| 20 | Gietz | 8300 | (null) | 10300 |

# 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
```

ORACLE



CASE expressions allow you to use the IF-THEN-ELSE logic in SQL statements without having to invoke procedures.

In a simple CASE expression, the Oracle server searches for the first WHEN ... THEN pair for which expr is equal to comparison_expr and returns return_expr. If none of the WHEN ... THEN pairs meet this condition, and if an ELSE clause exists, the Oracle server returns else_expr. Otherwise, the Oracle server returns a null. You cannot specify the literal NULL for all the return_exprs and the else_expr.

The expressions expr and comparison_expr must be of the same data type, which can be CHAR, VARCHAR2, NCHAR, or NVARCHAR2, NUMBER, BINARY_FLOAT, or BINARY_DOUBLE or must all have a numeric datatype. All of the return values (return_expr) must be of the same data type.

If all expressions have a numeric datatype, then Oracle determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that datatype, and returns that datatype.

# 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 |
|---|---|---|---|---|
| 1 | King | AD_PRES | 24000 | 24000 |
| ... | | | | |
| 4 | Hunold | IT_PROG | 9000 | 9900 |
| 5 | Ernst | IT_PROG | 6000 | 6600 |
| 6 | Lorentz | IT_PROG | 4200 | 4620 |
| 7 | Mourgos | ST_MAN | 5800 | 5800 |
| 8 | Rajs | ST_CLERK | 3500 | 4025 |
| 9 | Davies | ST_CLERK | 3100 | 3565 |
| 10 | Matos | ST_CLERK | 2600 | 2990 |
| 11 | Vargas | ST_CLERK | 2500 | 2875 |
| ... | | | | |
| 13 | Abel | SA_REP | 11000 | 13200 |
| 14 | Taylor | SA_REP | 8600 | 10320 |
| 15 | Grant | SA_REP | 7000 | 8400 |

ORACLE



In the SQL statement in the slide, the value of JOB_ID is decoded. If JOB_ID is IT_PROG, the salary increase is 10%; if JOB_ID is ST_CLERK, the salary increase is 15%; if JOB_ID is SA_REP, the salary increase is 20%. For all other job roles, there is no increase in salary.

The same statement can be written with the DECODE function.

The following code is an example of the searched CASE expression. In a searched CASE expression, the search occurs from left to right until an occurrence of the listed condition is found, and then it returns the return expression. If no condition is found to be true, and if an ELSE clause exists, the return expression in the ELSE clause is returned; otherwise, a NULL is returned.

```
SELECT last_name,salary,
(CASE WHEN salary<5000 THEN 'Low'
      WHEN salary<10000 THEN 'Medium'
      WHEN salary<20000 THEN 'Good'
      ELSE 'Excellent'
END) qualified_salary
FROM employees;
```

# 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])
```



The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic that is used in various languages. The DECODE function decodes *expression* after comparing it to each *search* value. If the expression is the same as *search*, *result* is returned.

If the default value is omitted, a null value is returned where a search value does not match any of the result values.

# 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 |
|---|---|---|---|---|
| ... | | | | |
| 4 | Hunold | IT_PROG | 9000 | 9900 |
| 5 | Ernst | IT_PROG | 6000 | 6600 |
| 6 | Lorentz | IT_PROG | 4200 | 4620 |
| 7 | Mourgos | ST_MAN | 5800 | 5800 |
| 8 | Rajs | ST_CLERK | 3500 | 4025 |
| 9 | Davies | ST_CLERK | 3100 | 3565 |
| 10 | Matos | ST_CLERK | 2600 | 2990 |
| 11 | Vargas | ST_CLERK | 2500 | 2875 |
| 12 | Zlotkey | SA_MAN | 10500 | 10500 |
| ... | | | | |
| 13 | Abel | SA_REP | 11000 | 13200 |
| 14 | Taylor | SA_REP | 8600 | 10320 |
| 15 | Grant | SA_REP | 7000 | 8400 |

ORACLE



In the SQL statement in the slide, the value of JOB_ID is tested. If JOB_ID is IT_PROG, the salary increase is 10%; if JOB_ID is ST_CLERK, the salary increase is 15%; if JOB_ID is SA_REP, the salary increase is 20%. For all other job roles, there is no increase in salary.

The same statement can be expressed in pseudocode as an IF-THEN-ELSE statement:

```
IF job_id = 'IT_PROG'     THEN  salary = salary*1.10
IF job_id = 'ST_CLERK'    THEN  salary = salary*1.15
IF job_id = 'SA_REP'      THEN  salary = salary*1.20
ELSE salary = salary
```

# 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;
```



This slide shows another example using the DECODE function. In this example, you determine the tax rate for each employee in department 80 based on the monthly salary. The tax rates are as follows:

| ***Monthly Salary Range*** | ***Tax Rate*** |
|---|---|
| $0.00–1,999.99 | 00% |
| $2,000.00–3,999.99 | 09% |
| $4,000.00–5,999.99 | 20% |
| $6,000.00–7,999.99 | 30% |
| $8,000.00–9,999.99 | 40% |
| $10,000.00–11,999.99 | 42% |
| $12,200.00–13,999.99 | 44% |
| $14,000.00 or greater | 45% |

| | LAST_NAME | SALARY | TAX_RATE |
|---|---|---|---|
| 1 | Zlotkey | 10500 | 0.42 |
| 2 | Abel | 11000 | 0.42 |
| 3 | Taylor | 8600 | 0.4 |