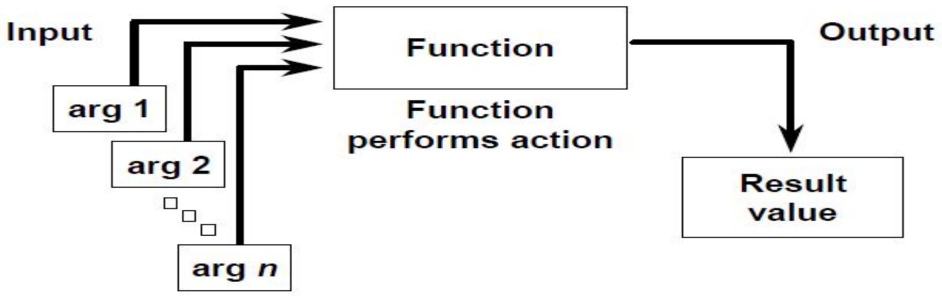
Single-Row Functions

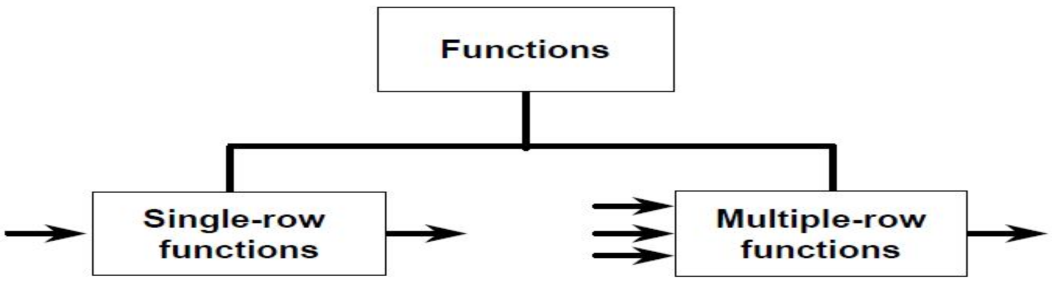
SQL Functions

Functions are a very powerful feature of SQL and can be used to do the following:
• Perform calculations on data
• Modify individual data items
• Manipulate output for groups of rows
• Format dates and numbers for display
• Convert column data types
SQL functions sometimes take arguments and always return a value.
Note: Most of the functions described in this lesson are specific to Oracle Corporation's version of SQL.

SQL Functions



Two Types of SQL Functions



Single-Row Functions

These functions operate on single rows only and return one result per row. There are different types of
single-row functions. This lesson covers the following ones:
• Character
• Number
• Date
• Conversion

Multiple-Row Functions

Functions can manipulate groups of rows to give one result per group of rows. These functions are known
as group functions. This is covered in a later lesson.

Single row functions:

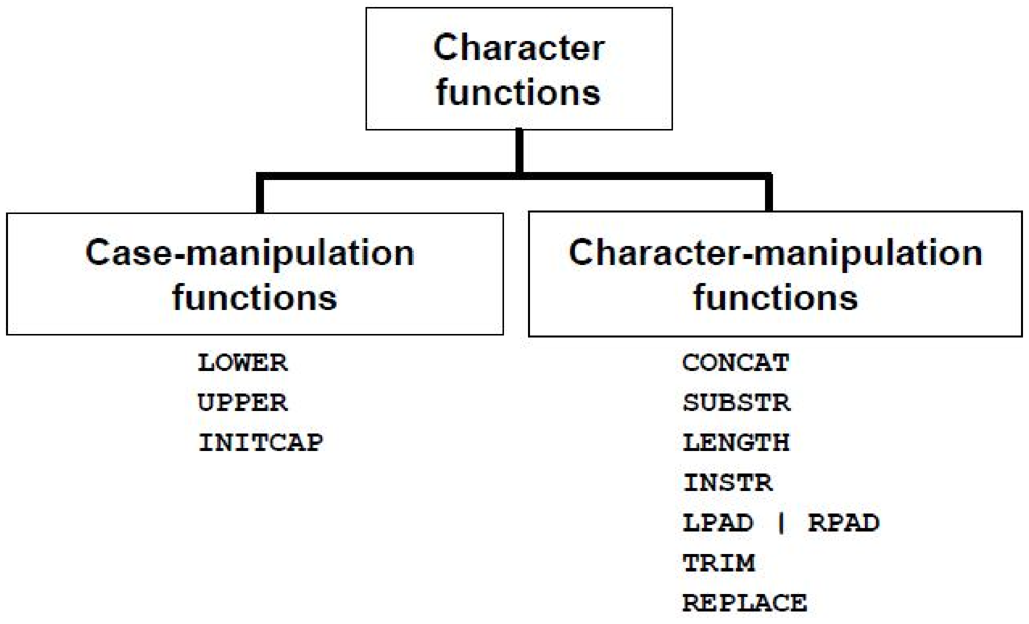
- **Manipulate data items**
- **Accept arguments and return one value**
- **Act on each row returned**
- **Return one result per row**
- **May modify the data type**
- **Can be nested**
- **Accept arguments which can be a column or an expression**

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

Features of single-row functions include:
• Acting on each row returned in the query
• Returning one result per row
• Possibly returning a data value of a different type than that referenced
• Possibly expecting one or more arguments
• Can be used in `SELECT`, `WHERE`, and `ORDER BY` clauses; can be nested

In the syntax:
`function_name` is the name of the function.
`arg1, arg2` is any argument to be used by the function.
This can be represented by a column name or expression.

Character Functions



Character Functions

Single-row character functions accept character data as input and can return both character and numeric values. Character functions can be divided into the following:

- Case-manipulation functions
- Character-manipulation functions

Function	Purpose
`LOWER(`*`column`*`|`*`expression`*`)`	Converts alpha character values to lowercase
`UPPER(`*`column`*`|`*`expression`*`)`	Converts alpha character values to uppercase
`INITCAP(`*`column`*`|`*`expression`*`)`	Converts alpha character values to uppercase for the first letter of each word, all other letters in lowercase
`CONCAT(`*`column1`*`|`*`expression1`*`, `*`column2`*`|`*`expression2`*`)`	Concatenates the first character value to the second character value; equivalent to concatenation operator (||)
`SUBSTR(`*`column`*`|`*`expression`*`,`*`m`* *`[,n]`*`)`	Returns specified characters from character value starting at character position *m*, *n* characters long (If *m* is negative, the count starts from the end of the character value. If *n* is omitted, all characters to the end of the string are returned.)

Function	Purpose
`LENGTH(column|expression)`	Returns the number of characters in the expression
`INSTR(column|expression, 'string', [,m], [n] )`	Returns the numeric position of a named string. Optionally, you can provide a position *m* to start searching, and the occurrence *n* of the string. *m* and *n* default to 1, meaning start the search at the beginning of the search and report the first occurrence.

`LPAD(column|expression, n, 'string')`
`RPAD(column|expression, n, 'string')`
	Pads the character value right-justified to a total width of *n* character positions
	Pads the character value left-justified to a total width of *n* character positions

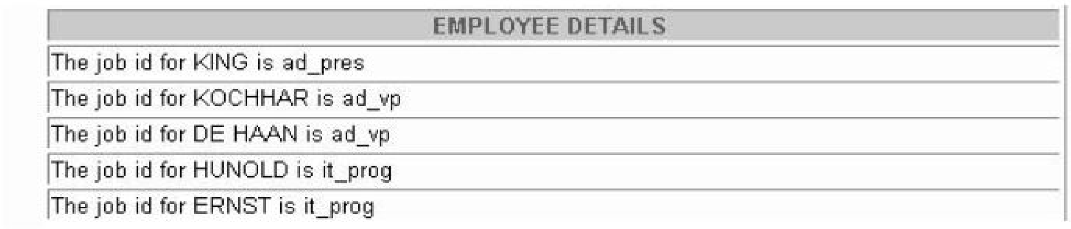
`TRIM(leading|trailing|both, trim_character FROM trim_source)`	Enables you to trim heading or trailing characters (or both) from a character string. If `trim_character` or `trim_source` is a character literal, you must enclose it in single quotes. This is a feature available from Oracle8*i* and later.
`REPLACE(text, search_string, replacement_string)`	Searches a text expression for a character string and, if found, replaces it with a specified replacement string

Case Manipulation Functions

LOWER, UPPER, and INITCAP are the three case-conversion functions.

- LOWER: Converts mixed case or uppercase character strings to lowercase
- UPPER: Converts mixed case or lowercase character strings to uppercase
- INITCAP: Converts the first letter of each word to uppercase and remaining letters to lowercase

```
SELECT 'The job id for '||UPPER(last_name)||' is '
        ||LOWER(job_id) AS "EMPLOYEE DETAILS"
FROM employees;
```



```
EMPLOYEE DETAILS
The job id for KING is ad_pres
The job id for KOCHHAR is ad_vp
The job id for DE HAAN is ad_vp
The job id for HUNOLD is it_prog
The job id for ERNST is it_prog
```

```
[illegible] id for [illegible] is ac_mgr
The job id for GIETZ is ac_account
```

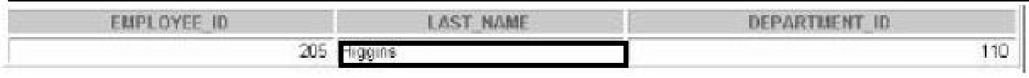
Function	Result
`LOWER('SQL Course')`	`sql course`
`UPPER('SQL Course')`	`SQL COURSE`
`INITCAP('SQL Course')`	`Sql Course`

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
205	Higgins	110

The WHERE clause of the first SQL statement specifies the employee name as higgins. Because all the data in the EMPLOYEES table is stored in proper case, the name higgins does not find a match in the table, and as a result no rows are selected.

The WHERE clause of the second SQL statement specifies that the employee name in the EMPLOYEES table is compared to higgins, converting the LAST_NAME column to lowercase for comparison purposes. Since both the names are lowercase now, a match is found and one row is selected. The WHERE clause can be rewritten in the following manner to produce the same result:

```
...WHERE last_name = 'Higgins'
```

The name in the output appears as it was stored in the database. To display the name capitalized, use the UPPER function in the SELECT statement.

```
SELECT employee_id, UPPER(last_name), department_id
FROM   employees
WHERE  INITCAP(last_name) = 'Higgins';
```

Character-Manipulation Functions

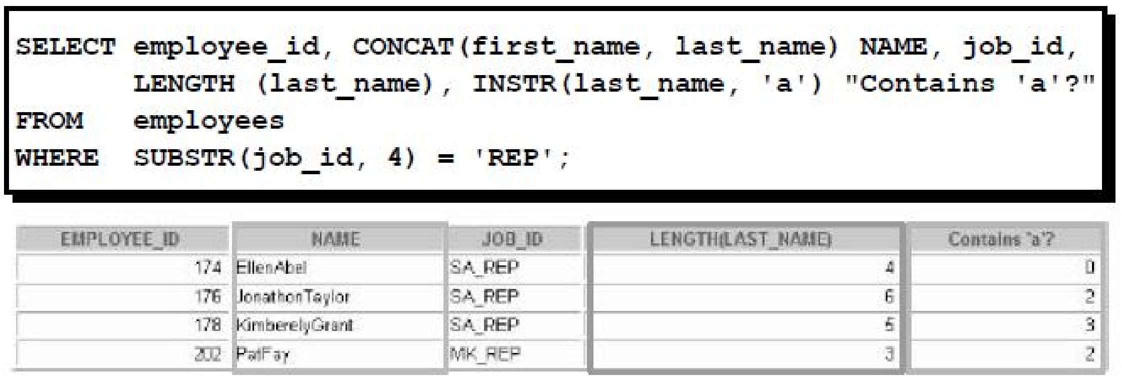
These functions manipulate character strings:

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*****
TRIM('H' FROM 'HelloWorld')	elloWorld

Character Manipulation Functions

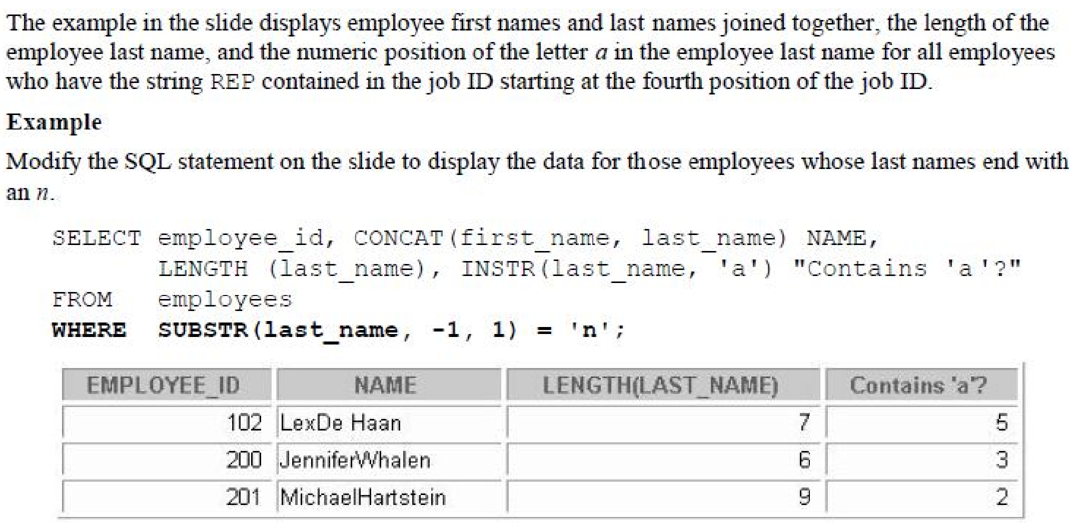
`CONCAT`, `SUBSTR`, `LENGTH`, `INSTR`, `LPAD`, `RPAD`, and `TRIM` are the character manipulation functions
covered in this lesson.

- `CONCAT`: Joins values together (you are limited to using two parameters with CONCAT
- `SUBSTR`: Extracts a string of determined length
- `LENGTH`: Shows the length of a string as a numeric value
- `INSTR`: Finds numeric position of a named character
- `LPAD`: Pads the character value right-justified
- `RPAD`: Pads the character value left-justified
- `TRIM`: Trims heading or trailing characters (or both) from a character string (If *`trim_character`*
or *`trim_source`* is a character literal, you must enclose it in single quotes.)



```
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



The example in the slide displays employee first names and last names joined together, the length of the employee last name, and the numeric position of the letter *a* in the employee last name for all employees who have the string `REP` contained in the job ID starting at the fourth position of the job ID.

Example

Modify the SQL statement on the slide to display the data for those employees whose last names end with an *n*.

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
       LENGTH (last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(last_name, -1, 1) = 'n';
```

EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
102	LexDe Haan	7	5
200	JenniferWhalen	6	3
201	MichaelHartstein	9	2

Number Functions

- ROUND: Rounds value to specified decimal

  ROUND(45.926, 2) $\longrightarrow$ 45.93

- TRUNC: Truncates value to specified decimal

  TRUNC(45.926, 2) $\longrightarrow$ 45.92

- MOD: Returns remainder of division

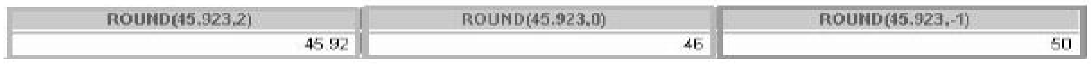
  MOD(1600, 300) $\longrightarrow$ 100

Number functions accept numeric input and return numeric values. This section describes some of the number functions.

Function	Purpose
ROUND(*column|expression, n*)	Rounds the column, expression, or value to *n* decimal places, or, if *n* is omitted, no decimal places. (If *n* is negative, numbers to left of the decimal point are rounded.)
TRUNC(*column|expression,n*)	Truncates the column, expression, or value to *n* decimal places, or, if *n* is omitted, then *n* defaults to zero
MOD(*m,n*)	Returns the remainder of *m* divided by *n*

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)
          45.92                   46                    50
```

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

ROUND Function

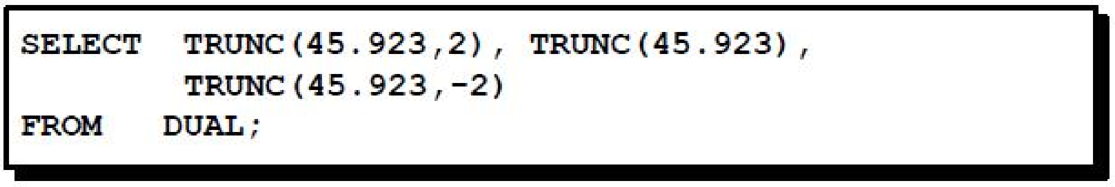
The ROUND function rounds the column, expression, or value to *n* decimal places. If the second argument is 0 or is missing, the value is rounded to zero decimal places. If the second argument is 2, the value is rounded to two decimal places. Conversely, if the second argument is -2, the value is rounded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

The ROUND function can also be used with date functions.

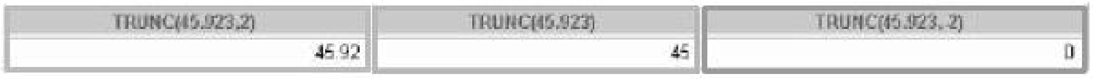
The DUAL Table

The DUAL table is owned by the user SYS and can be accessed by all users. It contains one column,DUMMY, and one row with the value X. The DUAL table is useful when you want to return a value once only: for instance, the value of a constant, pseudocolumn, or expression that is not derived from a table with user data. The `DUAL` table is generally used for `SELECT` clause syntax completeness, because both `SELECT` and `FROM` clauses are mandatory, and several calculations do not need to select from actual tables.

Using the `TRUNC` Function



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



TRUNC(45.923,2)	TRUNC(45.923)	TRUNC(45.923,-2)
45.92	45	0

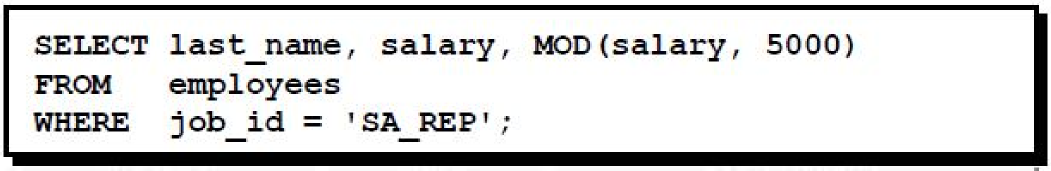
The TRUNC function truncates the column, expression, or value to *n* decimal places.

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. Conversely, if the second argument is -2, the value is truncated to two decimal places to the left.

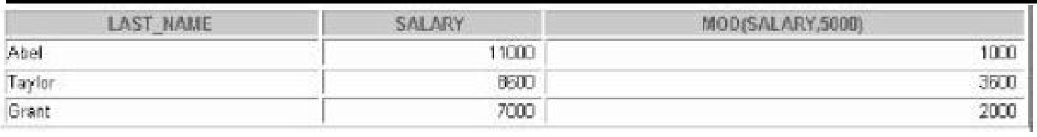
Like the ROUND function, the TRUNC function can be used with date functions.

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';
```

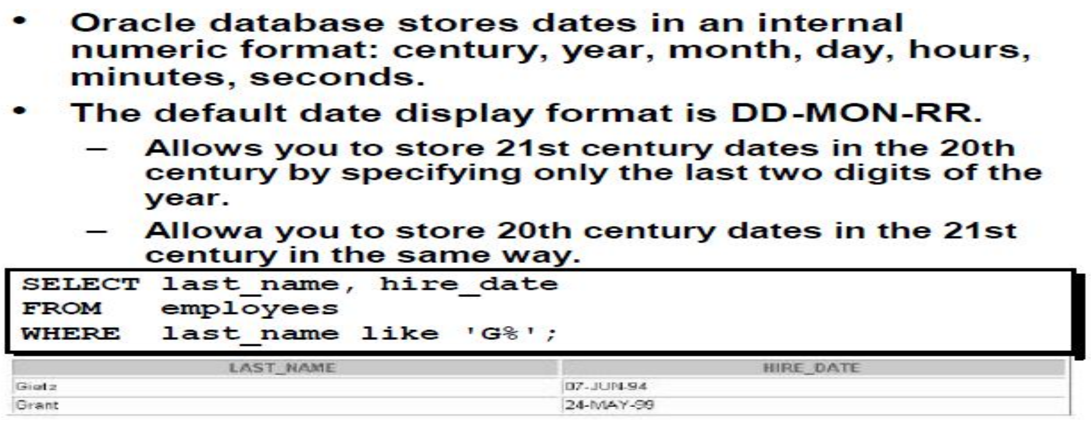


LAST_NAME	SALARY	MOD(SALARY,5000)
Abel	11000	1000
Taylor	8600	3600
Grant	7000	2000

The MOD function finds the remainder of value1 divided by value2. The slide example calculates the remainder of the salary after dividing it by 5,000 for all employees whose job ID is SA_REP.

Note: The MOD function is often used to determine if a value is odd or even.

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.
  - Allowa you to store 20th century dates in the 21st century in the same way.

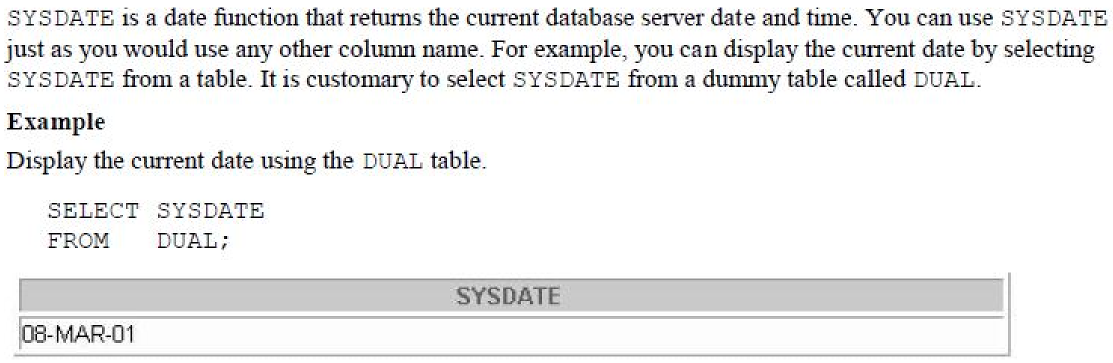
```
SELECT last_name, hire_date
FROM   employees
WHERE  last_name like 'G%';
```

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

Oracle Date Format
The Oracle database stores dates in an internal numeric format, representing the century, year, month, day,hours, minutes, and seconds.
The default display and input format for any date is DD-MON-RR. Valid Oracle dates are between January1, 4712 B.C. and A.D. December 31, 9999.
In the example in the slide, the `HIRE_DATE` for the employee Gietz is displayed in the default format DDMON-RR. However, dates are not stored in the database in this format. All the components of the date andtime are stored. So, although a `HIRE_DATE` such as 07-JUN-94 is displayed as day, month, and year, thereis also *time* and *century* information associated with it. The complete data might be June 7th, 1994 5:10:43p.m.
This data is stored internally as follows:
CENTURY YEAR MONTH DAY HOUR MINUTE SECOND 19 94 06 07 5 10 43
Centuries and the Year 2000
The Oracle Server is year 2000 compliant. When a record with a date column is inserted into a table, thecentury information is picked up from the `SYSDATE` function. However, when the date column is displayed on the screen, the century component is not displayed by default.
The `DATE` data type always stores year information as a four-digit number internally, two digits for the century and two digits for the year. For example, the Oracle database stores the year as 1996 or 2001, and not just as 96 or 01.

`SYSDATE` is a function that returns:

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



`SYSDATE` is a date function that returns the current database server date and time. You can use `SYSDATE` just as you would use any other column name. For example, you can display the current date by selecting `SYSDATE` from a table. It is customary to select `SYSDATE` from a dummy table called `DUAL`.

Example

Display the current date using the `DUAL` table.

```
SELECT SYSDATE
FROM   DUAL;
```

SYSDATE
08-MAR-01

Arithmetic with Dates

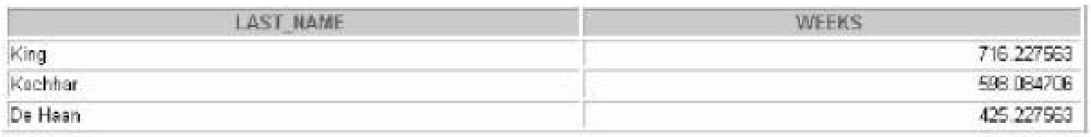
- **Add or subtract a number to or from a date for a resultant date value.**
- **Subtract two dates to find the number of days between those dates.**
- **Add hours to a date by dividing the number of hours by 24.**

Because the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as dates.

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

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



LAST_NAME	WEEKS
King	716.227563
Kochhar	598.084706
De Haan	425.227563

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.

Note: `SYSDATE` is a SQL function that returns the current date and time. Your results may differ from the example.

If a more current date is subtracted from an older date, the difference is a negative number.

Date Functions

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

Date functions operate on Oracle dates. All date functions return a value of DATE data type except MONTHS_BETWEEN, which returns a numeric value.

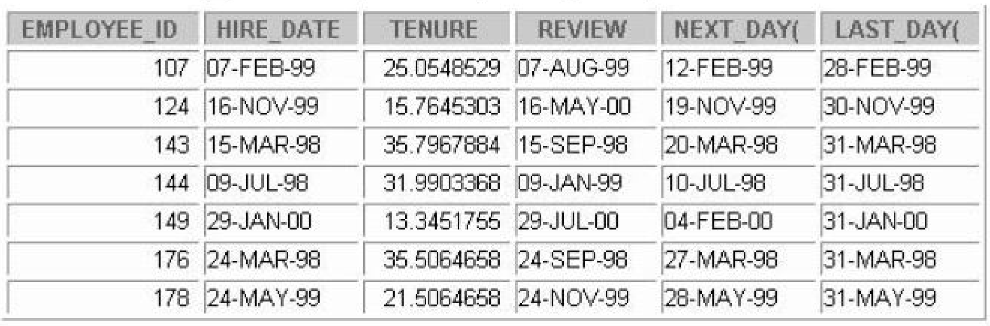
- MONTHS_BETWEEN(*date1, date2*): Finds the number of months between *date1* and *date2*. The result can be positive or negative. If *date1* is later than *date2*, the result is positive; if *date1* is earlier than *date2*, the result is negative. The noninteger part of the result represents a portion of the month.
- ADD_MONTHS(*date, n*): Adds *n* number of calendar months to *date*. The value of *n* must be an integer and can be negative.
- NEXT_DAY(*date*, '*char*'): Finds the date of the next specified day of the week ('*char*') following *date*. The value of *char* may be a number representing a day or a character string.
- LAST_DAY(*date*): Finds the date of the last day of the month that contains *date*.
- ROUND(*date*[,'*fmt*']): Returns *date* rounded to the unit specified by the format model *fmt*. If the format model *fmt* is omitted, *date* is rounded to the nearest day.
- TRUNC(*date*[, '*fmt*']): Returns *date* with the time portion of the day truncated to the unit specified by the format model *fmt*. If the format model *fmt* is omitted, *date* is truncated to the nearest day.

Using Date Functions

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

For example, display the employee number, hire date, number of months employed, six-month review date, first Friday after hire date, and last day of the month when hired for all employees employed for fewer than 36 months.

```
SELECT employee_id, hire_date,
       MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,
       ADD_MONTHS (hire_date, 6) REVIEW,
       NEXT_DAY (hire_date, 'FRIDAY'), LAST_DAY(hire_date)
FROM   employees
WHERE  MONTHS_BETWEEN (SYSDATE, hire_date) < 36;
```



EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(	LAST_DAY(
107	07-FEB-99	25.0548529	07-AUG-99	12-FEB-99	28-FEB-99
124	16-NOV-99	15.7645303	16-MAY-00	19-NOV-99	30-NOV-99
143	15-MAR-98	35.7967884	15-SEP-98	20-MAR-98	31-MAR-98
144	09-JUL-98	31.9903368	09-JAN-99	10-JUL-98	31-JUL-98
149	29-JAN-00	13.3451755	29-JUL-00	04-FEB-00	31-JAN-00
176	24-MAR-98	35.5064658	24-SEP-98	27-MAR-98	31-MAR-98
178	24-MAY-99	21.5064658	24-NOV-99	28-MAY-99	31-MAY-99

Assume SYSDATE = '25-JUL-95':

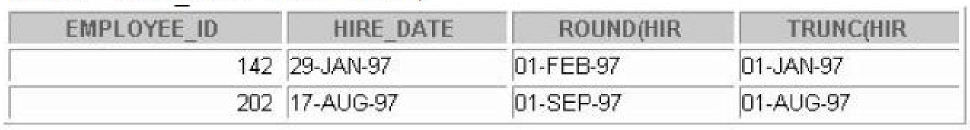
- **ROUND(SYSDATE,'MONTH')** $\longrightarrow$ **01-AUG-95**
- **ROUND(SYSDATE ,'YEAR')** $\longrightarrow$ **01-JAN-96**
- **TRUNC(SYSDATE ,'MONTH')** $\longrightarrow$ **01-JUL-95**
- **TRUNC(SYSDATE ,'YEAR')** $\longrightarrow$ **01-JAN-95**

The ROUND and TRUNC functions can be used for number and date values. When used with dates, these functions round or truncate to the specified format model. Therefore, you can round dates to the nearest year or month.

Example

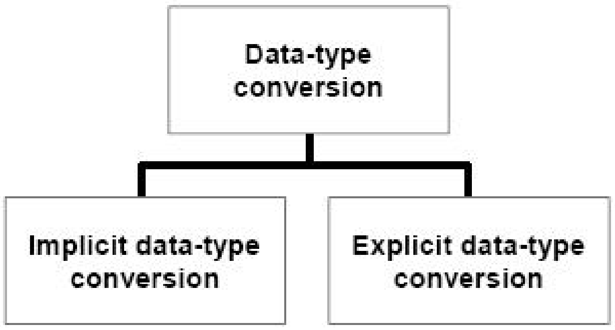
Compare the hire dates for all employees who started in 1997. Display the employee number, hire date, and month started using the ROUND and TRUNC functions.

```
SELECT employee_id, hire_date,
       ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH')
FROM   employees
WHERE  hire_date LIKE '%97';
```



EMPLOYEE_ID	HIRE_DATE	ROUND(HIR	TRUNC(HIR
142	29-JAN-97	01-FEB-97	01-JAN-97
202	17-AUG-97	01-SEP-97	01-AUG-97

Conversion Functions

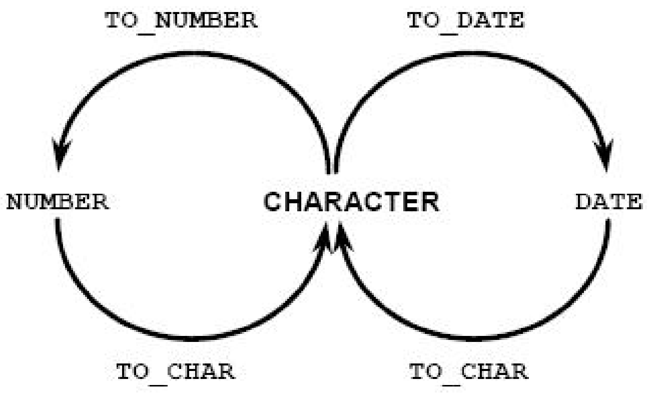


Implicit Data-Type Conversion

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



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 format model *`fmt`*. **Number Conversion:** The `nlsparams` parameter specifies the following characters, which are returned by number format elements: • Decimal character • Group separator • Local currency symbol • International currency symbol If `nlsparams` or any other parameter is omitted, this function uses the default parameter values for the session.
`TO_CHAR(number|date,[ fmt], [nlsparams])`	Specifies the language in which month and day names and abbreviations are returned. If this parameter is omitted, this function uses the default date languages for the session.
`TO_NUMBER(char,[fmt], [nlsparams])`	Converts a character string containing digits to a number in the format specified by the optional format model `fmt`. The `nlsparams` parameter has the same purpose in this function as in the `TO_CHAR` function for number conversion.
`TO_DATE(char,[fmt],[nlsparams])`	Converts a character string representing a date to a date value according to the `fmt` specified. If `fmt` is omitted, the format is DD-MON-YY. The `nlsparams` parameter has the same purpose in this function as in the `TO_CHAR` function for date 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**

Displaying a Date in a Specific Format

Previously, all Oracle date values were displayed in the DD-MON-YY format. You can use the `TO_CHAR` function to convert a date from this default format to one specified by you.

Guidelines

- The format model must be enclosed in single quotation marks and is case sensitive.
- The format model can include any valid date format element. Be sure to separate the date value from the format model by 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.
- You can format the resulting character field with the *i*SQL*Plus `COLUMN` command covered in a later lesson.

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

EMPLOYEE_ID	MONTH
205	06/94

Elements of the Date Format Model

YYYY	**Full year in numbers**
YEAR	**Year spelled out**
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**

Sample Format Elements of Valid Date Formats

Element	**Description**
SCC or CC	Century; server prefixes B.C. date with -
Years in dates YYYY or SYYYY	Year; server prefixes B.C. date with -
YYY or YY or Y	Last three, two, or one digits of year
Y,YYY	Year with comma in this position
IYYY, IYY, IY, I	Four, three, two, or one digit year based on the ISO standard
SYEAR or YEAR	Year spelled out; server prefixes B.C. date with -
BC or AD	B.C.A.D. indicator
B.C. or A.D.	B.C./A.D. indicator with periods
Q	Quarter of year
MM	Month: two-digit value
MONTH	Name of month padded with blanks to length of nine characters
MON	Name of month, three-letter abbreviation
RM	Roman numeral month
WW or W	Week of year or month
DDD or DD or D	Day of year, month, or week
DAY	Name of day padded with blanks to a length of nine characters
DY	Name of day; three-letter abbreviation
J	Julian day; the number of days since 31 December 4713 B.C.

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.**

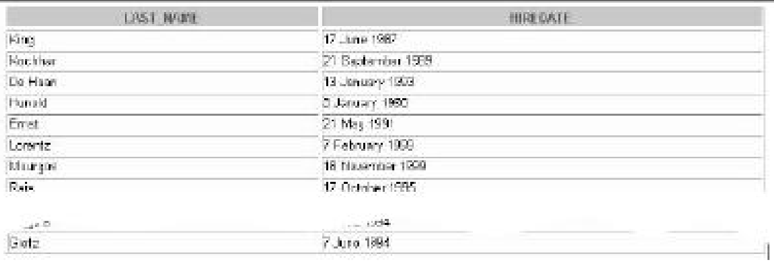
DD "of" MONTH	12 of OCTOBER

- **Number suffixes spell out numbers.**

ddspth	fourteenth

Using the `TO_CHAR` Function with Dates

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY') 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
Rajs	17 October 1995
[illegible]	[illegible] 1994
Gietz	7 June 1994

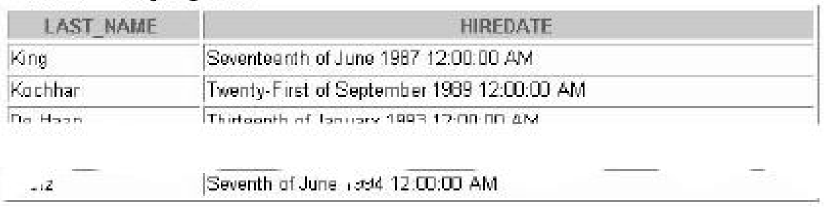
The `TO_CHAR` Function with Dates

The SQL statement on the slide displays the last names and hire dates for all the employees. The hire date appears as 17 June 1987.

Example

Modify the slide example to display the dates in a format that appears as Seventh of June 1994 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
King	Seventeenth of June 1987 12:00:00 AM
Kochhar	Twenty-First of September 1989 12:00:00 AM
De Haan	Thirteenth of January 1993 12:00:00 AM
[illegible]	Seventh of June 1994 12:00:00 AM

20 rows selected.

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

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:

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 thousand indicator

The TO_CHAR Function with Numbers

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 concatenati on.

Number Format Elements

If you are converting a number to the character data type, you can use the following format elements:

Element	Description	Example	Result
9	Numeric position (number of 9s determine display width)	999999	1234
0	Display leading zeros	099999	001234
$	Floating dollar sign	$999999	$1234
L	Floating local currency symbol	L999999	FF1234
.	Decimal point in position specified	999999.99	1234.00
,	Comma in position specified	999,999	1,234
MI	Minus signs to right (negative values)	999999MI	1234-
PR	Parenthesize negative numbers	999999PR	<1234>
EEEE	Scientific notation (format must specify four Es)	99.999EEEE	1.234E+03
V	Multiply by 10 *n* times (*n* = number of 9s after V)	9999V99	123400
B	Display zero values as blank, not 0	B9999.99	1234.00

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.

The TO_NUMBER and TO_DATE Functions

You may want to convert a character string to either a number or a date. To accomplish this task, you use the TO_NUMBER or TO_DATE functions. The format model you choose is based on the previously demonstrated format elements.

The fx modifier specifies exact matching for the character argument and date format model of a TO_DATE function:

- Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model.
- The character argument cannot have extra blanks. Without fx, the Oracle Server ignores extra blanks.
- Numeric data in the character argument must have the same number of digits as the corresponding element in the format model. Without fx, numbers in the character argument can omit leading zeroes.

Example

Display the names and hire dates of all the employees who joined on May 24, 1999. Because the fx modifier is used, an exact match is required and the spaces after the word "May" are not recognized.

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = TO_DATE('May       24, 1999', 'fxMonth DD, YYYY')
```

```
ERROR at line 3:
ORA-01858: a non-numeric character was found where a numeric was expected
```

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

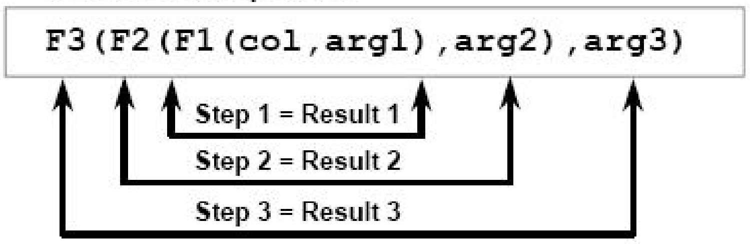
To find employees who were hired prior to 1990, the RR format can be used. Because the year is now greater than 1999, the RR format interprets the year portion of the date from 1950 to 1999.

The following command, on the other hand, results in no rows being selected because the YY format interprets the year portion of the date in the current century (2090).

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM   employees
WHERE  TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-1990';

no rows selected
```

Nesting Functions

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



```
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

Example

Display the date of the next Friday that is six months from the hire date. The resulting date should appear as Friday, August 13th, 1999. Order the results by hire date.

```
SELECT    TO_CHAR(NEXT_DAY(ADD_MONTHS
          (hire_date, 6), 'FRIDAY'),
          'fmDay, Month DDth, YYYY')
          "Next 6 Month Review"
FROM       employees
ORDER BY   hire_date;
```

General Functions

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

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

Function	Description
NVL	Converts a null value to an actual value
NVL2	If expr1 is not null, NVL2 returns expr2. If expr1 is null, NVL2 returns expr3. The argument expr1can have any data type.
NULLIF	Compares two expressions and returns null if they are equal, or the first expression if they are not equal
COALESCE	Returns the first non-null expression in the expression list

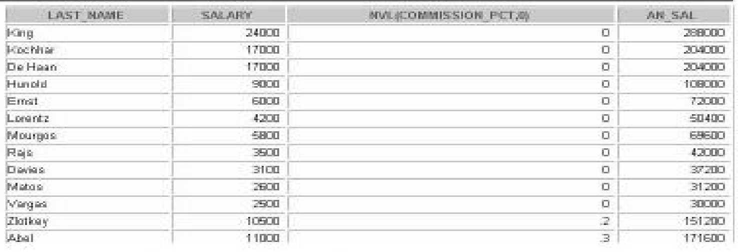
NVL Function

- **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')`

NVL Conversions for Various Data Types

Data Type	Conversion Example
NUMBER	NVL(*number_column*,9)
DATE	NVL(*date_column*, '01-JAN-95')
CHAR or VARCHAR2	NVL(*character_column*, 'Unavailable')

```
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
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
Davies	3100	0	37200
Matos	2600	0	31200
Vargas	2500	0	30000
Zlotkey	10500	.2	151200
Abel	11000	.3	171600

20 rows selected.

To convert a null value to an actual value, use the NVL function.

Syntax

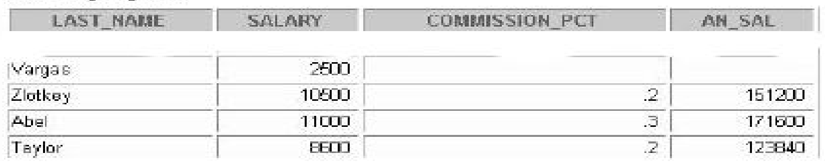
```
NVL (expr1, expr2)
```

In the syntax:

expr1	is the source value or expression that may contain a null
expr2	is the target value for converting the null

To calculate the annual compensation of all employees, you need to multiply the monthly salary by 12 and then add the commission percentage to it.

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

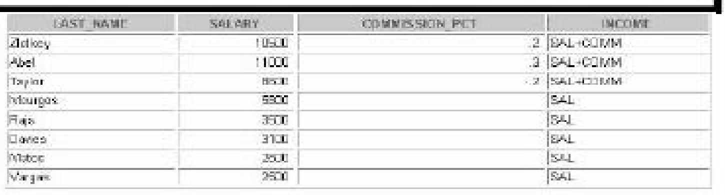


LAST_NAME	SALARY	COMMISSION_PCT	AN_SAL
Vargas	2500		
Zlotkey	10500	.2	151200
Abel	11000	.3	171600
Taylor	8600	.2	123840

20 rows selected.

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.

The NVL2 function examines the first expression. If the first expression is not null, then the NVL2 function returns the second expression. If the first expression is null, then the third expression is returned.

Syntax

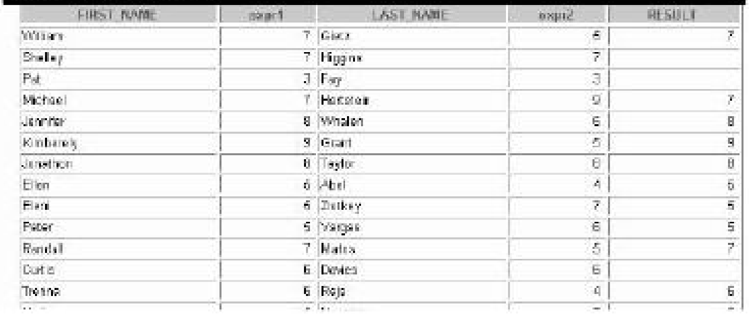
```
NVL(expr1, expr2, expr3)
```

In the syntax:

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

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
William	7	Gietz	5	7
Shelley	7	Higgins	7	
Pat	3	Fay	3	
Michael	7	Hartstein	9	7
Jennifer	8	Whalen	6	8
Kimberely	9	Grant	5	9
Jonathon	8	Taylor	6	8
Ellen	5	Abel	4	5
Eleni	5	Zlotkey	7	5
Peter	5	Vargas	6	5
Randall	7	Matos	5	7
Curtis	6	Davies	6	
Trenna	6	Rajs	4	6

20 rows selected.

The `NULLIF` function compares two expressions. If they are equal, the function returns null. If they are not equal, the function returns the first expression. You cannot specify the literal `NULL` for first expression.

Syntax

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

In the syntax:

expr1	is the source value compared to *expr2*
expr2	is the source value compared with *expr1*. (If it is not equal to *expr1*, *expr1* is returned.)

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.**

The `COALESCE` function returns the first nonnull expression in the list.

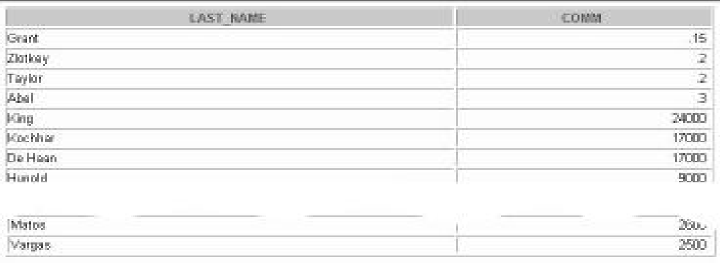
Syntax

```
COALESCE (expr1, expr2, ... exprn)
```

In the syntax:

expr1	returns this expression if it is not null
expr2	returns this expression if the first expression is null and this expression is not null
exprn	returns this expression if the preceding expressions are null

```
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
Matos	2600
Vargas	2500

20 rows selected.

Conditional Expressions

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

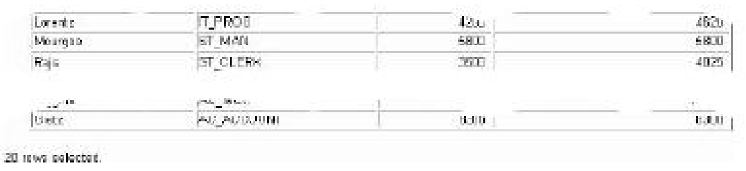
The `CASE` Expression

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
```

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;
```



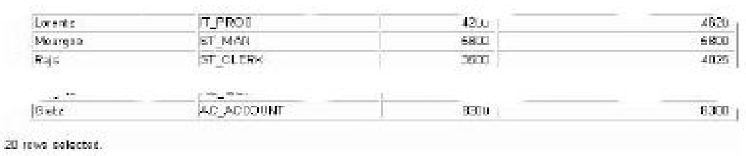
```
Lorentz   IT_PROG      4200   4620
Mourgos   ST_MAN       5800   5800
Rajs      ST_CLERK     3500   4025

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])
```

```
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;
```



```
Lorentz   IT_PROG      4200   4620
Mourgos   ST_MAN       5800   5800
Rajs      ST_CLERK     3500   4025

Gietz     AC_ACCOUNT   8300   8300

20 rows selected.
```

Example

This slide shows another example using the `DECODE` function. In this example, we determine the tax rate for each employee in department 80 based on the monthly salary. The tax rates are as per the values mentioned in the following data.

Monthly Salary Range	*Rate*
$0.00 - 1999.99	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%

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;
```

LAST_NAME	SALARY	TAX_RATE
Zlotkey	10500	.42
Abel	11000	.42
Taylor	8600	.4

Note: The `CASE` expression is new in the Oracle9*i* Server release. The `CASE` expression complies with ANSI SQL, `DECODE` is specific to Oracle syntax.

\

TABLES

In relational database systems (DBS) data are represented using *tables* (*relations*). A query issued against the DBS also results in a table. A table has the following structure:

Column 1	Column 2	...	Column n
...	...	...	...

⟵ Tuple (or Record)

A table is uniquely identified by its name and consists of *rows* that contain the stored information, each row containing exactly one *tuple* (or *record*). A table can have one or more columns. A *column* is made up of a column name and a data type, and it describes an attribute of the tuples. The structure of a table, also called *relation schema*, thus is defined by its attributes. The type of information to be stored in a table is defined by the data types of the attributes at table creation time.

SQL uses the terms *table*, *row*, and *column* for *relation*, *tuple*, and *attribute*, respectively. In this tutorial we will use the terms interchangeably.

A table can have up to 254 columns which may have different or same data types and sets of values (domains), respectively. Possible domains are alphanumeric data (strings), numbers and date formats. ORACLE offers the following basic data types:

- **char**(n): Fixed-length character data (string), n characters long. The maximum size for n is 255 bytes (2000 in ORACLE8). Note that a string of type **char** is always padded on right with blanks to full length of n. (☞ can be memory consuming).
  Example: **char**(40)
- **varchar2**(n): Variable-length character string. The maximum size for n is 2000 (4000 in ORACLE8). Only the bytes used for a string require storage. *Example:* **varchar2**(80)
- **number**(o, d): Numeric data type for integers and reals. o = overall number of digits, d = number of digits to the right of the decimal point.
  Maximum values: o =38, d= −84 to +127. *Examples:* **number**(8), **number**(5,2)
  Note that, e.g., **number**(5,2) cannot contain anything larger than 999.99 without resulting in an error. Data types derived from **number** are **int**[**eger**], **dec**[**imal**], **smallint** and **real**.
- **date**: Date data type for storing date and time.
  The default format for a date is: DD-MMM-YY. *Examples*: '13-OCT-94', '07-JAN-98'
  - **long**: Character data up to a length of 2GB. Only one **long** column is allowed per table.

Note: In ORACLE-SQL there is no data type **boolean**. It can, however, be simulated by using either **char**(1) or **number**(1).

As long as no constraint restricts the possible values of an attribute, it may have the special value *null* (for unknown). This value is different from the number 0, and it is also different from the empty string ''.

Further properties of tables are:

- the order in which tuples appear in a table is not relevant (unless a query requires an explicit sorting).
- a table has no duplicate tuples (depending on the query, however, duplicate tuples can appear in the query result).

A *database schema* is a set of relation schemas. The extension of a *database schema* at database run-time is called a *database instance* or *database*, for short.