Datatypes

A datatype associates a fixed set of properties with the values that can be used in a column of a table or in an argument of a procedure or function. These properties cause Oracle to treat values of one datatype differently from values of another datatype. For example, Oracle can add values of `NUMBER` datatype, but not values of `RAW` datatype.

Oracle supplies the following built-in datatypes:

- Character datatypes
  - `CHAR`
  - `NCHAR`
  - `VARCHAR2 and VARCHAR`
  - `NVARCHAR2`
  - `CLOB`
  - `NCLOB`
  - `LONG`
- `NUMBER` datatype
- Time and date datatypes:
  - `DATE`
  - `INTERVAL DAY TO SECOND`
  - `INTERVAL YEAR TO MONTH`
  - `TIMESTAMP`
  - `TIMESTAMP WITH TIME ZONE`
  - `TIMESTAMP WITH LOCAL TIME ZONE`
- Binary datatypes
  - `BLOB`
  - `BFILE`
  - `RAW`
  - `LONG RAW`

Table 3-1 summarizes the information about each Oracle built-in datatype.

Table 3-1 Summary of Oracle Built-In Datatypes

Datatype	**Description**	**Column Length and Default**
`CHAR (`*size* `[BYTE | CHAR])`	Fixed-length character data of length *size* bytes or characters.	Fixed for every row in the table (with trailing blanks); maximum size is 2000 bytes per row, default size is 1 byte per row. Consider the character set (single-byte or multibyte) before setting *size.*
`VARCHAR2 (`*size* `[BYTE | CHAR])`	Variable-length character data, with maximum length *size* bytes or characters.	Variable for each row, up to 4000 bytes per row. Consider the character set (single-byte or multibyte) before setting *size.* A maximum *size*must be specified.
`NCHAR (`*size*`)`	Fixed-length Unicode character data of length *size* characters.	Fixed for every row in the table (with trailing blanks). Column *size* is the number of characters. (The number of bytes is 2 times this number for the `AL16UTF16` encoding and 3 times this number for the `UTF8` encoding.) The upper limit is 2000 bytes per row. Default is 1 character.
`NVARCHAR2 (`*size*`)`	Variable-length Unicode character data of length *size* characters. A maximum *size* must be specified.	Variable for each row. Column *size* is the number of characters. (The number of bytes may be up to 2 times this number for a the `AL16UTF16`encoding and 3 times this number for the `UTF8` encoding.) The upper limit is 4000 bytes per row. Default is 1 character.
`CLOB`	Single-byte character data	Up to 2^{32} - 1 bytes, or 4 gigabytes.
`NCLOB`	Unicode national character set (`NCHAR`) data.	Up to 2^{32} - 1 bytes, or 4 gigabytes.
`LONG`	Variable-length character data.	Variable for each row in the table, up to 2^{32} - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.
`NUMBER (`*p*`,` *s*`)`	Variable-length numeric data. Maximum precision *p* and/or scale *s* is 38.	Variable for each row. The maximum space required for a given column is 21 bytes per row.
`DATE`	Fixed-length date and time data, ranging from Jan. 1, 4712 B.C.E. to Dec. 31, 4712 C.E.	Fixed at 7 bytes for each row in the table. Default format is a string (such as `DD-MON-RR`) specified by the `NLS_DATE_FORMAT` parameter.
`INTERVAL YEAR`	A period of time, represented as years and	Fixed at 5 bytes.

Datatype	Description	Column Length and Default
(*precision*) TO MONTH	months. The *precision* value specifies the number of digits in the `YEAR` field of the date. The precision can be from 0 to 9, and defaults to 2 for years.	
INTERVAL DAY (*precision*) TO SECOND (*precision*)	A period of time, represented as days, hours, minutes, and seconds. The *precision* values specify the number of digits in the `DAY` and the fractional `SECOND` fields of the date. The precision can be from 0 to 9, and defaults to 2 for days and 6 for seconds.	Fixed at 11 bytes.
TIMESTAMP (*precision*)	A value representing a date and time, including fractional seconds. (The exact resolution depends on the operating system clock.) The precision value specifies the number of digits in the fractional second part of the `SECOND` date field. The precision can be from 0 to 9, and defaults to 6	Varies from 7 to 11 bytes, depending on the precision. The default is determined by the `NLS_TIMESTAMP_FORMAT` initialization parameter.
TIMESTAMP (*precision*) WITH TIME ZONE	A value representing a date and time, plus an associated time zone setting. The time zone can be an offset from UTC, such as `'-5:0'`, or a region name, such as `'US/Pacific'`.	Fixed at 13 bytes. The default is determined by the `NLS_TIMESTAMP_TZ_FORMAT` initialization parameter.

Datatype	Description	Column Length and Default
`TIMESTAMP (`*`precision`*`) WITH LOCAL TIME ZONE`	Similar to `TIMESTAMP WITH TIME ZONE`, except that the data is normalized to the database time zone when stored, and adjusted to match the client's time zone when retrieved.	Varies from 7 to 11 bytes, depending on the precision. The default is determined by the `NLS_TIMESTAMP_FORMAT` initialization parameter.
`BLOB`	Unstructured binary data	Up to 2^{32} - 1 bytes, or 4 gigabytes.
`BFILE`	Binary data stored in an external file	Up to 2^{32} - 1 bytes, or 4 gigabytes.
`RAW (`*`size`*`)`	Variable-length raw binary data	Variable for each row in the table, up to 2000 bytes per row. A maximum *size* must be specified. Provided for backward compatibility.
`LONG RAW`	Variable-length raw binary data	Variable for each row in the table, up to 2^{31} - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.
`ROWID`	Binary data representing row addresses	Fixed at 10 bytes (extended `ROWID`) or 6 bytes (restricted `ROWID`) for each row in the table.
`CHAR (`*`size`* [BYTE | CHAR]`)`	Fixed-length character data of length *size* bytes or characters.	Fixed for every row in the table (with trailing blanks); maximum size is 2000 bytes per row, default size is 1 byte per row. Consider the character set (single-byte or multibyte) before setting *size.*
`VARCHAR2 (`*`size`* [BYTE | CHAR]`)`	Variable-length character data, with maximum length *size* bytes or characters.	Variable for each row, up to 4000 bytes per row. Consider the character set (single-byte or multibyte) before setting *size.* A maximum *size*must be specified.
`NCHAR (`*`size`*`)`	Fixed-length Unicode character data of length *size* characters.	Fixed for every row in the table (with trailing blanks). Column *size* is the number of characters. (The number of bytes is 2 times this number for the `AL16UTF16` encoding and 3 times this number for the `UTF8` encoding.) The upper limit is 2000 bytes per row. Default is 1 character.
`NVARCHAR2 (`*`size`*`)`	Variable-length Unicode character data of length *size* characters. A maximum *size* must be specified.	Variable for each row. Column *size* is the number of characters. (The number of bytes may be up to 2 times this number for a the `AL16UTF16`encoding and 3 times this

Datatype	Description	Column Length and Default
		number for the `UTF8` encoding.) The upper limit is 4000 bytes per row. Default is 1 character.
`CLOB`	Single-byte character data	Up to 2^{32} - 1 bytes, or 4 gigabytes.
`NCLOB`	Unicode national character set (`NCHAR`) data.	Up to 2^{32} - 1 bytes, or 4 gigabytes.
`LONG`	Variable-length character data.	Variable for each row in the table, up to 2^{32} - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.
`NUMBER (`*`p, s`*`)`	Variable-length numeric data. Maximum precision *p* and/or scale *s* is 38.	Variable for each row. The maximum space required for a given column is 21 bytes per row.
`DATE`	Fixed-length date and time data, ranging from Jan. 1, 4712 B.C.E. to Dec. 31, 4712 C.E.	Fixed at 7 bytes for each row in the table. Default format is a string (such as `DD-MON-RR`) specified by the `NLS_DATE_FORMAT` parameter.
`INTERVAL YEAR (`*`precision`*`) TO MONTH`	A period of time, represented as years and months. The *precision* value specifies the number of digits in the `YEAR` field of the date. The precision can be from 0 to 9, and defaults to 2 for years.	Fixed at 5 bytes.
`INTERVAL DAY (`*`precision`*`) TO SECOND (`*`precision`*`)`	A period of time, represented as days, hours, minutes, and seconds. The *precision* values specify the number of digits in the `DAY` and the fractional `SECOND` fields of the date. The precision can be from 0 to 9, and defaults to 2 for days and 6 for seconds.	Fixed at 11 bytes.

Datatype	**Description**	**Column Length and Default**
`TIMESTAMP (`*`precision`*`)`	A value representing a date and time, including fractional seconds. (The exact resolution depends on the operating system clock.) The precision value specifies the number of digits in the fractional second part of the `SECOND` date field. The precision can be from 0 to 9, and defaults to 6	Varies from 7 to 11 bytes, depending on the precision. The default is determined by the `NLS_TIMESTAMP_FORMAT` initialization parameter.
`TIMESTAMP (`*`precision`*`) WITH TIME ZONE`	A value representing a date and time, plus an associated time zone setting. The time zone can be an offset from UTC, such as `'-5:0'`, or a region name, such as `'US/Pacific'`.	Fixed at 13 bytes. The default is determined by the `NLS_TIMESTAMP_TZ_FORMAT` initialization parameter.
`TIMESTAMP (`*`precision`*`) WITH LOCAL TIME ZONE`	Similar to `TIMESTAMP WITH TIME ZONE`, except that the data is normalized to the database time zone when stored, and adjusted to match the client's time zone when retrieved.	Varies from 7 to 11 bytes, depending on the precision. The default is determined by the `NLS_TIMESTAMP_FORMAT` initialization parameter.
`BLOB`	Unstructured binary data	Up to 2^{32} - 1 bytes, or 4 gigabytes.
`BFILE`	Binary data stored in an external file	Up to 2^{32} - 1 bytes, or 4 gigabytes.
`RAW (`*`size`*`)`	Variable-length raw binary data	Variable for each row in the table, up to 2000 bytes per row. A maximum *size* must be specified. Provided for backward compatibility.
`LONG RAW`	Variable-length raw binary data	Variable for each row in the table, up to 2^{31} - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.

Datatype	**Description**	**Column Length and Default**
ROWID	Binary data representing row addresses	Fixed at 10 bytes (extended ROWID) or 6 bytes (restricted ROWID) for each row in the table.
CHAR (*size* [BYTE | CHAR])	Fixed-length character data of length *size* bytes or characters.	Fixed for every row in the table (with trailing blanks); maximum size is 2000 bytes per row, default size is 1 byte per row. Consider the character set (single-byte or multibyte) before setting *size*.
VARCHAR2 (*size* [BYTE | CHAR])	Variable-length character data, with maximum length *size* bytes or characters.	Variable for each row, up to 4000 bytes per row. Consider the character set (single-byte or multibyte) before setting *size.* A maximum *size*must be specified.
NCHAR (*size*)	Fixed-length Unicode character data of length *size* characters.	Fixed for every row in the table (with trailing blanks). Column *size* is the number of characters. (The number of bytes is 2 times this number for the AL16UTF16 encoding and 3 times this number for the UTF8 encoding.) The upper limit is 2000 bytes per row. Default is 1 character.
NVARCHAR2 (*size*)	Variable-length Unicode character data of length *size* characters. A maximum *size* must be specified.	Variable for each row. Column *size* is the number of characters. (The number of bytes may be up to 2 times this number for the AL16UTF16encoding and 3 times this number for the UTF8 encoding.) The upper limit is 4000 bytes per row. Default is 1 character.
CLOB	Single-byte character data	Up to 2^{32} - 1 bytes, or 4 gigabytes.
NCLOB	Unicode national character set (NCHAR) data.	Up to 2^{32} - 1 bytes, or 4 gigabytes.
LONG	Variable-length character data.	Variable for each row in the table, up to 2^{32} - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.
NUMBER (*p*, *s*)	Variable-length numeric data. Maximum precision *p* and/or scale *s* is 38.	Variable for each row. The maximum space required for a given column is 21 bytes per row.
DATE	Fixed-length date and time data, ranging from Jan. 1, 4712 B.C.E. to Dec. 31, 4712 C.E.	Fixed at 7 bytes for each row in the table. Default format is a string (such as DD-MON-RR) specified by the NLS_DATE_FORMAT parameter.

Datatype	Description	Column Length and Default
`INTERVAL YEAR (`*`precision`*`) TO MONTH`	A period of time, represented as years and months. The *precision* value specifies the number of digits in the `YEAR` field of the date. The precision can be from 0 to 9, and defaults to 2 for years.	Fixed at 5 bytes.
`INTERVAL DAY (`*`precision`*`) TO SECOND (`*`precision`*`)`	A period of time, represented as days, hours, minutes, and seconds. The *precision* values specify the number of digits in the `DAY` and the fractional `SECOND` fields of the date. The precision can be from 0 to 9, and defaults to 2 for days and 6 for seconds.	Fixed at 11 bytes.
`TIMESTAMP (`*`precision`*`)`	A value representing a date and time, including fractional seconds. (The exact resolution depends on the operating system clock.) The precision value specifies the number of digits in the fractional second part of the `SECOND` date field. The precision can be from 0 to 9, and defaults to 6	Varies from 7 to 11 bytes, depending on the precision. The default is determined by the `NLS_TIMESTAMP_FORMAT` initialization parameter.

Representing Character Data

Use the character datatypes to store alphanumeric data:

- `CHAR` and `NCHAR` datatypes store fixed-length character strings.

- VARCHAR2 and NVARCHAR2 datatypes store variable-length character strings. (The VARCHAR datatype is synonymous with the VARCHAR2 datatype.)
- NCHAR and NVARCHAR2 datatypes store Unicode character data only.
- CLOB and NCLOB datatypes store single-byte and multibyte character strings of up to four gigabytes.
- The LONG datatype stores variable-length character strings containing up to two gigabytes, but with many restrictions.
- This datatype is provided for backward compatibility with existing applications; in general, new applications should use CLOB and NCLOB datatypes to store large amounts of character data, and BLOB and BFILE to store large amounts of binary data.

When deciding which datatype to use for a column that will store alphanumeric data in a table, consider the following points of distinction:

Space Usage

- To store data more efficiently, use the VARCHAR2 datatype. The CHAR datatype blank-pads and stores trailing blanks up to a fixed column length for all column values, while the VARCHAR2 datatype does not add any extra blanks.

Comparison Semantics

- Use the CHAR datatype when you require ANSI compatibility in comparison semantics (when trailing blanks are not important in string comparisons). Use the VARCHAR2 when trailing blanks are important in string comparisons.

Future Compatibility

- The CHAR and VARCHAR2 datatypes are and will always be fully supported. At this time, the VARCHAR datatype automatically corresponds to the VARCHAR2 datatype and is reserved for future use.

CHAR, VARCHAR2, and LONG data is automatically converted from the database character set to the character set defined for the user session by the NLS_LANGUAGE parameter, where these are different.

Representing Numeric Data

Use the `NUMBER` datatype to store real numbers in a fixed-point or floating-point format. Numbers using this datatype are guaranteed to be portable among different Oracle platforms, and offer up to 38 decimal digits of precision. You can store positive and negative numbers of magnitude 1 x 10^{-130} through 9.99 x10^{125}, as well as zero, in a `NUMBER` column.

You can specify that a column contains a floating-point number, for example:

```
distance NUMBER
```

Or, you can specify a precision (total number of digits) and scale (number of digits to right of decimal point):

```
price NUMBER (8, 2)
```

Although not required, specifying precision and scale helps to identify bad input values. If a precision is not specified, the column stores values as given. The following table shows examples of how data different scale factors affect storage.

Table 3-2 How Scale Factors Affect Numeric Data Storage

Input Data	Specified As	Stored As
7,456,123.89	NUMBER	7456123.89
7,456,123.89	NUMBER (9)	7456124
7,456,123.89	NUMBER (9,2)	7456123.89
7,456,123.89	NUMBER (9,1)	7456123.9
7,456,123.89	NUMBER (6)	(not accepted, exceeds precision)
7,456,123.89	NUMBER (7, -2)	7456100

Representing Date and Time Data

Use the `DATE` datatype to store point-in-time values (dates and times) in a table. The `DATE` datatype stores the century, year, month, day, hours, minutes, and seconds.

Use the `TIMESTAMP` datatype to store precise values, down to fractional seconds. For example, an application that must decide which of two events occurred first might use `TIMESTAMP`. An application that needs to specify the time for a job to execute might use `DATE`.

Because `TIMESTAMP WITH TIME ZONE` can also store time zone information, it is particularly suited for recording date information that must be gathered or coordinated across geographic regions.

Use `TIMESTAMP WITH LOCAL TIME ZONE` values when the time zone is not significant. For example, you might use it in an application that schedules teleconferences, where each participant sees the start and end times for their own time zone.

The `TIMESTAMP WITH LOCAL TIME ZONE` type is appropriate for two-tier applications where you want to display dates and times using the time zone of the client system. You should not use it in three-tier applications, such as those involving a web server, because in that case the client is the web server, so data displayed in a web browser is formatted according to the time zone of the web server rather than the time zone of the browser.

Use `INTERVAL DAY TO SECOND` to represent the precise difference between two datetime values. For example, you might use this value to set a reminder for a time 36 hours in the future, or to record the time between the start and end of a race. To represent long spans of time, including multiple years, with high precision, you can use a large value for the days portion.

Use `INTERVAL YEAR TO MONTH` to represent the difference between two datetime values, where the only significant portions are the year and month. For example, you might use this value to set a reminder for a date 18 months in the future, or check whether 6 months have elapsed since a particular date.

Oracle uses its own internal format to store dates. Date data is stored in fixed-length fields of seven bytes each, corresponding to century, year, month, day, hour, minute, and second.

Date Format

For input and output of dates, the standard Oracle default date format is `DD-MON-RR`. For example:

```
'13-NOV-1992'
```

To enter dates that are not in the current default date format, use the `TO_DATE` function with a format mask. For example:

```
TO_DATE ('November 13, 1992', 'MONTH DD, YYYY')
```

Be careful using a date format like DD-MON-YY. The YY indicates the year in the current century. For example, **31-DEC-92** is **December 31, 2092**, not 1992 as you might expect. If you want to indicate years in any century other than the current one, use a different format mask, such as the default RR.

Checking If Two DATE Values Refer to the Same Day

To compare dates that have time data, use the SQL function TRUNC to ignore the time component.

Displaying the Current Date and Time

Use the SQL function SYSDATE to return the system date and time.

Time Format

Time is stored in 24-hour format, HH24:MI:SS. By default, the time in a DATE column is 12:00:00 A.M. (midnight) if no time portion is entered, or if the DATE is truncated. In a time-only entry, the date portion defaults to the first day of the current month. To enter the time portion of a date, use the TO_DATE function with a format mask indicating the time portion, as in:

```
INSERT INTO Birthdays_tab (bname, bday) VALUES
    ('ANNIE',TO_DATE('13-NOV-92 10:56 A.M.','DD-MON-YY HH:MI A.M.'));
```

Performing Date Arithmetic

Oracle provides a number of features to help with date arithmetic, so that you do not need to perform your own calculations on the number of seconds in a day, the number of days in each month, and so on.

Some useful functions include:

- ADD_MONTHS
- SYSDATE
- SYSTIMESTAMP
- TRUNC. When applied to a DATE value, it trims off the time portion so that it represents the very beginning of the day (the stroke of midnight). By truncating two DATE values and comparing them, you can check whether they refer to the same day. You can also use TRUNC along with a GROUP BY clause to produce daily totals.

- Arithmetic operators such as + and -.
- INTERVAL datatype. To represent constants when performing date arithmetic, you can use the INTERVAL datatype rather than performing your own calculations. For example, you might add or subtract INTERVAL constants from DATE values, or subtract two DATE values and compare the result to an INTERVAL.
- Comparison operators such as >, <, =, and BETWEEN.

Representing Geographic Coordinate Data

To represent Geographic Information System (GIS) or spatial data in the database, you can use the Oracle Spatial features, including the type MDSYS.SDO_GEOMETRY. You can store the data in the database using either an object-relational or a relational model, and manipulate and query the data using a set of PL/SQL packages.

Representing Image, Audio, and Video Data

Whether you store such multimedia data inside the database as BLOBs or BFILEs, or store it externally on a web or other kind of server, you can use interMedia to access the data using either an object-relational or a relational model, and manipulate and query the data using a set of object types.

Representing Large Data Types

In times gone by, the way to represent large data objects in the database was to use the LONG, RAW, and LONG RAW types. Oracle recommends that current applications use the various LOB types, such as CLOB, BLOB and BFILE, for this data.

Addressing Rows Directly with the ROWID Datatype

Every row in an Oracle table is assigned a ROWID that corresponds to the physical address of a row. If the row is too large to fit within a single data block, the ROWID identifies the initial row piece. Although ROWIDs are usually unique, different rows can have the same ROWID if they are in the same data block, but in different clustered tables.

Each table in an Oracle database has a pseudocolumn named ROWID.

How Oracle Converts Datatypes

In some cases, Oracle allows data of one datatype where it expects data of a different datatype. Generally, an expression cannot contain values with different datatypes. However, Oracle can use the following functions to automatically convert data to the expected datatype:

- `TO_NUMBER()`
- `TO_CHAR()`
- `TO_NCHAR()`
- `TO_DATE()`
- `HEXTORAW()`
- `RAWTOHEX()`
- `RAWTONHEX()`
- `ROWIDTOCHAR()`
- `ROWIDTONCHAR()`
- `CHARTOROWID()`
- `TO_CLOB()`
- `TO_NCLOB()`
- `TO_BLOB()`
- `TO_RAW()`

Implicit datatype conversions work according to the rules explained below.

Datatype Conversion During Assignments

For assignments, Oracle can automatically convert the following:

- `VARCHAR2`, `NVARCHAR2`, `CHAR`, or `NCHAR` to `NUMBER`
- `NUMBER` to `VARCHAR2` or `NVARCHAR2`
- `VARCHAR2`, `NVARCHAR2`, `CHAR`, or `NCHAR` to `DATE`
- `DATE` to `VARCHAR2` or `NVARCHAR2`
- `VARCHAR2`, `NVARCHAR2`, `CHAR`, or `NCHAR` to `ROWID`
- `ROWID` to `VARCHAR2` or `NVARCHAR2`
- `VARCHAR2`, `NVARCHAR2`, `CHAR`, `NCHAR`, or `LONG` to `CLOB`
- `VARCHAR2`, `NVARCHAR2`, `CHAR`, `NCHAR`, or `LONG` to `NCLOB`

- `CLOB` to `CHAR`, `NCHAR`, `VARCHAR2`, `NVARCHAR2`, and `LONG`
- `NCLOB` to `CHAR`, `NCHAR`, `VARCHAR2`, `NVARCHAR2`, and `LONG`
- `NVARCHAR2`, `NCHAR`, or `BLOB` to `RAW`
- `RAW` to `BLOB`
- `VARCHAR2` or `CHAR` to `HEX`
- `HEX` to `VARCHAR2`

The assignment succeeds if Oracle can convert the datatype of the value used in the assignment to that of the assignment's target.

For the examples in the following list, assume a package with a public variable and a table declared as in the following statements:

- `variable := expression`

  The datatype of *expression* must be either the same as, or convertible to, the datatype of *variable*. For example, Oracle automatically converts the data provided in the following assignment within the body of a stored procedure:

  ```
  VAR1 := 0;
  ```

- `INSERT INTO table VALUES (expression1, expression2, ...)`

  The datatypes of *expression1*, *expression2*, and so on, must be either the same as, or convertible to, the datatypes of the corresponding columns in *table*. For example, Oracle automatically converts the data provided in the following `INSERT` statement for `TABLE1` (see table definition above):

  ```
  INSERT INTO Table1_tab VALUES ('19');
  ```

- `UPDATE` *table* `SET` *column* = *expression*

  The datatype of *expression* must be either the same as, or convertible to, the datatype of *column*. For example, Oracle automatically converts the data provided in the following `UPDATE` statement issued against `TABLE1`:

  ```
  UPDATE Table1_tab SET col1 = '30';
  ```

- `SELECT` *column* `INTO` *variable* `FROM` *table*

The datatype of *column* must be either the same as, or convertible to, the datatype of *variable*. For example, Oracle automatically converts data selected from the table before assigning it to the variable in the following statement:

```
SELECT Col1 INTO Var1 FROM Table1_tab WHERE Col1 = 30;
```

Datatype Conversion During Expression Evaluation

For expression evaluation, Oracle can automatically perform the same conversions as for assignments. An expression is converted to a type based on its context. For example, operands to arithmetic operators are converted to `NUMBER`, and operands to string functions are converted to `VARCHAR2`.

Oracle can automatically convert the following:

- `VARCHAR2` or `CHAR` to `NUMBER`
- `VARCHAR2` or `CHAR` to `DATE`

Character to `NUMBER` conversions succeed only if the character string represents a valid number. Character to `DATE` conversions succeed only if the character string satisfies the session default format, which is specified by the initialization parameter `NLS_DATE_FORMAT`.

Some common types of expressions follow:

- Simple expressions, such as:
- `commission + '500'`
-
- Boolean expressions, such as:
- `bonus > salary / '10'`
-
- Function and procedure calls, such as:
- `MOD (counter, '2')`
-
- `WHERE` clause conditions, such as:
- `WHERE hiredate = TO_DATE('1997-01-01','yyyy-mm-dd')`
-
- `WHERE` clause conditions, such as:
- `WHERE rowid = 'AAAAaoAATAAAADAAA'`
-

In general, Oracle uses the rule for expression evaluation when a datatype conversion is needed in places not covered by the rule for assignment conversions.

In assignments of the form:

```
variable := expression
```

Oracle first evaluates *expression* using the conversion rules for expressions; *expression* can be as simple or complex as desired. If it succeeds, then the evaluation of *expression* results in a single value and datatype. Then, Oracle tries to assign this value to the target variable using the conversion rules for assignments.

Referencia bibliográfica.

Docs.oracle.com. (2018). *Selecting a Datatype*. [online] Available at: https://docs.oracle.com/cd/B10500_01/appdev.920/a96590/adfnstyp.htm [Accessed 8 Aug. 2018].