A
MLE Type Conversions

Supported conversions between JavaScript and PL/SQL, SQL, and JSON data types.

JavaScript target types include both native JavaScript types as well as SQL wrapper types. Supported SQL types are converted to the analogous JavaScript type by default where such a natural counterpart exists. If a conversion is attempted and there is no corresponding JavaScript type, conversion to a native JavaScript type is not supported and values are instead converted to the corresponding SQL wrapper type by default.

> **Note:**
>
> MLE does not provide functionality to prevent information loss that might occur between conversions from a customized database character representation to the built-in string representation of JavaScript (UTF-16).

> **See Also:**
>
> - Server-Side JavaScript API Documentation for information about using `mle-js-bindings` to change the default mappings when exchanging values between PL/SQL and JavaScript
> - Server-Side JavaScript API Documentation for information on how to use `mle-js-plsqltypes` to create SQL wrapper types, such as `OracleNumber`
> - Server-Side JavaScript API Documentation for information on using `mle-js-oracledb` to override the default conversions (as seen in Table A-1) when fetching column values from a `SELECT` statement

Date Conversions

JavaScript `Date` represents an instant (i.e., a single moment in time). Conversions can occur between the instant type `Date` and PL/SQL types `DATE` and `TIMESTAMP` that do not have time zone information. Conversions between instants on the JavaScript side and `DATE` and `TIMESTAMP` on the other side are handled as follows:

- When converting a `Date` to a `TIMESTAMP` or `DATE`, the instant is converted to a timezone-aware datatime value in the current session time zone. The local datatime portion of this value is stored in the target `DATE` or `TIMESTAMP` value.
- To convert a `TIMESTAMP` or `DATE` to a timezone-aware `Date`, the source datetime value is interpreted to be in the session time zone and is converted into an instant according to the session time zone.

Table A-1 Supported Mappings from SQL and PL/SQL Types to JavaScript Types

SQL Type	JavaScript Types (Bold Font Signifies Default)
NUMBER	**number** OracleNumber
BINARY_FLOAT	**number**
BINARY_DOUBLE	**number**
BINARY_INTEGER[1]	**number**
BOOLEAN	**boolean**
VARCHAR2	**string**
NVARCHAR2	**string**
CHAR	**string**
NCHAR	**string**
CLOB	**OracleCLOB** string
NCLOB	**OracleCLOB** string
BLOB	**OracleBLOB** Uint8Array (TypedArray)
RAW	**Uint8Array** (TypedArray)
DATE	**Date** OracleDate
TIMESTAMP	**Date** OracleTimestamp
TIMESTAMP WITH TIME ZONE	**Date** OracleTimestampTZ
TIMESTAMP WITH LOCAL TIME ZONE	**Date** OracleTimestampTZ
INTERVAL YEAR TO MONTH	OracleIntervalYearToMonth
INTERVAL DAY TO SECOND	OracleIntervalDayToSecond
NULL[2]	**null**
JSON	any (object, array, null)[3]

1 Note that BINARY_INTEGER is a PL/SQL type and not supported in SQL. MLE only supports BINARY_INTEGER on PL/SQL interfaces.

2 Although not technically a type, MLE converts a SQL NULL value into a JavaScript null value and vice versa. This is so that JavaScript can indicate to the database that a value passed into the database is absent (for example, the return value of a function or an IN bind in a SQL statement).

3 See MLE JavaScript Support for JSON for details

Table A-2 Supported Mappings from JavaScript Types to SQL Types

JavaScript Type	SQL Type
number boolean OracleNumber	NUMBER
number	BINARY_FLOAT
number	BINARY_DOUBLE
number boolean	BINARY_INTEGER
number OracleNumber boolean	BOOLEAN
string	VARCHAR2
string	CHAR
string	NCHAR
string	NVARCHAR2
string OracleCLOB	CLOB
string OracleCLOB	NCLOB
string	UROWID
Uint8Array OracleBlob	BLOB
UintArray	RAW
Date OracleDate	DATE
Date OracleTimestamp	TIMESTAMP
Date OracleTimestampTZ	TIMESTAMP WITH (LOCAL) TIME ZONE
OracleIntervalYearToMonth	INTERVAL YEAR TO MONTH
OracleIntervalDayToSecond	INTERVAL DAY TO SECOND
null	NULL (any supported SQL type)

Table A-2 (Cont.) Supported Mappings from JavaScript Types to SQL Types

JavaScript Type	SQL Type
`number`	JSON[2]
`string`	
`boolean`	
`null`	
`undefined`	
`Date`	
`Uint8Array`	
`OracleNumber`	
`OracleDate`	
`OracleTimestamp`	
`OracleTimestampTZ`	
`OracleIntervalYearToMonth`	
`OracleIntervalDayToSecond`	
object[1]	

1 JavaScript objects and arrays that do not match one of the classes listed above

2 See MLE JavaScript Support for JSON for details

- MLE JavaScript Support for JSON
  Supported conversions between JavaScript and the JSON data type.
- MLE JavaScript Support for the VECTOR Data Type
  Oracle Multilingual Engine (MLE) supports conversions between JavaScript TypedArrays and SQL vectors with formats `INT8`, `FLOAT32`, and `FLOAT64`. Data exchanges between JavaScript and the `VECTOR` data type are supported by the MLE JavaScript SQL driver, MLE call specifications, and MLE JavaScript bindings.

MLE JavaScript Support for JSON

Supported conversions between JavaScript and the JSON data type.

Values of the SQL `JSON` type can be converted to and from JavaScript values. The type mapping between the SQL `JSON` type and JavaScript values is aligned with type mappings employed by the `node-oracledb` driver.

> **Note:**
>
> For more information about `node-oracledb` and the `JSON` data type, see the node-oracledb documentation.

Values of the SQL `JSON` type are converted to JavaScript values as follows:

- If the `JSON` value is an object, it is converted to an equivalent JavaScript object by converting all fields of the input object.

- If the `JSON` value is an array, it is converted to an equivalent JavaScript array by converting all elements of the input array.
- If the `JSON` value is a scalar value, it is converted to an equivalent value according to the type mapping in Table A-3.

Table A-3 Mapping from JSON Attribute Types and Values to JavaScript Types and Values

JSON Attribute Type or Value	JavaScript Type or Value
`null`	`null`
`false`	`false`
`true`	`true`
`NUMBER`	`Number`
`VARCHAR2`	`String`
`RAW`	`Uint8Array`
`CLOB`	`String`
`BLOB`	`UintArray`
`DATE`	`Date`
`TIMESTAMP`	`Date`
`INVERVAL YEAR TO MONTH`	`OracleIntervalYearToMonth`
`INTERVAL DAY TO SECOND`	`OracleIntervalDayToSecond`
`BINARY_DOUBLE`	`Number`
`BINARY_FLOAT`	`Number`
Arrays	Array
Objects	A plain JavaScript Object

Values of a JavaScript type are converted to the SQL `JSON` type as follows:

- If the JavaScript value matches one of the scalar types in the first column of Table A-4, it is converted to a JSON value of the corresponding type.
- If the JavaScript value is an array, it is converted to a JSON array by converting all elements of the array. Note that `Uint8Array` values are treated as scalars as opposed to arrays, so `Uint8Array` values are converted to the type `RAW`, not to a JSON array.
- If the JavaScript value is an object that is neither an array nor matches any of the JavaScript types/ classes listed in Table A-4, it is converted to a JSON object. Each field of the object is converted according to the appropriate mappings.

Table A-4 Mapping from JavaScript Types and Values to JSON Attributes and Values

JavaScript Type or Value	JSON Attribute Type or Value
`null`	`null`
`undefined`	`null`
`string`	`VARCHAR2`
`true`	`true`
`false`	`false`
`Uint8Array`	`RAW`
`Number`	`NUMBER`
`Date`	`DATE`

Table A-4 (Cont.) Mapping from JavaScript Types and Values to JSON Attributes and Values

JavaScript Type or Value	JSON Attribute Type or Value
`OracleNumber`	`NUMBER`
`OracleDate`	`DATE`
`OracleTimestamp`	`TIMESTAMP`
`OracleTimestampTZ`	`TIMESTAMP WITH TIME ZONE`
`OracleIntervalYearToMonth`	`INVERVAL YEAR TO MONTH`
`OracleIntervalDayToSecond`	`INTERVAL DAY TO SECOND`
Array	Array
Object	Object

MLE JavaScript Support for the VECTOR Data Type

Oracle Multilingual Engine (MLE) supports conversions between JavaScript TypedArrays and SQL vectors with formats `INT8`, `FLOAT32`, and `FLOAT64`. Data exchanges between JavaScript and the `VECTOR` data type are supported by the MLE JavaScript SQL driver, MLE call specifications, and MLE JavaScript bindings.

The `VECTOR` data type can appear as an `IN`, `OUT`, and `IN OUT` bind argument, as well as a return type. The `SIGNATURE` clause of an MLE call specification supports the following JavaScript types:

- `Float32Array`
- `Float64Array`
- `Int8Array`

Table A-5 Mapping from VECTOR Data Type to JavaScript Types

SQL Type	JavaScript Type
`VECTOR(*, float32)`	`Float32Array` (TypedArray)
`VECTOR(*, float64)`	`Float64Array` (TypedArray)
`VECTOR(*, int8)`	`Int8Array` (TypedArray)
`VECTOR(*)`	`Float64Array`[1] (TypedArray)

[1] When no vector format is specified, `Float64Array` is used by default

Table A-6 Mapping from JavaScript Types to VECTOR Data Type

JavaScript Type	SQL Type
`Float32Array`	`VECTOR(*, float32)`
`Float64Array`	`VECTOR(*, float64)`
`Int8Array`	`VECTOR(*, int8)`
`Array`	`VECTOR(*, float64)`

> **See Also:**
>
> - *Oracle Database AI Vector Search User's Guide* for more information about the `VECTOR` data type and Oracle AI Vector Search capabilities

Example A-1 Use VECTOR Data Type with MLE

This example demonstrates support of the `VECTOR` data type used in arguments and as return type in MLE call specifications.

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE MLE MODULE vec_mod
LANGUAGE JAVASCRIPT AS

/**
 * Add two vectors
 * @param v1 the first vector
 * @param v2 the second vector
 * @returns the resulting vector after adding v1 and v2
 */
export function addVectors(v1, v2){
  return v1.map((element, index) => element + v2[index]);
}

/**
 * Subtract two vectors
 * @param v1 the first vector
 * @param v2 the second vector
 * @returns the resulting vector after subtracting v2 from v1
 */
export function subtractVectors(v1, v2){
  return v1.map((element, index) => element - v2[index]);
}
/

CREATE OR REPLACE PACKAGE mle_vec_pkg AS

  FUNCTION addVectors(
    input_vector1 IN VECTOR,
    input_vector2 IN VECTOR
  )
  RETURN VECTOR
    AS MLE MODULE vec_mod
    SIGNATURE 'addVectors';

  FUNCTION subtractVectors(
    input_vector1 IN VECTOR,
    input_vector2 IN VECTOR
  )
  RETURN VECTOR
    AS MLE MODULE vec_mod
    SIGNATURE 'subtractVectors';
```

```
END mle_vec_pkg;
/


SELECT mle_vec_pkg.addVectors(
  VECTOR('[1, 2]'),
  VECTOR('[3, 4]')
) AS result;
```

Result:

```
RESULT
--------------------------------------------
[4.0E+000,6.0E+000]


SELECT mle_vec_pkg.subtractVectors(
  VECTOR('[3, 4]'),
  VECTOR('[1, 2]')
) AS result;
```

Result:

```
RESULT
--------------------------------------------
[2.0E+000,2.0E+000]
```