313
JSON Data Structures

To work with JSON data in PL/SQL, you can use these data structures.

- JSON_ELEMENT_T Object Type
- JSON_OBJECT_T Object Type
- JSON_ARRAY_T Object Type
- JSON_SCALAR_T Object Type
- JSON_KEY_LIST Type

> **Note:**
>
> The time component in the `DATE` value of the `get_Date` function is truncated. If you want to preserve the time component, use `get_Timestamp` and then use `CAST AS DATE` to convert to a SQL date with the time component.

Related Topics

- *Oracle Database JSON Developer's Guide*
- The JSON standard

JSON_ELEMENT_T Object Type

`JSON_ELEMENT_T` is the supertype for the `JSON_OBJECT_T`, `JSON_SCALAR_T`, and `JSON_ARRAY_T` object types.

Description

Note the following:

- To create an instance of JSON_ELEMENT_T, use the `parse` or `load` function. See the "Constructors" section below for details.
- You cannot create an empty JSON_ELEMENT_T instance. To create an empty JSON container, create it based on one of the subtypes.
- To cast a JSON_ELEMENT_T into a subtype (for example, JSON_OBJECT_T), you need to perform an explicit cast using TREAT AS. For example:

```
TREAT (elem AS JSON_OBJECT_T)
```

Constructors

You create a JSON_ELEMENT_T instance using the `parse` or `load` function. The `parse` function takes as input VARCHAR2, CLOB, or BLOB data, and returns a JSON_ELEMENT_T instance. Similarly, the `load` function takes JSON type as an input.

```
STATIC FUNCTION parse(jsn VARCHAR2) RETURN JSON_ELEMENT_T
STATIC FUNCTION parse(jsn CLOB)     RETURN JSON_ELEMENT_T
STATIC FUNCTION parse(jsn BLOB)     RETURN JSON_ELEMENT_T
STATIC FUNCTION parse(jsn BLOB, FORMAT IN VARCHAR2) RETURN JSON_ELEMENT_T
STATIC FUNCTION load(jsn JSON)      RETURN JSON_Element_T,
```

Only UTF8-encoded JSON is passed as a BLOB.

The `parse` function takes a JSON string as input and sets up an internal representation of the JSON data. If the provided input is not valid JSON, then an error message is raised. Valid JSON has to pass the lax check of the “IS JSON” SQL condition. The input for the `load` function is a JSON type, therefore, no JSON syntax check is needed.

Serialization and Conversions

A `JSON_ELEMENT_T` instance (and all subtypes) can be serialized to a JSON string, converted to a JSON type or (if it is a scalar value) converted to a SQL value like Date or Number. Serialization is the inverse of the parse function: a string representation of the in-memory representation of the JSON data is being generated and returned as an appropriate SQL type.

The serialization and conversion functions are:

```
MEMBER FUNCTION to_String    RETURN VARCHAR2
MEMBER FUNCTION stringify    RETURN VARCHAR2

MEMBER FUNCTION to_Clob      RETURN CLOB
MEMBER FUNCTION to_Blob      RETURN BLOB
MEMBER PROCEDURE to_Clob(c IN OUT CLOB)
MEMBER PROCEDURE to_Blob(c IN OUT BLOB)

MEMBER FUNCTION to_Json      RETURN JSON

MEMBER FUNCTION to_Number    RETURN NUMBER
MEMBER FUNCTION to_Date      RETURN DATE
MEMBER FUNCTION to_Timestamp RETURN TIMESTAMP
MEMBER FUNCTION to_Boolean   RETURN BOOLEAN
```

The FUNCTION stringify is synonym of to_String. It has the same functionality.

The `to_Clob` and `to_Blob` *procedures* accept a CLOB or BLOB input and enable you to provide a LOB to be used as the serialization destination. For instance, `EMPTY_LOB` can be provided. The input LOB cannot be `NULL`.

If the `to_Clob` *function* is used, then a new CLOB is created. If you do not want to create a CLOB first, then you can use the `to_Clob` or `to_Blob` *functions*, which take no parameter and generate a temp LOB.

`to_Blob` serializes to UTF8 format only.

Introspection

Introspection enables you to discover properties of JSON objects without modifying them. The introspection functions are:

```
MEMBER FUNCTION has(key VARCHAR2) RETURN BOOLEAN,
MEMBER FUNCTION is_Object         RETURN BOOLEAN
MEMBER FUNCTION is_Array          RETURN BOOLEAN
MEMBER FUNCTION is_Scalar         RETURN BOOLEAN
```

```
MEMBER FUNCTION is_String         RETURN BOOLEAN
MEMBER FUNCTION is_Number         RETURN BOOLEAN
MEMBER FUNCTION is_Boolean        RETURN BOOLEAN
MEMBER FUNCTION is_True           RETURN BOOLEAN
MEMBER FUNCTION is_False          RETURN BOOLEAN
MEMBER FUNCTION is_Null           RETURN BOOLEAN
MEMBER FUNCTION is_Date           RETURN BOOLEAN
MEMBER FUNCTION is_Timestamp      RETURN BOOLEAN
MEMBER FUNCTION get_Size          RETURN NUMBER
```

The `has` function checks if a given name exists in the JSON_OBJECT_T object.

The return value of the `get_size` function depends on the JSON type:

- For a scalar, it returns `1`.
- For an object, it returns the number of keys.
- For an array, it returns the number of items.

Note that textual JSON does not support dates and timestamps natively. Instead they are typically modeled as strings. You can use the JSON type to preserve dates and timestamps natively. The Document Object Model (DOM) interface enables you to add dates and timestamps as scalar values and preserve them until serialization to JSON, where they are printed as string following the ISO 8601 format. If a SQL value of type date or timestamp has been added, then the `is_Date` and `is_Timestamp` functions return true. If a date has been added as a string (e.g. as ISO 8601), then the `is_Date` and `is_Timestamp` functions return false. You can use Oracle conversion function `to_Date` and `to_Timestamp` to convert a string representation of a date, and timestamp to the Oracle representation.

Comparing

These functions enables the users to compare two JSON elements or objects or arrays or scalars . The comparing functions are:

```
MEMBER FUNCTION equals(cmp <JSON_ELEMENT_T or JSON>) RETURN BOOLEAN
MEMBER FUNCTION equals(cmp JSON_ELEMENT_T,
order_by_compare BOOLEAN DEFAULT FALSE)              RETURN NUMBER
MEMBER FUNCTION equals(cmp JSON,
order_by_compare BOOLEAN DEFAULT FALSE)              RETURN NUMBER
MEMBER FUNCTION compare_to(JSON_SCALAR_T)            RETURN NUMBER
```

The comparing functions returns a numeric values corresponding to the outcome of the comparison function. 0 indicates that the compared values are same. -1 indicates the value is smaller than the compared value. +1 indicates that the value is greater than the compared value. NULL value is returned if the values can not be compared due to type incompatibility. If the Boolean argument `order_by_compare` is set to true, incompatible types could also be compared. In this case, a comparison value of NULL would never be returned.

Error Handling

You can set the level of error handling for JSON processing. You might not want an error to be raised for every mismatch. The `on_Error` procedure enables you to specify when errors should be raised:

```
MEMBER PROCEDURE on_Error(value NUMBER)
```

The `on_Error` procedure defines what happens if an error is encountered during a PL/SQL operation, for instance, a get call.

The default is to not raise an error but to return NULL instead.

You invoke `On_error` on a JSON_ELEMENT_T instance and it sets the error behavior for all subsequent calls. To reset the behavior to the default, you can call `on_Error(0)`.

Values for the `value` parameter are:

Table 313-1 Values for the value Parameter in the ON_ERROR Procedure

Value	Description
0	Reset to the default behavior, which is to return NULL instead of raising an error.
1	Raise all errors.
2	Raise an error if no value is detected.
3	Raise an error if the data types do not match, for example, if you call GET_NUMBER on a string value.
4	Raise an error if the input is invalid, for example, if the array is out of bounds.

You can combine values. For example, you can specify 7 to indicate a combination of 3 and 4.

In the following example, an error is raised because the value of “a” is “xyz”, which cannot be converted to a number. If the `on_Error` procedure had not been called, then NULL would be returned, and no error would be raised.

```
declare
     jo JSON_OBJECT_T;
begin
     jo := JSON_OBJECT_T.parse('{a:"xyz"}');
     jo.On_error(1);
     dbms_output.put_line(jo.get_Number('a'));
end;
/
```

JSON_OBJECT_T Object Type

`JSON_OBJECT_T` is a subtype of the JSON_ELEMENT_T object type. It corresponds to the JSON object structure.

Constructors

You can create an empty JSON_OBJECT_T instance using the following constructor.

```
   CONSTRUCTOR FUNCTION JSON_OBJECT_T RETURN SELF AS RESULT
```

You can create a JSON_OBJECT_T instance using one of the following `parse` functions:

```
STATIC FUNCTION parse(jsn VARCHAR2) RETURN JSON_OBJECT_T
STATIC FUNCTION parse(jsn CLOB)     RETURN JSON_OBJECT_T
STATIC FUNCTION parse(jsn BLOB)     RETURN JSON_OBJECT_T
STATIC FUNCTION parse(jsn BLOB, FORMAT IN VARCHAR2) RETURN JSON_OBJECT_T
```

You can also create a JSON_OBJECT_T instance using one of the following constructors:

```
CONSTRUCTOR FUNCTION JSON_OBJECT_T(jsn VARCHAR2) RETURN SELF AS RESULT
CONSTRUCTOR FUNCTION JSON_OBJECT_T(jsn CLOB) RETURN SELF AS RESULT
CONSTRUCTOR FUNCTION JSON_OBJECT_T(jsn BLOB) RETURN SELF AS RESULT
CONSTRUCTOR FUNCTION JSON_Object_T(jsn BLOB, FORMAT IN VARCHAR2) RETURN SELF AS RESULT
```

```
CONSTRUCTOR FUNCTION JSON_OBJECT_T(jsn JSON) RETURN SELF AS RESULT
CONSTRUCTOR FUNCTION JSON_OBJECT_T(e JSON_ELEMENT_T) RETURN SELF AS RESULT
```

Only UTF8-encoded JSON is passed as a BLOB.

The `parse` function takes a JSON string as input and sets up an internal representation of the JSON data. If the provided input is not a valid JSON object, then an error message is raised. The input has to specify a JSON object, not an array.

Get Functions and Procedures

The following functions and procedures enable you to retrieve the value of the JSON object:

```
MEMBER FUNCTION get(key VARCHAR2)              RETURN JSON_ELEMENT_T
MEMBER FUNCTION get_String(key VARCHAR2)       RETURN VARCHAR2
MEMBER FUNCTION get_Number(key VARCHAR2)       RETURN NUMBER
MEMBER FUNCTION get_Date(key VARCHAR2)         RETURN DATE
MEMBER FUNCTION get_Timestamp(key VARCHAR2)    RETURN TIMESTAMP
MEMBER FUNCTION get_TimestampTZ(key VARCHAR2) RETURN TIMESTAMP WITH TIME ZONE
MEMBER FUNCTION get_Boolean(key VARCHAR2)      RETURN BOOLEAN
MEMBER FUNCTION get_Clob(key VARCHAR2)         RETURN CLOB
MEMBER FUNCTION get_Blob(key VARCHAR2)         RETURN BLOB
MEMBER FUNCTION get_Object(key VARCHAR2)       RETURN JSON_OBJECT_T
MEMBER FUNCTION get_Array(key VARCHAR2)        RETURN JSON_ARRAY_T
MEMBER FUNCTION get_Json(key NUMBER)           RETURN JSON

MEMBER PROCEDURE get_Json(key NUMBER, j OUT NOCOPY JSON)
MEMBER PROCEDURE get_Clob(key NUMBER, c IN OUT CLOB)
MEMBER PROCEDURE get_Blob(key NUMBER, c IN OUT BLOB)
```

Note:

- The `get` function has reference semantics. This means that if the returned JSON_ELEMENT_T is modified, then the containing JSON_ELEMENT_T is also changed. See the Reference Semantics section below for details.
- The `GET_STRING` function converts the value to a string if it is not already a string. Thus, the `GET_STRING` function returns a non-null value even if IS_STRING returns false.
- All the “get” functions perform a conversion if possible. If no conversion is possible, then an error might be raised, depending on what `ON_ERROR` is set to.

The `GET_CLOB` and `GET_BLOB` *procedures*, which accept a CLOB or BLOB as input, enable you to provide a LOB to be used as serialization destination. For instance, EMPTY_LOB can be provided. If you use the `GET_CLOB` *function* instead, then a new CLOB will be created implicitly. The input LOB cannot be NULL. If you do not want to create a BLOB or CLOB first, you can use the GET_CLOB or GET_BLOB functions, which take no parameter and generates a temp LOB.

`GET_BLOB` serializes to UTF8 format only.

Set Procedures

The following procedures enable you to set the value of a JSON object. Existing values, if present, will be overwritten.

```
MEMBER PROCEDURE put(key VARCHAR2, val JSON_ELEMENT_T)
MEMBER PROCEDURE put(key VARCHAR2, val VARCHAR2)
MEMBER PROCEDURE put(key VARCHAR2, val NUMBER)
MEMBER PROCEDURE put(key VARCHAR2, val BOOLEAN)
MEMBER PROCEDURE put(key VARCHAR2, val DATE)
MEMBER PROCEDURE put(key VARCHAR2, val TIMESTAMP)
```

```
MEMBER PROCEDURE put_Null(key VARCHAR2)
MEMBER PROCEDURE put(key VARCHAR2, val JSON)
```

Introspection Functions

Introspection enables you to discover properties of JSON objects without modifying them. The introspection functions are:

```
MEMBER FUNCTION has(key VARCHAR2)      RETURN BOOLEAN
MEMBER FUNCTION get_Type(key VARCHAR2) RETURN VARCHAR2
MEMBER FUNCTION get_Keys               RETURN JSON_KEY_LIST
```

The get_Keys function returns an object type of JSON_KEY_LIST, which is a varray of VARCHAR2(4000). The varray contains the names of keys in the JSON object. The get_Keys function returns at most 32767 field names for a given JSON object. An error is raised if it is applied to an object with more than 32767 fields.

The example below walks through the items of the varray to build a JSON_ARRAY_T object that contains all key names.

```
declare
  jo JSON_OBJECT_T;
  ja JSON_ARRAY_T;
  keys JSON_KEY_LIST;
  keys_string VARCHAR2(100);

begin
  ja := new JSON_ARRAY_T;
  jo := JSON_OBJECT_T.parse('{"name":"fred",
                             "jobTitle":"codemonkey",
                             "projects":["json", "xml"]}');
  keys := jo.get_keys;
  for i in 1..keys.count loop
     ja.append(keys(i));
  end loop;

  keys_string := ja.to_string;
  dbms_output.put_line(keys_string);
end;
/
```

The output is:

```
["name","jobTitle","projects"]
```

Modification Procedures

The following procedures enable you to remove or rename keys in a JSON object.

```
MEMBER PROCEDURE remove(key VARCHAR2)
MEMBER PROCEDURE rename_Key(keyOld VARCHAR2, keyNew VARCHAR2)
```

Duplicate key names are not supported and will raise an error.

Clone Function

This function makes a copy of the JSON object. Reference semantics is changed to value semantics.

```
MEMBER FUNCTION clone RETURN JSON_OBJECT_T
```

Reference Semantics

Calling the `get` function that returns a JSON_ELEMENT_T object always returns a reference to the complex values instead of a copy. This means that changing the returned value affects its container. See the example below.

```
declare
  data    JSON_OBJECT_T;
  address JSON_OBJECT_T;
  zip     number;

begin

  data := new JSON_OBJECT_T('{
    "first": "John",
    "last": "Doe",
    "address": {
        "country": "USA",
        "zip": "94065"
    }
  }');

  address := data.get_object('address');
  dbms_output.put_line(address.to_string);

  -- 1) VALUE SEMANTICS for scalar values
  -- (changing the value has no effect on container)
  zip := address.get_number('zip');
  dbms_output.put_line(zip);
  zip := 12345;
  dbms_output.put_line(zip);
  -- address is still the same
  dbms_output.put_line(address.to_string);


  -- 2) REFERENCE SEMANTICS for complex values
  --    'address' is a reference to the complex address values inside 'data'
  address.put('zip', 12345);
  address.put('street', 'Detour Road');
  dbms_output.put_line(data.to_string);
end;
/
```

In cases where you do not want the reference semantics, you can use the `clone` function to create a copy of the returned object. This decouples the value from its container. In the example above, you can create a copy of the “address” object by replacing this line:

```
address := data.get_object('address');
```

with this line:

```
address := data.get_object('address').clone;
```

After this, changing the address with have no effect on the value of the ‘data’ containing object.

Update Example

The following example updates the price of an item by 10%.

```
WITH
  FUNCTION updatePrice(jsonTxt in VARCHAR2 ) RETURN VARCHAR2 IS
```

```
  jo JSON_OBJECT_T;
  oldPrice NUMBER;

  BEGIN
    jo := new JSON_OBJECT_T(jsonTxt);
    oldPrice := jo.get_number('price');
    jo.put('price', oldPrice * 1.1);
    RETURN jo.to_string();
  END;
SELECT updatePrice(col)
FROM   t1;
```

JSON_ARRAY_T Object Type

`JSON_ARRAY_T` is a subtype of the `JSON_ELEMENT_T` object type. `JSON_ARRAY_T` corresponds to the JSON array structure.

Constructors

You can create an empty JSON_ARRAY_T instance using the following constructor.

```
CONSTRUCTOR FUNCTION JSON_ARRAY_T RETURN SELF AS RESULT
```

You can create a JSON_ARRAY_T instance using one the following `parse` functions:

```
STATIC FUNCTION parse(jsn VARCHAR2)                   RETURN JSON_ARRAY_T
STATIC FUNCTION parse(jsn CLOB)                       RETURN JSON_ARRAY_T
STATIC FUNCTION parse(jsn BLOB)                       RETURN JSON_ARRAY_T
STATIC FUNCTION parse(jsn BLOB, FORMAT IN VARCHAR2)   RETURN JSON_ARRAY_T
```

You can also create a JSON_ARRAY_T instance using one the following constructors:

```
CONSTRUCTOR FUNCTION JSON_Array_T(jsn VARCHAR2) RETURN SELF AS RESULT
CONSTRUCTOR FUNCTION JSON_Array_T(jsn CLOB) RETURN SELF AS RESULT
CONSTRUCTOR FUNCTION JSON_Array_T(jsn BLOB) RETURN SELF AS RESULT
CONSTRUCTOR FUNCTION JSON_Array_T(e JSON_ELEMENT_T) RETURN SELF AS RESULT
```

Only UTF8-encoded JSON is passed as a BLOB.

The `parse` function takes a JSON string as input and sets up an internal representation of the JSON data. If the provided input is not valid JSON, then an error message is raised. The input has to specify a JSON array, not an object.

Get Functions and Procedures

The following functions and procedures enable you to retrieve the value of the JSON array:

```
MEMBER FUNCTION get(pos NUMBER)           RETURN JSON_ELEMENT_T
MEMBER FUNCTION get_String(pos NUMBER)    RETURN VARCHAR2
MEMBER FUNCTION get_Number(pos NUMBER)    RETURN NUMBER
MEMBER FUNCTION get_Date(pos NUMBER)      RETURN DATE
MEMBER FUNCTION get_Timestamp(pos NUMBER) RETURN TIMESTAMP
MEMBER FUNCTION get_Boolean(pos NUMBER)   RETURN BOOLEAN
MEMBER FUNCTION get_Clob(pos NUMBER)      RETURN CLOB
MEMBER FUNCTION get_Blob(pos NUMBER)      RETURN BLOB
MEMBER FUNCTION get_Json(pos NUMBER)      RETURN JSON

MEMBER PROCEDURE get_Json(pos NUMBER, j IN OUT NOCOPY JSON)
```

```
MEMBER PROCEDURE get_Clob(pos NUMBER, c IN OUT CLOB)
MEMBER PROCEDURE get_Blob(pos NUMBER, c IN OUT BLOB)
```

Note:

- The `get` function has reference semantics. This means that if the returned JSON_ELEMENT_T is modified, then the containing JSON_ELEMENT_T is changed too.
- The `GET_STRING` function converts the value to a string if it is not already a string. The function returns a non-null value even if `IS_STRING` returns false.
- All the “get” functions perform a conversion if possible. If no conversion is possible, then an error might be raised, depending on what `ON_ERROR` is set to.

The `GET_CLOB` and `GET_BLOB` *procedures*, which accept a `CLOB` or `BLOB` as input, enable you to provide a LOB to be used as serialization destination. For instance, `EMPTY_LOB` can be provided. If you use the `GET_CLOB` *function* instead, then a new `CLOB` will be created implicitly. The input `LOB` cannot be NULL. If you do not want to create a `BLOB` or `CLOB` first, you can use the `GET_CLOB` or `GET_BLOB` functions, which take no parameter and generates a temp LOB.

`GET_BLOB` serializes to UTF8 format only.

Set Procedures

The following procedures enable you to set the value at the specified position in the JSON array. These procedures insert (not overwrite) at the specified position unless overwrite is requested.

```
MEMBER PROCEDURE put(pos NUMBER, val VARCHAR2, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, val NUMBER, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, val BOOLEAN, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, val DATE, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, val JSON_ELEMENT_T, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, val JSON, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put(pos NUMBER, val TIMESTAMP, overwrite BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE put_Null(pos NUMBER, overwrite BOOLEAN DEFAULT FALSE)
```

The following procedures append the specified value to the end of the JSON array:

```
MEMBER PROCEDURE append(val JSON_ELEMENT_T)
MEMBER PROCEDURE append(val VARCHAR2)
MEMBER PROCEDURE append(val NUMBER)
MEMBER PROCEDURE append(val BOOLEAN)
MEMBER PROCEDURE append(val DATE)
MEMBER PROCEDURE append(val JSON)
MEMBER PROCEDURE append(val TIMESTAMP)
MEMBER PROCEDURE append_all(arr JSON_ELEMENT_T, excl_exis BOOLEAN DEFAULT FALSE)
MEMBER PROCEDURE append_Null
```

> **Note:**
>
> No sorting happens in append_all procedure. Deduplication happens only in the arr2, the new array that is being appended. If the original array has duplicates, they are left unaltered.

Introspection Function

Introspection enables you to discover properties of the JSON array without modifying them.

```
MEMBER FUNCTION get_Type(pos NUMBER) RETURN VARCHAR2
MEMBER FUNCTION has_value(val <data_type>) RETURN BOOLEAN
```

The `has_value` function can be called on a JSON_ARRAY_T type instance. It checks if the array contains the `val` argument and returns TRUE; otherwise it returns FALSE. The function takes an argument `val` of data type VARCHAR2, NUMBER, BOOLEAN, DATE, TIMESTAMP, JSON, and JSON_ELEMENT_T.

Modification Procedure

The following procedures enables you to remove the value at the specified position, removes the duplicate values, or sort the elements of the array targeted by the specified path in the JSON array.

```
MEMBER PROCEDURE remove(pos NUMBER)
MEMBER PROCEDURE sort(ascending BOOLEAN DEFAULT TRUE)
MEMBER PROCEDURE deduplicate
```

The `deduplicate` procedure does not reorder the items.

Clone Function

This function makes a copy of the JSON array. Reference semantics is changed to value semantics.

```
MEMBER FUNCTION clone RETURN JSON_ARRAY_T
```

JSON_SCALAR_T Object Type

`JSON_SCALAR_T` is a subtype of the `JSON_ELEMENT_T` object type.

Description

A `JSON_SCALAR_T` instance captures one scalar values, for example, the string “fred” or the number 1. This type does not have any functions or procedures other than the ones inherited from `JSON_ELEMENT_T`. You cannot create an instance of this type directly.

JSON_KEY_LIST Type

`JSON_KEY_LIST` is a varray of `VARCHAR2(4000)`.

Description

This type is used by the `get_Keys` function in the JSON_OBJECT_T Object Type.