Uwe Simon

Dynamic APEX UI for JSON data with the JSON Region plugin

Thu, 20 March, 10:55 | Bach 1 & 2

‘ReeHorst’, Ede

#APEXWorld2025

About Me

• Name:	Uwe Simon
• Oracle-DB:	Oracle-DB experience since 1992 starting with Oracle 5 Database modelling Performance tuning of huge multi 100TB databases Database analysis of “unknown” mission critical databases during major incident procedures DB-migrations Proof of Concepts
• APEX:	Started 1998 with OAS/OWS, HTML-DB, APEX ...24.2
• Development:	SQL, PL/SQL, C++, JavaScript, Java, HTML/CSS, ...
• Other DBs:	DB2, MySQL, PostgreSQL

- Located in Cologne
  during the last years partial on-site in The Netherlands, Czech Republic, India
- In partial retirement and Freelancer since 2023

Agenda

History of the plug-in

- The development of the JSON-Region-Plugin started October 2023

- First presentation of the plug-in at APEX-connect2024 in Germany,
- Michael Hichwa (Oracle) gave the feedback that my plug-in is great and that “Oracle definitely needs such a solution for JSON in APEX”
- Oracle introduced a solution using new JSON/Duality-source in APEX 24.2, implementing parts of the plug-ins functions

- The plug-in is continuously updated with new features for APEX 20.2-24.2

JSON-Schema in nutshell

- Documentation of JSON-Schema could be found at https://json-schema.org/
- JSON-schema is a description for the structure of JSON-data and could be used for the validation of JSON-data
  - A JSON-schema is an object or an array (“type“: “object“, “type”: “array“) with attributes (“properties”, “items”)
  - For each attribute it defines it’s data type (“type“) and format (“format“),
    if it’s “mandatory” (“required“),
    if it’s an enumeration (“enum“),
    must match a pattern (“pattern“), ...
  - An attribute could be an object or an array, .
- Oracle23ai supports JSON-schema-validations on a CLOB/JSON-columns and collection tables/views, duality-views

```
{
  "type": "object",
  "required": ["enum", "short_string"],
  "properties": {
    "enum":         { "type": "string", "enum": [ "val1", "val2" ]},
    "short_string": { "type": "string" },
    "long_string":  { "type": "string", "maxLength": 400},
    "bool":         { "type": "boolean"},
    "int":          { "type": "integer" },
    "number":       { "type": "number" },
    "date":         { "type": "string", "format": "date"},
    "date_time":    { "type": "string", "format": "date-time"},
    "email":        { "type": "string", "format": "email"},
    "uri":          { "type": "string", "format": "uri"},
    "pattern":      { "type": "string", "pattern": "[0-9]{4}( [0-9]{4}){3}"}
  }
}
```

So what could be more obvious than using JSON Schema for the APEX UI.

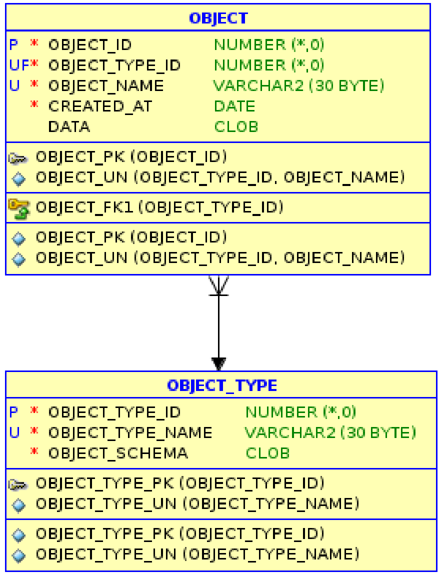
Idea of the plug-in

- APEX <24.2 offers no Out-Of-The-Box-solution for input/output of JSON-data
- The plug-in generates at runtime a dynamic APEX-UI based on a JSON-schema
- Support of all major APEX-item-types
- Use of variable JSON-schemas for each row of a table
- Modifying the JSON-schema changes the UI immediately without any modification in the APEX-code
- Transparent for user, “look like APEX-UI”, so same error-handling, same items, ...
- Customization of the UI with JSON-schema-extention by new property **`“apex”: {...}`**
- Support of JSON-schema-references **`“$ref”: “...”`**
- Support of conditional JSON-schema for dynamic UIs

  ```
  “dependentRequired”: {},
  “dependentSchemas”: {},
  ”$if”: {}, “$then”: {}, “$else”: {}
  ```

Possible usecases of the plug-in

- Configurable workflows:
  The data for the workflow is stored in JSON-columns
- Configurable Asset-Management-Systems:
- Attributes depending on asset types are stored in JSON-columns
- Form-tools:
  Form-structure is a JSON-schema and form-data is stored in JSON-columns
- Polling-tools:
  Questions and list of answers are stored in a JSON-schema and the data is stored in a JSON-column
- APEX-applications with customization by customer:
  Customizing of the application is possible with JSON-columns.



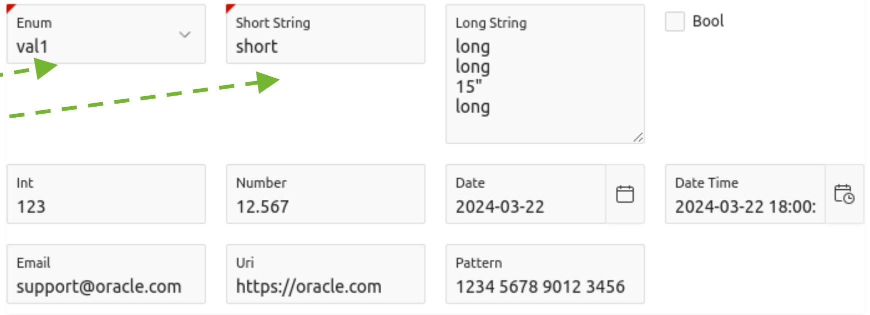
```
OBJECT
P  * OBJECT_ID          NUMBER (*,0)
UF*  OBJECT_TYPE_ID     NUMBER (*,0)
U  * OBJECT_NAME        VARCHAR2 (30 BYTE)
   * CREATED_AT         DATE
     DATA               CLOB
OBJECT_PK (OBJECT_ID)
OBJECT_UN (OBJECT_TYPE_ID, OBJECT_NAME)
OBJECT_FK1 (OBJECT_TYPE_ID)
OBJECT_PK (OBJECT_ID)
OBJECT_UN (OBJECT_TYPE_ID, OBJECT_NAME)

OBJECT_TYPE
P  * OBJECT_TYPE_ID     NUMBER (*,0)
U  * OBJECT_TYPE_NAME   VARCHAR2 (30 BYTE)
   * OBJECT_SCHEMA      CLOB
OBJECT_TYPE_PK (OBJECT_TYPE_ID)
OBJECT_TYPE_UN (OBJECT_TYPE_NAME)
OBJECT_TYPE_PK (OBJECT_TYPE_ID)
OBJECT_TYPE_UN (OBJECT_TYPE_NAME)
```

Plug-in vs. JSON-support of APEX-24.2

Feature	**Plug-in**	**APEX 24.2**
VARCHAR2/CLOB/JSON column	√	√
collection-table	√	√
collection-view	√	–
JSON-duality-view	√	√
Fixed JSON-schema	√	√ JSON-source
Variable JSON-schema	√	–
Evaluation of JSON-schema	At runtime	In page-designer
JSON-schema from DB (23ai)	√	√– (Duality-source only)
Dynamic UI	√	–
Schema references	√	–
Conditional JSON-schema	√	–
Interactive grid/report	√– with JSON-Item-Plug-in	√

JSON-schema, JSON-data and APEX-UI

```
{
  "type": "object",
  "required": ["enum", "short_string"],
  "properties": {
    "enum":         { "type": "string", "enum": [ "val1", "val2" ]},
    "short_string": { "type": "string" },
    "long_string":  { "type": "string", "maxLength": 400},
    "bool":         { "type": "boolean"},
    "int":          { "type": "integer" },
    "number":       { "type": "number" },
    "date":         { "type": "string", "format": "date"},
    "date_time":    { "type": "string", "format": "date-time"},
    "email":        { "type": "string", "format": "email"},
    "uri":          { "type": "string", "format": "uri"},
    "pattern":      { "type": "string", "pattern": "[0-9]{4}( [0-9]{4}){3}"}
  }
}
```

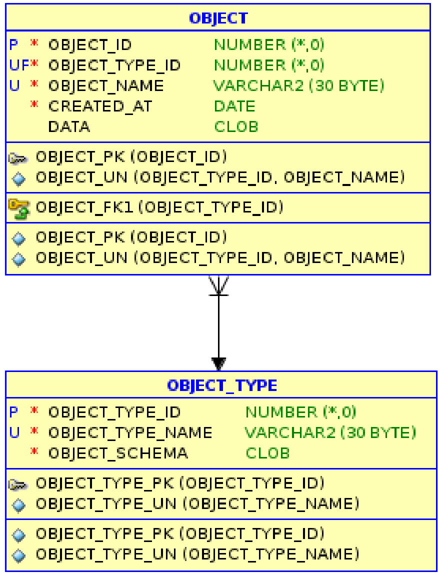


Enum: val1	Short String: short	Long String: long long 15" long	Bool
Int: 123	Number: 12.567	Date: 2024-03-22	Date Time: 2024-03-22 18:00:
Email: support@oracle.com	Uri: https://oracle.com	Pattern: 1234 5678 9012 3456	

```
{
  "enum":"val1",
  "short_string":"short",
  "long_string":"long\nlong\n15\"\nlong",
  "bool":false,
  "int":123,
  "number":12.567,
  "date":"2024-03-22",
  "date_time":"2024-03-22T18:00:00",
  "email":"support@oracle.com",
  "uri":"https://oracle.com",
  "pattern":"1234 5678 9012 3456"
}
```

Demo

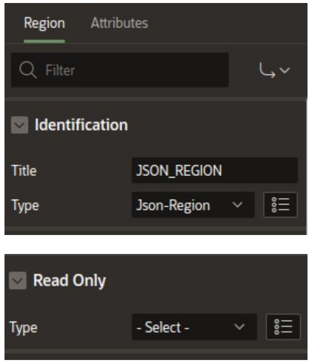
- A demo says more than 1000 slides
- My demo uses a simple data-model
  - Table “OBJECT” for „generic“ objects in JSON-column “DATA” (CLOB/JSON)
  - Table “OBJECT_TYPE” containing the valid object-types with column “OBJECT_SCHEMA” for the JSON-schema of each type
  - JSON-collection-table/view “TABLE23AI”, “VIEW23AI”
  - JSON-duality-view “JSON23AI”
  - APEX-24.2 JSON-source, Duality-source
  - When time left: 2 applications using the plug-in



```
OBJECT
P  * OBJECT_ID          NUMBER (*,0)
UF* OBJECT_TYPE_ID      NUMBER (*,0)
U  * OBJECT_NAME        VARCHAR2 (30 BYTE)
   * CREATED_AT         DATE
     DATA               CLOB
OBJECT_PK (OBJECT_ID)
OBJECT_UN (OBJECT_TYPE_ID, OBJECT_NAME)
OBJECT_FK1 (OBJECT_TYPE_ID)
OBJECT_PK (OBJECT_ID)
OBJECT_UN (OBJECT_TYPE_ID, OBJECT_NAME)

OBJECT_TYPE
P  * OBJECT_TYPE_ID       NUMBER (*,0)
U  * OBJECT_TYPE_NAME     VARCHAR2 (30 BYTE)
   * OBJECT_SCHEMA        CLOB
OBJECT_TYPE_PK (OBJECT_TYPE_ID)
OBJECT_TYPE_UN (OBJECT_TYPE_NAME)
OBJECT_TYPE_PK (OBJECT_TYPE_ID)
OBJECT_TYPE_UN (OBJECT_TYPE_NAME)
```

Plug-in configuration in APEX-dialog-editor

- Simple configuration
  - JSON-Item
  - Source for JSON-schema
    - Static JSON-schema
    - SQL-Query, returning the JSON-schema
    - Generated JSON-schema generated based on JSON-data
- Other configurations
  - UI: column width, limit when "textarea" is used, item-template to use
  - Generate headers for sub-objects
  - Hide the page-item containing the JSON-data
  - Keep additional attribute in JSON
  - Remove empty/null properties from JSON-data
  - Read-only-attribute of region is used for all items in the region

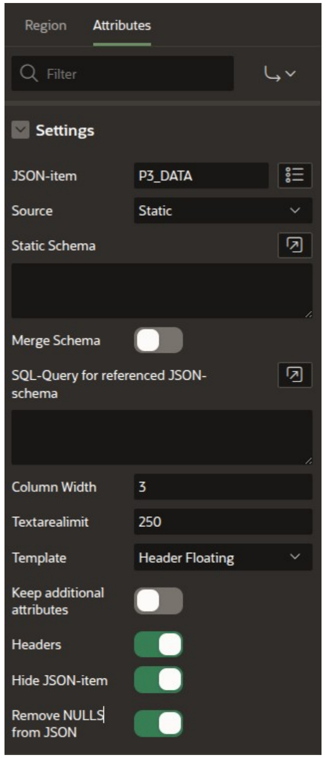


Identification

Title	JSON_REGION
Type	Json-Region

Read Only

Type	- Select -



Settings

JSON-item	P3_DATA
Source	Static
Static Schema	
Merge Schema	Off
SQL-Query for referenced JSON-schema	
Column Width	3
Textarealimit	250
Template	Header Floating
Keep additional attributes	Off
Headers	On
Hide JSON-item	On
Remove NULLS from JSON	On

Transformation JSON-Schema to APEX-UI

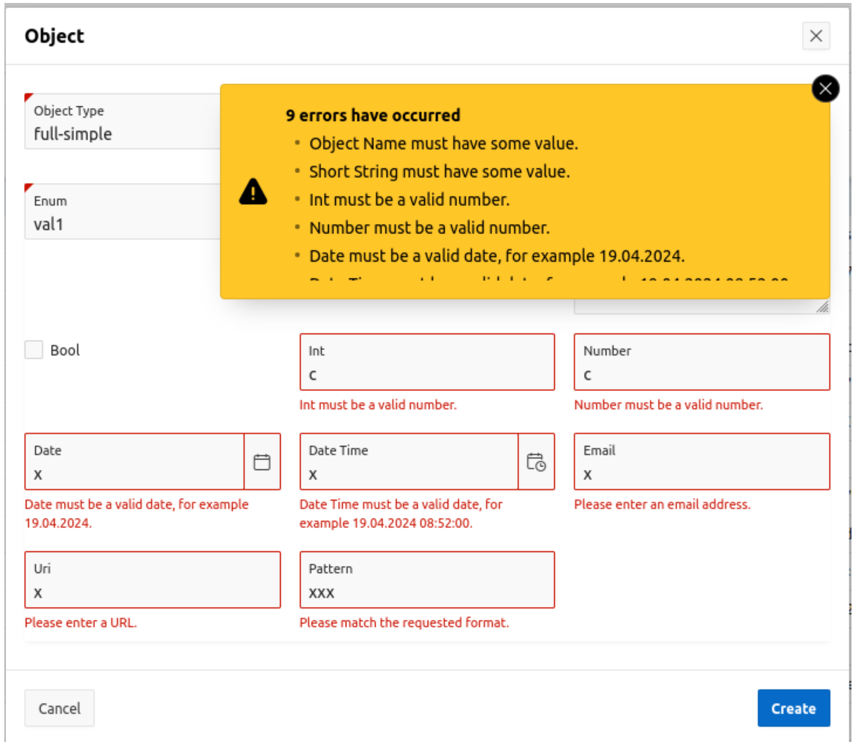
- The attributes are displayed in the same order ad defined in the JSON-schema.
- Depending on “type“/“format“/”pattern” in the JSON-schema a matching „APEX-Item-Type“ is used for input/output.

string	”Text Field”/“Text, ”Image” when “contentEncoding“: “base64“ and “contentMediaType” are defined
integer/number	”Number field”
boolean	”Checkbox”,
date/date-time/time	”Date/Date-Picker/Time-Picker
enum	”Selectlist”
email	”Text Field” with Subtype “Email”
uri	”Text Field” with Subtype “URL”
ipv4/v6/UUID	”Text Field” with pattern”
…	

- Display name of APEX-items is per default the attribute name
  (1. char capital, replace _- by “ “, same behaviour as for default-titles in page-designer)

item validation and error messages

- Supported validations
  - mandatory
  - integer, number, currency
  - date, date-time, time
  - regexp-pattern
  - email-address, URI
  - ipv4/v6 address, UUID
  - min, max
  - string length
  - Enum (list of values)
- Standard system error messages on validation errors



Object

9 errors have occurred
- Object Name must have some value.
- Short String must have some value.
- Int must be a valid number.
- Number must be a valid number.
- Date must be a valid date, for example 19.04.2024.

Object Type: full-simple		
Enum: val1		
Bool	Int: c Int must be a valid number.	Number: c Number must be a valid number.
Date: x Date must be a valid date, for example 19.04.2024.	Date Time: x Date Time must be a valid date, for example 19.04.2024 08:52:00.	Email: x Please enter an email address.
Uri: x Please enter a URL.	Pattern: xxx Please match the requested format.	

extended JSON-schema attributes

- Constant value
  - “const“: „constValue“
  - constant value for string/number/integer/boolean

- Binary data in strings (display only)
  - “contentEncoding“: “base64“
  - “contentMediaType“: “image/png“, “image/jpg““image/gif“
  - for images on PNG, JPG or GIF format

- Recursive
  - “type“: “object“
  - “type“: “array“, “items“: […]

JSON-schema references

- Local schema-references
  - “$ref“: “#/defs/...“
    The reference starts with “#/” and references in the current JSON-schema.
    To avoid redundancies. For example when multiple addresses are required in a JSON-schema.
- Static schema-reference from DB-server
  - “$ref“: “/defs/...“
    Callback to database for selecting a static “sub-schema”, which is used in multiple JSON-schemas. The reference starts with “/”.
- Dynamic schema-reference from DB-server
  - “$ref“: “/defs/...”
    Callback to database for generating a “sub-schema”. For example to dynamically generate a select-list or a hierarchy of select-lists from a hierarchical query.
    The reference starts with “/”.

Conditional JSON-schema...

- “dependentRequired“
  items become mandatory when another item is not empty

```
"dependentRequired":{
  "payment": ["card", “validity”, “securitycode”]
}
```

- “dependentSchema“
  A “sub-schema” is only available, when an item is not empty.

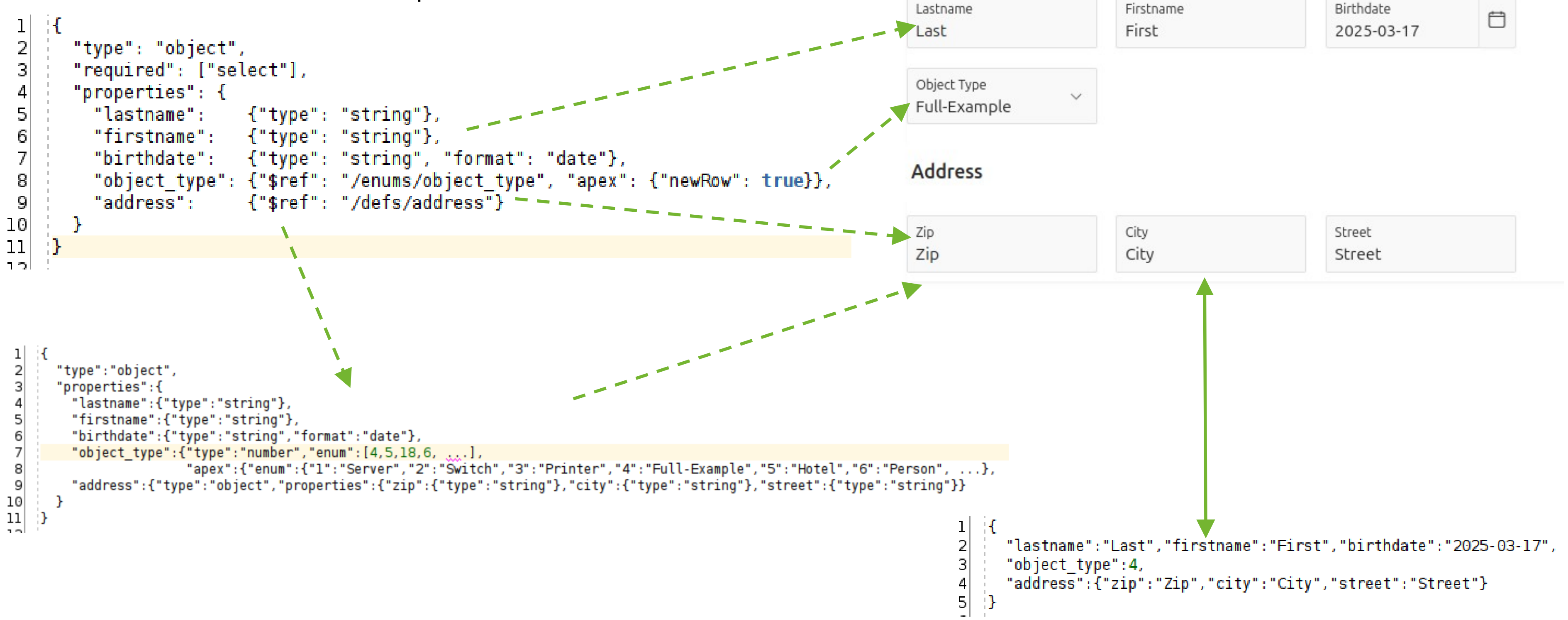
```
"dependentSchemas": {
  "payment": {
    "type": "object",
    "properties": { "creditcard": {"$ref": "#/$defs/creditcard"} }
  }
}
```

...Conditional JSON-schema

- “if“, “then“, “else“
  Depending on a condition additional items are available.
```
"if":   { "properties": { "deliverytohome": { "const": true} } },
"then": { "properties": { "home_address": {"$ref": "#/$defs/address"} } },
"else": { "properties": { "delivery_info": {"type": "string"} } }
```
- “allOf“, “oneOf”, “anyOf“, “not“ in an “if“ condition
- “allOf” for schema concatenation
  Useful for simplifying complex “if/then/else” conditions

Complex JSON-schema attributes...

Schema-references: “$ref”: “/xxx/xxx”



```
{
  "type": "object",
  "required": ["select"],
  "properties": {
    "lastname":    {"type": "string"},
    "firstname":   {"type": "string"},
    "birthdate":   {"type": "string", "format": "date"},
    "object_type": {"$ref": "/enums/object_type", "apex": {"newRow": true}},
    "address":     {"$ref": "/defs/address"}
  }
}
```

Lastname: Last	Firstname: First	Birthdate: 2025-03-17
Object Type: Full-Example		

Address

Zip: Zip	City: City	Street: Street

```
{
  "type":"object",
  "properties":{
    "lastname":{"type":"string"},
    "firstname":{"type":"string"},
    "birthdate":{"type":"string","format":"date"},
    "object_type":{"type":"number","enum":[4,5,18,6, ...],
                   "apex":{"enum":{"1":"Server","2":"Switch","3":"Printer","4":"Full-Example","5":"Hotel","6":"Person", ...},
    "address":{"type":"object","properties":{"zip":{"type":"string"},"city":{"type":"string"},"street":{"type":"string"}}
  }
}
```

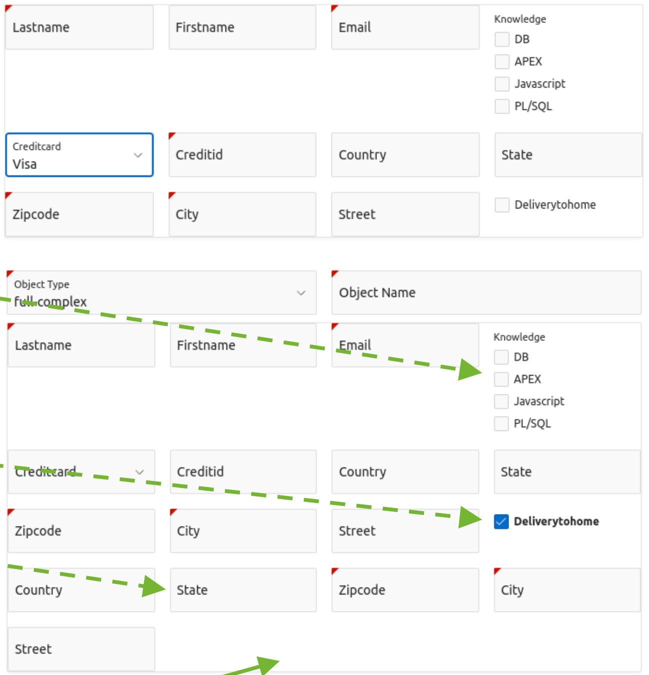
```
{
  "lastname":"Last","firstname":"First","birthdate":"2025-03-17",
  "object_type":4,
  "address":{"zip":"Zip","city":"City","street":"Street"}
}
```

...Complex JSON-schema attributes

- object/array, conditional schema:

```
{
  "type": "object",
  "required": ["lastname", "email"],
  "dependentRequired":{
    "creditcard": ["creditid"],
    "creditid":   ["creditcard"]
  },
  "properties": {
    "lastname":    {"type": "string", "maxLength": 30},
    "firstname":   {"type": "string", "maxLength": 30},
    "email":       {"type": "string", "format": "email"},
    "knowledge":   {"type":"array","items":{"type":"string", "enum":["DB","APEX","Javascript","PL/SQL"]}},
    "creditcard":  {"type": "string", "enum":["Visa", "Mastercard", "Amex", "Diners"]},
    "creditid":    {"$ref": "#/$defs/cardid"},
    "office_address":     {"$ref": "#/$defs/address"},
    "deliverytohome":     { "type": "boolean"}
  },
  "if": {
      "properties": {
        "deliverytohome": { "const": true}
      }
  },
  "then": {
     "properties": {
       "home_address": {"$ref": "#/$defs/address"}
     }
  },
  "$defs":{
      "name": {"type": "string", "maxLength": 30},
      "address": {
        "type": "object",
        "required": ["zipcode", "city"],
        "properties": {
           "country": {"type": "string"},
           "state":   {"type": "string"},
           "zipcode": {"type": "string"},
           "city":    {"type": "string"},
           "street":  {"type": "string"}
          }
       },
      "cardid": {"type": "string", "pattern": "[0-9]{4}( [0-9]{4}){3}"}
   }
}
```

```
{
  "lastname":"Simon",
  "firstname":"Uwe",
  "email":"usdbc@magenta.de",
  "knowledge":["DB","APEX","Javascript","PL/SQL"],
  "creditcard":"Visa",
  "creditid":"1234 5678 9012 3456",
  "office_address":{
    "country":"D","state":"NRW","zipcode":"50000","city":"City1","street":"Street1"
  },
  "deliverytohome":true,
  "home_address":{
    "country":"D","state":"NRW","zipcode":"50000","city":"City2","street":"Street2"}
  }
}
```



Customizing the APEX-UI ...

- The APEX-UI provides multiple item-type for a data-type
- All APEX-specific Configurations are stored below `“apex“: {…}`
  - “itemtype": "starrating" — Numeric as “Starrating” `“apex“: {“itemtype”: “starrating”}`
  - "itemtype": "switch"/”radio” — Boolean as “Switch”/”Radio”
  - “itemtype“: “password“ — Password item
  - “itemtype“: “pctgraph“ — Numeric as a bar in % (0-100, display only)
  - “itemtype“: “currency“ — Integer/Number as currency
  - >=APEX-23.2
    - "itemtype": "richtext" — The “Richtext-editor” for long strings
    - "itemtype": "combobox" — A multiselect combobox with „Chips“
    - „itemtype“: „qrcode“ — Display of QR-Codes
  - >=APEX 24.1
    - “itemtype”: “Selectone” — For a single-select
    - “itemtype”: “Selectmany” — A multiselect combobox

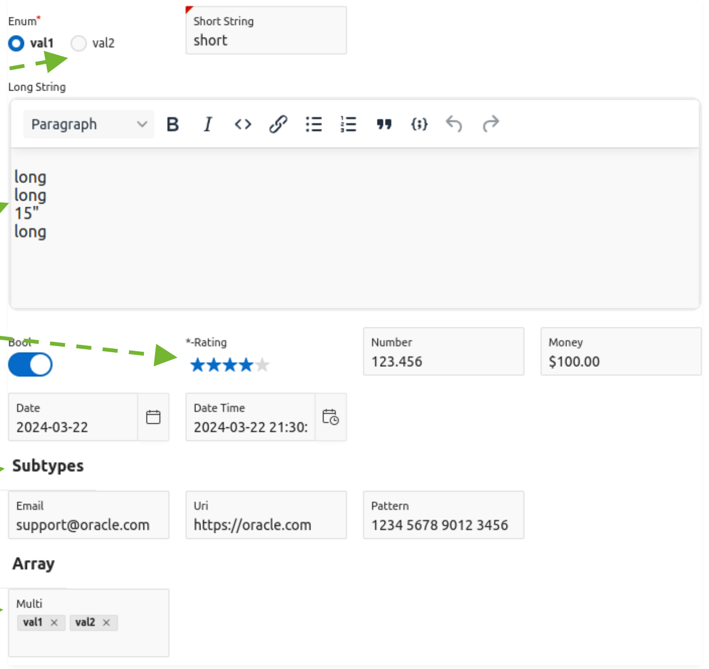
...Customizing the APEX-UI ...

- Other attributes below „apex“

„label“: “Text“	Text as item label
"newRow": true	Start a new row before the item,
"textBefore": "Text"	Static “Text” in front of item
"lines": 10	Number of rows for Textarea/Richtext-Editor
"colSpan": 6	Width of the item (1-12)
“readonly“: true	Item is display only
“direction“: “horizontal“	”Radio”/”Checkbox” group horizontally

… Customizing der APEX-UI

```
{
  "type": "object",
  "required": ["enum", "short_string"],
  "properties": {
    "enum":         { "type": "string", "enum": [ "val1", "val2" ],
                      "apex": {"itemtype": "radio", "direction": "horizontal"}},
    "short_string": { "type": "string" },
    "long_string":  { "type": "string", "maxLength": 400,
                      "apex": {"itemtype": "richtext", "lines": 4, "colSpan": 12}},
    "bool":         { "type": "boolean",
                      "apex": {"itemtype": "switch"}},
    "int":          { "type": "integer", "maximum": 5,
                      "apex": {"itemtype": "starrating", "label": "*-Rating"}},
    "number":       { "type": "number" },
    "money":        { "type": "number",
                      "apex": {"format": "currency"}},
    "date":         { "type": "string", "format": "date"},
    "date_time":    { "type": "string", "format": "date-time"},
    "email":        { "type": "string", "format": "email",
                      "apex": {"textBefore": "Subtypes"}},
    "uri":          { "type": "string", "format": "uri"},
    "pattern":      { "type": "string", "pattern": "[0-9]{4}( [0-9]{4}){3}"},
    "multi":  { "type": "array",
                "items": { "type": "string", "enum":  ["val1",  "val2"]},
                "apex":  {"itemtype": "combobox", "textBefore": "Array"}
              }
    }
}
```



Enum* [✓ val1] val2

Short String: short

Long String

Paragraph

long
long
15"
long

Bool

*-Rating: ★★★★☆

Number: 123.456

Money: $100.00

Date: 2024-03-22

Date Time: 2024-03-22 21:30:

Subtypes

Email: support@oracle.com

Uri: https://oracle.com

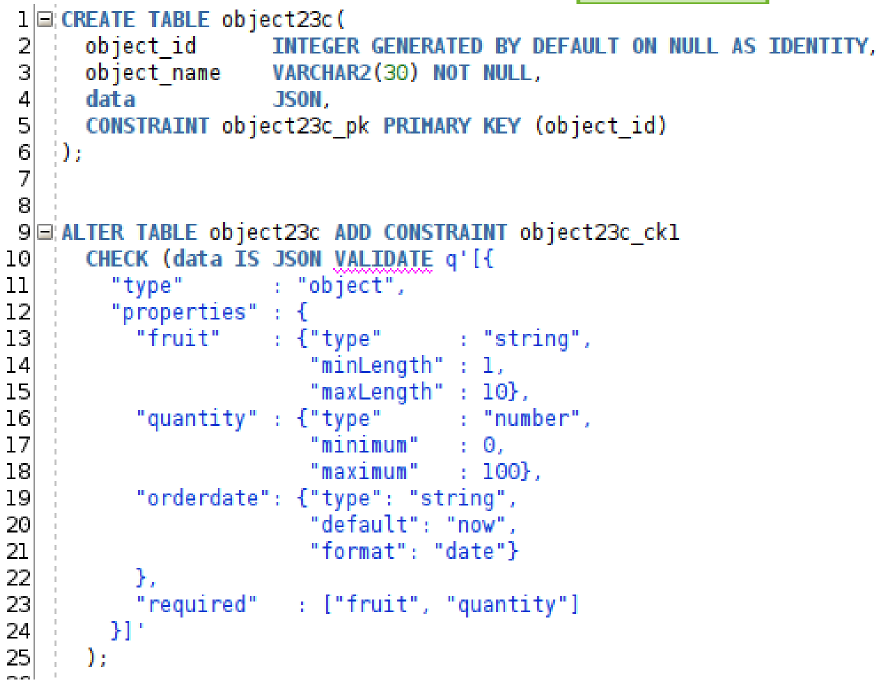
Pattern: 1234 5678 9012 3456

Array

Multi: val1 × val2 ×

JSON-Schema generated from JSON-validation

- Starting with Oracle23ai, it is possible to define an “IS JSON VALIDATE” Check-Constraint for a VARCHAR2/CLOB/JSON-columns or collection-tables.
- So, why not use the JSON-schema in this constraints for generating the APEX-UI
- **Caution**:
  - Oracle doesn’t support all “complex” JSON-schema configurations.
    For example “$ref“: “…“ ignored / returns error
  - In JSON-duality-views Oracle uses some extensions like “extendedType“, which is supported by the plug-in



```
CREATE TABLE object23c(
  object_id      INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY,
  object_name    VARCHAR2(30) NOT NULL,
  data           JSON,
  CONSTRAINT object23c_pk PRIMARY KEY (object_id)
);


ALTER TABLE object23c ADD CONSTRAINT object23c_ck1
  CHECK (data IS JSON VALIDATE q'[{
    "type"       : "object",
    "properties" : {
      "fruit"    : {"type"      : "string",
                    "minLength" : 1,
                    "maxLength" : 10},
      "quantity" : {"type"      : "number",
                    "minimum"   : 0,
                    "maximum"   : 100},
      "orderdate": {"type": "string",
                    "default": "now",
                    "format": "date"}
      },
      "required"   : ["fruit", "quantity"]
    }]'
  );
```

Experiences during development...

Different behaviours of UI-items in JavaScript

- apex.item(…).setValue(…)
  - Destroys Date/Date-Time-Picker in APEX<=22.2 when called after the UI-item is rendered
  - For item-types “QRCode” and “RichTextEditor” (introduced in APEX-23.2) must wait until the rendering of the UI-Items has finished
  - QR-Code is generated in PL/SQL, uses an AJAX-request via AJAX-callback
  - RichTextEditor is initialized asynchronously, must be finished before using apex.item().setValue
- <input ...>
  - All common browsers support additional attributes like minLength=“..“, „pattern=“..“, type=“time“, …
    Error-messages for this <input>-tags is generated by the browsers, unfortunately in the language of the browser-UI but not in the language of the current page

...Experiences during development...

- The first plug-in-version, supporting the “simple” property types „string“, „integer“, „number“, „boolean“, was implemented quite fast, the next versions supporting different APEX/DB-versions and additions APEX-item-types (‘Richtext”,”Switch”, “Combobox”, …) sub-object and arrays took much more time.
- **APEX-JavaScript-code**
  - JavaScript-code .../images/apex/libraries is available in minimized and “readable” format. The inline Documentation inside source code not always matching 100% / is partially missing
  - Depending on item/widget different implementation pattern with different behaviour
  - Oracle changes the used JavaScript-library for advanced item-types like “Richttext”, Date-Picker”, … which changes in behaviours/APIs

...Experiences during development

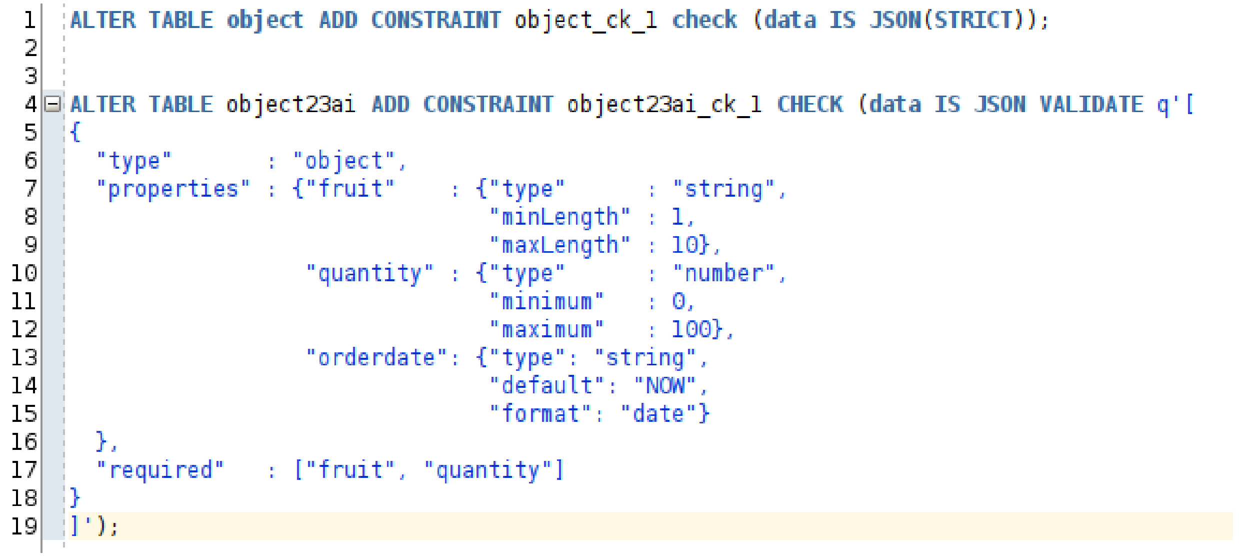
- **PL/SQL**
  - There is no PL/SQL-constant for the current APEX-Release like in JavaScript
    `apex.env.APEX_VERSION`
    The support of features (QR-code, collection-tables, …) in new APEX/DB-releases requires conditional compile in PL/SQL
    Workarround (here for APEX für 23.2):

```
 $if wwv_flow_api.c_current>=20231031 $then
  …
 $end
```

Miscellaneous...

- When using JSON in CLOB use check-constraint
  IS JSON(STRICT)

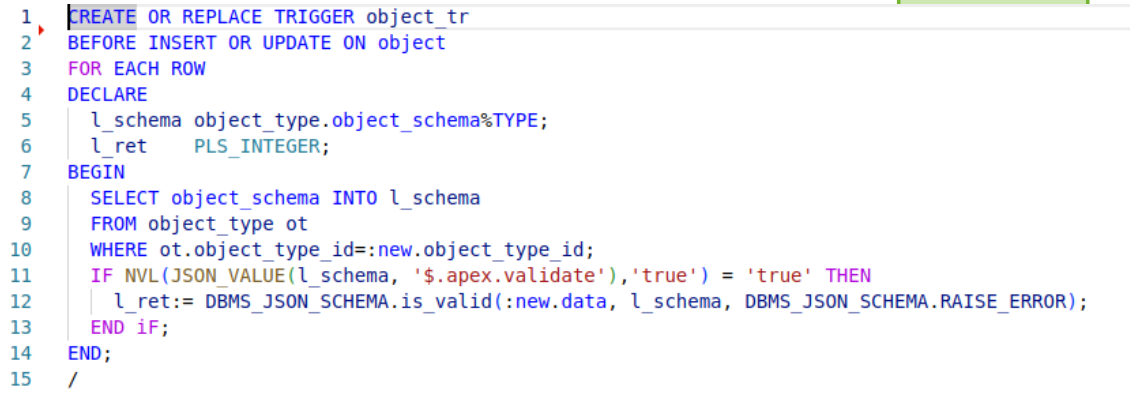
- When using a static JSON-schema for a table. Starting with Oracle23ai use check-constraint
  IS JSON VALIDATE
  ‘{….}’



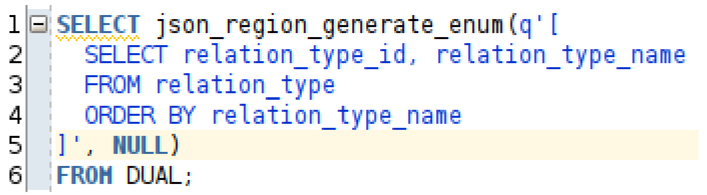
```
ALTER TABLE object ADD CONSTRAINT object_ck_1 check (data IS JSON(STRICT));


ALTER TABLE object23ai ADD CONSTRAINT object23ai_ck_1 CHECK (data IS JSON VALIDATE q'[
{
  "type"        : "object",
  "properties" : {"fruit"    : {"type"      : "string",
                                "minLength" : 1,
                                "maxLength" : 10},
                 "quantity" : {"type"      : "number",
                                "minimum"   : 0,
                                "maximum"   : 100},
                 "orderdate": {"type": "string",
                                "default": "NOW",
                                "format": "date"}
  },
  "required"   : ["fruit", "quantity"]
}
]');
```

...Miscellaneous

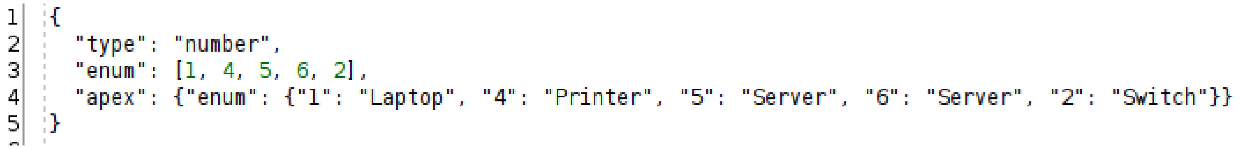
- When a JSON-column can contain data with different JSON-schema, the data could be verified with a row-trigger
- Often the JSON-schema attribute “enum” must be in sync with a lookup-table.
  2 solutions
  - A statement-Trigger on the lookup-table
  - Use a “dynamic” “$ref”: “/enum/….” to generate the values for an “enum” or a list of dependend “enum”



```
CREATE OR REPLACE TRIGGER object_tr
BEFORE INSERT OR UPDATE ON object
FOR EACH ROW
DECLARE
  l_schema object_type.object_schema%TYPE;
  l_ret     PLS_INTEGER;
BEGIN
  SELECT object_schema INTO l_schema
  FROM object_type ot
  WHERE ot.object_type_id=:new.object_type_id;
  IF NVL(JSON_VALUE(l_schema, '$.apex.validate'),'true') = 'true' THEN
    l_ret:= DBMS_JSON_SCHEMA.is_valid(:new.data, l_schema, DBMS_JSON_SCHEMA.RAISE_ERROR);
  END iF;
END;
/
```



```
SELECT json_region_generate_enum(q'[
  SELECT relation_type_id, relation_type_name
  FROM relation_type
  ORDER BY relation_type_name
]', NULL)
FROM DUAL;
```



```
{
  "type": "number",
  "enum": [1, 4, 5, 6, 2],
  "apex": {"enum": {"1": "Laptop", "4": "Printer", "5": "Server", "6": "Server", "2": "Switch"}}
}
```

Known usages of the JSON-Region-Plugin

- USA:
  One of the hugest county offices
  Workflow of complex forms translated into JSON

- India:
  PoC for a generic workflow

- Germany:
  Flows4Apex: An open-source BPEL-engine for APEX

Known issues with plugin

JSON-collection-table:

```
INSERT INTO table23ai VALUES('{"fruit":"Banana","quantity":10,"orderdate":"2025-
02-20"}');
UPDATE table23ai SET data='{"fruit":"Banana","quantity":10,"orderdate":"2025-02-
20"}'
where rowid='AAArpxAAQAAAAL1AAA';
```

Oracle DB < 23.7: JSON-collection-table:

```
    INSERT, UPDATE
ORA-00932: inconsistent datatypes: expected VARCHAR, BLOB, CLOB, FILE, BINARY,
JSON – got -
ORA-54059: cannot update an immutable column ("UWE"."TABLE23AI"."RESID")
```

Oracle DB >= 23.7: JSON-collection-table with JSON VALIDATE constraint:

```
    INSERT works
    UPDATE
ORA-00932: inconsistent datatypes: expected VARCHAR, BLOB, CLOB, FILE, BINARY,
JSON – got -
```

Known issues APEX 24.2

- JSON-Source:
  - Defining a source with a complex JSON-schema returns – not very helpful - error
    `Could not compute a Data Profile for the new JSON Source, because of the following error: ORA-06503: PL/SQL: function returned without value`.
- Duality-View:
  - While defining a source on an Oracle >=23.7 DB, I get the – the not very helpful – error
    `ORA-06503: PL/SQL: function returned without value`
    In Oracle 23.7 the output of dbms_json_schema.describe has changed.
- Reason:
  JSON-schema contains some unknown keywords for the JSON-schema-parser of APEX 24.2, should be fixed with an upcoming patch.

Others

- Ideas for further improvements
  - Support of JSON-schema from OpenAPI https://swagger.io/specification/
  - Support of JSON-schema from JSON-Form https://jsonforms.io/
  -
- Flows4Apex https://flowsforapex.org/ uses the JSON-Region-Plugin
  - Here is a stand-alone APEX-application “JSON-Simple-Form-Builder” for defining and testing of forms. It also generates the JSON-schema used by the “JSON-Region-Plugin”
  - There is a “Simple Process Starter” application, which uses forms generated by the JSON-Region-Plugin

Nearly done

Thank You for your attention

Time for questions ??

Bedankt voor uw aandacht

Tijd voor vragen ??

APEX:	https://apex.world (JSON-Region)
Github:	https://github.com/simonuwe/oracle-apex-json-region
Email:	usdbc@magenta.de
LinkedIn:	https://www.linkedin.com/in/uwe-simon-cologne/

NL.OUG
NL ORACLE USER GROUP

Please fill in your evaluations