JSON-Region-Plugin

Dynamische APEX-UI für JSON-Daten

mit dem JSON-Region-Plugin

Uwe Simon Database Consulting
2025-05-15
APEX-Connect-2025

Über mich

- Name: Uwe Simon
- Oracle-DB: Erfahrung mit Oracle-DB seit 1992 ab Oracle 5
- Erstellen von Datenbank-Modellen
- Performance-Tuning großer (>100TB) Datenbanken
- Fehleranalyse in Mission-Critical Datenbanken im Rahmen von Major-Incident-Verfahren
- DB-Migrationen
- Proof of Concepts
- APEX: 1998 erste Applikation mit OAS/OWS, HTML-DB, APEX ...24.2
- Entwicklung: SQL, PL/SQL, C++, JavaScript, Java, HTML/CSS, ...
- Andere DBs: DB2, MySQL, PostgreSQL

- Wohnort Köln (Schäl Sick)
  Arbeitseinsätze in Niederlande, Tschechien, Indien
- Seit Sommer 2023 Altersteilzeit und Freelancer

Agenda

- Idee hinter dem Plug-in
- Beispiele für Anwendungsfällen für das Plug-in
- JSON-Schema in Kürze
- JSON-Schema und APEX-UI
- JSON-Region-Plugin vs. JSON-Sources in APEX 24.2
- Demo
- Konfiguration des Plugins
- Komplexe JSON-schema
- Sonstiges

Idee hinter dem Plug-in

- Das Plugin generiert die APEX-UI-Items zur Laufzeit basierend auf einem JSON-Schema
- Es sollen alle wichtigen APEX-Itemtypen unterstützt werden
- Für jeden Datentypen im JSON-Schema gibt es eine „Default”-Darstellung
- Es kann für jeden Datensatz ein anderes JSON-Schema genutzt werden
- Änderungen des JSON-Schema ändern direkt die APEX-UI ohne Anpassungen im APEX-Code
- UI ist transparent für den Anwender, “look like APEX-UI”, also auch identische Darstellung bei Eingabefehlern etc.
- Konfigurierbarkeit der APEX-UI über eine JSON-Schema-Erweiterung mit dem neuen Key
  `“apex”: {...}`
- APEX <24.2 bietet keine Out-Of-The-Box-Lösung für die einfache Ein-/Ausgabe von JSON-Daten. Mit APEX-24.2 gibt es statische JSON-Schema-Untersützung

Werdegang des Plug-in

- Die Entwicklung des JSON-Region-Plugin begann in Oktober 2023

- Erste Vorstellung auf der APEX-Connect2024 in Düsseldorf,
- Letzes Jahr fand Michael Hichwa mein Plugin super und meinte “Oracle definitely needs such a solution for JSON in APEX”
- Mit APEX-24.2 führte Oracle die neuen Datasources JSON/Duality-Source ein. Dies implementiert den „statischen Anteil“ meines Plugins

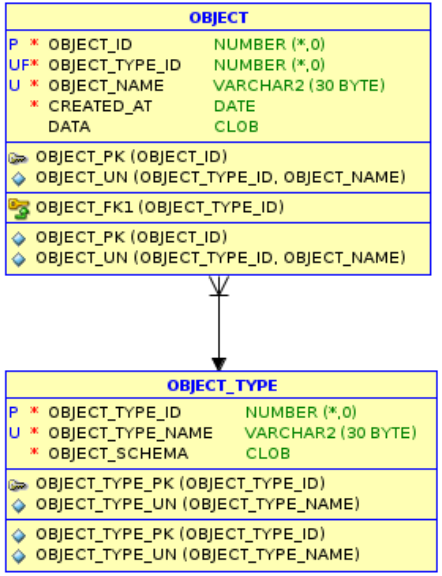
- Das Plug-in wird seit APEX-21.2 kontinuierlich um neue APEX-Features erweitert (z.B. APEX-24.1 „SelectMany”, „QRCode“, ...)

Agenda

- Idee hinter dem Plug-in
- Beispiele für Anwendungsfällen für das Plug-in
- JSON-Schema in Kürze
- JSON-Schema und APEX-UI
- JSON-Region-Plugin vs. JSON-Sources in APEX 24.2
- Demo
- Konfiguration des Plugins
- Komplexe JSON-schema
- Sonstiges

Beispiele für Anwendungsfällen für das Plug-in

- Konfigurierbare Workflows mit Daten in einer JSON-Spalte
- Konfigurierbare Asset-Management-Systeme mit Attributen in einer JSON-Spalte, die vom Asset-Typ abhängen
- Formular-Tools mit einer Formularstruktur als JSON-Schema und den Formulardaten in einer JSON-Spalte
- Umfrage-Tools mit Fragen und der Liste der möglichen Antworten als JSON-Schema gespeichert und den Antwortdaten in einer JSON-Spalte
- Anpassung einer APEX-Anwendung durch den Kunden und einfache Versionsupdates:
  Anpassungen sind an vorgesehenen Stellen ohne Änderungen im Page-Designer möglich.



Demo

2 einfache Beispiele zur Motivation für das Plug-In

- Asset-Management
- Workflow zur Bestellung von Hard-/Software etc. mit APEX-Workflow

Agenda

- Idee hinter dem Plug-in
- Beispiele für Anwendungsfällen für das Plug-in
- JSON-Schema in Kürze
- JSON-Schema und APEX-UI
- JSON-Region-Plugin vs. JSON-Sources in APEX 24.2
- Demo
- Konfiguration des Plugins
- Komplexe JSON-schema
- Sonstiges

JSON-Schema in Kürze

- Die Dokumentation von JSON-Schema befindet sich unter https://json-schema.org/
- JSON-Schema ist eine Beschreibung der Struktur von JSON-Daten
  - Ein JSON-Schema ist ein Objekt oder ein Array (“type“: “object“, “type”: “array“)
  - Jedes Attribute definiert den Datentyp (“type“) und ggf. das Format (“format“),
  - Es gibt Mussfelder (“required“),
  - Für Felder gibt es Wertelisten (“enum“), und Muster (“pattern“), ...
- Oracle23ai unterstützt JSON-Schema-Validierungen von CLOB/JSON-Spalten und Collection-Tables

```
{
  "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}"}
  }
}
```

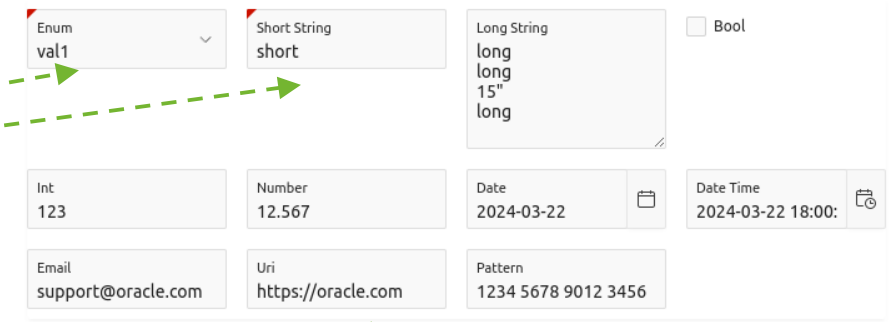
Warum also nicht ein JSON-Schema zur Generierung einer APEX-UI nutzen.

Agenda

- Idee hinter dem Plug-in
- Beispiele für Anwendungsfällen für das Plug-in
- JSON-Schema in Kürze
- JSON-Schema und APEX-UI
- JSON-Region-Plugin vs. JSON-Sources in APEX 24.2
- Demo
- Konfiguration des Plugins
- Komplexe JSON-schema
- Sonstiges

JSON-Schema, JSON-Daten und APEX-UI

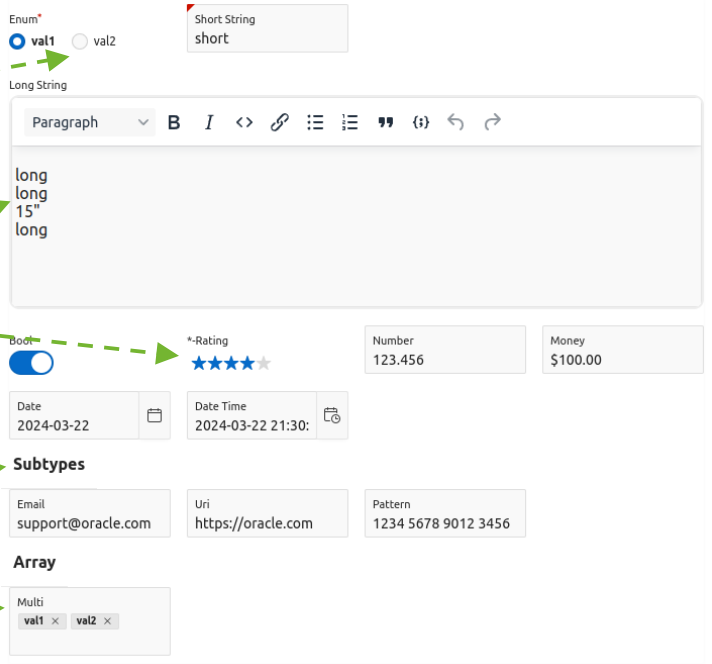
```
{
  "type": "object",
  "required": ["enum", "short_string"],
  "properties": {
    "enum":         { "type": "string", "enum": [ "val1", "val2" ]},
    "short_string": { "type": "string" },
    "long_string":  { "type": "string", "maxLength": 400},
    "bool":         { "type": "boolean"},
    "int":          { "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\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"
}
```

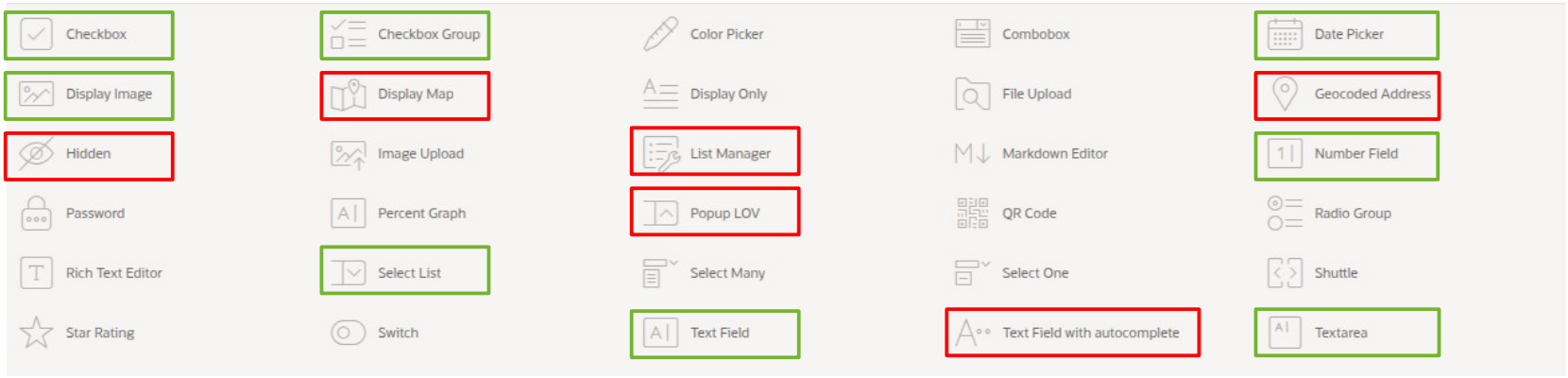
Anpassung der APEX-UI

```
{
  "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"}
              }
    }
}
```



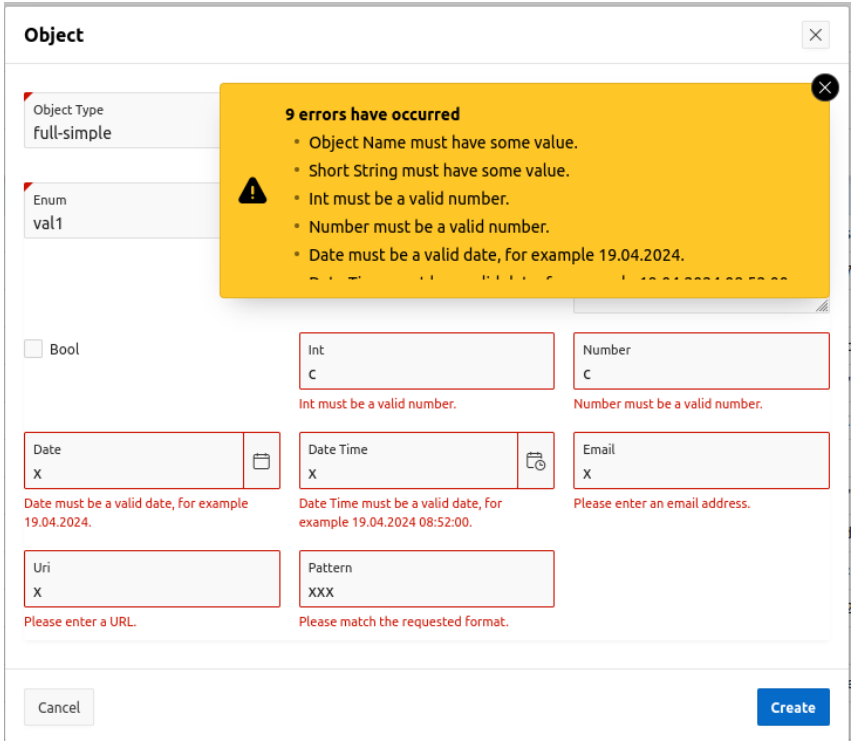
Unterstützte APEX-Page-Itemtypes

- Grün: Default basierend auf JSON-Schema-Properties “type”, “format”, “enum”
- Rot: (Noch) nicht unterstützt
- Sonstige: Konfiguration über JSON-Schema **“apex”: {“itemtype”: “???”}**
- Plugin-Only: **“itemtype“: “time“**, nutzt den Browser-Time-Selektor, APEX unterstützt dies nicht.



Feld-Validierungen und Fehlermeldungen

- Unterstützte Feld-Validierungen
  - required
  - integer, number, currency
  - date, date-time, time
  - regexp-Muster
  - email-address, URI
  - ipv4/v6 address, UUID
  - min, max
  - String Länge
  - Enum (list of values)
- Standard APEX-Fehlermeldungen bei Validierungsfehlern



Agenda

- Idee hinter dem Plug-in
- Beispiele für Anwendungsfällen für das Plug-in
- JSON-Schema in Kürze
- JSON-Schema und APEX-UI
- JSON-Region-Plugin vs. JSON-Sources in APEX 24.2
- Demo
- Konfiguration des Plugins
- Komplexe JSON-schema
- Sonstiges

Plug-in vs. JSON-Sources in APEX-24.2

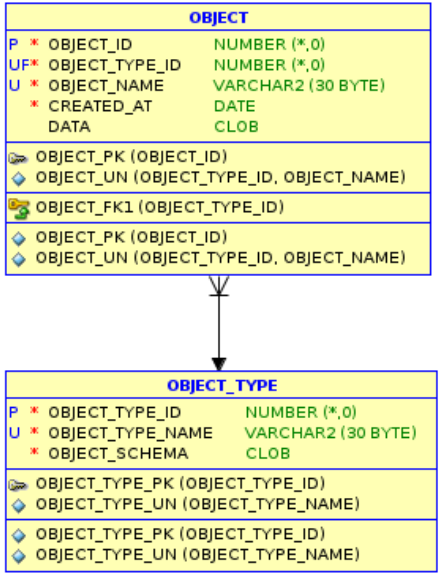
Feature	Plug-in	APEX 24.2
VARCHAR2/CLOB/JSON column	√	√
collection-table	√	√
collection-view	√	–
JSON-duality-view	√	√
Fixed JSON-schema	√	√ JSON-source
Variable JSON-schema	√	–
Evaluation of JSON-schema	At runtime	In page-designer
JSON-schema from DB (23ai)	√	√– (Duality-source only)
Dynamic UI	√	–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

Agenda

- Idee hinter dem Plug-in
- Beispiele für Anwendungsfällen für das Plug-in
- JSON-Schema in Kürze
- JSON-Schema und APEX-UI
- JSON-Region-Plugin vs. JSON-Sources in APEX 24.2
- Demo
- Konfiguration des Plugins
- Komplexe JSON-schema
- Sonstiges

Demo

- Eine Demo sage mehr als 1000 Folien
- Demo nutzt einfaches Datenmodel
  - Tabelle “OBJECT” mit „generischen“ Objekten in der JSON-Spalte “DATA” (CLOB/JSON)
  - Tabelle “OBJECT_TYPE” enthält die gültigen Object-Typen und deren jeweiliges JSON-Schema in Spalte “OBJECT_SCHEMA”
  - JSON-collection-table/view “TABLE23AI”, “VIEW23AI”
  - JSON-duality-view “JSON23AI”
  - Wenn noch Zeit:
    Nutzung APEX-24.2 JSON-Source, Duality-Source

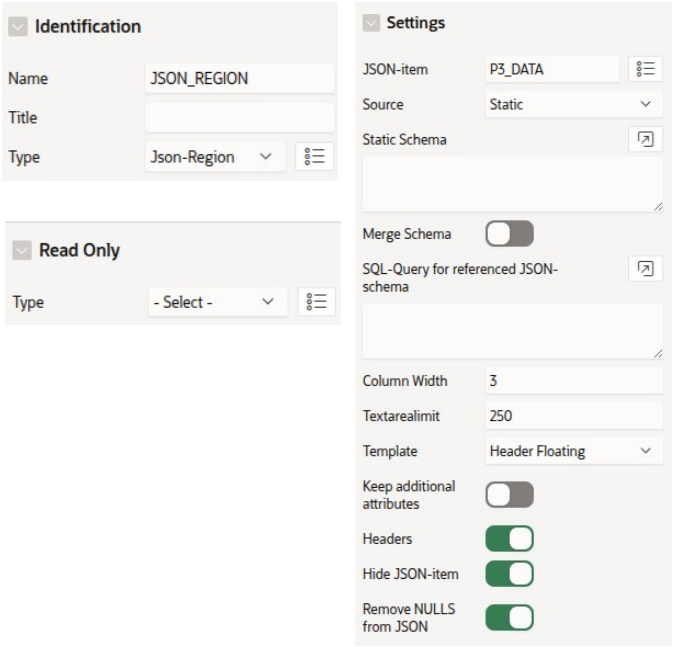


Agenda

- Idee hinter dem Plug-in
- Beispiele für Anwendungsfällen für das Plug-in
- JSON-Schema in Kürze
- JSON-Schema und APEX-UI
- JSON-Region-Plugin vs. JSON-Sources in APEX 24.2
- Demo
- Konfiguration des Plugins
- Komplexe JSON-schema
- Sonstiges

Plug-in configuration in APEX-dialog-editor

- Einfache Konfiguration
  - JSON-Item
  - Source für das JSON-schema
    - Statisches JSON-schema
    - SQL-Query, die das JSON-Schema liefert
    - Generiere JSON-Schema basierend auf den JSON-Data
- Weitere Konfigurationen
  - UI: Breite der Spalten, Grenzwet, abdem Textarea genutzt wird, Template für Eingabefelder
  - Generiere Titel für Sub-Objekte
  - Page-Item mit den JSON-Daten unsichtbar schalten
  - Zusätzliche Attribute in den JSON-Daten erhalten
  - Lösche Properties die Leer/Null sind aus den JSON-Daten
  - Das Read-only-Attribut wird für die Region genutzt



Agenda

- Idee hinter dem Plug-in
- Beispiele für Anwendungsfällen für das Plug-in
- JSON-Schema in Kürze
- JSON-Schema und APEX-UI
- JSON-Region-Plugin vs. JSON-Sources in APEX 24.2
- Demo
- Konfiguration des Plugins
- Komplexe JSON-schema
- Sonstiges

JSON-Schema Referenzen

- Lokale Schema-Referenzen
  - “$ref“: “#/defs/...“
    The reference starts with “#/” and references in the current JSON-schema.
    To avoid redundancies. For example when multiple addresses are required in a JSON-schema.
- Statische Schema-Referenzen vom DB-server
  - “$ref“: “/defs/...“
    Callback to database for selecting a static “sub-schema”, which is used in multiple JSON-schemas. The reference starts with “/”.
- Dynamische Schema-Referenzen vom 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 “/”.

Bedingte JSON-Schema...

- “dependentRequired“: Items werden „Mandatory“, wenn ein anderes Feld „Nicht leer“ ist (payment nicht leer, dann card, validity und securitycode)

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

- “dependentSchema“: Ein “Sub-Schema” steht nur zur Verfügung, wenn ein Item „Nicht leer“ ist (pyment nicht leer, dann auch creditcard)

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

...Bedingte JSON-Schema

- “if“, “then“, “else“
  Abhängig von einer Bedingung werden Items sichtbar

```
"if":   { "properties": { "deliverytohome": { "const": true} } },
"then": { "properties": { "home_address": {"$ref": "#/$defs/address"} } },
"else": { "properties": { "delivery_info": {"type": "string"} } }
```

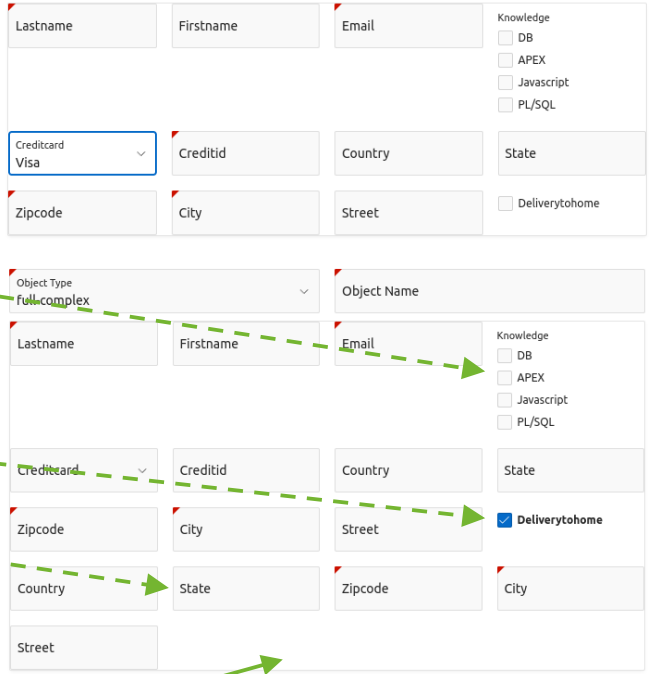
- “allOf“, “oneOf”, “anyOf“, “not“ in den “if“-Bedingungen
- “allOf” für die Schema-Concatenation
  Vereinfacht komplexe “if/then/else”-Bedingungen

Komplexes JSON-Schema

- 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"}
  }
}
```



Agenda

- Idee hinter dem Plug-in
- Beispiele für Anwendungsfällen für das Plug-in
- JSON-Schema in Kürze
- JSON-Schema und APEX-UI
- JSON-Region-Plugin vs. JSON-Sources in APEX 24.2
- Demo
- Konfiguration des Plugins
- Komplexe JSON-schema
- Sonstiges

Übernahme des JSON-Schema aus JSON-Validierung

- Mit Oracle23ai, kann man nun auch eine **“IS JSON VALIDATE”** Check-Constraint für VARCHAR2/CLOB/JSON-Spalten angeben.
- Die JSON-Duality-Views haben auch ein JSON-Schema als Beschreibung
- Warum also nicht diese Informationen direkt für das JSON-Schema des Plug-Ins nutzen.
- **Achtung**:
  - Oracle unterstützt nur einen Subset von JSON-Schema. Komplexe Konstrukte wie z.B. **“$ref“: “…“** werden ignoriert oder liefern bei der Validierung Fehler
  - Bei den JSON-Duality-Views nutzt Oracle einige Erweiterungen wie **“extendedType“**, diese werden durch das Plugin auch unterstützt

```
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"]
    }]'
  );
```

Bekannte Fallen mit APEX 24.2...

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 mit JSON VALIDATE constraint:

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

...Bekannte Fallen mit APEX 24.2

- JSON-Source:
  - Beim Definieren einer JSON-Source mit einem „komplexen“ JSON-schema bekommt man - den nicht wirklich hilfreichen - Fehler
    `Could not compute a Data Profile for the new JSON Source, because of the following error: ORA-06503: PL/SQL: function returned without value`.
    Das JSON-Schema enthält Key/Werte, die der JSON-Schema-Parser von APEX24.2 nicht versteht
- Duality-View:
  - Beim Definieren einer Duality-Source mit OracleDB >=23.7, gibt es - den nicht wirklich hilfreichen - Fehler
    `ORA-06503: PL/SQL: function returned without value`
    In OracleDB 23.7 hat sich die Ausgabe von dbms_json_schema.describe geändert.
- Sollte mit einem der nächsten Patches gefixed sein.

Sonstiges

Nutzung des Plugin

- Feedback von 2 Kundenanwendungen (USA, Indien)
- Das Produkt „Flows4Apex“ https://flowsforapex.org/ (Dev-Talk am 13.05.)
  - Hier gibt es auch eine „Standalone-APEX-Applikation“ „JSON-Simple-Form-Builder” mit der ohne JSON-Schema-Kenntnisse Formulare erstellt und getestet werden können. Die so generierten JSON-Schema können dann per Copy&Paste übernommen werden
  - Es gibt auch eine “Simple Process Starter”-Applikation, die JSON-Schema zum Generieren von Formularen mittels dem JSON-Region-Plugin nutzt.

Weitere Ideen für Erweiterungen des Plug-Ins

- Unterstützung des JSON-Schema aus OpenAPI https://swagger.io/specification/
- Unterstützung des JSON-Schema aus JSON-Form https://jsonforms.io/

Fast geschafft

Vielen Dank für die Aufmerksamkeit

Zeit für Fragen ?

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/

Anhang

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
  '{….}'

```
1  ALTER TABLE object ADD CONSTRAINT object_ck_1 check (data IS JSON(STRICT));
2
3
4  ALTER TABLE object23ai ADD CONSTRAINT object23ai_ck_1 CHECK (data IS JSON VALIDATE q'[
5  {
6    "type"       : "object",
7    "properties" : {"fruit"    : {"type"      : "string",
8                                  "minLength" : 1,
9                                  "maxLength" : 10},
10                  "quantity" : {"type"      : "number",
11                                 "minimum"   : 0,
12                                 "maximum"   : 100},
13                  "orderdate": {"type": "string",
14                                 "default": "NOW",
15                                 "format": "date"}
16   },
17   "required"   : ["fruit", "quantity"]
18 }
19 ]');
```

...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 dependent “enum”

```
1  CREATE OR REPLACE TRIGGER object_tr
2  BEFORE INSERT OR UPDATE ON object
3  FOR EACH ROW
4  DECLARE
5    l_schema object_type.object_schema%TYPE;
6    l_ret    PLS_INTEGER;
7  BEGIN
8    SELECT object_schema INTO l_schema
9    FROM object_type ot
10   WHERE ot.object_type_id=:new.object_type_id;
11   IF NVL(JSON_VALUE(l_schema, '$.apex.validate'),'true') = 'true' THEN
12     l_ret:= DBMS_JSON_SCHEMA.is_valid(:new.data, l_schema, DBMS_JSON_SCHEMA.RAISE_ERROR);
13   END iF;
14 END;
15 /
```

```
1  SELECT json_region_generate_enum(q'[
2    SELECT relation_type_id, relation_type_name
3    FROM relation_type
4    ORDER BY relation_type_name
5  ]', NULL)
6  FROM DUAL;
```

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

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
```