

# JSON-Region-Plugin

Ein Plugin zur dynamischen Generierung  
von APEX-UI-Items für die Ein-/Ausgabe  
von Attributen von JSON-Feldern

Uwe Simon Database Consulting  
2024-04-24  
APEX-Connect-2024

# Über Mich

Name: Uwe Simon

Oracle-DB: seit 1992 mit Oracle 5

- Datenmodellierung
- Performanz-Tuning
- Migrationen
- Proof of Concepts
- Performanz-Tests

APEX: 1998 mit OAS/OWS, HTML-DB, APEX

Entwicklung: SQL, PL/SQL, C++, Javascript, Java, ...

Weitere DBs: MySQL, PostgreSQL

Seit 2023 Freiberuflich



# Agenda

Idee des Plugins

JSON-Schema und APEX-UI

Komplexe JSON-Schema und APEX-UI

Customizing der APEX-UI

Oracle 23c

Sonstiges



## Idee des Plugins

JSON-Schema und APEX-UI

Komplexe JSON-Schema und APEX-UI

Customizing der APEX-UI

Oracle 23c

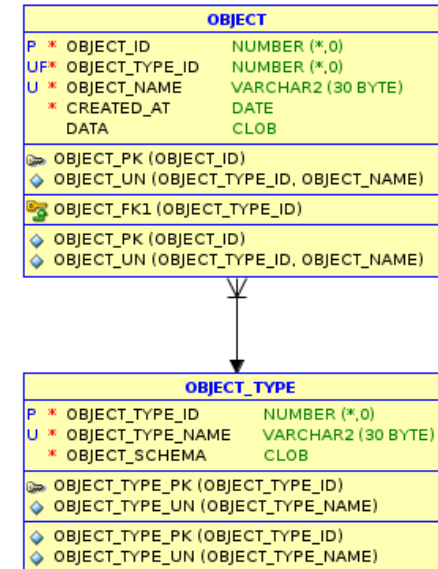
Sonstiges

# Idee

- JSON-Felder werden in immer mehr Datenbank-Schemas genutzt
- APEX liefert keine Out-Of-The-Box-Lösung für die Ein-/Ausgabe der Attribute von JSON-Feldern
- Idee:
  - Aus einem JSON-Schema dynamisch zur Laufzeit eine APEX-UI generieren,
  - Je Datensatz ggf. je nach „Datensatztyp“ unterschiedliche JSON-Schema
  - Keine Modifikationen am APEX-Code bei Änderungen des JSON-Schema
  - Anpassungsmöglichkeiten des UI-Layouts zur Unterstützung von weiteren APEX-Item-Typen durch erweiterte Attribute im JSON-Schema

# Einsatzmöglichkeiten des Plugins

- Konfigurierbare Workflows:  
Die Daten für den Workflow sind in JSON-Feldern abgelegt
- Konfigurierbare Asset-Management-Systeme:  
Attribute die vom Assettyp abhängen liegen in JSON-Feldern
- Formular-Tools:  
Formularstruktur liegt im JSON-Schema und Formulardaten liegen in JSON-Feldern
- Umfrage-Tools:  
Fragen liegen in JSON-Schema und Daten dann in JSON-Feldern
- Durch den Kunden anpassbare Anwendungen:  
Customizing erfolgt über JSON-Felder.
- ...





Idee des Plugins

JSON-Schema und APEX-UI

Komplexe JSON-Schema und APEX-UI

Customizing der APEX-UI

Oracle 23c

Sonstiges

# JSON-Schema in Kürze

- Dokumentation zu JSON-Schema befindet sich unter <https://json-schema.org/>
- JSON-Schema wird zur Beschreibung und Validierung von JSON-Daten genutzt
  - Für jedes Attribut wird dabei Mussfeld (“required”), Datentyp (“type”) und Format (“format”), Aufzählung (“enum”) bzw. Muster (“pattern”) festgelegt.
  - Ein Attribut kann wiederum ein JSON-Schema bzw. ein Array sein (“type”: “object“ bzw. “array“) sein.
- Oracle23c unterstützt JSON-Schema-Validierungen auch als CHECK-Constraint einer CLOB/JSON-Spalte.

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

Was liegt also näher, als auch JSON-Schema für die APEX-UI zu nutzen.



# 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  
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

- Ich hab da mal was vorbereitet
- Die kleine Demo nutzt ein sehr einfaches Datenmodell
  - Tabelle für „generische“ Objekte mit einer JSON-Spalte (CLOB/JSON)
  - Tabelle mit Objekttypen und Spalte mit dem JSON-Schema

# Konfiguration des Plugins im Dialog-Editor

- Simple Konfiguration
  - JSON-Item
  - Source Static/SQL-Query
  - Statisches JSON-Schema bzw. SQL-Query, die eine Zeile mit dem JSON-Schema zurückliefert
- Weitere Konfiguration
  - Zusätzliche Attribute im JSON behalten
  - Für Subobjekte Überschriften generieren
  - Das JSON-Feld automatisch nicht anzeigen
  - Leere Felder aus dem JSON entfernen
  - Das Readonly-Attribut für die Region wird übernommen

The screenshot shows the 'Region' tab of a configuration dialog. Under the 'Identification' section, the 'Title' field is set to 'JSON\_REGION' and the 'Type' dropdown is set to 'Json-Region'.

The screenshot shows the 'Read Only' section, which is checked. The 'Type' dropdown is set to '- Select -'.

The screenshot shows the 'Attributes' tab of a configuration dialog. Under the 'Settings' section, the 'JSON-item' is 'P3\_DATA', the 'Source' is 'SQL-Query', and the 'SQL-Query' field contains the query: `select object_schema from object_type where object_type_id=:P3_OBJECT_TYPE_ID`. Other settings include 'Column width' set to 3, 'Textarealimit' set to 250, and three toggle switches: 'Keep additional attributes' (off), 'Headers' (off), and 'Hide JSON-item' (on). The 'Remove NULLS from JSON' toggle is also on.

# Abbildung JSON-Schema nach APEX-UI

- Die Attribute werden in der gleichen Reihenfolge wie im JSON-Schema angezeigt.
- Je nach “type”/“format” wird per Default ein passender „APEX-Item-Typ“ für die Ein-/Ausgabe genutzt
  - string Text Field bzw. Textarea (je nach Länge)
  - integer/number Numerisches Feld
  - boolean Checkbox
  - date/date-time Date-Picker/ Date-Picker+Time
  - enum Pulldown
  - email Text Field mit Subtyp Email
  - uri Text Field mit Subtype URL
  - ...
- Anzeigename des APEX-Items ist per Default der Name des Attributes  
(1. Zeichen je Wort groß \_- ersetzen, ... wie Default-Title im Page-Designer)

# Feldvalidierungen und Fehlermeldungen

- Unterstützte Validierungen
  - Integer, Number
  - Date, Date-Time
  - Regex-Pattern
  - Email-Adresse
  - URI
  - Minimum, Maximum
  - Maximale Länge
- Standard-Fehlermeldungen bei Validierungsfehlern

**10 errors have occurred**

- Object Type must have some value.
- Object Name must have some value.
- Short String must have some value.
- Int must be a valid number.
- Number must be a valid number.

**Int**  
X  
Int must be a valid number.

**Number**  
X  
Number must be a valid number.

**Date**  
X  
Date must be a valid date, for example 2024-03-22.

**Date Time**  
X  
Date Time must be a valid date, for example 2024-03-22 11:54:00.

**Email**  
X  
Die E-Mail-Adresse muss ein @-Zeichen enthalten. In der Angabe "x" fehlt ein @-Zeichen.

**Uri**  
X  
Gib eine URL ein.

**Pattern**  
X  
Deine Eingabe muss mit dem geforderten Format übereinstimmen.



Idee des Plugins

JSON-Schema und APEX-UI

Komplexe JSON-Schema und APEX-UI

Customizing der APEX-UI

Oracle 23c

Sonstiges

# Komplexe JSON-Schema Attribute

- Konstante Werte
  - “const”: “constant Value”
- Rekursive
  - “type”: “object”
  - “type”: “array”, “items”: [...] (Unterstützung nur für “multiselect”/“checkbox-group”)
- Schema-Referenzen
  - “\$ref”: “#/...” (Unterstützung nur für Referenzen im gleichen JSON-Schema)  
Vermeiden von Redundanzen in der Schema-Definition, z.B. wenn es mehrere Anschriften gibt
- Conditional Schema
  - “dependentRequired”  
Feld wird Mussfeld, wenn ein anderes Feld nicht leer ist
  - “dependentSchema”  
Subschema muss eingegeben werden, wenn ein anderes Feld nicht leer ist (Kreditkartentyp, Nr.)
  - “if”, “then”, “else”  
Je nach Wert eines Feldes, weitere Eingabefelder (z.B. bei „abweichende Rechnungsanschrift“, 2. Anschrift)

# Komplexe JSON-Schema Attribute

```
1 {
2   "type": "object",
3   "required": ["lastname", "email"],
4   "dependentRequired": {
5     "creditcard": ["creditid"],
6     "creditid": ["creditcard"]
7   },
8   "properties": {
9     "lastname": {"type": "string", "maxLength": 30},
10    "firstname": {"type": "string", "maxLength": 30},
11    "email": {"type": "string", "format": "email"},
12    "knowledge": {"type": "array", "items": {"type": "string", "enum": ["DB", "APEX", "Javascript", "PL/SQL"]}},
13    "creditcard": {"type": "string", "enum": ["Visa", "Mastercard", "Amex", "Diners"]},
14    "creditid": {"$ref": "#/$defs/cardid"},
15    "office_address": {"$ref": "#/$defs/address"},
16    "deliverytohome": {"type": "boolean"}
17  },
18  "if": {
19    "properties": {
20      "deliverytohome": {"const": true}
21    }
22  },
23  "then": {
24    "properties": {
25      "home_address": {"$ref": "#/$defs/address"}
26    }
27  },
28  "$defs": {
29    "name": {"type": "string", "maxLength": 30},
30    "address": {
31      "type": "object",
32      "required": ["zipcode", "city"],
33      "properties": {
34        "country": {"type": "string"},
35        "state": {"type": "string"},
36        "zipcode": {"type": "string"},
37        "city": {"type": "string"},
38        "street": {"type": "string"}
39      }
40    },
41    "cardid": {"type": "string", "pattern": "[0-9]{4}([0-9]{4}){3}"},
42  }
43 }
```

```
1 {
2   "lastname": "Simon",
3   "firstname": "Uwe",
4   "email": "usdbc@agenta.de",
5   "knowledge": ["DB", "APEX", "Javascript", "PL/SQL"],
6   "creditcard": "Visa",
7   "creditid": "1234 5678 9012 3456",
8   "office_address": {
9     "country": "D", "state": "NRW", "zipcode": "50000", "city": "City1", "street": "Street1"
10  },
11  "deliverytohome": true,
12  "home_address": {
13    "country": "D", "state": "NRW", "zipcode": "50000", "city": "City2", "street": "Street2"
14  }
15 }
```

Lastname	Firstname	Email	Knowledge <input type="checkbox"/> DB <input type="checkbox"/> APEX <input type="checkbox"/> Javascript <input type="checkbox"/> PL/SQL
Creditcard Visa	Creditid	Country	State
Zipcode	City	Street	<input type="checkbox"/> Deliverytohome

Object type full-complex	Object Name		
Lastname	Firstname	Email	Knowledge <input type="checkbox"/> DB <input type="checkbox"/> APEX <input type="checkbox"/> Javascript <input type="checkbox"/> PL/SQL
Creditcard	Creditid	Country	State
Zipcode	City	Street	<input checked="" type="checkbox"/> Deliverytohome
Country	State	Zipcode	City
Street			





Idee des Plugins

JSON-Schema und APEX-UI

Komplexe JSON-Schema und APEX-UI

Customizing der APEX-UI

Oracle 23c

Sonstiges

# Customizing der APEX-UI ...

- APEX hat in der UI für einige Datentypen mehrere Darstellungsformen
- APEX-spezifische Konfiguration unter "apex": {...}
- Attribute "itemtype" zur Konfiguration des APEX-UI-Items
  - "itemtype": "starrating" Integer-Feld als Starrating
  - "itemtype": "switch" Boolean-Feld als Switch
  - "itemtype": "richtext" Für lange Strings Richtext-Editor
  - "itemtype": "combobox" Für Multiselect Combobox (mit „Chips“, ab APEX23.2)
- Weitere Attribute unter „apex“
  - „label“: "Text" Text als Label für das Feld
  - "newRow": true Neue Zeile vor dem Feld,
  - "textBefore": "Text" statische Text vor dem Feld
  - "lines": 10 Anzahl der Zeilen bei Textarea/Richtext-Editor
  - "colSpan": 6 Breite des Feldes (1-12)
  - "readonly": true Feld ist nur zur Anzeige
  - "direction": "horizontal" Radio/Checkbox horizontal

# ... Customizing der APEX-UI

```
1 {
2   "type": "object",
3   "required": ["enum", "short_string"],
4   "properties": {
5     "enum": {
6       "type": "string", "enum": ["val1", "val2"],
7       "apex": {"itype": "radio", "direction": "horizontal"}},
8     "short_string": { "type": "string" },
9     "long_string": { "type": "string", "maxLength": 400,
10      "apex": {"itype": "richtext", "lines": 4, "colSpan": 12}},
11     "bool": {
12       "type": "boolean",
13       "apex": {"itype": "switch"}},
14     "int": {
15       "type": "integer", "maximum": 5,
16       "apex": {"itype": "starrating", "label": "*-Rating"}},
17     "number": {
18       "type": "number" },
19     "money": {
20       "type": "number",
21       "apex": {"format": "currency"}},
22     "date": {
23       "type": "string", "format": "date"},
24     "date_time": {
25       "type": "string", "format": "date-time"},
26     "email": {
27       "type": "string", "format": "email",
28       "apex": {"textBefore": "Subtypes"}},
29     "uri": {
30       "type": "string", "format": "uri"},
31     "pattern": {
32       "type": "string", "pattern": "[0-9]{4}([0-9]{4}){3}"},
33     "multi": {
34       "type": "array",
35       "items": { "type": "string", "enum": ["val1", "val2"]},
36       "apex": {"itype": "combobox", "textBefore": "Array"}
37     }
38   }
39 }
```

The image shows the APEX-UI rendering of the JSON schema. Dashed green arrows map specific schema properties to their corresponding UI controls:

- enum**: Maps to a radio button group with options **val1** (selected) and **val2**.
- short\_string**: Maps to a text input field labeled **Short String** with the value **short**.
- long\_string**: Maps to a rich text editor (Paragraph) containing the text **long long 15" long**.
- bool**: Maps to a toggle switch labeled **Bool**, currently in the 'on' position.
- int**: Maps to a star rating control labeled **\*-Rating**, showing 4 stars.
- number**: Maps to a numeric input field labeled **Number** with the value **123.456**.
- money**: Maps to a currency input field labeled **Money** with the value **\$100.00**.
- date**: Maps to a date picker labeled **Date** with the value **2024-03-22**.
- date\_time**: Maps to a date-time picker labeled **Date Time** with the value **2024-03-22 21:30**.
- email**: Maps to an email input field labeled **Email** with the value **support@oracle.com**.
- uri**: Maps to a URI input field labeled **Uri** with the value **https://oracle.com**.
- pattern**: Maps to a pattern input field labeled **Pattern** with the value **1234 5678 9012 3456**.
- multi**: Maps to a multi-select combobox labeled **Multi** with options **val1** and **val2**.



Idee des Plugins

JSON-Schema und APEX-UI

Komplexe JSON-Schema und APEX-UI

Customizing der APEX-UI

Oracle 23c

Sonstiges

# Schema aus JSON-Validierung

- Mir Oracle23c kann man im Check-Constraint einer JSON-Spalte auch das Schema angeben.
- Was liegt da näher, als dieses auch für die APEX-UI zu nutzen
- Achtung:
  - Leider unterstützt Oracle nicht die kompletten Möglichkeiten des JSON-Schema z.B. wird "\$ref": "..." ignoriert
  - Es gibt Oracle-spezifische Erweiterungen z.B. "extendedType": "..." (Wird von Plugin unterstützt)

```
1 CREATE TABLE object23c(  
2   object_id      INTEGER GENERATED BY DEFAULT ON NULL AS IDENTITY,  
3   object_name    VARCHAR2(30) NOT NULL,  
4   data           JSON,  
5   CONSTRAINT object23c_pk PRIMARY KEY (object_id)  
6 );  
7  
8  
9 ALTER TABLE object23c ADD CONSTRAINT object23c_ck1  
10 CHECK (data IS JSON VALIDATE q'[{  
11   "type"       : "object",  
12   "properties" : {  
13     "fruit"    : {"type"       : "string",  
14                  "minLength" : 1,  
15                  "maxLength" : 10},  
16     "quantity" : {"type"       : "number",  
17                  "minimum"    : 0,  
18                  "maximum"    : 100},  
19     "orderdate": {"type": "string",  
20                  "default": "now",  
21                  "format": "date"}  
22   },  
23   "required"   : ["fruit", "quantity"]  
24   }]'  
25 );
```



Idee des Plugins

JSON-Schema und APEX-UI

Komplexe JSON-Schema und APEX-UI

Customizing der APEX-UI

Oracle 23c



Sonstiges

# Sonstiges

- JSON in CLOB

Für den CLOB den  
Check-Constraint  
IS JSON(STRICT) nutzen

- Häufig müssen in einem  
JSON-Schema “enum“-  
Attribute mit Lookup-Tabellen  
synchron gehalten werden

Lösung:  
Statement-Trigger auf  
Lookup-Tabelle

```
1 ALTER TABLE object ADD CONSTRAINT object_ck_1 check (data IS JSON(STRICT));
2
3
4 CREATE TABLE hotel_feature(
5     feature VARCHAR2(100) NOT NULL,
6     CONSTRAINT hotel_feature_pk PRIMARY KEY(feature)
7 );
8
9 CREATE OR REPLACE TRIGGER hotel_feature_tr
10 AFTER INSERT OR UPDATE OR DELETE ON hotel_feature
11 DECLARE enum VARCHAR2(32000);
12 BEGIN
13     SELECT listagg(''||REPLACE(feature,',', '\')||',' WITHIN GROUP (ORDER BY feature)
14     INTO enum
15     FROM hotel_feature;
16     UPDATE object_type SET object_schema =
17         json_mergepatch(object_schema, '{"properties": {"features": {"items": {"enum": ['||enum||']}}}')
18     WHERE object_type_name='Hotel';
19 END;
20 /
```

# Bekannte Nutzungen des Plugins

- USA: Eine der größten Kommunal-Behörden  
Workflow mit komplexen auf JSON basierenden Formularen  
Go-Live April/May
- Indien:  
PoC: Generische Workflows
- Deutschland:  
PoC: keine Info erhalten



# Weitere Ideen

- Formatierung von JSON-Spalten in Listen/Reports mittels JSON-Path (Rel 0.9.0)

```
"apex": {  
  "display": { "list1": "Model: #$.model#, Vendor: #$.vendor#" }  
}
```

- Images (Rel 0.9.0)

- JSON-Schema unterstützt Strings mit base64 codierten Daten.

```
{  
  "type": "string", "contentEncoding": "base64", "contentMediaType": "image/png"  
}
```

- JSON-Relational-Duality

UI aus der Oracle23c JSON-Duality-View generieren

- Weitere Unterstützung von “array“

Analog Interactive Grid

# Das Ende

## Fragen und Antworten

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