

Dynamische APEX-UI für JSON-Daten mit dem JSON-Region-Plugin

APEX-Connect 2025

Uwe Simon Database Consulting

2025-05-15

Inhaltsverzeichnis

Dynamische APEX-UI für JSON-Daten mit dem JSON-Region-Plugin.....	1
Einleitung.....	3
Idee.....	4
APEX-24.2 und JSON.....	5
JSON-Schema.....	8
JSON-Schema und APEX-UI.....	11
Fehlermeldungen.....	12
Konfiguration im Page-Designer.....	14
Anpassung der APEX-UI.....	16
Unterstützung von Oracle23c-Features.....	18
Dynamische JSON-Schema.....	19
Erfahrung während der Entwicklung.....	20
Sonstiges.....	23
Weiteres hilfreiches Plugin.....	24
Next Steps.....	24

Einleitung

Um das Jahr 2000 mit der Verbreitung von SOAP (*Simple Object Access Protocol*) wurden die generische Datenstrukturen mittels XML (Extensible Markup Language) abgebildet. Für die Beschreibung der Struktur der XML-Daten wird dabei XSD (XML-Schema-Definition, <https://www.w3.org/XML/Schema>) genutzt. Der Nachteil von XML ist der relativ hohe Overhead durch die Tags (jeweils <xxxxxx>abc</xxxxxx>), besonders, wenn die Tags „sprechend“ sind.

Um 2014 wurde dann REST (Representational State Transfer) eingeführt, hier wird nun JSON genutzt, JSON hat den großen Vorteil, dass es auch „lesbarer Text“ ist, aber deutlich weniger Overhead als XML hat („xxxx“:“abc“). Für die Beschreibung der Struktur der JSON-Daten wird dabei JSON-Schema genutzt (<https://json-schema.org>).

Mit Oracle 9i wird XML mit dem Datentyp XML Type unterstützt. Seit Oracle 12c kam die erste Unterstützung von JSON in VARCHAR2/CLOB-Spalten (Check-Constraint IS JSON) und Funktionen für den Zugriff auf JSON-Attribute dazu. Mit Oracle 23c gibt es jetzt auch einen JSON-Datentyp, den Constraint IS JSON VALIDATE ‘...’, die Relational-JSON-Duality, etc.

Es gibt etliche Anwendungsfälle, in denen JSON-Daten genutzt werden, wie z.B.

- 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.

JSON-Schema wird auch zur Beschreibung/Validierung von REST-APIs mit OpenAPI (bzw. Swagger <https://www.openapis.org/>)..

Vor Oracle-APEX-24.2 gab es keine Out-Of-The-Box-Lösung für die Ein-/Ausgabe der Attribute von JSON-Feldern.

So entstand die Idee diese Funktionalität durch ein **APEX-Plugin** bereitzustellen.

Mit Oracle-APEX-24.2 wurden im Page-Designer unter „Shared Components“ die neuen Datasources „Duality Views“ und „JSON Sources für „JSON-Collection-Table“ und „Table with JSON-Column“ eingeführt. Diese neuen Datasources unterstützen aber nur „statische JSON-Schema“. Jegliche Änderungen am JSON-Schema bedeuten hier aber Änderungen in der Applikation.

Das **APEX-Plugin** ermöglicht hier aber auch dynamische JSON-Schema, die erst zur Laufzeit ausgewertet werden und somit die UI der Applikation immer synchron zum JSON-Schema ist.

Idee

Die APEX-UI soll dabei durch das JSON-Schema der JSON-Daten beschrieben werden.

Da die JSON-Daten typischerweise mehr als ein Attribut haben, wird dies mit dem Region-Plugin **JSON-Region** implementiert.

Das Plugin sollte dabei so flexibel wie möglich sein.

Anforderungen:

- 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 APEX-UI-Layouts zur Unterstützung von weiteren APEX-Item-Typen
- Dynamische Ui – Felder werden abhängig von anderen Feldern angezeigt - durch JSON-Schema „\$if/\$then/\$else“, „dependentRequired“, „dependentSchema“,
- Dynamische generierte JSON-Schema durch „\$ref“

Für die flexible Nutzung von JSON-Daten und JSON-Schema wird man typischerweise in dem Datenmodell Tabellen mit den JSON-Daten und Lookup-Tabellen mit dem zugehörenden JSON-Schema enthalten.

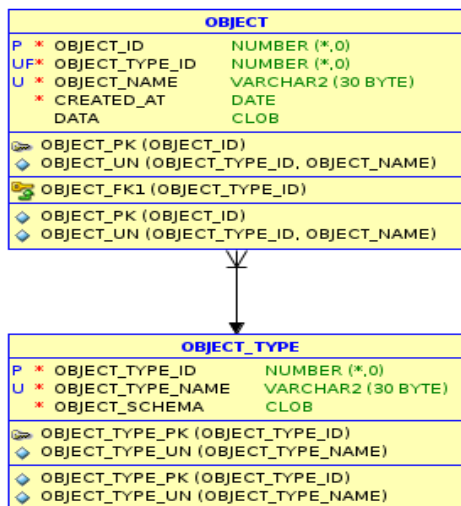


Abbildung 1: JSON-Daten und JSON-Schema

APEX-24.2 und JSON

Oracle 23ai unterstützt JSON-Daten über

- JSON Columns
- Collection Tables
- Collection Views
- Duality Views

Mit APEX-24.2 hat Oracle eine erste Unterstützung von JSON-Daten und JSON-Schema implementiert.

Unter Shared-Components → Data Sources gibt es jetzt

- JSON Sources
 - Table with JSON Column
 - JSON Collection Table
- Duality View

In APEX-24.2 werden Collection-Views (noch) nicht unterstützt.

Diese „Data Sources“ können wie jede andere „Data Source“ im Page-Designer verwendet werden. Hiermit unterstützt APEX jeweils ein fixes JSON-Schema je JSON-Spalte etc. Es können aber mehrere Datasource mit jeweils einem anderen JSON-Schema zu einer Tabelle angelegt werden.

Die Einrichtung der JSON-basierten Datasources ist recht einfach.

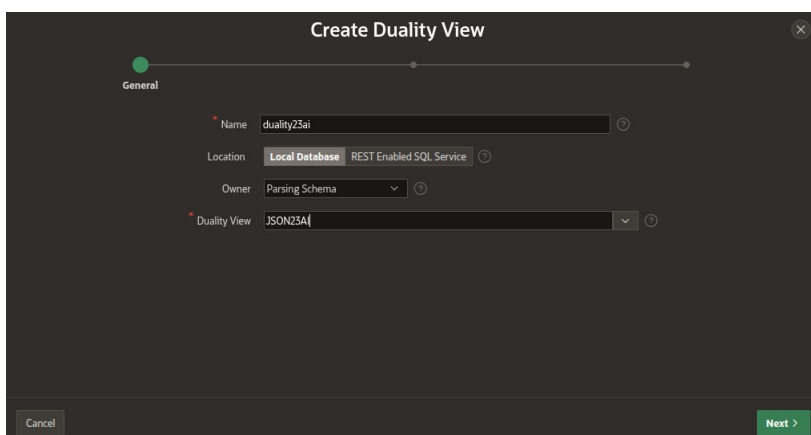


Abbildung 2: Anlegen einer „Data Source“ für eine „Duality View“

Achtung: Das JSON-Schema für eine Duality-View in Oracle23.6 liefert einen DB-Fehler
ORA-06503: PL/SQL: function returned without value

Der Grund ist hier eine Änderung des Outputs der Funktion (Bug 37538056)

DBMS_JSON_SCHEMA.DESCRIBE

die von APEX-24.3-Patch 2 noch nicht unterstützt wird. Duality-Views mit Oracle <23.6 funktionieren ohne Probleme.

Die Definition einer JSON-Source funktioniert analog.

Hier kann zwischen Tabelle mit JSON-Spalte und Table-Collection unterschieden werden.

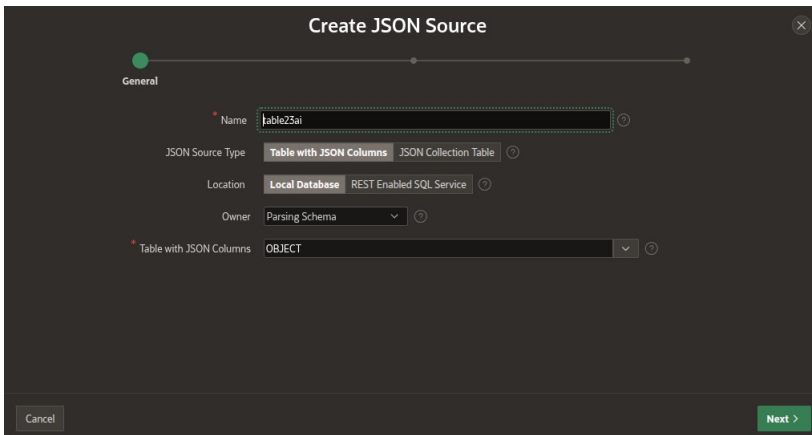


Abbildung 3: „Data Source“ für eine „Table with JSON Column“

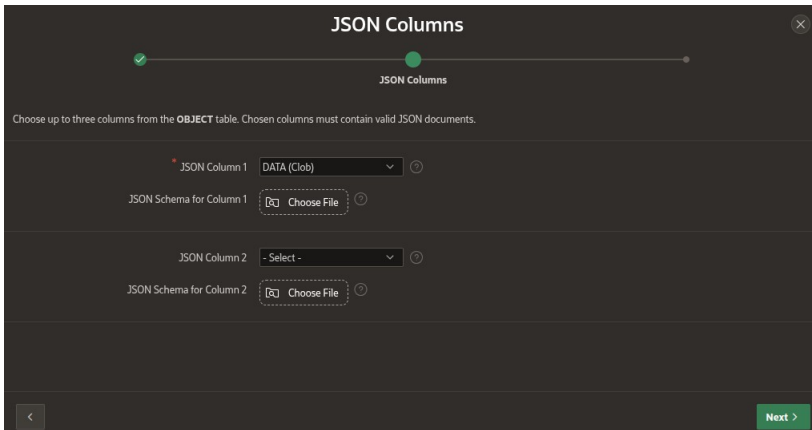


Abbildung 4: JSON-Schema für „Table with JSON Column“

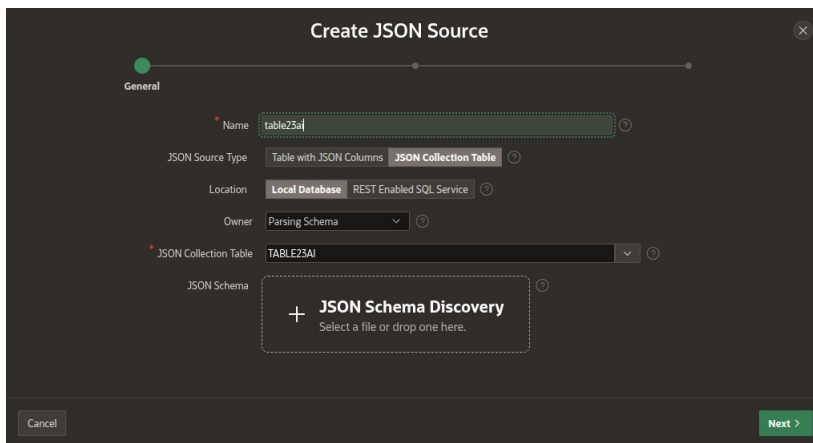


Abbildung 5: „Data Source“ „JSON Collection Table“

Leider extrahiert hier APEX-24.2 nicht das JSON-Schema aus einem ggf. vorhandenen JSON VALIDATE-Constraint. Das JSON-Schema muss hier jeweils manuell extrahiert werden und dann als File importiert werden. Es kann ein JSON-Schema aber basierend auf den Daten in der JSON-Spalte/Collection erzeugt werden.

Dynamische Zuordnungen von JSON-Schemata zu JSON-Spalten sind mit der APEX-24.2 Implementierung aber nicht möglich. Jede Änderung am JSON-Schema bedeutet zuerst eine Synchronisierung in der „Data Source“ und anschließend in den betroffenen Pages.

Das **JSON-Region-Plugin** bietet hier dann die weiteren **dynamischen** Optionen um ein **JSON-Schema** ohne Änderung der Applikation im Page-Designer sogar zur Laufzeit zu ändern. Ferner unterstützt es zusätzliche JSON-Schema Funktionalitäten wie Schema-References, Dependencies und if/then/else, Schema-References, etc.

JSON-Schema

Da die Konfiguration des JSON-Region-Plugins auf einem JSON-Schema basiert, hier eine kurze Beschreibung von JSON-Schema. Die komplette Dokumentation zu JSON-Schema befindet sich unter <https://json-schema.org/>. Ein JSON-Schema wird durch eine JSON-Struktur beschrieben.

Die Beschreibung jedes Feld (Property) besteht dabei aus

- Mussfeld ("required"),
- Datentyp ("type")
- Format ("format")
- Aufzählung ("enum")
- Muster ("pattern")

Ein einfaches JSON-Schema sieht dabei wie folgt aus

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

Abbildung 6: Einfaches JSON-Schema

In einem JSON-Schema können auch komplexere Strukturen abgebildet werden.

- Konstante Werte: `"const": "constant Value"`
- Binärdaten in Textfeldern (aktuell unterstützt das Plugin nur Bilder als Anzeige). Binärdaten werden mittels `"contentEncoding": "base64"` definiert. Die Bedeutung des Inhaltes beschreibt `"contentType"`. Hier sind Werte `"image/png"`, `"image/jpg"` `"image/gif"` für Bilder im PNG, JPG bzw. GIF Format möglich.
Beispiel: `"contentType": "image/png"`
- Rekursive: `{ "type": "object", "properties": {...} }`
- Listen: `{ "type": "array", "items": [...] }`

Plugin-Unterstützung: Tabellen und Multiselect-Felder

- Schema-Referenzen zur Vermeidung von Redundanzen
`"$ref": "#/$defs/schemaX"`
im gleichen Schema und
`"$ref": "/$defs/schemaX"`
aus DB-Tabelle bzw. SQL-Queries..
Plugin-Unterstützung: Referenzen aus DB können für dynamische Sub-schemata genutzt werden, z.B. Auswahllisten oder Auswahl-Hierarchien.
- Conditional Required, ein Feld wird Mussfeld, wenn ein andere Felder nicht leer ist
`"dependentRequired": { "field1": ["field2", ...] }`
z.B. Kreditkartentyp, Kreditkartennummer, Gültigkeit, Securitycode
- `"dependentSchema"`, die Daten eines Subschema werden benötigt, wenn ein anderes Feld nicht leer ist,
- Conditional Schema, je nach Wert eines Feldes, weitere Felder (z.B. bei „abweichende Rechnungsanschrift“ = true, Felder der 2. Anschrift) mittels
`"if": {...}, "then": {...}, "else": {...}`
Hier werden auch `"allOf"` (AND), `"anyOf"` (OR) und `"not"` (NOT) für komplexere Bedingungen unterstützt

Ein komplexes JSON-Schema sieht dann z.B. wie folgt aus

```
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 }
```

Abbildung 7: Komplexes JSON-Schema

JSON-Schema und APEX-UI

Mit einem JSON-Schema kann das Plugin nun eine Region in der APEX-UI generieren.

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/time Date-Picker / Date-Picker+Time / Time-Picker
- enum Pulldown
- email Text Field mit Subtyp Email
- uri Text Field mit Subtype URL
- ...

Analog zum Verhalten des Page-Designers ist der Anzeigename des APEX-Items ist standardmäßig der Name des Attributes (1. Zeichen je Wort groß und „_“ bzw. „-“ werden durch „ „ ersetzt, ... wie Default-Title im Page-Designer). Hier ein Beispiel wie aus dem JSON-Schema die APEX-UI-Region erzeugt wird.

```

{
  "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}" }
  }
}

```

Abbildung 8: JSON-Schema

Abbildung 9: APEX-UI

Fehlermeldungen

Oracle-APEX unterstützt die Validierung von Eingaben. Dies erfolgt bei dem „JSON-Region-Plugin“ identisch. Es werden die Standard-Validierungen und Meldungen von APEX genutzt (mit den gleichen „Problemchen“).

Vom Plugin unterstützte Validierungen

- Integer, Number
- Date, Date-Time
- Regex-Pattern
- Email-Adresse
- URI
- Minimum, Maximum
- Maximale Länge

Die Darstellung sieht hier dann z.B. wie folgt aus. Hierbei sind in der ersten Zeile der Seite „normale“ APEX-Items.

Object

Object Type

full-simple

Enum

val1

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.

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.

Cancel

Create

Abbildung 10: Meldungen bei Validierungsfehlern

2025-05-15

Uwe Simon Database Consulting

13

Konfiguration im Page-Designer

Die Konfiguration im APEX-Page-Designer ist recht einfach. Nachdem auf der Seite eine „JSON-Region“ eingefügt wurde, muss hier nur das Feld angegeben werden, in dem die JSON-Daten stehen und das JSON-Schema. Beim JSON-Schema kann ein „statisches Schema“ direkt in den Attributen der Region angegeben werden, bzw. eine Query, die mit Hilfe eines Typfeldes das Schema per SQL-Query ermittelt, diese Query muss genau eine Zeile mit einer Spalte, die das JSON-Schema enthält, zurückliefern. Ferner unterstützt das Plugin auch das „Read Only“ Attribut.

Read Only

Type: - Select -

Region | Attributes

Filter

Identification

Title: JSON_REGION

Type: Json-Region

Settings

JSON-item: P11_DATA

Source: SQL-Query

SQL-Query

```
select object_schema
from object_type
where
```

Merge Schema: ☐

SQL-Query for referenced JSON-schema

```
SELECT schema, sqlquery
FROM json_region_schema
WHERE path=:p1
```

Column Width: 4

Textarealimit: 250

Template: Header Floating

Keep additional attributes: ☒

Headers: ☒

Hide JSON-item: ☒

Remove NULLS from JSON: ☒

Abbildung 11: Plug-Konfiguration

Konfigurationen:

- JSON-Item Das Item mit den JSON-Daten
- Source „Statisch“, „SQL-Query“, „Generate“
- Statisches Schema Wenn Source=“Static“, dann das JSON-Schema, Wenn leer, Nutzung des JSON-Validate-Constraints
- SQL-Query Wenn Source=“SQL-Query“, dann die SQL-Query
- Merge Schema Ein fixes JSON-Schema, welches mit dem Schema aus der SQL-Query gemerged wird
- SQL-Query for referenced ... Query, die für “\$ref”: “/xxx“ das Schema aus der DB liest
- Keep additional Attributes Wenn die JSON-Daten mehr Attribute als das Schema enthalten, bleiben diese Attribute erhalten
- Headers Ausgabe der Namen bei Subschema als Überschrift
- Hide-JSON-Item Das JSON-Feld wird automatisch unsichtbar
- Remove NULLS from JSON Zur Reduktion der Größe des JSON können leere Felder aus dem JSON entfernt werden
- Column Width Breite der Spalten (1-12)
- Textarea Limit Länge, ab der Strings als Textarea angezeigt werden
- Template Template für Items, „Header Floating“, „Label Above“, „Label Left“, „Label Hidden“

Anpassung der APEX-UI

APEX hat in der UI für einige Datentypen mehrere Darstellungsformen., die durch das Plugin auch genutzt werden können. Ferner soll die gesamte Darstellung auch anpassbar sein.

Ein JSON-Schema kann durch eigene Properties erweitert werden. Darum unterstützt das Plugin für APEX-spezifische Konfiguration das neue Property

```
"apex": {  
    ...  
}
```

Attribute "itemtype" zur Konfiguration des APEX-UI-Items

- "itemtype": "starrating" Integer-Feld als Starrating
- "itemtype": "switch" Boolean-Feld als Switch
- "itemtype": "password" Kennwortfeld
- "itemtype": "pctgraph" Anzeige als Balken in % (0-100)
- "itemtype": "currency" Anzeige von Integer/Number als Währung

Ab APEX 23.2

- "itemtype": "richtext" Für lange Strings Richtext-Editor
- "itemtype": "combobox" Für Multiselect Combobox mit „Chips“
- "itemtype": "qrcode" Darstellung von String/Integer/Number als QR-Code

Ab APEX 24.1

- "itemtype": "selectone" SelectOne Page-Item
- "itemtype": "selectmany" Selectmany Page-Item
- "itemtype": "fileupload" Beliebige Daten aus Dateien als base64
- "itemtype": "imageupload" Bilder aus Dateien als base64

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
- “mimetypes“: “.jpg,.png“ Mimetypes für fileupload/imageupload
- “maxFileSize“: 100 Maximale Dateigröße für Fileuploads in KB.

Die Itemtypes “fileupload“ und “imageupload“ haben die Besonderheit, dass hier im JSON nicht ein einzelnes Feld genutzt wird sondern ein Datei-Objekt, welches neben dem Dateiinhalte noch die Dateigröße und den Dateinamen enthält.

```
"file": {  
  "name": "filename",  
  "size": 123456,  
  "content": "base64..."  
}
```

Unterstützung von Oracle23c-Features

Mit Oracle23c kann man im Check-Constraint einer JSON-Spalte auch das JSON-Schema angeben. Was liegt da näher, als dieses auch für die APEX-UI zu nutzen. Damit wird dann eine Änderung am CHECK-Constraint sofort in der APEX-UI sichtbar. Zur Konfiguration bleibt dabei im Page-Designer für die Plugin-Region das „Static Schema“ leer.

Achtung:

Leider unterstützt Oracle nicht die kompletten Möglichkeiten des JSON-Schema z.B. wird „\$ref“: „...“ ignoriert

Ferner gibt es noch Oracle-spezifische Erweiterungen z.B. „extendedType“: „...“, die durch das Plugin unterstützt werden .

```
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 );
```

Abbildung 12: Oracle 23C JSON-Validierung


Fruit bbb	Quantity 1	Orderdate 31-MAR-24	
--------------	---------------	------------------------	---

Abbildung 13: Darstellung in der Plugin-Region

Dynamische JSON-Schema

In der Konfiguration des Plugins kann eine Query zur Auflösung von Schema-Referenzen abgelegt werden, die je Plugin-Nutzung überschrieben werden kann.

Hier wird hierzu eine Tabelle mit 3 Spalten

url	VARCHAR2	URL der Referenz z.B. „/defs/enum“
schema	VARCHAR2	Wenn nicht leer, ein fixes Schema
query	CLOB	Eine SQL-Query, die das JSON-Schema generiert.

benötigt.

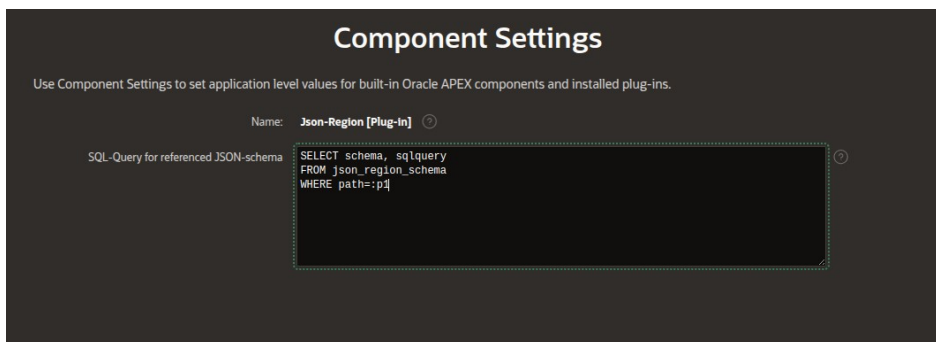


Abbildung 14: SQL-Query zur Auflösung externer Schema-Referenzen

Die hier abgelegt SQL-Query löst dann die Referenz der “\$ref” zur Laufzeit auf. Hierdurch wird das „statische“ JSON-Schema dynamisiert.

Beispiele für dieses Feature

- Nutzung von einheitlichen Sub-Schema wie Adressen etc.
- Dynamische Enums basierend auf den aktuellen Daten einer Lookup-Tabelle erzeugt werden.
- Ferner kann hiermit auch aus einer Werte-Hierarchie eine Auswahllisten-Hierarchie erzeugt werden, in der die Werte von Auswahllisten basierend auf einer anderen Auswahl angezeigt werden können. Hierzu werden die Auswahllisten im JSON-Schema mittels “\$if”/“\$then”/“\$else” verbunden.

Für einen „enum“ enthält dann entweder die Spalte „schema“ das statische Subschema oder die „query“ generierte dieses Subschema

```
{ "type": "string", "enum": ["val1", "val2", "val3"] }
```

für einen einfachen Enum oder

```
{ "type": "string", "enum": [1,2,3],  
  "apex": {"enum": {1: "val1", 2: "val2", 3, "val3"}  
}
```

für einen Enum mit Mapping der Werte auf Strings

oder dieses Schema wird durch dir Sqlquery generiert.

Beispiel-Code für Funktionen gibt es im Github-Repository des Plugins unter **examples/create_json_region_schema.sql** und **examples/create_json_region_hierarchie.sql**

Erfahrung während der Entwicklung

Idee bei der Entwicklung war, die Logik für die UI-Erzeugung in Javascript abzulegen, Zum Einen belastet diese Logik dann die DB nicht, zum Anderen kann man in PL/SQL ja auch JSON bearbeiten/nutzen, es ist gegenüber Javascript aber doch „etwas unhandlich“

Die erste Version mit den „einfachen Typen „string“, „integer“, „number“, „boolean“ war recht schnell erstellt. Im „Prinzip“ ist die Implementierung des Region-Plugins recht einfach.

- Für alle Items
 - HTML-Tag erzeugen
- HTML an die Region per `$('#REGIONID').html(...)` hängen
- Für alle UI-Items
 - `apex.item.create('NAME')`
 - `apex.item('NAME')`

Die HTML-Tags werden alle mittels Templates erzeugt, hierbei ist `#XX#` jeweils ein Platzhalter , der dann durch Aufruf von `apex.util.applyTemplate(...)` mit den passenden Werten ersetzt wird. Das einfachste ist hierbei der „einfache String“

```
<input type="text" id="#ID#" name="#ID#" #REQUIRED#  
#MINLENGTH# #MAXLENGTH# #PATTERN# class="text_field apex-item-text"  
data-trim-spaces="#TRIMSPACES#" aria-describedby="#ID#_error">
```

Dies wird dann ersetzt durch

```
<input type="text" id="P2_DATA_string" name="P2_DATA_string" required=""  
class=" text_field apex-item-text"  
data-trim-spaces="BOTH" aria-describedby="P2_DATA_string_error">
```

Die erste Hürde war das das Default-Verhalten bei Speichern der Daten. Hier wird versucht auch die Eingabe-Daten der neu erzeugen UI-Items mit abzuspeichern, was zu Fehlermeldungen führt. Dies lässt sich beim Anlegen der UI-Items verhindern, wenn sie kein Attribut `name="NAME"` haben. Das Attribut „name“ wird aber bei der Validierung benötigt, ohne dies werden die Felder nicht in der Default-Validierung und den Default-Fehlermeldungen berücksichtigt. Lösung: Mit Attribut „name“ anlegen, dann aber vor dem Speichern löschen.

Die Erste Version war recht schnell erstellt, die nächsten Erweiterungen waren dann doch deutlich aufwändiger (Richtext-Editor, QRCode, ...).

Nach der Veröffentlichung unter „apex.world“ wünschte sich ein Anwender auch den Support für APEX 22.1 (ich hatte mit 22.2 gestartet). Die Version 22.1 hat aber andere Date/Date-Time/Picker. Dies brachte dann die erste versionsabhängige Unterscheidung in Javascript.

```
if(apex.env.APEX_VERSION >='22.2.0'){
    ...
} else {
    ...
}
```

Mit APEX23.2 kam dann durch das neuen „QRCode“-Item auch eine Unterscheidung in PL/SQL, da die Funktion zum Generieren der QRCode nur im PL/SQL von APEX enthalten ist. Die UI nutzt hierbei dann einen AJAX-Callback.

Hierdurch kam dann auch eine Versionsabhängigkeit in PL/SQL dazu. Leider gibt es aktuell in PL/SQL keine Konstante für das aktuelle APEX-Release (analog Javascript „apex.env.APEX_VERSION“)

Man kann aber die Version der WWV_FLOW_API abfragen, die wohl dem Releasedatum der APEX-Version entspricht.

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

Die Implementierung wäre ja zu einfach, wenn es bei APEX nicht unterschiedliches Verhalten von UI-Items in Javascript gäbe (wohl bedingt durch die lange Historie).

Hier hat schon ein einfaches apex.item(...).setValue(...) seine Tücken.

Beim Date/Date-Time-Picker vor 22.2 wird der Inhalt in der UI zerstört, wenn man es „nach dem Rendern“ aufruft. Bei QRCode und RichttextEditor gibt es Fehler wenn man es „vor dem Rendern“ aufruft.

Der QRCode ist dahingehend trickreich, dass beim Anlegen des HTML-Tags direkt das „Image“ direkt als SVG erwartet. Mittels apex.item(...).setValue() kann an einen neuen Wert erst nach dem Rendern angegeben (vorher wird er ignoriert).

Beim RichTextEditor kommt erschwerend hinzu, dass die Initialisierung der Javascript-Libraries asynchron abläuft und je nach Browser, Debug-Output on/off, .. ggf. erst nach dem Rendern abgeschlossen ist. Lösung hier: Bei Nutzung des UI-Items warten bis Initialisierung abgeschlossen ist. Dafür gibt es Freundlicherweise eine Funktion, der Aufruf ist vereinfacht

```
$('#a-rich-text-editor').editorElement[0].getEditor()
```

Beim Tag <input ...> unterstützen alle gängigen aktuellen Browser noch weitere Attribute wie minLength="--“, „pattern=“..“, type="time“, ... Diese werden direkt im Browser geprüft. Hier werden auch die Fehlermeldungen dazu erzeugt. Dummerweise aber nicht in der Sprache der Webseite, sondern in Sprache der Browser-Oberfläche (die kommen also z.B. in Deutsch, auch wenn die Webseite in Englisch angezeigt wird).

Manche Tücken kamen auch durch die Dokumentation im APEX-Javascript-Code, die ist nicht immer 100% korrekt.

Sonstiges

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

Lösung:

Ein Statement-Trigger auf die Lookup-Tabelle.

Beispiel:

Hier wird bei Änderungen in der Tabelle HOTEL_FEATURES im JSON-Schema des Objekttypes „Hotel“ das Feld „features“

```
{
  "properties": {
    "features": {
      "items": {
        "enum": []
      }
    }
  }
}
```

aktualisiert.

```
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 /
```

Abbildung 15: Trigger zur Synchronisierung von "enum"

Weiteres hilfreiches Plugin

Zur Darstellung von dynamischen JSON-Feldern in Listen gibt es von mir noch das Plugin „**json-item-display**“.

Dies ist ebenfalls unter „apex.world“ zu finden.

Hiermit können JSON-Spalten in Listen/Reports mittels JSON-Path (Rel 0.9.0)

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

dargestellt werden. „display“ ist hierbei ein symbolischer Name für die Liste. So können z.B. „Übersichtslisten“ bzw. „Detaillisten“ unterstützt werden und die Konfiguration befindet sich an einem zentralen Ort.

Next Steps

- JSON-Schema aus OpenAPI
- JSON-Schema aus JSON-Forms
- ...