Université IBM i 2018

16 et 17 mai

IBM Client Center Paris

Session S09 – DB2 et support de JSON

Nathanaël BONNET
Gaia
nathanael.bonnet@gaia.fr

Gaia

- Conseil et formation IBM i depuis 1995
  - Inter et intra entreprise
- Base de connaissance en ligne
  - https://know400.gaia.fr
- Organisateur des matinées 400 iday
  - https://www.gaia.fr/400iday-3

GAIA

https://www.gaia.fr

https://twitter.com/GaiaFrance

Plan de la présentation

- Introduction à JSON
- Possibilités DB2 pour JSON
- Publication de document JSON
- Consommation de document JSON

Introduction à JSON

JSON

- JavaScript Object Notation
  - Format de représentation des données
  - Originellement amené par JavaScript
    - Mais utilisable par ailleurs
- Très répandu lors de l'utilisation
  - Services web REST
  - Open Data
  - Stockage local pour des applications
  - …
- Plus léger que XML
  - En nombre de caractères
  - Mais ne permet pas de validation

JSON

- Le langage JSON est simple et efficace
  - 2 types d'éléments structurels :
    - Paires nom / valeur
    - Listes ordonnées de valeurs
  - 3 types de données :
    - Objets
    - Tableaux
    - Valeurs génériques de type tableau, objet, booléen, nombre, chaine ou null
- Encodage
  - Supporte uniquement UTF-8 (recommandé), UTF-16 et UTF-32
- Référence
  - https://tools.ietf.org/html/rfc7159

JSON

- Syntaxes
  - Dérivée de JavaScript, sous forme de couples nom:valeur
    - "Nom": "Valeur"
  - Début / fin
    - { … }
  - Objet
    - { X, X, X, X }
  - Types simples
    - String entre double-quote ", nombre et booléen (true/false) sans délimiteur
    - Les caractères de contrôles sont échappés par \
  - Tableau
    - [X, X, X, X]

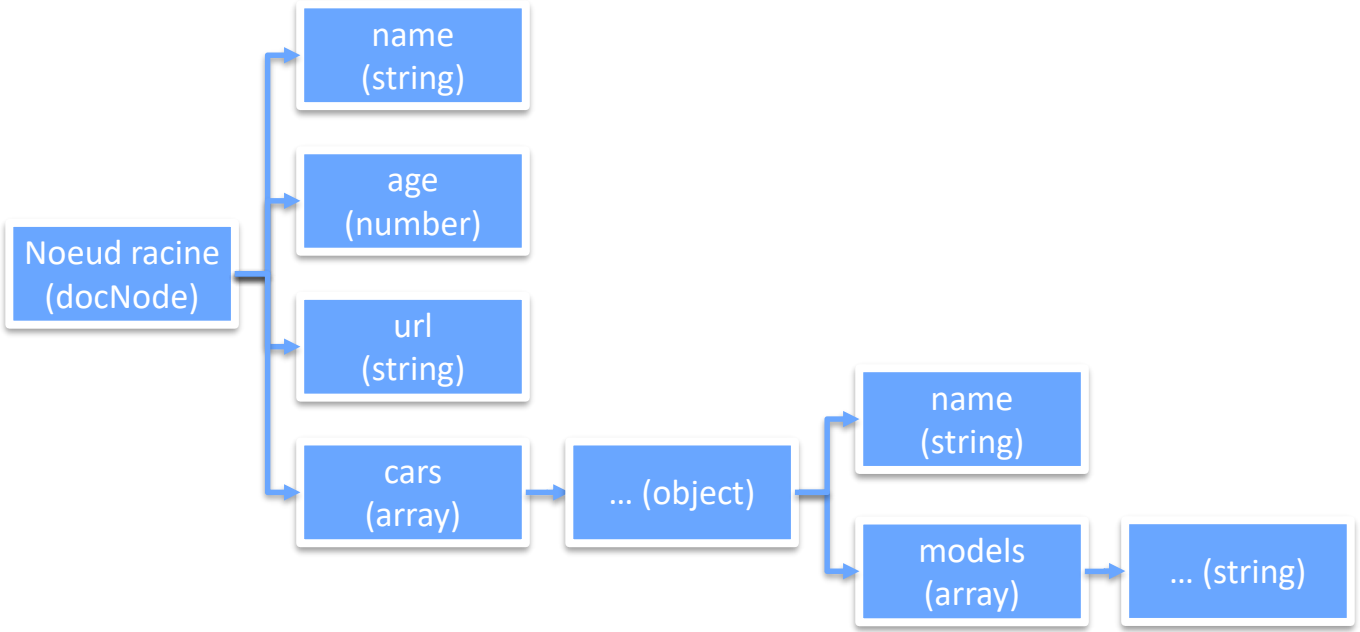
JSON

- Exemple

```
{
    "name":"John",
    "age":30,
    "url":"http:\/\/www.john.io\/",
    "cars": [
        { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },
        { "name":"BMW", "models":[ "320", "X3", "X5" ] },
        { "name":"Fiat", "models":[ "500", "Panda" ] }
    ]
 }
```

Arbre JSON

- Le flux JSON est manipulé par les parseurs (YAJL par exemple) sous forme d'un arbre



JSON vs XML

- Les deux langages permettent de représenter et d'échanger des données
  - JSON est plus léger en taille et ressources nécessaires
  - XML est plus robuste (grammaire)

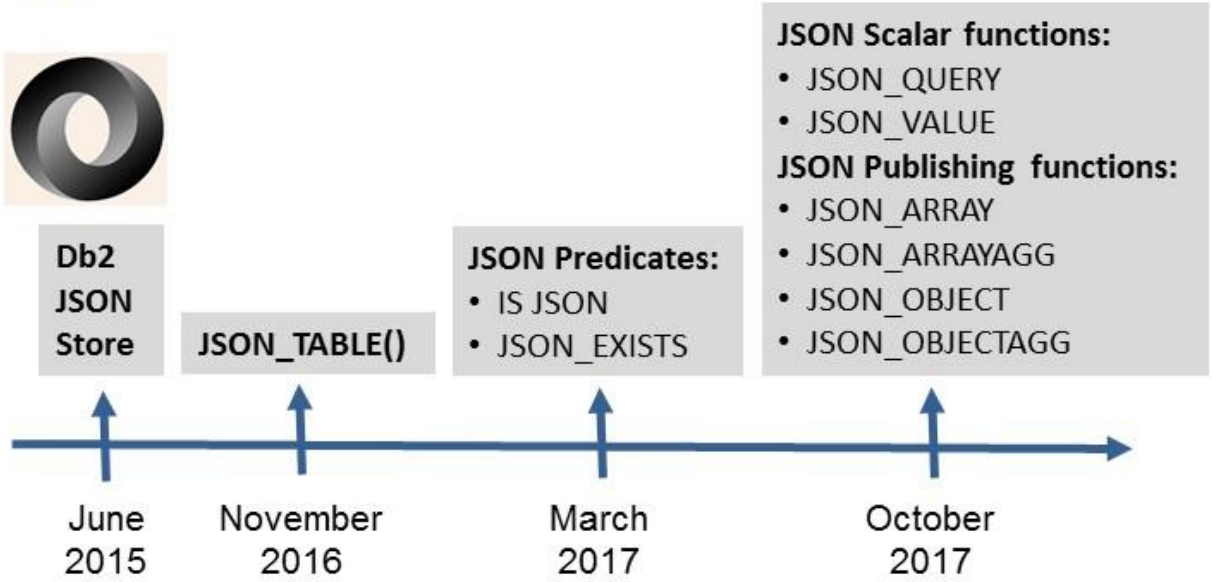
```json
{ "menu": {
    "id": "file",
    "value": "File",
    "popup": {
         "menuitem": [
           { "value": "New", "onclick": "CreateNewDoc()"
},
           { "value": "Open", "onclick": "OpenDoc()" },
           { "value": "Close", "onclick": "CloseDoc()" }
                    ]
              }
           }
}
```

```xml
<menu id="file" value="File">
  <popup>
    <menuitem value="New">CreateNewDoc()</menuitem>
    <menuitem value="Open">OpenDoc()</menuitem>
    <menuitem value="Close">CloseDoc()</menuitem>
  </popup>
</menu>
```

DB2 for i

Cognitive Systems

JSON support in Db2 for i



Source : https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/IBM%20i%20Technology%20Updates/page/JSON%20Scalar%20Functions

Possibilités DB2 pour JSON

Type et stockage

- Contrairement à XML
  - Aucun type de donnée JSON disponible
  - ➔ pas de validation syntaxique à l'insertion
- Pour le stockage, on utilisera une colonne
  - CHAR ou VARCHAR
  - CLOB
  - BINARY ou VARBINARY
  - BLOB

Fonctions

Nom	Usage
BSON2JSON	Transforme un BLOB en string contenant du JSON
JSON2BSON	Transforme un string contenant du JSON en BLOB
BSON_VALIDATE	Valide des données JSON
JSON_BINARY	Extrait un élément, y compris si c'est un tableau
JSON_LEN	Retourne le nombre de postes du tableau JSON
JSON_TABLE JSON_TABLE_BINARY	Extrait les données du document JSON sous forme relationnelle
JSON_TYPE	Retourne le type de l'élément JSON
JSON_VAL	Valeur d'un élément JSON
JSON_GET_POS_ARR_INDEX	Permet de trouver le rang d'un élément dans le tableau JSON
JSON_UPDATE	Permet de mettre à jour une partie du document JSON Non documenté …

Fonctions (suite) et prédicat

- Avec la 7.3 TR3

Nom	Usage
JSON_QUERY	Retourne une valeur au format JSON
JSON_VALUE	Extrait une valeur dans un document JSON
JSON_ARRAY	Génère un tableau JSON
JSON_ARRAYAGG	Génère un tableau JSON depuis GROUP BY
JSON_OBJECT	Génère un objet JSON
JSON_OBJECTAGG	Génère un objet JSON depuis GROUP BY

- Prédicats

Nom	Usage
IS (NOT) JSON	Une valeur est-elle au format JSON ?
JSON_EXISTS	Un élément JSON est-il présent ?

Autres fonctions utiles

- UDF

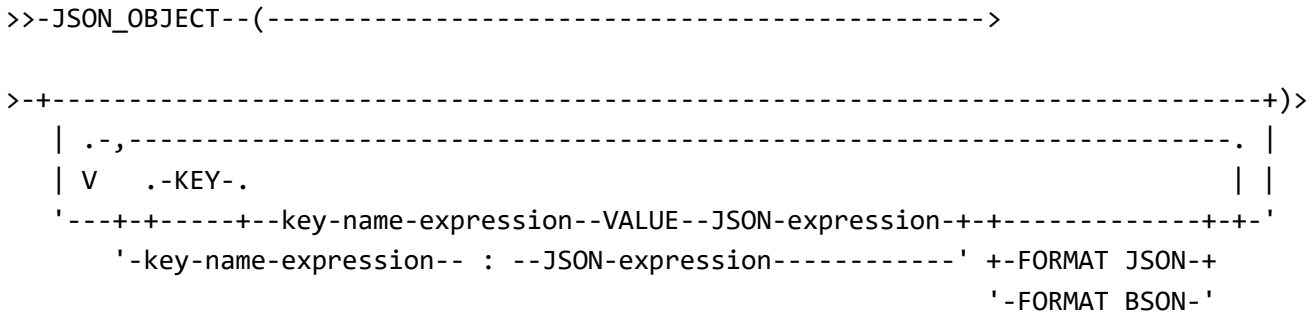
Nom	Usage
URLENCODE	Encoder une valeur au format URL
URLDECODE	Décoder une valeur au format URL
GET_BLOB_FROM_FILE	Charge un BLOB depuis un fichier IFS
GET_CLOB_FROM_FILE	Charge un CLOB depuis un fichier IFS
HTTP*	Invocation de ressources web

Publication de document JSON

Fonctions de publication

JSON_OBJECT

- Génère un objet JSON
  - A partir de couples de valeurs sous la forme CLE : VALEUR
  - Syntaxe minimale (nombreuses options)



JSON_OBJECT

- Exemples

```
VALUES (JSON_OBJECT(
            KEY 'first' VALUE 'John',
            KEY 'last' VALUE 'Doe'));
```

Est equivalent à

```
{"first":"John","last":"Doe"}
```

```
VALUES (JSON_OBJECT(
            'first' : 'John',
            'last' : 'Doe'));
```

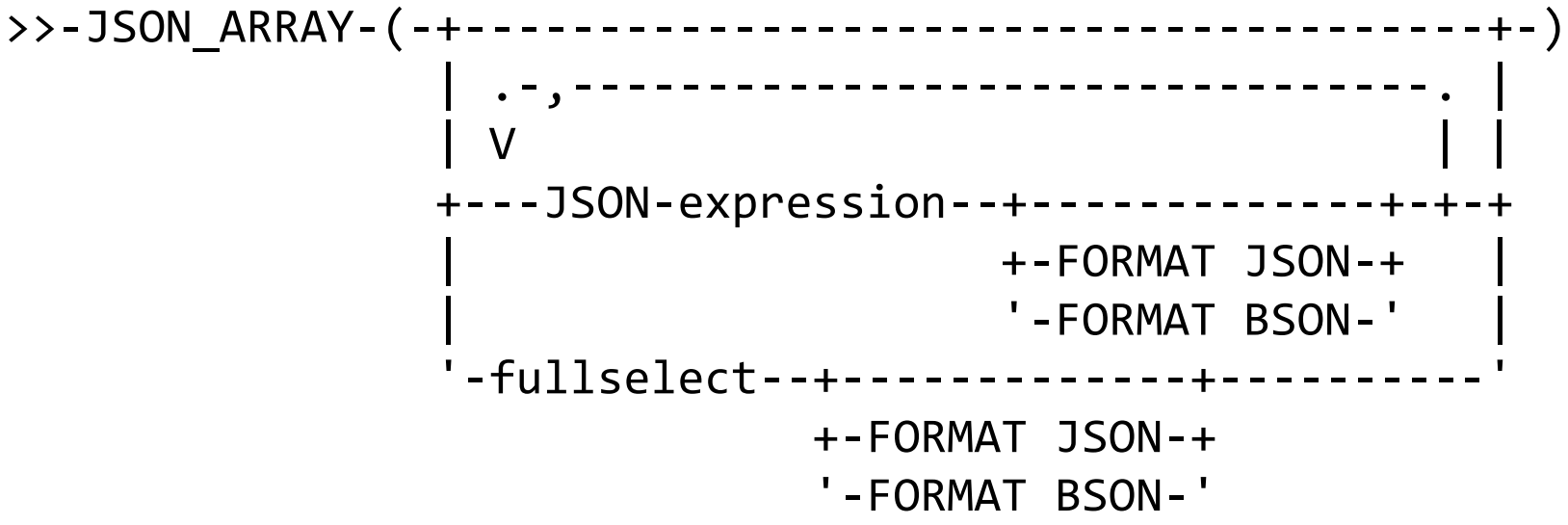
JSON_OBJECT

```
SELECT JSON_OBJECT(
                'horodatage'   : JOTSTP,
                'Système'      : JOSYNM,
                'Utilisateur' : JOUSPF)
FROM exploit.QAUDIT
WHERE jotstp between current timestamp - 2 days
      and current timestamp
ORDER BY jotstp desc ;
```


```
{"horodatage":"2018-01-10-22.59.00.031760","Système":"NEPTUNE ","Utilisateur":"QTMHHTTP  "}
{"horodatage":"2018-01-10-22.58.54.544336","Système":"NEPTUNE ","Utilisateur":"QTCP      "}
{"horodatage":"2018-01-10-22.58.54.544288","Système":"NEPTUNE ","Utilisateur":"QSECOFR   "}
{"horodatage":"2018-01-10-22.58.54.544096","Système":"NEPTUNE ","Utilisateur":"QTCP      "}
{"horodatage":"2018-01-10-22.58.54.543888","Système":"NEPTUNE ","Utilisateur":"QSECOFR   "}
{"horodatage":"2018-01-10-22.58.53.017840","Système":"NEPTUNE ","Utilisateur":"CTL4I     "}
{"horodatage":"2018-01-10-22.58.52.966752","Système":"NEPTUNE ","Utilisateur":"QTMHHTTP  "}
{"horodatage":"2018-01-10-22.58.52.966128","Système":"NEPTUNE ","Utilisateur":"QTMHHTTP  "}
```

JSON_ARRAY

- Génère un tableau JSON
  - Syntaxe minimale (nombreuses options)



JSON_ARRAY

- Exemples

```
VALUES (JSON_ARRAY('Lyon', 'Paris', 'Nantes'));
```

```
00001
["Lyon","Paris","Nantes"]
```

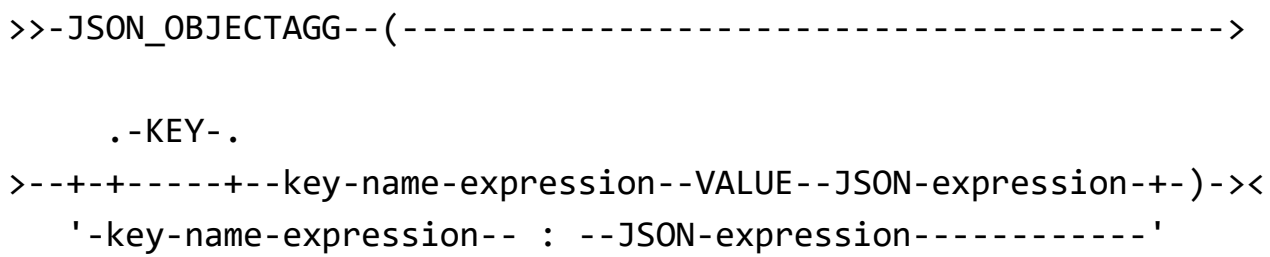
```
values( json_array((
   select distinct JOCODE concat JOENTT
   from exploit.qaudit
   where jotstp between current timestamp - 1 months and
current timestamp )) );
```

- Remarque
  - 2 parenthèses ouvrantes/fermantes pour une sous-requête

```
00001
["TSV","TSK","TPS","TPA","TLD","TJS","TIP","TIM","TGS","TGR","TDO","TCO","TCA","TAF","JPR"]
```

JSON_OBJECTAGG

- Génère une série d’objets JSON depuis un GROUP BY
  - Syntaxe minimale (nombreuses options)



JSON_OBJECTAGG

- Exemples

```
SELECT JSON_OBJECTAGG(deptno VALUE mgrno ABSENT ON NULL)
FROM nbsqlsmpl.dept ;
```

```
{"A00":"000010","B01":"000020","C01":"000030","D11":"000060","D21":"000070",
```

```
SELECT deptno,
       JSON_OBJECTAGG(projno VALUE projname) AS projlist
FROM nbsqlsmpl.proj
WHERE deptno LIKE 'D%’
GROUP BY deptno ;
```

DEPTNO	PROJLIST
D01	{"AD3100":"ADMIN SERVICES","MA2100":"WELD LINE AUTOMATION"}
D11	{"MA2110":"W L PROGRAMMING","MA2111":"W L PROGRAM DESIGN","MA2112":"W L ROBOT DESIGN","MA2113":"W
D21	{"AD3110":"GENERAL ADMIN SYSTEMS","AD3111":"PAYROLL PROGRAMMING","AD3112":"PERSONNEL PROGRAMMING",

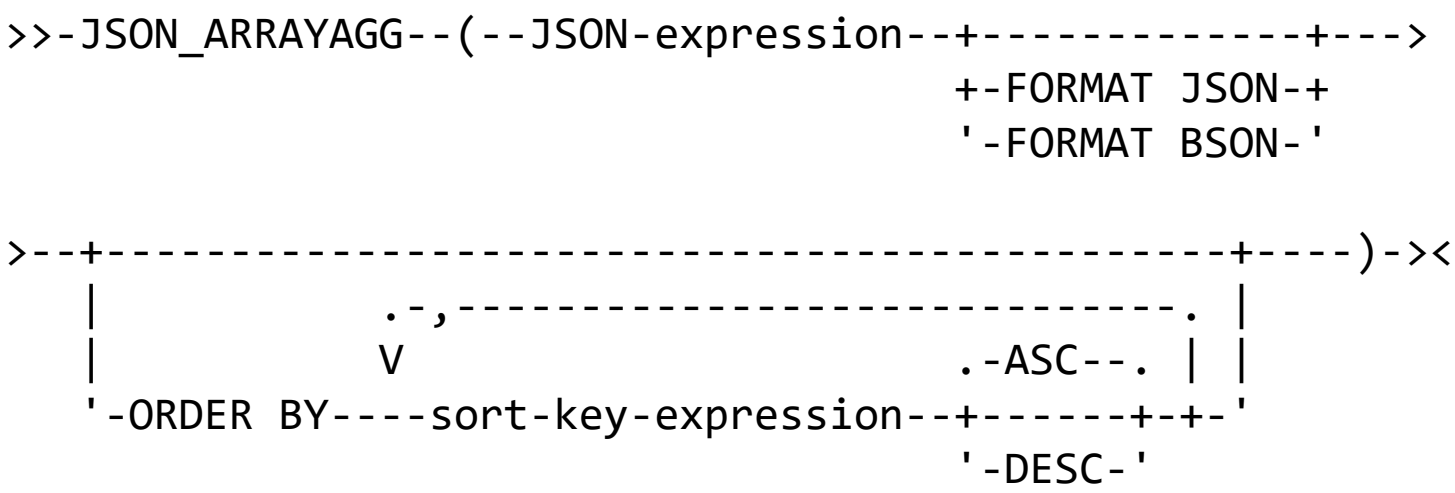
JSON_OBJECTAGG

```
select JOCODE, JOENTT,
       json_objectagg(char(JOSEQN) value JOUSPF)
FROM EXPLOIT.QAUDIT
WHERE jotstp between current timestamp - 2 days
      and current timestamp
GROUP BY JOCODE, JOENTT ;
```

JOCODE	JOENTT	00003					
J	PR	{"2787277	":"QSYS	","3024100	":"QSYS	"}	
T	CA	{"2787278	":"QSECOFR	","2787281	":"QSECOFR	","2787288	
T	CO	{"2787655	":"QTMHHTTP	","2787864	":"QTMHHTTP	","2788077	
T	DO	{"2787680	":"QTMHHTTP	","2787869	":"QTMHHTTP	","2788095	
T	GS	{"2791498	":"QSECOFR	","2791499	":"QSECOFR	","2791500	
T	IM	{"2791048	":"QSYS	","2791340	":"QSYS	","2791751	
T	IP	{"2788613	":"QSYS	","2788614	":"QSYS	","2789506	

JSON_ARRAYAGG

- Génère une tableau JSON depuis un GROUP BY
  - Syntaxe minimale (nombreuses options)



JSON_ARRAYAGG

- Exemples

```
select JSON_ARRAYAGG(DEPTNO)
FROM nbsqlsmpl.DEPT
WHERE DEPTNAME LIKE 'BRANCH OFFICE%' ;
```

00001
["F22","G22","H22","I22","J22"]

```
select JOCODE, json_arrayagg(JOENTT)
FROM exploit.qaudit
WHERE jotstp between current timestamp - 1 months and current
timestamp
GROUP BY jocode ;
```

JOCODE	00002
J	["PR","PR","PR","PR","PR","PR","PR"
T	["AF","AF","AF","AF","CA","CA","CA"

Exemples

On rassemble les pièces !

- Possibilité de créer des « fragments » individuels
- Ces fragments peuvent s'imbriquer les uns dans les autres :

```
values json_object( key 'client' value json_object(
key 'numclient' value 123456 ) ) ;
```

```
{"client":{"numclient":123456}}
```

```
values json_object( key 'clients' value json_array(
123456, 789012, 345678 ) ) ;
```

```
{"clients":[123456,789012,345678]}
```

Employés par département

```
with lst_dept as (
  select workdept, json_arrayagg( json_object(
              KEY 'id' VALUE EMPNO,
              KEY 'firstnme' VALUE FIRSTNME,
              KEY 'lastname' VALUE LASTNAME,
              KEY 'workdept' VALUE WORKDEPT) ) as emp
  from nbsqlsmpl.employee
  group by workdept )
select json_objectagg( workdept value emp format json )
from lst_dept ;
```

Employés par département


```
{
    "A00": [{
            "id": "000010",
            "firstnme": "CHRISTINE",
            "lastname": "HAAS",
            "workdept": "A00"
        }, {
            "id": "000110",
            "firstnme": "VINCENZO",
            "lastname": "LUCCHESSI",
            "workdept": "A00"
        }, {
            "id": "000120",
            "firstnme": "SEAN",
            "lastname": "O'CONNELL",
            "workdept": "A00"
        }, {
            "id": "200010",
            "firstnme": "DIAN",
            "lastname": "HEMMINGER",
            "workdept": "A00"
        }, {
            "id": "200120",
            "firstnme": "GREG",
            "lastname": "ORLANDO",
            "workdept": "A00"
        }
    ],
    "B01": [{
            "id": "000020",
            "firstnme": "MICHAEL",
            "lastname": "THOMPSON",
            "workdept": "B01"
        }
    ],
    "C01": [{
            "id": "000030",
            "firstnme": "SALLY",
            "lastname": "KWAN",
            "workdept": "C01"
```

Extraction des employés

- Soit les tables suivantes
  - PERSONNE

ID	NOM	PRENOM	TEL_BUREAU	TEL_DOMICILE
901	Duck	Donald	555-7242	555-3762
902	Pan	Peter	555-8925	-
903	Mouse	Mickey	555-4311	555-6312
904	Gonzales	Speedy	-	-

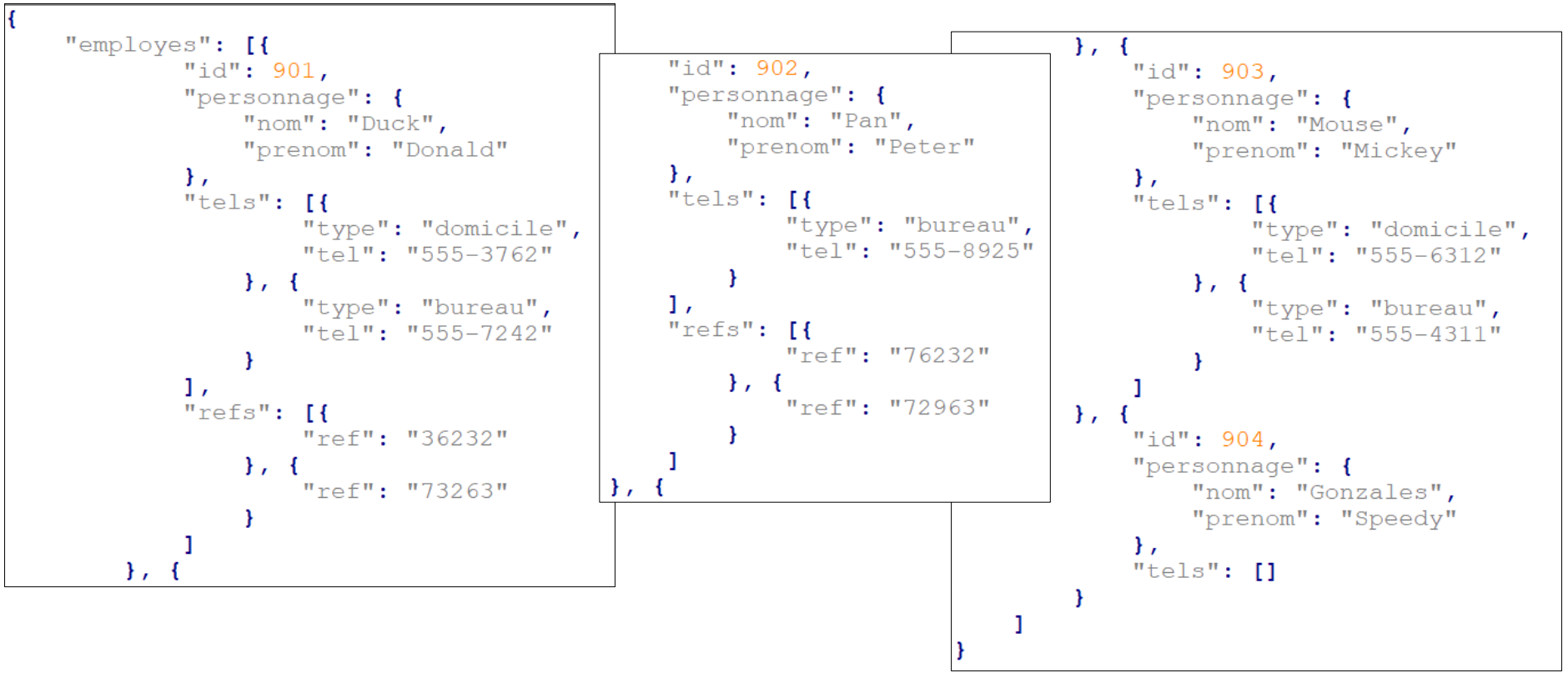
  - PERSONNE_REF

ID	REFERENCE
901	36232
901	73263
902	76232
902	72963

Extraction des employés

```
select json_object('employes' value json_arrayagg(
       json_object ('id' value id,
                    'personnage' value json_object ( 'nom' value nom,
                                                     'prenom' value prenom),
                    'tels' value json_array
                                (case when tel_domicile is not null then
                                    json_object('type' value 'domicile',
                                                'tel' value tel_domicile
                                                ) end format json,
                                 case when tel_bureau is not null then
                                    json_object('type' value 'bureau',
                                                 'tel' value tel_bureau
                                                 ) end format json),
                    'refs' value (select json_arrayagg(
                                                json_object('ref' value reference))
                                  from nbsqlsmpl.personne_ref a
                                  where a.id = e.id group by id) format json
                    absent on null)))
from nbsqlsmpl.personne e;
```

Extraction des employés



Cas spécifiques

Gestion des valeurs nulles

- Il est possible d'indiquer le comportement à adopter
  - Indiquer null (supporté par JSON)
  - Omettre la valeur
- Exemple

```
values json_object('null' : cast( null as char(1))
null on null ) ;
```

```
{"null":null}
```

```
values json_object('null' : cast( null as char(1))
absent on null ) ;
```

```
{}
```

Gestion des booléens

- Non supporté
  - Pas de type booléen dans DB2 !

Autres infos

- Concaténation non supportée pour JSON
  - Existe pour XML
  - Bien pratique …
- Imbrication de fonctions d'imbrication non supportée
  - Oblige généralement à passer par un CTE

Consommation de document JSON

Fonctions de consommation

JSON_VALUE

- Retourne une valeur scalaire SQL depuis une valeur JSON et une expression Xpath
  - Syntaxe minimale

```
>>-JSON_VALUE--(--JSON-expression--+-------------+--,--->
                                   +-FORMAT JSON-+
                                   '-FORMAT BSON-'

>--sql-json-path-expression--+---------------+--------)-><
                             '-AS--path-name-’
```

  - Avec
    - JSON expression
      - retournant une valeur caractère
    - sql-json-path-expression
      - retournant une valeur caractère

JSON_VALUE

- De nombreux attributs
  - Type de valeur retour
  - CCSID
  - Normalisation

- Cf doc en ligne
  - https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/db2/rbafzscajsonvalue.htm

JSON_VALUE

- Exemples

```
VALUES (JSON_VALUE('{"id":"987"}’,
                   '$.id’
                   RETURNING INTEGER));
```

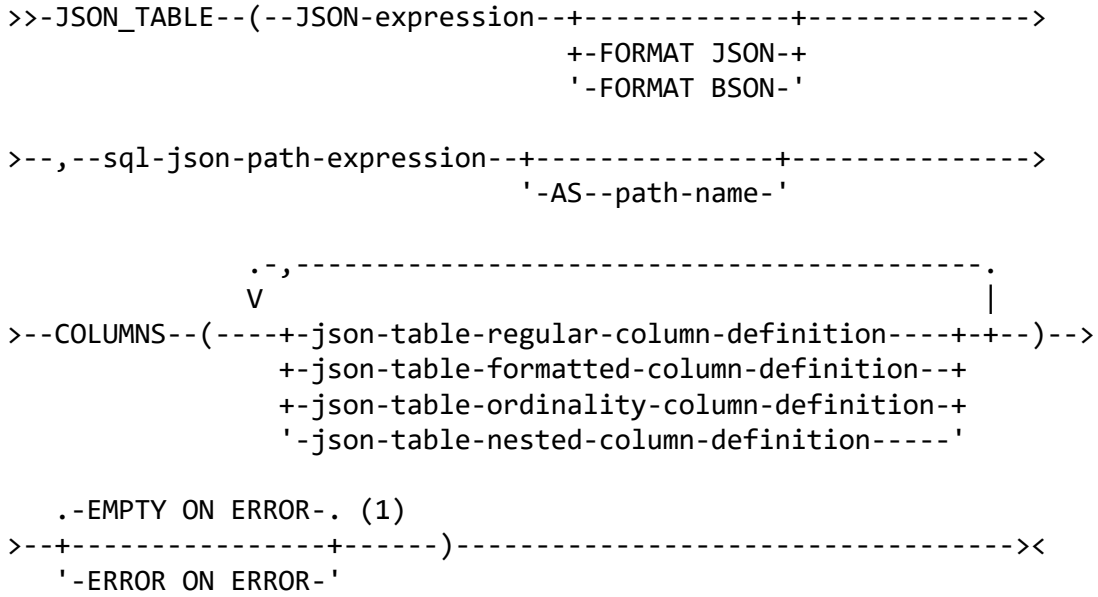
00001
987

```
VALUES (JSON_VALUE('{"friends":["John","Lisa"]}',
                   'strict $.friends'
                   DEFAULT 'Not found' ON ERROR));
```

00001
Not found

JSON_TABLE et JSON_TABLE_BINARY

- Extrait le contenu JSON sous forme d'une table résultat



  - Retourne NULL si l'élément n'est pas trouvé

JSON_TABLE et JSON_TABLE_BINARY

- Exemple

```
SELECT U."id", U."nom", U."prenom"
  FROM JSON_TABLE('{ "id" : 901,
                     "name" : { "first":"John", "last":"Doe" },
                     "phones" : [{"type":"home", "number":"555-3762"},
                                 {"type":"work", "number":"555-7252"}] }',
                  'lax $'
                      COLUMNS( "id"      INTEGER     PATH 'lax $.id' ,
                               "nom"     VARCHAR(20) PATH 'lax $.name.last',
                               "prenom"  VARCHAR(20) PATH 'lax $.name.first' )
                  ) AS U ;
```

- Produit

id	nom	prenom
901	Doe	John

JSON_TABLE et JSON_TABLE_BINARY

- Types de valeurs supportés
  - 1 Chiffre
  - 2 Chaîne
  - 3 Objet
  - 4 Array (tableau)
  - 8 Booléen
  - 10 Null
  - 16 Integer

- JSON_TABLE_BINARY
  - Idem, mais retourne un BSON (binaire)

TYPE	VALUE
16	107B000000
16	10C8010000

BSON2JSON

- Permet de convertir un BLOB contenant un document JSON en chaine de caractères
  - Paramètres
    - Entrée : BLOB (16Mo)
    - Sortie : CLOB (16Mo), encodé en UTF-8

```
select systools.bson2json( data ) as data from jsontst."JClient" ;
```

DATA
{"id":123,"raisonsoc":"IBM","satut":"actif","datcrt":20150731}

BSON2JSON

- La valeur de retour étant un CLOB
  - Certaines interfaces ne l'afficheront pas
    - Avec STRSQL

```
Première ligne à afficher . . _____
....+....1....+....2....+....3..
DATA
*POINTER
********  Fin de données  ********
```

  - Dans ce cas, il faut changer le type
    - `select cast( systools.bson2json( data ) as varchar(1024))`
    - `from jsontst."JClient"`

```
DATA
{"id":123,"raisonsoc":"IBM","satut":"actif","datcrt":20150731}
```

JSON2BSON

- Permet de convertir un CLOB contenant un document JSON en chaine de caractères en format binaire pour stockage
  - Paramètres
    - Entrée : CLOB (16Mo)
    - Sortie : BLOB de 16Mo
  - Exemple

```
insert into jsontst."JClient" (id, data)
values ('id456',
          systools.json2bson(
'{"id":456,"raisonsoc":"GAIA","satut":"actif","datcrt":20150824}'
 )
       )  ;
```

JSON2BSON

- C’est l’opération inverse de BSON2JSON

- Exemple

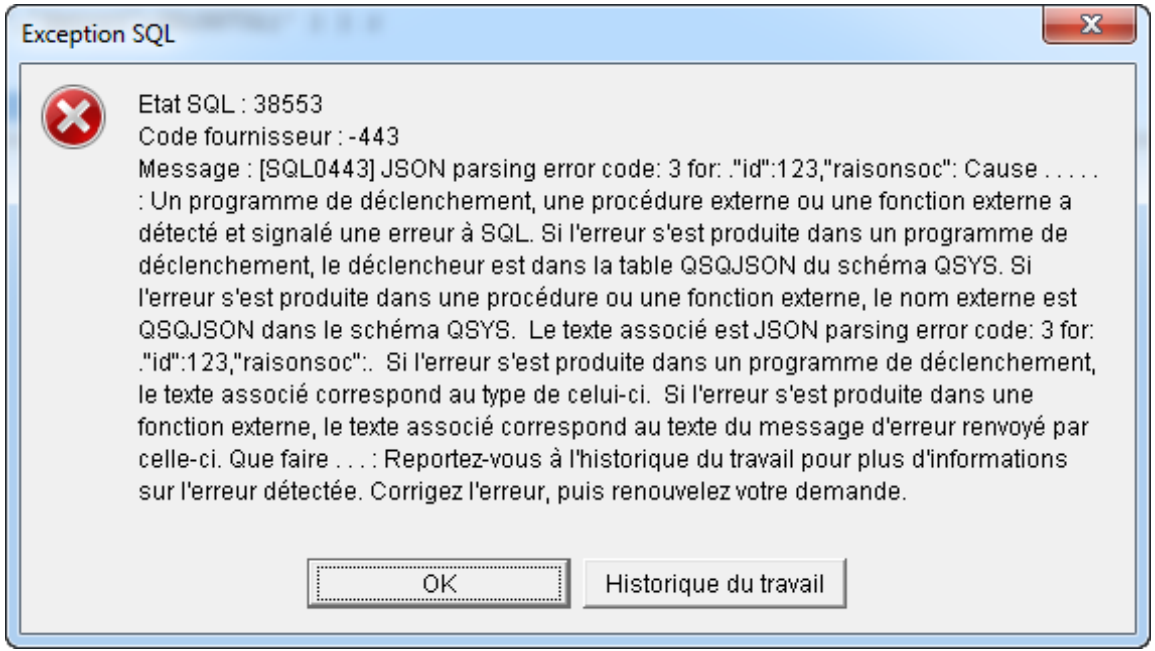
```
values systools.bson2json(
   systools.json2bson(
'{"id":123,"raisonsoc":"IBM","satut":"actif","datcrt":20150731}'
)   ) ;
```

- Produit

```
{"id":123,"raisonsoc":"IBM","satut":"actif","datcrt":20150731}
```

JSON2BSON

- En cas de valeur invalide du JSON, la fonction retourne l'erreur suivante



BSON_VALIDATE

- Valide syntaxiquement un BSON
  - Paramètres
    - Entrée : BLOB (16Mo)
    - Sortie : INT
    - 0 → invalide
    - 1 → valide
  - Exemples

```
values systools.bson_validate(
   systools.json2bson(
'{"id":123,"raisonsoc":"IBM","satut":"actif","datcrt":20150731}' ) ) ;

select id,
       systools.bson_validate(data) as valide,
       systools.bson2json( data )
from jsontst."JClient" ;
```

JSON_VAL

- Retourne la valeur d'un élément JSON dans le type demandé
  - Paramètres
    - Entrée :
      - JSON BLOB (16Mo)
      - Élément VARCHAR(2048)
      - Type de retour VARCHAR(100)
    - Sortie :
      - Valeur VARCHAR(2048)
  - Retourne NULL si l'élément n'est pas trouvé

JSON_VAL

- Exemple

```
select id,
       systools.bson2json( data ),
       json_val(data, 'id', 'i') as id
from jsontst."JClient" ;
```

- Produit

ID	00002	ID
í|ù0ß⊗0□0]]m	{"id":123,"raisonsoc":"IBM","satut":"actif","datcrt":20150731}	123
id456	{"id":456,"raisonsoc":"GAIA","satut":"actif","datcrt":20150824}	456

JSON_VAL

- Types de valeurs supportés
  - ‘n’ DECFLOAT
  - ‘i’ INT
  - ‘l’ BIGINT
  - ‘f’ DOUBLE
  - ‘d’ DATE
  - ‘ts’ TIMESTAMP
  - ‘t’ TIME
  - ‘s:n’ VARCHAR(n)
  - ‘b:n’ VARCHAR(n) FOR BIT DATA
  - ‘u’ INT
- Suffixe optionnel
  - ‘na’ No Array : ne retourne pas la valeur si c’est un tableau (NULL)
  - Sinon, le 1er élément est retourné

JSON_LEN

- Retourne le nombre d’occurrences d’un tableau
  - Paramètres
    - Entrée :
      - JSON BLOB (16Mo)
      - Élément VARCHAR(2048)
    - Sortie :
      - Valeur INTEGER
  - Retourne NULL si l’élément n’est pas trouvé, ou si ce n’est pas un tableau

JSON_LEN

- Exemple
  - Insertion d'un tableau

```
insert into jsontst."JClient" (id, data)
values ('array', json2bson(
'{ "menu": {
  "id": "file",
  "value": "File",
  "popup": { "menuitem": [
    { "value": "New", "onclick": "CreateNewDoc()" },
    { "value": "Open", "onclick": "OpenDoc()" },
    { "value": "Close", "onclick": "CloseDoc()" }
      ]
    }
  }
}
' ) ) ;
```

JSON_LEN

- Comptage

```
select id,
       bson2json( data ),
       json_len(data, 'menu.popup.menuitem') as len
from jsontst."JClient" ;
```

ID	00002	LEN
í|ù0ß⊗0□0]]m	{"id":123,"raisonsoc":"IBM","satut":"actif","datcrt":20150731}	-
id456	{"id":456,"raisonsoc":"GAIA","satut":"actif","datcrt":20150824}	-
array	{"menu":{"id":"file","value":"File","popup":{"menuitem":[{"val...	3

- Ce qui correspond à :

```
"menuitem": [
    { "value": "New", "onclick": "CreateNewDoc()" },
    { "value": "Open", "onclick": "OpenDoc()" },
    { "value": "Close", "onclick": "CloseDoc()" }
      ]
```

JSON_TYPE

- Retourne le type de l'élément recherché
  - Paramètres
    - Entrée :
      - JSON BLOB (16Mo)
      - Élément VARCHAR(2048)
      - Taille maxi INTEGER
    - Sortie :
      - Valeur INTEGER
  - Retourne NULL si l'élément n'est pas trouvé
  - Les types en retour sont identiques à ceux gérés par JSON_TABLE

JSON_TYPE

- Exemple

```
select id,
       systools.bson2json( data ),
       systools.json_type(data, 'id', 10) as len,
       systools.json_type(data, 'menu.popup.menuitem',
1024) as len2
from jsontst."JClient" ;
```

  - Produit

ID	00002	LEN	LEN2
í|ù0ß⊗0□0]]m	{"id":123,"raisonsoc":"IBM","satut":"actif","datcrt":20150...	16	-
id456	{"id":456,"raisonsoc":"GAIA","satut":"actif","datcrt":2015...	16	-
array	{"menu":{"id":"file","value":"File","popup":{"menuitem":[{...	-	4

JSON_BINARY

- Retourne la valeur d'un élément JSON en binaire
- Fonctionne également avec les tableaux, contrairement à JSON_VAL
  - Paramètres
    - Entrée :
      - JSON BLOB (16Mo)
      - Élément VARCHAR(2048)
      - Longueur maxi INTEGER
    - Sortie :
      - Valeur VARCHAR(2050)
  - Retourne NULL si l'élément n'est pas trouvé

JSON_BINARY

- Exemple

```
select id,
       bson2json( data ),
       json_binary(data, 'id', 10) as id ,
       json_binary(data, 'menu.popup.menuitem', 512) as
menuitem
from jsontst."JClient" ;
```

  - Produit

ID	00002	ID	MENUITEM
í|ù0ß☼0□□0]]m	{"id":123,"raisonsoc":"IBM...	□£□□□	-
id456	{"id":456,"raisonsoc":"GAI...	□H□□□	-
array	{"menu":{"id":"file","valu...	-	□q□□□□□□□□□□□I/%IA□□□□□+AI□□?>A%NA,□□□□□äEA/EA+AI@?A□□□□□□□□□□...

JSON_QUERY

- Retourne une valeur au format JSON
  - Syntaxe minimale

```
>>-JSON_QUERY--(--JSON-expression--+-------------+->
                                   +-FORMAT JSON-+
                                   '-FORMAT BSON-'

>--,--sql-json-path-expression--+---------------+-)-><
                                '-AS--path-name-’
```

  - Avec
    - JSON expression
      - retournant une valeur caractère
    - sql-json-path-expression
      - retournant une valeur caractère

JSON_QUERY

- De nombreux attributs
  - Type de valeur retour
  - Gestion des tableaux
  - Gestion des séparateurs
  - Gestion de valeur par défaut
  - Gestion des erreurs

- Cf doc en ligne
  - https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/db2/rbafzscajsonquery.htm

JSON_QUERY

- Exemples

```
VALUES JSON_QUERY(
'{"id":"701", "name":{"first":"John", "last":"Doe"}}',
'$.name');
```

```
00001
{"first":"John","last":"Doe"}
```

```
VALUES JSON_QUERY('{"id":"701", "name":{"first":"John",
"last":"Doe"}}', '$.name.first');
```

```
00001
"John"
```

```
VALUES JSON_QUERY('{"id":"701", "name":{"first":"John",
"last":"Doe"}}', '$.first');
```

```
00001
-
```

Exemples

Extraction d'informations

- Une table PERSONNAGE contenant les informations suivantes


```
{
    "id": 901,
    "personnage": {
        "prenom": "Verbal",
        "nom": "Kint"
    },
    "films": [{
            "type": "policier",
            "titre": "Usual Suspects"
        }
    ]
}
```

```
{
    "id": 902,
    "personnage": {
        "prenom": "Keyser",
        "nom": "Söze"
    },
    "films": [{
            "type": "policier",
            "titre": "Usual Suspects"
        }
    ]
}
```

```
{
    "id": 1027,
    "personnage": {
        "prenom": "François",
        "nom": "Pignon"
    },
    "films": [{
            "type": "comédie",
            "titre": "L'emmerdeur"
        }, {
            "type": "comédie",
            "titre": "Les compères"
        }, {
            "type": "comédie",
            "titre": "Les fugitifs"
        }, {
            "type": "comédie",
            "titre": "Le diner de cons"
        }, {
            "type": "comédie",
            "titre": "Le placard"
        }, {
            "type": "comédie",
            "titre": "La doublure"
        }, {
            "type": "comédie",
            "titre": "L'Emmerdeur"
        }
    ]
}
```

Extraction d'informations

- Liste des personnages / films

```
SELECT U."id", U."nom",U."prenom",U."type film",U."titre film"
  FROM nbsqlsmpl.personnage E,
    JSON_TABLE(E.jsonval,
               'lax $'
               COLUMNS( "id" INTEGER,
                        "nom"  VARCHAR(20) PATH 'lax $.personnage.nom',
                        "prenom"   VARCHAR(20) PATH 'lax $.personnage.prenom',
                        "type film"  VARCHAR(20) PATH 'lax $.films[0].type',
                        "titre film"  VARCHAR(20) PATH 'lax $.films[0].titre')
                      ) AS U ;
```

Extraction d’informations

- Résultats

id	nom	prenom	type film	titre film
901	Kint	Verbal	policier	Usual Suspects
902	Söze	Keyser	policier	Usual Suspects
1027	Pignon	François	comédie	L'emmerdeur

Extraction d'informations

- Liste des personnages / films

```
SELECT U."id", U."prenom",U."nom",U."type film",U."titre" as "titre"
  FROM nbsqlsmpl.personnage E,
    JSON_TABLE(E.jsonval,
               'lax $'
               COLUMNS( "id" INTEGER,
                        "nom"  VARCHAR(20) PATH 'lax $.personnage.nom',
                        "prenom"   VARCHAR(20) PATH 'lax $.personnage.prenom',
                         NESTED PATH 'lax $.films[*]'
                           COLUMNS (
                                    "type film"  VARCHAR(20) PATH 'lax $.type',
                                    "titre"  VARCHAR(20) )
                                        )
                       ) AS U ;
```

Extraction d’informations

- Résultats

id	prenom	nom	type film	titre
901	Verbal	Kint	policier	Usual Suspects
902	Keyser	Söze	policier	Usual Suspects
1027	François	Pignon	comédie	L'emmerdeur
1027	François	Pignon	comédie	Les compères
1027	François	Pignon	comédie	Les fugitifs
1027	François	Pignon	comédie	Le diner de cons
1027	François	Pignon	comédie	Le placard
1027	François	Pignon	comédie	La doublure
1027	François	Pignon	comédie	L'Emmerdeur

Cas réel : iTop

```
SELECT t.* FROM JSON_TABLE(
     SYSTOOLS.HTTPPOSTCLOB ('https://services.gaia.fr/webservices/rest.php',
        CAST ('<httpHeader><header name="Content-Type"
               value="application/x-www-form-urlencoded"/></httpHeader>' AS CLOB(1K)),
        'auth_user=' || systools.urlencode('XXX', '') || '&auth_pwd=' ||
        systools.urlencode('ZZZ', '') || '&version=' || systools.urlencode('1.3', '') ||
        '&json_data=' || systools.urlencode('{"operation": "core/get", "class": "UserRequest",
           "key": "SELECT UserRequest", "output_fields": "operational_status, ref, org_name, caller_name, team_name,
                agent_name, title, start_date, end_date, last_update, close_date" }', '') ),
     'lax $' COLUMNS (
                nested path 'lax $.objects.*[*]'
                    columns ( ref                varchar(20)  PATH 'lax $.fields.ref',
                              operational_status varchar(20)  PATH 'lax $.fields.operational_status',
                              org_name           varchar(40)  PATH 'lax $.fields.org_name',
                              caller_name        varchar(40)  PATH 'lax $.fields.caller_name',
                              team_name          varchar(40)  PATH 'lax $.fields.team_name',
                              agent_name         varchar(40)  PATH 'lax $.fields.agent_name',
                              title              varchar(100) PATH 'lax $.fields.title',
                              start_date         timestamp    PATH 'lax $.fields.start_date',
                              end_date           timestamp    PATH 'lax $.fields.end_date',
                              last_update        timestamp    PATH 'lax $.fields.last_update',
                              close_date         timestamp    PATH 'lax $.fields.close_date' ) ) ) AS t ;
```

Cas réel : iTop

- Résultat

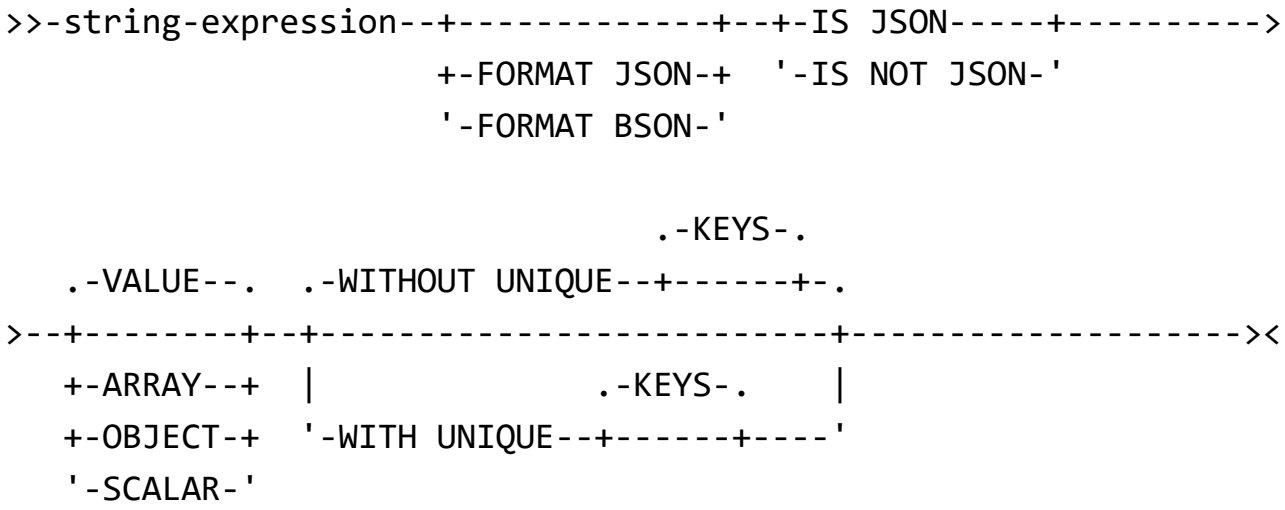
REF	OPERATIONAL_STATUS	ORG_NAME	CALLER_NAME	TEAM_NAME	AGENT_NAME	TITLE	START_DATE
R-000156	resolved	[redacted]	[redacted]	services@gaia.fr		PB Suivi des vagues d'Impression	2018-05-04 12:24:48.
R-000155	ongoing	[redacted]	[redacted]	services@gaia.fr		Accès partage QLJB	2018-05-02 12:22:26.
R-000154	closed	[redacted]	GAIA_CS	services@gaia.fr		Demande d'extractions de données IBM i	2018-04-27 11:53:39.
R-000153	ongoing	[redacted]	[redacted]	services@gaia.fr		Modification dossier image pixopolitan	2018-04-26 17:19:15.
R-000152	closed	[redacted]	Autre utilisateur	services@gaia.fr		OPCON - Gestion liste de bibliothèques job SI820K	2018-04-26 11:40:09.
R-000151	ongoing	[redacted]	[redacted]	services@gaia.fr		Profil utilisateur à voir et modifier	2018-04-25 09:58:42.
R-000150	resolved	[redacted]	[redacted]	services@gaia.fr		Problème mise en ruche	2018-04-24 16:07:30.
R-000149	resolved	[redacted]	[redacted]	services@gaia.fr		Marge produit hors norme pixo	2018-04-24 14:42:18.
R-000148	resolved	[redacted]	[redacted]	services@gaia.fr		Interface Commandes Pixopolitan=> Minos	2018-04-24 13:23:39.
R-000147	closed	[redacted]	[redacted]	services@gaia.fr		Ticket - Restauration donnée AS400 - ID 11906	2018-04-23 11:06:11.
R-000146	closed	[redacted]	[redacted]	services@gaia.fr		Demande pour une petite modif de programme UM9508	2018-04-17 17:25:29.
R-000145	closed	[redacted]	[redacted]	services@gaia.fr		Demande pour modification du PGM d'édition des factures Papier	2018-04-17 16:44:39.
R-000144	closed	[redacted]	[redacted]	services@gaia.fr		problème mise en ruche	2018-04-17 12:07:30.
R-000143	ongoing	[redacted]	Autre utilisateur	services@gaia.fr		RELANCES	2018-04-17 12:06:57.
R-000142	closed	[redacted]	[redacted]	services@gaia.fr		Ticket - Demande de restauration - ID 11859	2018-04-16 16:44:37.
R-000141	closed	[redacted]	[redacted]	services@gaia.fr		commande ecom - taxe DEEE	2018-04-13 13:39:16.
R-000140	closed	[redacted]	[redacted]	services@gaia.fr		plantage CRT_DALIM	2018-04-12 16:01:51.
R-000139	ongoing	[redacted]	[redacted]	services@gaia.fr		IBM PMR 35932,299,706 AUXRCT Problème de sauvegarde quotidienne	2018-04-10 12:28:11.
R-000138	closed	[redacted]	[redacted]	services@gaia.fr		Déplacer 2 jobs de QBATCHNUIT vers QBATCH	2018-04-04 14:53:14.
R-000137	closed	[redacted]	[redacted]	services@gaia.fr		Edition 1012 pages pour le service sociétaires	2018-03-30 12:17:41.
R-000134	closed	[redacted]	[redacted]	services@gaia.fr		Plantage travaux de nuit, SOLOEXPORT	2018-03-30 09:55:45.
R-000133	closed	[redacted]	[redacted]	services@gaia.fr		Edition des étiquettes marquage	2018-03-29 16:03:16.
R-000132	closed	[redacted]	GAIA_CS	services@gaia.fr		détermination chop caisse américaine pour article pixo et nsp	2018-03-29 13:55:02.
R-000131	closed	[redacted]	[redacted]	services@gaia.fr		Problème bon d'impression Premium	2018-03-27 17:41:23.
R-000130	closed	[redacted]	[redacted]	services@gaia.fr		Bon Chop Premium - règle de calcul	2018-03-27 11:46:16.
R-000128	ongoing	[redacted]	GAIA_CS	services@gaia.fr		[redacted]	2018-03-23 10:24:15.
R-000127	ongoing	[redacted]	GAIA_CS	services@gaia.fr		Edition Rappels avec code OMR (mensuel)	2018-03-23 10:18:36.
R-000126	ongoing	[redacted]	GAIA_CS	services@gaia.fr		Réédition relevés de bases et envoi par mail	2018-03-23 10:15:50.

Prédicats

IS (NOT) JSON

- Détermine si une valeur est au format JSON

  - Syntaxe



IS (NOT) JSON

- Remarques sur les attributs
  - Format
    - FORMAT JSON (défaut)
      - Si string-expression est binaire, la valeur est interprétée en UTF-8 ou UTF-16
  - Type de valeur
    - VALUE
      - N'importe quelle valeur JSON valide de type ARRAY, OBJECT ou SCALAR
    - ARRAY
      - Une valeur JSON représentant un tableau
    - OBJECT
      - Une valeur JSON représentant un objet
    - SCALAR
      - Valeur alpha, numérique, ou les valeurs literals null, true ou false

IS (NOT) JSON

- Exemples

```
SELECT CASE WHEN donnee IS JSON then 'JSON' ELSE 'AUTRE' END,
       a.*
FROM nb.auditjson a ;
```

00001	ID	JOCODE	JOENTT	DONNEE
JSON	1	J	PR	{"2787277 ":"QSYS ","3024100 ":"QSYS "}
JSON	21	T	LD	{"18":"QTMHHTTP","38":"QTMHHTTP","58":"QTMHHTTP","78":"QTMH
JSON	22	T	OM	{"17":"QTMHHTTP","37":"QTMHHTTP","57":"QTMHHTTP","77":"QTMH
JSON	23	T	SG	{"132":"QSYS","133":"QSYS","134":"QSYS","135":"QSYS","136":
JSON	24	T	SK	{"131":"QSYS","137":"QSYS","407":"QSYS","408":"QSYS","410":
JSON	25	T	SV	{"8":"QSECOFR","769":"QSECOFR","836":"QSECOFR"}
AUTRE	26	X	X1	*** Erreur ***
AUTRE	27	X	X2	OK
AUTRE	28	X	X3	{OK}
AUTRE	29	X	X4	{"OK"}

JSON_EXISTS

- Permet de tester l'existence d'un élément JSON (sa clé)

  - Syntaxe

```
>>-JSON_EXISTS--(--json-expression--+-------------+--,---------->
                                   +-FORMAT JSON-+
                                   '-FORMAT BSON-'

>--sql-json-path-expression--+---------------+----------------->
                             '-AS--path-name-'

   .-FALSE ON ERROR--------.
>--+-----------------------+--)------------------------------><
   '-+-TRUE----+--ON ERROR-'
     +-UNKNOWN-+
     '-ERROR---'
```

JSON_EXISTS

- Remarques sur les attributs
  - Format
    - FORMAT JSON (défaut)
      - Si string-expression est binaire, la valeur est interprétée en UTF-8 ou UTF-16
  - sql-json-path-expression
    - Expression retournant une valeur de type caractère
  - Erreur
    - FALSE ON ERROR (défaut)
      - Retourne faux en cas d'erreur encountered
    - TRUE ON ERROR
      - Retourne vrai en cas d'erreur
    - UNKNOWN ON ERROR
      - Résultat inconnu en cas d'erreur
    - ERROR ON ERROR
      - Provoque une erreur en cas d'erreur

JSON_EXISTS

- Exemples

```
select *
from nb.auditjson a
where json_exists (donnee, '$."137"');
```

ID	JOCODE	JOENTT	DONNEE
24	T	SK	{"131":"QSYS","137":"QSYS","407":"QSYS","408":"QSYS","410":"QSYS"}

```
select *
from nb.auditjson a
where json_exists (donnee, '$.Utilisateur');
```

ID	JOCODE	JOENTT	DONNEE
30	Z	ZZ	{"horodatage":"2018-01-10-22.02.04.721552","Système":"NEPTUNE ","Utilisateur":"QSYS "}
31	Z	ZZ	{"horodatage":"2018-01-10-22.02.04.781152","Système":"NEPTUNE ","Utilisateur":"QSECOFR "}
32	Z	ZZ	{"horodatage":"2018-01-10-22.02.04.787472","Système":"NEPTUNE ","Utilisateur":"QSECOFR "}
33	Z	ZZ	{"horodatage":"2018-01-10-22.02.04.787696","Système":"NEPTUNE ","Utilisateur":"QPGMR "}

JSON_EXISTS

- Utilisation JSON_EXISTS (clé) et JSON_VALUE (valeur) ensemble

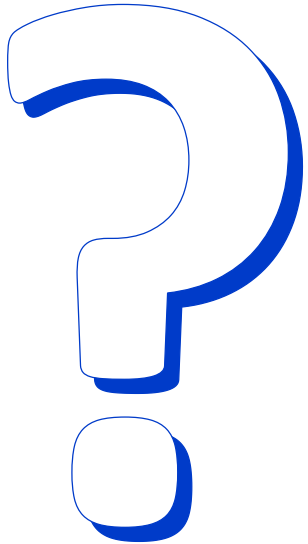
```
select *
from nb.auditjson a
where JSON_EXISTS(donnee, '$."Utilisateur"') and
      JSON_VALUE(donnee, '$.Utilisateur' RETURNING CHAR(10))
        not in ( 'QSYS', 'QPGMR', 'QSECOFR', 'QTMHHTTP', 'QTCP');
```

ID	JOCODE	JOENTT	DONNEE
11420	Z	ZZ	{"horodatage":"2018-01-10-22.08.07.207520","Système":"NEPTUNE ","Utilisateur":"CTL4I "}
11430	Z	ZZ	{"horodatage":"2018-01-10-22.08.18.110800","Système":"NEPTUNE ","Utilisateur":"*NONE "}
11440	Z	ZZ	{"horodatage":"2018-01-10-22.08.20.321376","Système":"NEPTUNE ","Utilisateur":"CTL4I "}
11460	Z	ZZ	{"horodatage":"2018-01-10-22.08.29.362752","Système":"NEPTUNE ","Utilisateur":"CTL4I "}

Références

- IBM
  - https://www.ibm.com/support/knowledgecenter/fr/ssw_ibm_i_73/sqlp/rbafyjson.htm
  - https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/IBM%20i%20Technology%20Updates/page/JSON%20Publishing%20Functions
  - https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/IBM%20i%20Technology%20Updates/page/JSON%20Scalar%20Functions
  - https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/IBM%20i%20Technology%20Updates/page/JSON_TABLE%20table%20function
- Articles
  - https://www.ibm.com/developerworks/ibmi/library/i-json-table-trs/index.html
- Open Source
  - https://github.com/BirgittaHauser/Generate-XML-and-JSON

Q/R



Merci de votre attention