# JSON

DB2 for i

# 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

# Syntaxe

- Sous la forme
  - Nom: valeur
- Les valeurs peuvent être
  - Objet
  - Tableau
  - Booléen, nombre, chaine ou null
- Les commentaires ne sont pas admis

- Référence
  - https://tools.ietf.org/html/rfc7159

# Exemple

**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" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

# DB2 for i

- La TR2 de la 7.2 (TR10 pour la 7.1) apporte un début de support de JSON dans DB2
  - En Technology Preview

- Le produit « DB2 NoSQL » supporte plusieurs interfaces
  - Ligne de commande
    - Sur l'IBM i avec QSH, ou sur une machine distante
  - Java
    - Via API
  - DB2
    - Fonctionnalité partielle via une fonction SQL

# Installation

- Le produit est disponible ici

  `/QIBM/ProdData/OS/SQLLIB`

  - Contient un répertoire bin avec les exécutables
- Une installation est nécessaire côté IBM i
  - Sous QSH

    `/QIBM/ProdData/OS/SQLLIB/bin/db2nosql -setup enable`

```
IBM DB2 NoSQL JSON API 1.1.0.0 version 1.4.8
Licensed Materials - Property of IBM
(c) Copyright IBM Corp. 2013,2015 All Rights Reserved.

ExÃ©cution SQL...
CDJSN1209I La crÃ©ation des artefacts de la base de donnÃ©es a abouti.
```

Le produit ne semble pas prévu pour gérer les caractères accentués …

  - Cela créé un ensemble de fonctions dans SYSTOOLS

# Possibilités

- En ligne de commande
  - Créer des collections JSON
    - Une table contenant une colonne de type BLOB
  - Insérer une document JSON dans une collection JSON
  - Retrouver un document JSON
- Via DB2 SQL
  - Convertir une document JSON (stocké en BLOB) en caractères
    - Fonction SYSTOOLS.BSON2JSON()

# Ligne de commande

# Ligne de commande

- IBM i : QSH
  - Vous pouvez aussi

    Copier le répertoire `/QIBM/ProdData/OS/SQLLIB` sur votre poste de travail

    Et lancer `db2nosql` pour Linux et Unix

    `db2nosql.bat` pour Windows
- Fonctions supportées
  - Démarrer db2nosql

```
db2nosql
```

  - Démarrer db2nosql avec un autre profil sur IBM i

```
db2nosql -user utilisateur motdepasse
```

  - Démarrer db2nosql avec un autre profil sur une autre plateforme

```
db2nosql -hostname NomAs400 -user utilisateur –password motdepasse
db2nosql -url jdbc:as400:NomAs400 -user utilisateur –password motdepasse
```

  - Aide sur la syntaxe db2nosql

```
db2nosql -help
```

# db2nosql

- Une fois `db2nosql` lancé
  - `help` affiche la liste des commandes supportées par l'interpréteur

```
Exemples de commande :
db.friends.insert({name:"Joe", age:5})       [InsÃ©rer dans la collection '"friends"'.]
db.friends.remove()                    [Supprimer toutes les lignes de la collection '"friends
db.friends.find({name:"Joe"})                [Trouver des amis dont le nom est '"Joe"'.]
db.friends.find({age:{$gt:5}})       [Trouver des amis dont l'Ã¢ge est supÃ©rieur Ã  "5".]
db.friends.find().sort({name:1})                        [Trier par nom, par ordre croissant.]
db.friends.find().sort({name:-1}).limit(5)              [Trier par nom, par ordre dÃ©croissant,
]
db.friends.find().sort({name:-1}).limit(5).skip(10)   [Trier par nom, par ordre dÃ©croissant,
 ignorer les "10" premiÃ¨res lignes.]
db.friends.importFile("C:\\myfriends.js")         [Importation de masse dans la collection]
quit    [Quitter l'interprÃ©teur de commandes.]

Liste des commandes :
 -- Commandes gÃ©nÃ©rales      Entrez 'help <mÃ©thode>' pour des informations dÃ©taillÃ©es. N
    debug                disable              enable
    help                 use                  show
```

# db2nosql

- Principales commandes
  - `db.sqlUpdate("instruction SQL")`

    Exécute l'instruction SQL

```
  nosql>
> db.sqlUpdate("create schema jsontst")
  db.sqlUpdate("create schema jsontst")
  0
```

  - `use nomBibliothèque`

    Indique le schéma (collection) en cours

```
  nosql>
> use jsontst
  use jsontst
  CDJSN1214I Passage au schÃ©ma "JSONTST" rÃ©alisÃ©
```

# db2nosql

- `db.createCollection("Nom")`

  Créer une collection JSON

```
  nosql>
> db.createCollection("JClient")
  db.createCollection("JClient")
  CDJSN1006I Collection : "db.JSONTST."JClient"" crÃ©Ã©e. Utilisez "JClient".
```

C'est-à-dire une table « `JClient` » dans la bibliothèque identifiée par la bd en cours, et contenant 3 colonnes

| | |
|---|---|
| `ID : CHAR(12) for bit data` | → clé |
| `DATA : BLOB` | → le document JSON |
| `ROWCREATED : TIMESTAMP` | → horodatage de création |

- Remarque

  Noms sensibles à la casse !

# db2nosql

- db.*NomCollectionJSON*.insert(...)

  Insérer un document JSON

```
nosql>
> db.JClient.insert({id:123, raisonsoc:"IBM", satut:"actif", datcrt:20150731})
db.JClient.insert({id:123, raisonsoc:"IBM", satut:"actif", datcrt:20150731})
CDJSN1000I La commande a abouti.
```

  Le document JSON doit être syntaxiquement valide

```
nosql>
> db.JClient.insert({id:123, raisonsoc:IBM", satut:"actif", datcrt:20150731})
db.JClient.insert({id:123, raisonsoc:IBM", satut:"actif", datcrt:20150731})
CDJSN0114E Syntaxe de la commande non valide
Les dÃ©tails de l'erreur sont affichÃ©s ci-dessous, mais ils ne sont peut-Ãªtre pas la source du problÃ¨me.
db.JClient.insert({id:123, raisonsoc:IBM", satut:"actif", datcrt:20150731})
                                           ^
DÃ©tails de l'erreur : il manque '}' après une liste de propriétés
NumÃ©ro de colonne : 50
```

  L'insertion créé un enregistrement dans la table correspondante

```
select * from jsontst."JClient" ;
```

| ID | DATA | ROWCREATED |
|---|---|---|
| í\|ùOß»0□□0]]m | 033D0000001069640078000000027261697 36F6E736F6300040000000494 24D... | 2015-07-31 14:32:44.245282878906 |

# db2nosql

- De nombreuses commandes de manipulation sont disponibles
  - Recherche totale ou partielle
  - Liste
  - Modification, suppression
  - …
- L'outil est documenté
  - Sous qsh

    ```
    db2nosql puis help
    ```

  - Référence developerWorks

    https://www.ibm.com/developerworks/data/library/techarticle/dm-1306nosqlforjson2/
- API Java également disponible

  https://www.ibm.com/developerworks/data/library/techarticle/dm-1307nosqlforjson3/

# SQL

# Fonctions disponibles

| 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<br>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 | **Technology Preview**<br>Permet de trouver le rang d'un élément dans le tableau JSON |
| JSON_UPDATE | **Technology Preview**<br>Permet de mettre à jour une partie du document JSON |

# 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 BLOB contenant un document JSON en chaine de caractères
  - 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":201
50824}'                          )
       )  ;
```

# JSON2BSON

- C’est l’opération inverse de BSON2JSON

- Exemple

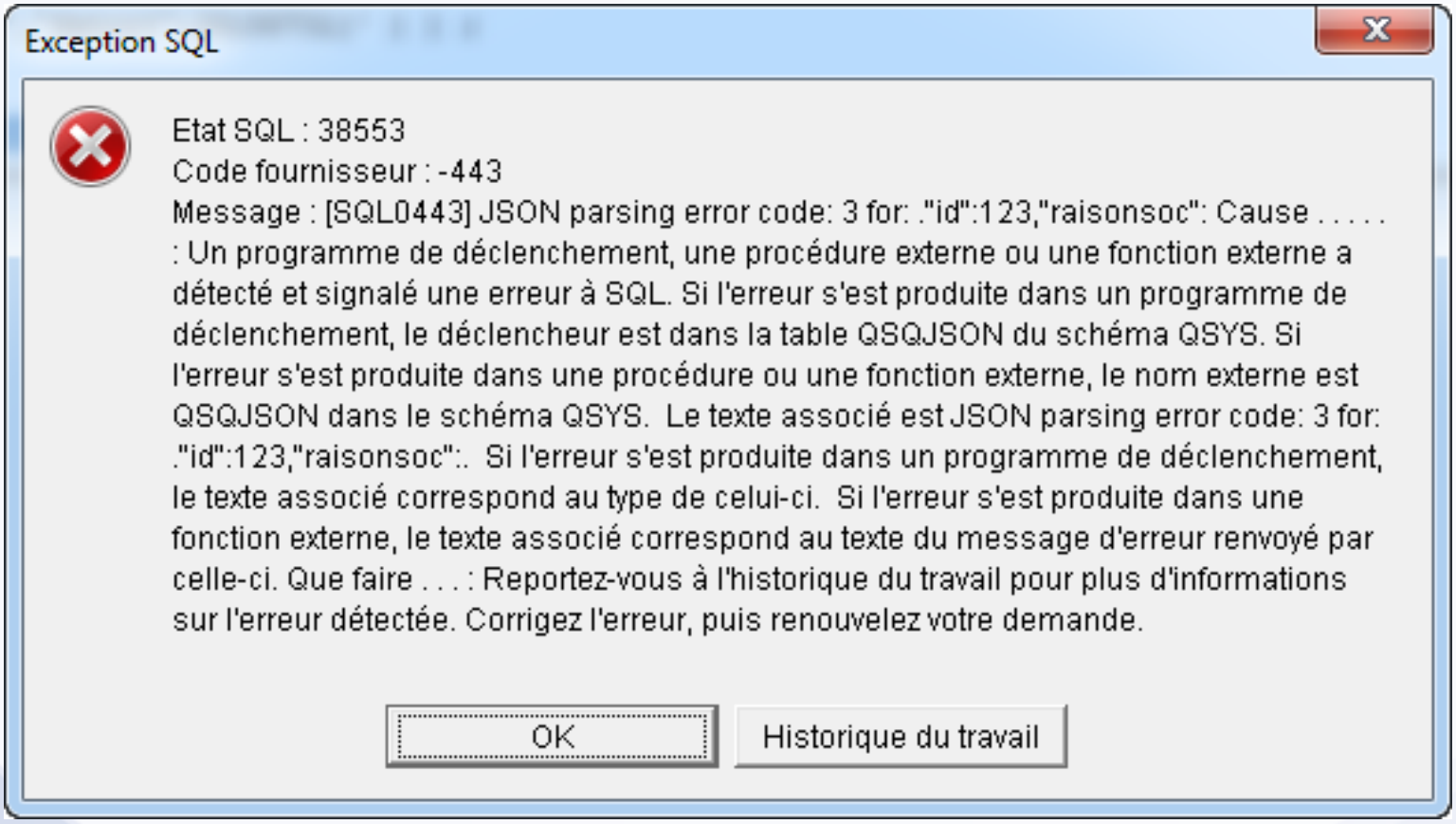
```
values systools.bson2json(
   systools.json2bson(
'{"id":123,"raisonsoc":"IBM","satut":"actif","datcrt":2
0150731}'            )   ) ;
```

- Produit

```
{"id":123,"raisonsoc":"IBM","satut":"actif","datcrt":20
150731}
```

# 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":2015
0731}' ) ) ;

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_TABLE et JSON_TABLE_BINARY

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

    Entrée :

    | | |
    |---|---|
    | JSON | BLOB (16Mo) |
    | Élément | VARCHAR(2048) |
    | Type de retour | VARCHAR(100) |

    Sortie :

    | | |
    |---|---|
    | Type | INTEGER |
    | Valeur | VARCHAR(2048) |

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

# JSON_TABLE et JSON_TABLE_BINARY

- Exemple

```
select j.*
from jsontst."JClient" ,
     TABLE( json_table( data, 'id', 'i' )) as j ;
```

- Produit

| TYPE | VALUE |
|---|---|
| 16 | 123 |
| 16 | 456 |

# 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 |

# 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) |
    | Longeur 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□□□□□□□□□□... |

# Usage

- JSON s'est imposé comme un langage d'échange de données, au même titre que XML
  - Devient incontournable dans les échanges automatisés, comme avec les services web par exemple
- Nous disposons maintenant d'un outillage SQL qui nous permet d'effectuer des traitements comparables avec le stockage XML natif
- Il est bien sur possible, et fort utile, d'exploiter ces fonctions SQL dans les programmes RPG avec SQL embarqué