Processing JSON with SQL
Speaker: Paul Tuohy

Simplifying IBM i Application Management with X-Analysis
Speaker: Ray Everhart

Paul’s handout at
ray.Everhart@freschesolutions.com

See more Summit Lunch & Learn webinars at SystemiDeveloper.com/LunchLearn
Processing JSON with SQL
Paul Tuohy, author of "Re-engineering RPG Legacy Applications" and "The Programmer's Guide to iSeries Navigator", is one of the most prominent consultants and trainer/educators for application modernization and development technologies on the IBM Midrange. He currently holds positions as CEO of ComCon, a consultancy firm based in Dublin, Ireland, and founding partner of System i Developer, the consortium of top educators who produce the acclaimed RPG & DB2 Summit conference. Previously, he worked as IT Manager for Kodak Ireland Ltd. and Technical Director of Precision Software Ltd.

In addition to hosting and speaking at the RPG & DB2 Summit, Paul is an award-winning speaker at COMMON, COMMON Europe Congress and other conferences throughout the world. His articles frequently appear in iProDeveloper, The Four Hundred Guru, RPG Developer and other leading publications. Paul also hosts the popular iTalk with Tuohy podcast interviews.

This presentation may contain small code examples that are furnished as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. We therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All code examples contained herein are provided to you "as is". THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.
Agenda

Sample JSON

Consuming JSON
  ▶ The JSON_TABLE Table Function
  ▶ JSON_TABLE Examples
  ▶ Generate Column List
  ▶ Get an IFS File with SQL
  ▶ JSON Processing with Embedded SQL

Generating JSON
  ▶ The JSON_OBJECT and JSON_ARRAY Functions
  ▶ JSON_OBJECT and JSON_ARRAY Examples
  ▶ Generate Column List
  ▶ Get an IFS File with SQL
  ▶ JSON Processing with Embedded SQL
```json
{
  "order_id":10248,
  "customer_id":"VINET",
  "employee_id":5,
  "order_date":"2021-07-04",
  "required_date":"2021-08-01",
  "order_details": [
    { "product_id":11, "unit_price":14.0000, "quantity":12, "discount":0 },
    { "product_id":42, "unit_price":9.8000, "quantity":10, "discount":0 },
    { "product_id":72, "unit_price":34.8000, "quantity":5, "discount":0 }
  ]
}
```
```json
{
  "agent_Id": "AMAZON",
  "batch_Id": "AMZ0001",
  "order_count": 2,
  "orders": [
    {
      "order_id": 10248,
      "customer_id": "VINET",
      "employee_id": 5,
      "order_date": "2021-07-04",
      "required_date": "2021-08-01",
      "order_details": [
        {"product_id": 11, "unit_price": 14.0000, "quantity": 12, "discount": 0},
        {"product_id": 42, "unit_price": 9.8000, "quantity": 10, "discount": 0},
        {"product_id": 72, "unit_price": 34.8000, "quantity": 5, "discount": 0}
      ]
    },
    {
      "order_id": 10249,
      "customer_id": "TOMSP",
      "employee_id": 6,
      "order_date": "2021-07-05",
      "required_date": "2021-08-16",
      "order_details": [
        {"product_id": 14, "unit_price": 18.6000, "quantity": 9, "discount": 0},
        {"product_id": 51, "unit_price": 42.4000, "quantity": 40, "discount": 0}
      ]
    }
  ]
}
```
Consume JSON with SQL
The JSON_TABLE Table Function

Returns a result table from the evaluation of SQL/JSON path expressions

- Each item in the result sequence of the row SQL/JSON path expression represents one or many rows in the result table

```
JSON_TABLE ( JSON-expression
, FORMAT JSON
, FORMAT BSON
, COLUMNS
, AS path-name
, json-table-regular-column-definition
, json-table-formatted-column-definition
, json-table-ordinality-column-definition
, json-table-nested-column-definition
, EMPTY ON ERROR
, ERROR ON ERROR
)
```

© ComCon & System i Developer, LLC 2005-2023
JSON_TABLE function consists of three parts.

- The JSON object to be deconstructed
- A path expression that generates zero or more rows from the JSON object
- The definition of the result columns to be returned. This includes
  - The column name
  - The result data type
  - A path expression to use to locate the column information

JSON_TABLE is a table function

- Used in place of a table name in a SELECT statement

```sql
JSON_TABLE(json_object, path_expression, result_columns)
```
```sql
select *
from json_table(singleorder, '$_'
  COLUMNS (  
    order_id DECIMAL(11, 0) PATH '.$.order_id',
    customer_id CHAR(5)        PATH '.$.customer_id',
    employee_id DECIMAL(11, 0) PATH '.$.employee_id',
    order_date DATE           PATH '.$.order_date',
    required_date DATE         PATH '.$.required_date',
    NESTED PATH '.$.order_details'
      COLUMNS (  
        product_id DECIMAL(11, 0) PATH '.$.product_id',
        unit_price DECIMAL(19) PATH '.$.unit_price',
        quantity DECIMAL(7)  PATH '.$.quantity',
        discount DECIMAL(3)  PATH '.$.discount'
      )
  )
) AS myJSON
```

© ComCon & System i Developer, LLC 2005-2023

Page 10 of 30
select * from json_table(batchorder, '$'  
COLUMNS (  agent_Id    char(6) PATH '$.agent_Id',  
batch_Id    char(7) PATH '$.batch_Id',  
order_count integer PATH '$.order_count',  
NESTED PATH '$.orders'  
COLUMNS (  order_id         DECIMAL(11, 0) PATH '$.order_id',  
customer_id      CHAR(5)        PATH '$.customer_id',  
employee_id      DECIMAL(11, 0) PATH '$.employee_id',  
order_date       DATE           PATH '$.order_date',  
required_date    DATE           PATH '$.required_date',  
NESTED PATH '$.order_details'  
COLUMNS (  product_id DECIMAL(11, 0) PATH '$.product_id',  
unit_price DECIMAL(19, 4) PATH '$.unit_price',  
quantity DECIMAL(7, 0) PATH '$.quantity',  
discount DECIMAL(3, 1) PATH '$.discount')
)
) AS myJSON ;
Generate Column List

Generating a column list from the system catalog

```sql
-- Get the longest name length - this example returns 16
select max(length(column_name))
from qsys2.syscolumns
where (table_schema, table_name) = ('JONSAMPLE', 'ORDERS');

select rpad(lower(column_name), 17) concat
  rpad( case when data_type in ('DATE','TIME','TIMESTAMP') then data_type
  when data_type in ('NUMERIC','DECIMAL')
  then 'DECIMAL(' concat length
      concat ', '
      concat numeric_scale
      concat ')
  else data_type concat '(' concat length concat ')' end
   , 17)
  , ' PATH ''$.' concat lower(column_name) concat '''
from qsys2.syscolumns
where (table_schema, table_name) = ('JONSAMPLE', 'ORDERS')
order by ordinal_position;
```
Getting an IFS File in SQL

Use the GET_CLOB_FROM_FILE() function

When using an SQL Client (like Run SQL Scripts)

- Global variables are your friend
- Or use the function directly in the SELECT statement
  - Less legible

```
create variable singleorder varchar(32000);
create variable batchorder varchar(32000);
```

```
set singleorder = qsys2.GET_CLOB_FROM_FILE('/home/paris/jsonstuff/singleorder.json');
set batchorder = qsys2.GET_CLOB_FROM_FILE('/home/paris/jsonstuff/batchorder.json');
```
Getting an IFS File with Embedded SQL

dcl-s gv_ifs_File SQLType(CLOB_FILE) ccsid(*utf8);

dcl-proc du_get_ifsFile export;
  dcl-pi *n varChar(32000) ccsid(*utf8);
    fileName varChar(250) const;
  end-Pi;

  dcl-s fileContent varChar(32000) ccsid(*utf8);

  gv_ifs_File_FO   = SQFRD;
  gv_ifs_File_NAME = fileName;
  gv_ifs_File_NL   = %len(fileName);

  exec SQL
    values :gv_ifs_File into :fileContent;

  return fileContent;
end-Proc;
dcl-s gv_ifs_File SQLType(CLOB_FILE) ccsid(*utf8);

dcl-proc du_write_ifsFile export;
  dcl-pi *n 
    fileName   varchar(250) const;
    fileContent varchar(32000) ccsid(*utf8) const;
  end-Pi;

  gv_ifs_File_FO   = SQFOVR;
  gv_ifs_File_NAME = fileName;
  gv_ifs_File_NL   = %len(fileName);

  exec SQL
    values :fileContent into :gv_ifs_File;

  return fileContent;
end-Proc;
merge into order_details as OLD
  using (select *
    from json_table(singleorder, '$'
      COLUMNS (  
        order_id        DECIMAL(11, 0) PATH '$.order_id',
        NESTED PATH '$.order_details'
        COLUMNS (  
          product_id DECIMAL(11, 0) PATH '$.product_id',
          unit_price DECIMAL(19) PATH '$.unit_price',
          quantity   DECIMAL(7) PATH '$.quantity',
          discount   DECIMAL(3) PATH '$.discount'
        )
      )
    ) AS myJSON) as NEW
on (old.order_id, old.product_id) = (new.order_id, new.product_id)
when NOT MATCHED then
  insert (order_id, product_id, unit_price, quantity, discount)
  values (new.order_id, new.product_id, new.unit_price, new.quantity, new.discount)
when MATCHED then
  update set (unit_price, quantity, discount) = (new.unit_price, new.quantity, new.discount);

With MERGE

Most powerful when used to directly manipulate database from JSON

- Example updates or inserts rows in the ORDER_DETAILS table
In RPG - Declare Host Variable and Get JSON

```rpg
  dcl-ds order_T extName('JONSAMPLE/ORDERS') alias template qualified end-ds;
  dcl-ds order_details_T extName('JONSAMPLE/ORDDETAIL') alias template qualified end-ds;

  dcl-ds order qualified;
    order_id   like(order_T.order_id);
    customer_id like(order_T.customer_id);
    employee_id like(order_T.employee_id);
    order_date  like(order_T.order_date);
    required_date like(order_T.required_date);
    num_order_details int(5);
  end-ds;

  dcl-ds order_details dim(99) qualified;
    product_id   like(order_details_T.product_id);
    unit_price    like(order_details_T.unit_price);
    quantity     like(order_details_T.quantity);
    discount     like(order_details_T.discount);
  end-ds;

  dcl-s myJSON varchar(32000) ccsid(*utf8);

  myJSON = du_get_ifsFile('/home/paris/jsonstuff/singleorder.json');
```
exec SQL
  select order_id, customer_id, employee_id, order_date, required_date, 0
  into :order
  from json_table(:myJSON, '$'
      COLUMNS (  order_id         DECIMAL(11, 0) PATH '$.order_id',
                 customer_id      CHAR(5)        PATH '$.customer_id',
                 employee_id      DECIMAL(11, 0) PATH '$.employee_id',
                 order_date       DATE           PATH '$.order_date',
                 required_date    DATE           PATH '$.required_date'
      )
  ) AS myJSON ;

exec SQL
  declare get_details scroll cursor for
  select product_id, unit_price, quantity, discount
  from json_table(:myJSON, '$.order_details'
      COLUMNS (  product_id DECIMAL(11, 0)    PATH '$.product_id',
                 unit_price DECIMAL(19, 4)    PATH '$.unit_price',
                 quantity   DECIMAL(7, 0)     PATH '$.quantity',
                 discount   DECIMAL(3, 1)     PATH '$.discount'
      )
  ) AS myJSON ;

exec SQL
  open get_details;

exec SQL
  fetch first from get_details for 99 rows into :order_details;

order.num_order_details = SQLERRD(3);

exec SQL
  close get_details;
Processing JSON with SQL
Speaker: Paul Tuohy

Simplifying IBM i Application Management with X-Analysis
Speaker: Ray Everhart

Paul’s handout at
ray.Everhart@freschesolutions.com

See more Summit Lunch & Learn webinars at
SystemiDeveloper.com/LunchLearn
Generate JSON with SQL
JSON Functions

Main functions (scalar)
- `JSON_OBJECT`
- `JSON_ARRAY`

Can also make use of (scalar)
- `JSON_QUERY`
- `JSON_TO_BSON`
- `JSON_VALUE`

Aggregate Functions
- `JSON_OBJECTAGG`
- `JSON_ARRAYAGG`
The JSON_OBJECT Scalar Function

Generates a JSON object using the specified *key:*value pairs

- If no *key:*value pairs are provided, an empty object is returned
The JSON_ARRAY Scalar Function

Generates a JSON array

- By either explicitly listing the array elements or by using a query
- If no JSON-expression is provided, the fullselect returns no values
- If all values are null and ABSENT ON NULL is specified, an empty array is returned
select json_object(
    'order_id'      : order_id,
    'customer_id'   : customer_id,
    'employee_id'   : employee_id,
    'order_date'    : order_date,
    'required_date' : required_date,
    'order_details' : json_array( (select json_object(
        'product_id'   : product_id,
        'unit_price'   : unit_price,
        'quantity'     : quantity,
        'discount'     : discount
    )
    from order_details
    where order_id = 10248 ))
) format json
from orders
where order_id = 10248;
values json_object(
    'agent_Id': 'AMAZON',
    'batch_Id': 'AMZ0001',
    'order_count': (select count(*) from orders where order_id in (10248, 10249)),
    'orders':
        json_array( (select json_object(
            'order_id'      : order_id
            ,'customer_id'   : customer_id
            ,'employee_id'   : employee_id
            ,'order_date'    : order_date
            ,'required_date' : required_date
            ,'order_details' :
                json_array( (select json_object(
                    'product_id'   : product_id
                    ,'unit_price'   : unit_price
                    ,'quantity'     : quantity
                    ,'discount'     : discount
                    )
                    from order_details
                    where order_id = 10248 )
                format json)
        from orders
        where order_id in (10248, 10249)) format json))
);
Things to Watch Out For in the SQL

**FORMAT JSON**
- Required when nesting functions

**ABSENT ON NULL or NULL ON NULL**
- NULL ON NULL is the default

**WITHOUT UNIQUE KEYS or WITH UNIQUE KEYS**
- WITHOUT UNIQUE KEYS is the default
Things to Watch Out For in RPG Programs

The host variable (to receive the JSON) should be defined as UTF8

- The JSON scalar functions return a UTF8 value

If the host variable is greater than 32K, it must be defined as a CLOB

- This is an SQL limitation, not an RPG limitation

```sql
DCL-S gen_JSON SQLType(CLOB: 10000000) CCSID(*utf8);

// Results in
DCL-DS GEN_JSON;
   GEN_JSON_LEN UNS(10);
   GEN_JSON_DATA CHAR(10000000) CCSID(1208);
END-DS GEN_JSON;
```

```sql
DCL-S gen_JSON SQLType(CLOB: 10000000) CCSID(*utf8);
DCL-S gen_JSON_var varchar(10000000) CCSID(*utf8) based(gen_JSON_var_p);
DCL-S gen_JSON_var_p pointer inz(%addr(gen_JSON));

// Above definition is more efficient than
// gen_JSON_var = '';
// if (gen_JSON_len > 0);
//   gen_JSON_var = %subst(gen_JSON_data: 1: gen_JSON_len);
// endIf;
```
dcl-s gv_ifs_File SQLType(CLOB_FILE) ccsid(*utf8);

dcl-proc du_get_ifsFile export;
  dcl-pi *n varChar(32000) ccsid(*utf8);
    fileName varChar(250) const;
  end-Pi;

dcl-s fileContent varChar(32000) ccsid(*utf8);

  gv_ifs_File_FO = SQFRD;
  gv_ifs_File_NAME = fileName;
  gv_ifs_File_NL = %len(fileName);

  exec SQL
    values :gv_ifs_File into :fileContent;

  return fileContent;
end-Proc;
dcl-s gv_ifs_File SQLType(CLOB_FILE) ccsid(*utf8);

dcl-proc du_write_ifsFile export;
dcl-pi *n varchar(32000) ccsid(*utf8);
    fileName varchar(250) const;
    fileContent varchar(32000) ccsid(*utf8) const;
end-Pi;

gv_ifs_File_FO = SQFOVR;
gv_ifs_File_NAME = fileName;
gv_ifs_File_NL = %len(fileName);

exec SQL
    values :fileContent into :gv_ifs_File;

return fileContent;
end-Proc;
Simplifying IBM i Application Management with X-Analysis
Speaker: Ray Everhart

Processing JSON with SQL
Speaker: Paul Tuohy

See more Summit Lunch & Learn webinars at SystemiDeveloper.com/LunchLearn

Sponsored by Fresche Solutions

ray.Everhart@freschesolutions.com