Processing JSON with SQL

Speaker: Paul Tuohy

Simplifying IBM i Application Management with X-Analysis

Speaker: Ray Everhart

Paul's handout at

Sponsored by

FRESCHE SOLUTIONS

ray.Everhart@freschesolutions.com

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

Processing JSON with SQL

Paul Tuohy
ComCon
System *i* Developer
5, Oakton Court,
Ballybrack
Co. Dublin
Ireland

Phone: +353 1 282 6230
e-Mail: paul@systemideveloper.com
Web: www.systemideveloper.com
www.ComConAdvisor.com

IBM CHAMPION

System *i* Developer
The Summit of Education Excellence

ComCon

Paul Tuohy

System i Developer
The Summit of Education Excellence
ComCon

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

System i Developer
The Summit of Education Excellence
ComCon

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

Sample JSON - Simple


```
{   "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
        }
    ]
}
```

Sample JSON - Slightly More Complex


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

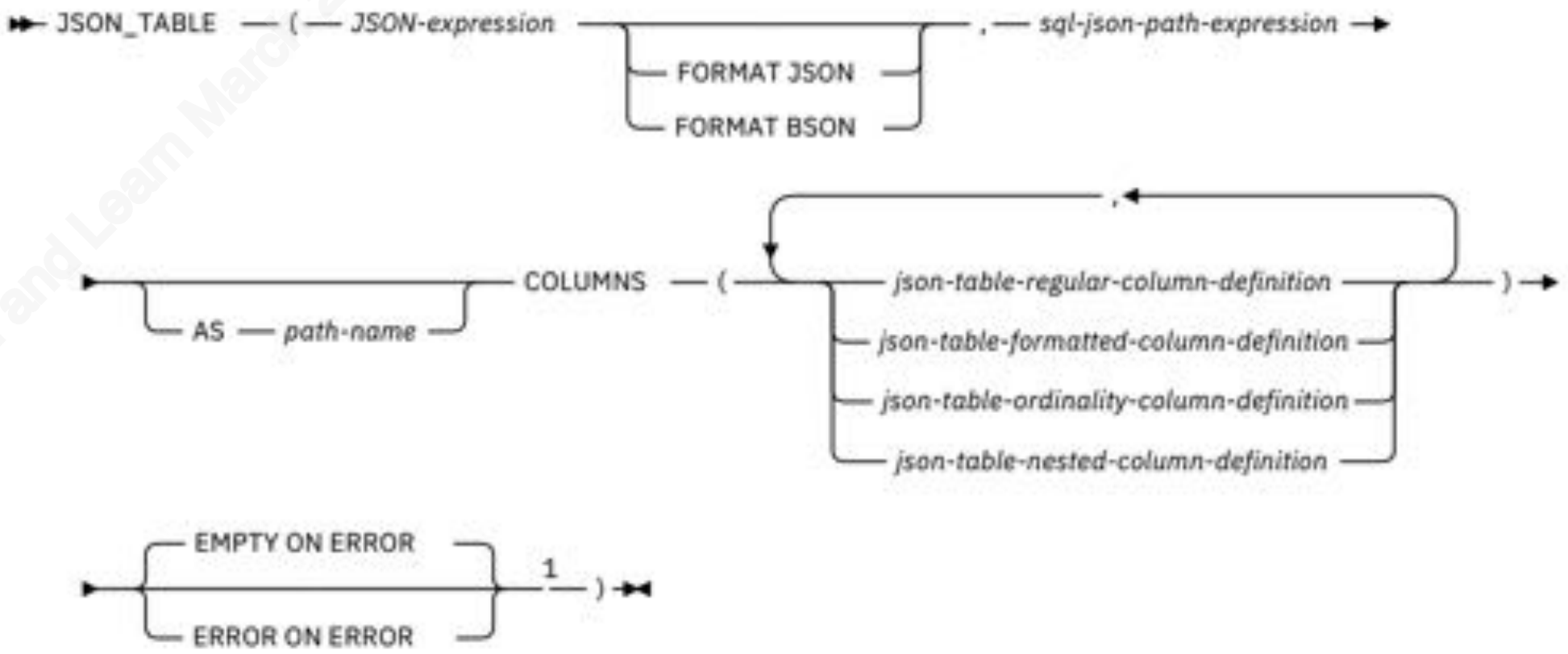
Consume JSON with SQL

The JSON_TABLE Table Function

System i Developer
The Summit of Education Excellence
ComCon

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_TABLE function consists of three parts.

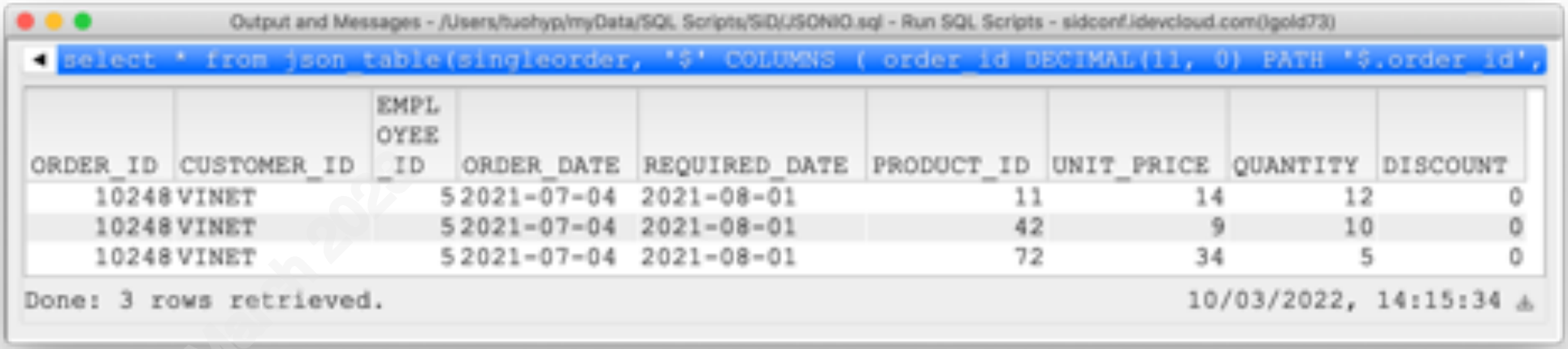
- The JSON object to be deconstructed
- A path expression that generates zero or more rows from the JSON object
- The definition of the result columns to be returned. This includes
  - The column name
  - The result data type
  - A path expression to use to locate the column information

JSON_TABLE is a table function

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

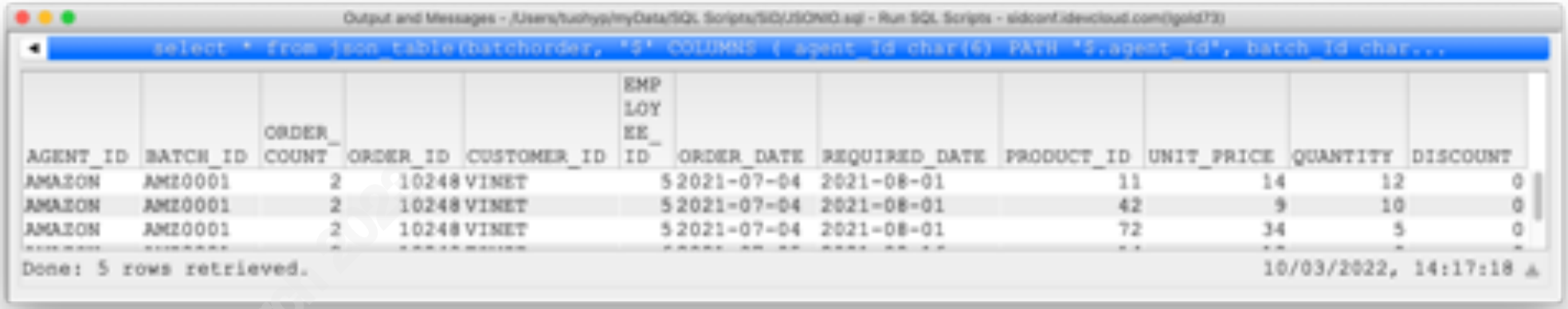

```
JSON_TABLE(json_object, path_expression, result_columns)
```

JSON_TABLE With Simple Sample




```
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 ;
```

JSON_TABLE With More Complex Sample



```
select * from json_table(batchorder, '$' COLUMNS ( agent_Id char(6) PATH '$.agent_Id', batch_Id char...
```

AGENT_ID	BATCH_ID	ORDER_COUNT	ORDER_ID	CUSTOMER_ID	EMPLOYEE_ID	ORDER_DATE	REQUIRED_DATE	PRODUCT_ID	UNIT_PRICE	QUANTITY	DISCOUNT
AMAZON	AMZ0001	2	10248	VINET	5	2021-07-04	2021-08-01	11	14	12	0
AMAZON	AMZ0001	2	10248	VINET	5	2021-07-04	2021-08-01	42	9	10	0
AMAZON	AMZ0001	2	10248	VINET	5	2021-07-04	2021-08-01	72	34	5	0

Done: 5 rows retrieved. 10/03/2022, 14:17:18

```
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

System i Developer
The Summit of Education Excellence
ComCon

Generating a column list from the system catalog

```
-- 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)
       concat ' 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;
```

Writing an IFS File with Embedded SQL


```
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;
```

With MERGE

Most powerful when used to directly manipulate database from JSON

- Example updates or inserts rows in the ORDER_DETAILS table

```
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);
```

In RPG - Declare Host Variable and Get JSON


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

Get Order and Order Details Seperately


```
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

Sponsored by

FRESCHE SOLUTIONS

ray.Everhart@freschesolutions.com

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

Generate JSON with SQL

JSON Functions

System i Developer
The Summit of Education Excellence
ComCon

Main functions (scalar)

- JSON_OBJECT
- JSON_ARRAY

Can also make use of (scalar)

- JSON_QUERY
- JSON_TO_BSON
- JSON_VALUE

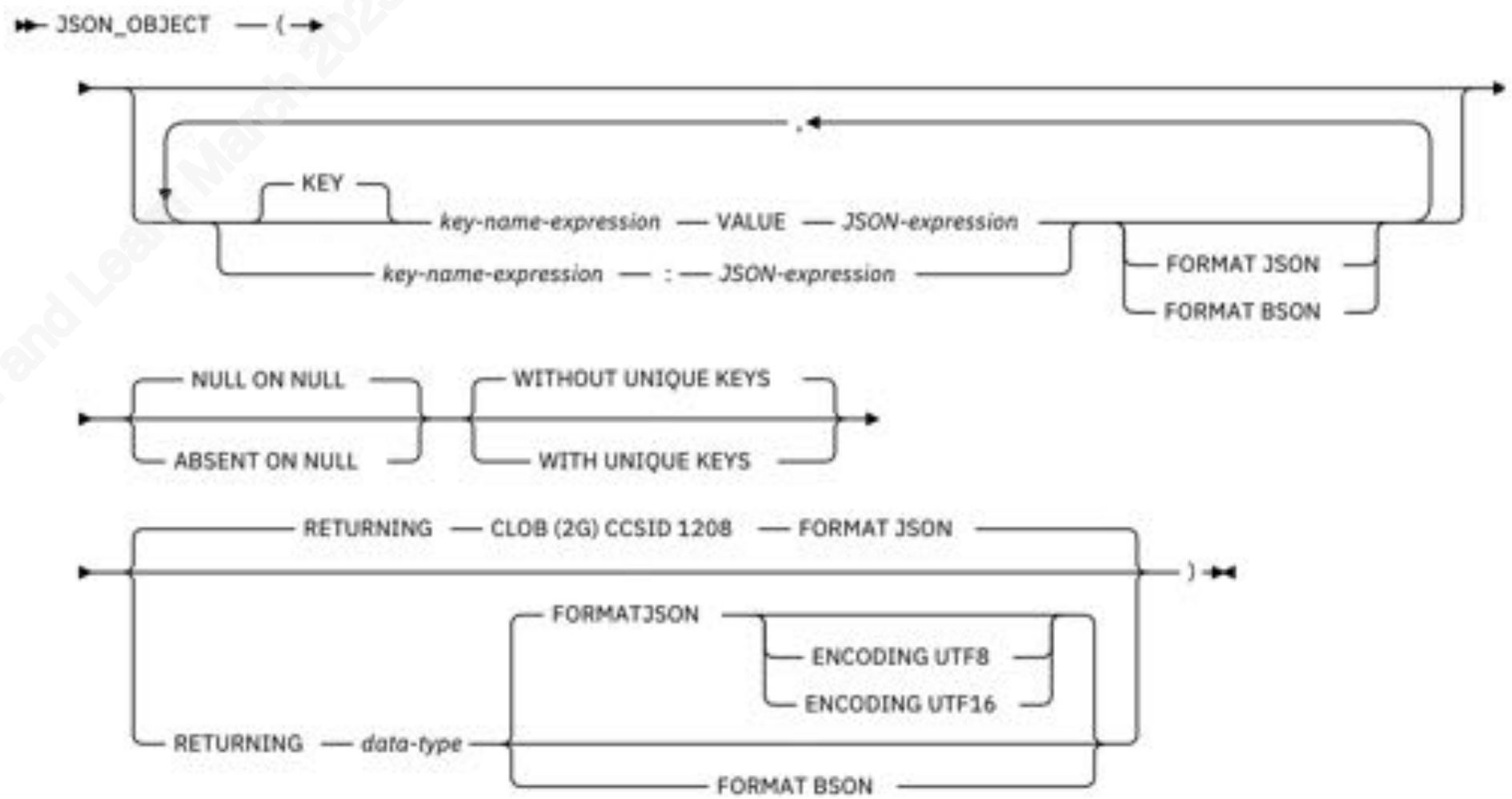
Aggregate Functions

- JSON_OBJECTAGG
- JSON_ARRAYAGG

The JSON_OBJECT Scalar Function

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

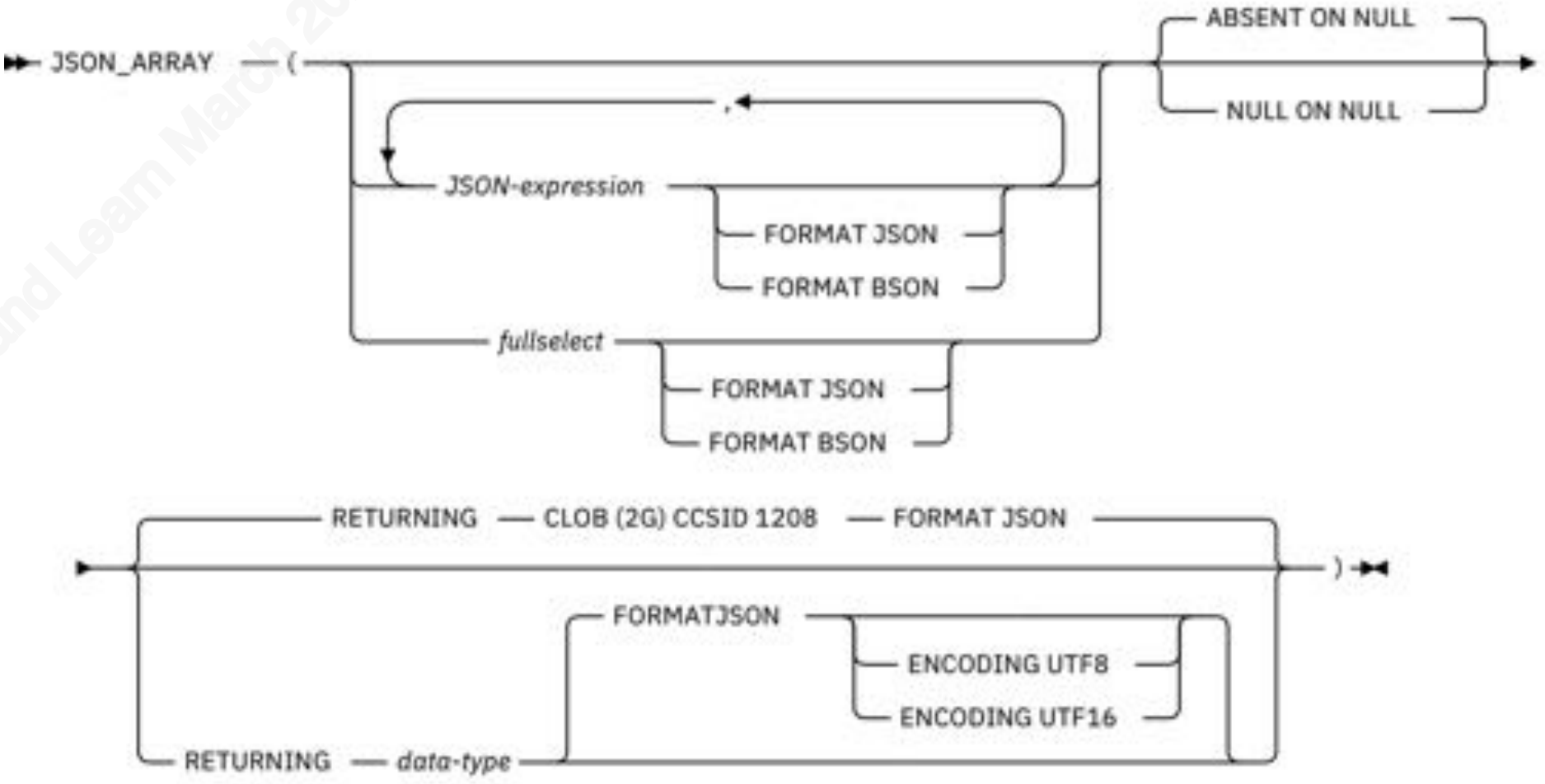
- If no *key:value* pairs are provided, an empty object is returned



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



Generating the Simple Sample


```
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;
```

Generating the More Complex Sample


```
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

```
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;
```

```
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 efficent than
// gen_JSON_var = '';
// if (gen_JSON_len > 0);
//   gen_JSON_var = %subst(gen_JSON_data: 1: gen_JSON_len);
// endIf;
```

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;
```

Writing an IFS File with Embedded SQL


```
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;
```

Processing JSON with SQL

Speaker: Paul Tuohy

Paul's handout at

Simplifying IBM i Application Management with X-Analysis

Speaker: Ray Everhart

Sponsored by

FRESCHE SOLUTIONS

ray.Everhart@freschesolutions.com

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