Actian Hybrid Data Conference
2017 London
Disclaimer

This document is for informational purposes only and is subject to change at any time without notice. The information in this document is proprietary to Actian and no part of this document may be reproduced, copied, or transmitted in any form or for any purpose without the express prior written permission of Actian.

This document is not intended to be binding upon Actian to any particular course of business, pricing, product strategy, and/or development. Actian assumes no responsibility for errors or omissions in this document. Actian shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials. Actian does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.
Plans and Possibilities for Ingres 11.1 and Beyond

Karl Schendel
Software Development Architect
Abstract

This session is a grab bag of planned capabilities for Ingres 11.1, and wild possibilities for the future. I'll talk a bit about what the international SQL standards committee has done and is doing, and how it applies to 11.1 and might apply to future work.
Short Biography

• Telling computers what to do since 1975
• Ingres user since 1990
• Ingres consultant 1997-2003
• Ingres (and Vector!) developer 2003-present
• Ingres/Vector Development team manager
The SQL Standard and Ingres/Vector

▪ Many core Ingres features don't agree with the Standard
  – Example: character string basics
▪ Ingres (and Vector) do adhere to the Standard in many ways
  – JOIN ... ON syntax
  – "ANSI" date/time types vs ingresdate
  – Window functions
  – Coercion semantics
  – Grouping sets
  – etc etc
▪ New features conform to the Standard as much as possible
The SQL Standard

▪ A National (ANSI) and International (ISO/IEC) Standard
▪ US: TAG (Technical Advisory Group) (DM32.2) members are companies, including Actian
▪ International: Working Group (JTC1 / SC 32 / WG 3) members are individual accredited experts
  – Working group recommendations are acted on at the SC level
  – SC votes are national body votes
▪ Actian reps to both US and ISO groups are Karl Schendel (primary), Gordy Thorpe (alternate)
The SQL Standard

- Nine parts of ISO/IEC 9075
  - Part 1: Framework (some definitions and terms)
  - Part 2: Foundation (the bulk of the Standard)
  - Part 4: PSM (Persistent Stored Modules)
  - Part 11: Schemata
  - Part 14: XML
  - and others of less relevance / interest: CLI, MED, OLB
  - Some parts no longer published

- Coming in 2018: Part 15: Multi-dimensional arrays
  - Currently in Draft International Standard (DIS) ballot stage
The SQL Standard

- New revisions published every few years
  - Most recent edition is SQL:2016
  - Technical Corrigenda (TC) published in between to fix errors
- Companion publications: Technical Reports 19075-nn
  - Non-normative, more descriptive material
  - 7 TR's published so far
  - MDA and OLAP (window function) TR's in progress
  - Freely available if you know the magic words:
    http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html
    and search for 19075
New in SQL:2016

- Lots and lots of bug fixes, presentation improvements
- Row Pattern Recognition (RPR)
  - Enhanced FROM clause to match for patterns across rows
  - Think of as a WHERE clause that works down rows rather than across columns
- Polymorphic Table Functions (PTF)
  - Functions whose input and return types need not be determined until query time
  - Very complicated and wildly flexible
- JSON support – the reason for this digression!
JSON Basics

- JSON = JavaScript Object Notation
- JSON values are numbers, strings, special words (true, false, null), objects or arrays
- JSON objects are lists of key:value pairs:
  ```
  { name: "Ingres 11.1", presenter: "Karl", duration : 45 }
  ```
- JSON arrays are ordered lists of values:
  ```
  [ "Karl", "schka01", 9111, { gender: "male", age: null }, true ]
  ```
- Arrays and objects are fully nestable in one another
- Inherently schema-less
JSON in SQL

- JSON columns are string valued columns
  - No separate data type as was done for XML

- Primary use cases:
  - JSON data ingestion and storage
    - column values are just character strings
  - JSON data generation from relational data
    - JSON constructor functions
  - Querying JSON as part of an SQL query
    - JSON query functions and path language
JSON Query Functions

- There are four primary JSON query functions
- JSON_EXISTS: test search condition on JSON value
- JSON_VALUE: extract SQL scalar from JSON value
- JSON_QUERY: extract JSON value from JSON value
- JSON_TABLE: generate "table" from JSON value
- There is also some additional SQL syntax such as "IS JSON"
Query Functions: JSON.EXISTS

- JSON.EXISTS tests whether a JSON value satisfies a search condition

```
SELECT T.thing
FROM T
WHERE JSON_EXISTS (T.Json, 'lax $.friends[*].rank')
```

- Quoted string in JSON.EXISTS is a *path expression*
  - describes what JSON values we're looking for
  - a "friends" member (key) that is an array of objects containing a "rank" key
  - can be "lax" (error tolerant) or "strict" (deviations or omissions cause error)
  - can pass in other column data using a PASSING clause, beyond the scope of this presentation
Query Functions: JSON_VALUE

- JSON_VALUE extracts an SQL scalar value from a JSON object
- Result can be used anywhere in the SQL query

```sql
SELECT T.thing,
    JSON_VALUE (T.Json, 'lax $.who') AS Who,
    JSON_VALUE (T.Json, 'lax $.friends[0].rank'
        RETURNING INTEGER NULL ON EMPTY) AS Rank
FROM T
WHERE JSON_EXISTS(T.Json, '.....')
    AND JSON_VALUE(T.filterJson, 'strict $.dayofweek')
    = 'monday'
```
Query Functions: JSON_QUERY

- JSON_QUERY extracts a JSON value from a JSON value
  - most commonly to return the value to the user
  - result could also be used as input to another JSON query function

```sql
SELECT T.thing,
       JSON_VALUE (T.Json, 'lax $.who') AS Who,
       JSON_VALUE (T.Json, 'lax $.where' NULL ON EMPTY) AS locn,
       JSON_QUERY (T.Json, 'lax $.friends.name'
                     WITH ARRAY WRAPPER)
       AS FriendsNames
FROM T
```

- a returned FriendsNames value might look like:
  ```json
  [ "Roy", "Betty" ]
  ```
- or [] if there are no friends with names
Query Functions: JSON_TABLE

- JSON_TABLE constructs a table from a JSON value
  - Not part of the first 11.1 JSON support plan

- Takes three parameters:
  - the JSON value to operate on
  - a path expression to select the desired JSON data from that value
  - a COLUMNS clause to specify the shape of the result table

- Can be very complex because of the need to deal with JSON nesting
  - COLUMNS clause can contain a NESTED PATH expression followed by a (nested) COLUMNS clause which can contain ...
  - PLAN clause to express parent-child relationship, e.g. join, union, etc
Constructors: JSON_OBJECT

- **JSON_OBJECT** builds a JSON object value from a specified key and value.

```sql
SELECT
    JSON_OBJECT( KEY 'deptno' VALUE d.deptno, 'deptname' VALUE d.deptname ) AS D314
FROM depts AS d
WHERE d.deptno = 314
```

- might return:
  ```json
  { "deptno" : 314, "deptname" : "Engineering" }
  ```
- returned value is an SQL string value
Constructors: JSON_ARRAY

- JSON_ARRAY constructs a JSON array value from either a list of values or a subquery
  
  JSON_ARRAY(d.deptno, d.deptname)
  JSON_ARRAY(SELECT deptname FROM dept)

- The subquery form must supply exactly one column, array values are rows
- might return:
  
  [ 200, "Engineering" ] for the first example
  [ "Engineering", "R&D", "Sales", "Finance", ... ] for the second
Constructors: Aggregates

- JSON_OBJECTAGG builds a JSON object from a grouped input

```sql
SELECT j.job_seq,
       JSON_OBJECTAGG( j.job_attrib KEY j.job_attval
                         RETURNING VARCHAR(200) )
       AS attributes
FROM jobs AS j
GROUP BY j.job_seq
```

- Generates a JSON object `{ "job_attrib": job_attval, ... }` for each job_seq group
Constructor Functions: Aggregates

- JSON_ARRAYAGG builds a JSON array from a grouped input
  - Query to create a JSON object for each department listing all employees and their salary in order of increasing salary might look like:

```sql
SELECT d.deptno,
       JSON_OBJECT ('department' VALUE d.name,
                     'employees' VALUE JSON_ARRAYAGG
                       ( JSON_OBJECT( 'employee' VALUE e.name,
                                       'salary' VALUE e.salary)
                     ORDER BY e.salary ASC)
       ) as departments
FROM depts d, employees e
WHERE d.deptno = e.dept_id
GROUP BY d.deptno, d.name
```
Constructor Example Output

<table>
<thead>
<tr>
<th>DEPARTMENTS</th>
</tr>
</thead>
</table>
| { "department" : "Sales",
  "employees" : [ { "employee" : "James", "salary" : 7000},
                { "employee" : "Rachel", "salary" : 9000},
                { "employee" : "Logan", "salary" : 10000} ] } |


JSON in Ingres

- Ingres 11.1 will follow the Standard as closely as reasonable
  - no JSON_TABLE initially
  - limited or no FORMAT, PASSING, other esoteric clauses

- The JSON Technical Report is 19075-6
  - highly recommended if you're interested
  - much more detail than I can go into here
  - much much easier to follow than the actual specifications in Foundation
Other SQL:2016 Features?

▪ Row Pattern Matching looks interesting
  – Read Technical Report 19075-5 to see what it is
  – Give us feedback if you would be interested
  – Lots of work but not out of reach for some future version
    • Not especially well suited for X100 processing

▪ Polymorphic Table Functions
  – Read Technical Report 19075-7
  – Wildly complicated feature
  – Unlikely to happen any time soon
    • would need significant new underpinnings especially on the X100 side

▪ MDA Multi-dimensional arrays (2018) ????
Standards Work In Progress

- **Property Graph Queries**
  - Query property graphs stored in relational database
  - Might be SQL extension, might be a separate (sub)language
  - International ad-hoc committee working on this (under the auspices of the US national body DM32.2)
    - Actian is following, not actively participating at the moment

- **Streams and Streaming Queries**
  - Aimed at IoT data that need not all be stored, just watched
  - Implies "continuous" queries, stream tapping, stream views, lots more
  - Active area of work in the Working Group (WG3)
Other Ingres 11.1 Features

- Mixed Ingres/Vector queries
  - X100 is primary execution engine, Ingres streams Ingres data to X100
  - QEF executes fragments that are Ingres-only
    - usually just Orig or Proj-Rest, i.e., scans and simple filters
  - Joins execute preferentially in X100

- X100 table references in DB Procedures
  - Ingres is DBP controller, X100 queries run in X100, return results to QEF
  - Intended for basic logic and business rule processing in the DBMS
  - Not ideal for processing of large datasets via looping in the DBP
Other Ingres 11.1 Features

▪ Partition Management operations (finally!)
  – Minimum: split/merge range partitions, truncate partition
  – Intent is to avoid touching partitions that don't change

▪ Data masking
  – Basic masking, not full anonymization

▪ CREATE/DROP STATISTICS statement, both X100 and Ingres
  – Including background "auto-stats" for Ingres tables
  – Similar feature for X100 already exists in Vector 5.0
WG3 Meeting in Cape Town South Africa
WG3 Meeting in Okayama Japan
WG3 Meeting in Granville Ohio USA
Questions?
Thank you!