What's next for the GQL Standard?

Keith W. Hare

Convenor, ISO/IEC JTC1 SC32 WG3 Database Languages

21st Graph Data Council TUC Meeting

September 4, 2026

Old North Church, Boston, October 2024

Who Am I – Keith Hare?

- JCC Consulting, Inc.
  - High performance database systems
  - Data replication and migration
  - Database Administration
- Standards – SQL and GQL
  - Convenor, ISO/IEC JTC1 SC32 WG3 Database Languages
  - Vice Chair, ANSI INCITS Data Management
- Neo4j Languages, Standards, and Research (LANGSTAR)
  - Developing the GQL standard
  - Standards Diplomacy
  - GQL Strategy

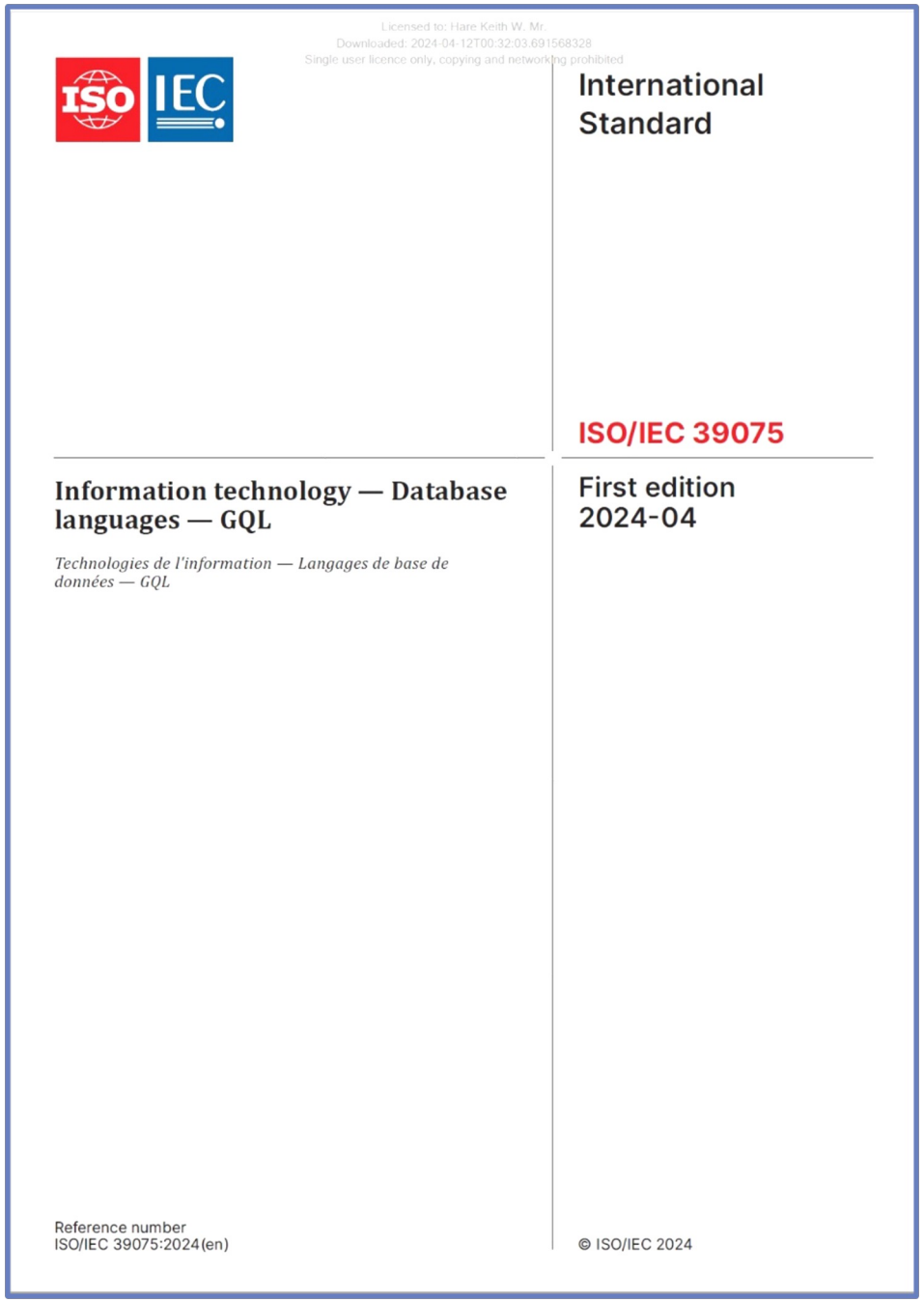
Database Language Standards

- ISO/IEC 9075 *Information technology — Database languages — SQL*
  - Published June 2023
  - 11 parts
  - Includes 9075:16 SQL/PGQ – property graph queries in SQL
- ISO/IEC 39075 *Information technology — Database languages — GQL*
  - Published April 2024
  - 1 part
- Work in progress on the next editions of both GQL and SQL

GQL:2024

Foundational Achievements

- Unified definition of property graph model
- Support for:
  - Schema-less
  - Graph types – schema model for property graphs
- Very powerful graph pattern matching language
  - Aligned with SQL/PGQ:2023 graph pattern language
- Execution model – sessions, transactions & standard exception codes
- Modernized type system – support for static and dynamic typing
- Composable queries
- SQL compatible



What's in the next edition of GQL?

The next edition of the GQL standard is being developed now.

- New capabilities in the next edition of GQL
- Corrections and bug fixes
- Anything else?
- Timing for the next edition
- Summary

New capabilities in the next edition of GQL

The main changes (so far) for the next edition of GQL are:

- Major capabilities
  - Conditional execution of statements
  - VECTOR data type and vector operations
  - Integrity constraints
  - Abstract element types
- Incremental enhancements
  - Reference individual list elements (using list[i] syntax)
  - PRODUCT in aggregate functions
  - GROUP BY ALL shorthand
  - Endpoint predicates
  - CAST error behavior
  - NULLS NOT DISTINCT
  - Match and Run
  - Unicode Case-folding
- Improvements for GQL Implementers

Conditional Procedures

- Example:

```
MATCH (n)
  WHERE EXISTS {
    WHEN n.age_in_month > 1
    THEN
      MATCH (n)<-[:IS_PARENT]-(x:Person)
      RETURN x
    ELSE
      MATCH (n)<-[:IN_HOSPITAL]-(x:Hospital)
      RETURN x
  }
RETURN n, x
```

Vector Data Types

- Support AI and retrieval augmented generation (RAG) techniques
- Atomic data type with three components
  - Vector of values
  - Type of the vector values
  - Dimension – count of vector values
- Store vectors as GQL properties in nodes and edges
- Vector Distance Function for comparing vectors
- Additional vector operators

Vector Distance Function

- VECTOR_DISTANCE (<vector 1>, <vector 2>, <vector distance metric>)
- Result is a relative distance between the two vectors
  - The smaller the distance, the more similar the vectors
- Distance metrics are common algorithms for comparing vectors
  - EUCLIDEAN
  - EUCLIDEAN_SQUARED
  - MANHATTAN
  - COSINE
  - DOT
  - HAMMING

Search for things that are similar

- GQL Example

```
MATCH (a: Articles)
RETURN a.author, a.article_text
      ,VECTOR_DISTANCE($ParamVec, a.article_vector, EUCLIDEAN) as distance
ORDER BY distance
LIMIT 10
```

- Calculate Vector Distance between a parameter and an Article property
- Ascending sort by the calculated distance
- Return 10 results

Additional vector operators

- VECTOR(vector-string, dimension, coordinate type)
  - Constructor function
- VECTOR_DIMENSION_COUNT(vector-name) – return size of a vector
- VECTOR_NORM(vector-name, vector distance metric)
  - Distance between a vector parameter and a zero vector
  - Only makes sense for EUCLIDEAN and MANHATTAN
- VECTOR_SERIALIZE(vector-name)
  - Return vector as a character string

Constraints and Keys

- Examples

```
CONSTRAINT emp_pk
// Covers (:Person&Employee), (:Robot&Employee), and (:Employee)
  FOR (n:Employee)
  REQUIRE n.emp_nr IS KEY  // key properties cannot be null
CREATE CONSTRAINT my_constraint
  FOR (n:Person)
  REQUIRE n.name IS UNIQUE    // unique properties can be null
```

- Very powerful capabilities, but room for additional capabilities

ABSTRACT GQL Element Types

- Examples – Nodes

```
ABSTRACT (:Person { name STRING }),
(:Employee => :Person { name STRING, ssn STRING, salary UINT }),
(:Manager => :Employee { name STRING, ssn STRING, salary UINT,
          reports UINT }),
(:Contractor => :Person { name STRING, tax_id STRING,
         compensation UINT })
```

- Example – Edges

```
// Edge type family definition
ABSTRACT (:Person)-[:knows { since :: DATE }]->(:Person)
// Compatible repetition of family characteristics
ABSTRACT (:Person)-[:knows { since :: DATE }]->(:City)
// Regular extension/inheritance
(:Employee)-[:learns => knows]->(:Handbook)
```

LIST (array) enhancements

- GQL V1 did not include ability to access an element of a LIST type
- Example:

```
match (n:list)
 return n.name, n.MyList[0], n.MyList[1], n.MyList[2], n.MyList[3]
 +-------------------------------------------------------------------+
 | n.name   | n.MyList[0] | n.MyList[1] | n.MyList[2] | n.MyList[3] |
 +-------------------------------------------------------------------+
 | "Keith" | 5           | 8           | 7           | NULL        |
 +-------------------------------------------------------------------+
```

- SET SESSION command to control list element access behaviors
  - Property graph style behavior

```
SESSION SET LIST BEHAVIOR (INDEX OFFSET, NEGATIVE FROM END, OUT OF RANGE NULL)
```

  - SQL style behavior

```
SESSION SET LIST BEHAVIOR (INDEX ORDINAL, NEGATIVE OUT OF RANGE, OUT OF RANGE
ERROR)
```

SUM and PRODUCT

- SUM
  - Current aggregation function – NULL if empty set or all values are NULL
  - Add "0 ON EMPTY" or "NULL ON EMPTY"

```
MATCH (a) WHERE a.PartColor = 'red' AND a.PartColor = 'blue'
RETURN SUM(a.Cost, 0 ON EMPTY)
```

- PRODUCT
  - New aggregation function – NULL if empty set or all values are NULL
  - Includes "1 ON EMPTY" or "NULL ON EMPTY"

```
MATCH (a) WHERE a.PartColor = 'red' AND a.PartColor = 'blue'
RETURN PRODUCT(a.SomeFactor, 1 ON EMPTY)
```

- Useful capabilities by themselves, needed for OPTIMAL PATH

GROUP BY ALL

- Example:

```
MATCH (e:employees)-[:worksIn]->(job:Job)
            -[locatedIn]->(state:State)
            -[locatedIn]->(country:Country)
            -[locatedIn]->(region:Region)
RETURN COUNT(*) AS total_employees,
       job, state, country, region,
       SUM(e.salary) AS total_salary,
       AVG(e.salary) AS average_salary,
       MIN(e.salary) AS min_salary,
       MAX(e.salary) AS max_salary
GROUP BY ALL
```

- Groups by non-aggregated properties:
  - job_id, state, country, region

Endpoint predicates

- Particularly useful for undirected edges
- Examples:

```
MATCH (a)-[b]-(c)
RETURN a IS ENDPOINT OF b
```

```
MATCH (a)-[b]-(c)
RETURN a IS NOT ENDPOINT OF b
```

CAST error behavior

- Avoid exceptions when using CAST
- Examples – Default

```
RETURN CAST ('abc' AS INT DEFAULT NULL ON CONVERSION ERROR)
RETURN CAST ('abc' AS INT DEFAULT 0 ON CONVERSION ERROR)
```

- Examples – Error

```
RETURN CAST ('abc' AS INT) // Current behavior
RETURN CAST ('abc' AS INT ERROR ON CONVERSION ERROR)
```

NULLS NOT DISTINCT

- Expansion of constraints
- Modeled on SQL capability
- Control what should happen when values referenced by a constraint are NULL
- Examples

```
CREATE CONSTRAINT n1_unique
FOR (a:N1)
REQUIRE a.n1_key IS UNIQUE NULLS DISTINCT

CREATE CONSTRAINT n2_unique
FOR (a:N2)
REQUIRE a.n2_key IS UNIQUE NULLS NOT DISTINCT
```

Match and Run

- Expansion of conditional procedures
- Example:

```
MATCH (a:Person)
CALL {
  WHEN BINDING (a)-[:OWNS]->(b:Movie)
  THEN
      RETURN "has at least movies" AS msg, count(b) AS num
  WHEN BINDING (a)-[:OWNS]->(b:Book)
  THEN
    RETURN "has only books" AS msg, count(b) AS num
  ELSE RETURN "has no media" AS msg, 0 AS num
  }
RETURN *
```

Unicode Case-folding

- Unicode defines titlecase as a distinct casing operation
  - Some characters have dedicated titlecase forms that differ from their uppercase forms
  - Example (dž, DŽ, and Dž are single characters):

```
UPPER('džungla') => 'DŽUNGLA'
TITLECASE('džungla') => 'Džungla'
```

- Case folding and lowercase mapping are distinct operations
  - Lowercase mapping transforms text into a lowercase form suitable for presentation
  - Case folding transforms text into a canonical form suitable for caseless comparison
  - Examples:

```
LOWER('ΟΣ') => 'ος'
CASEFOLD('ΟΣ') => 'οσ'
```

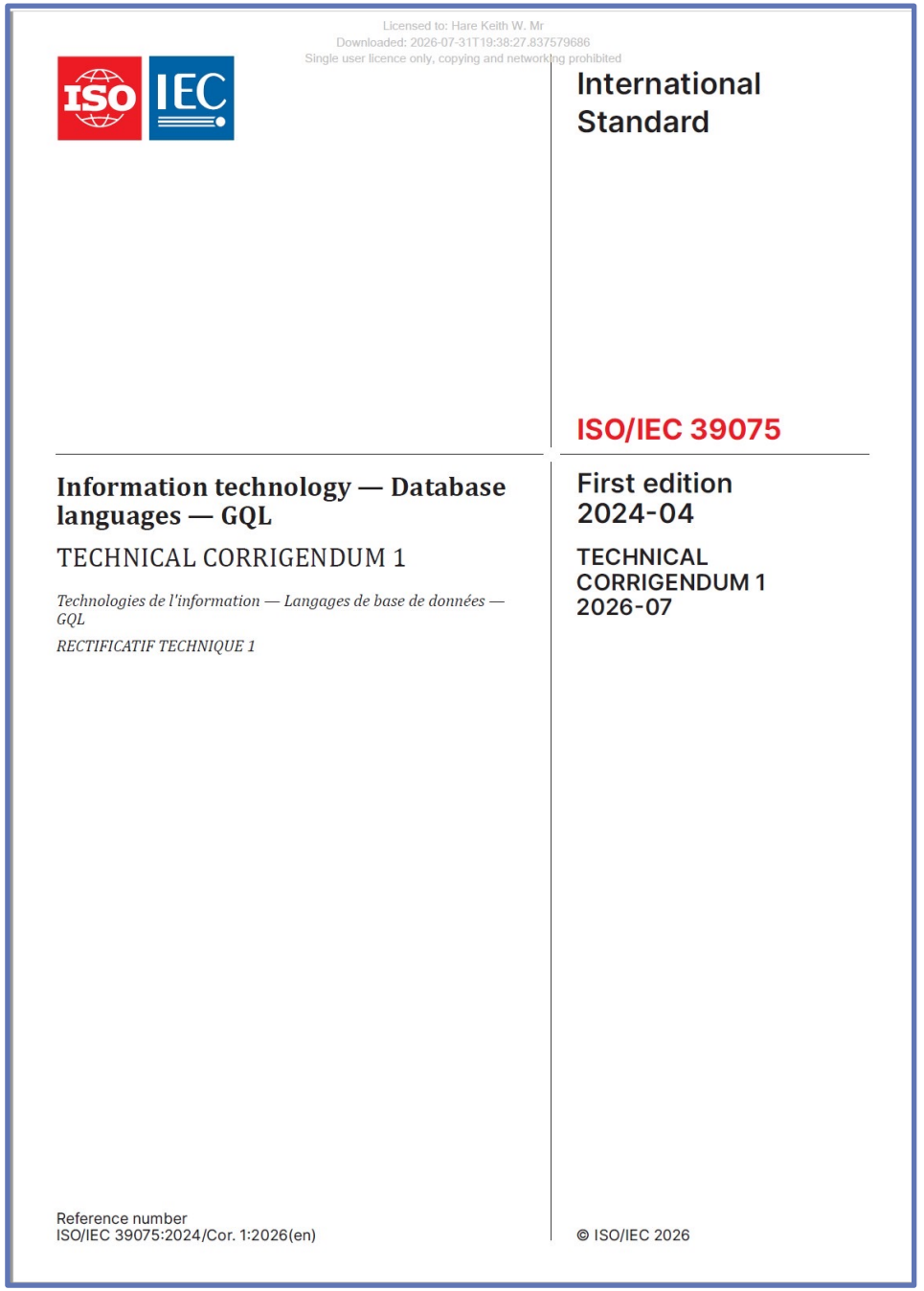
Improvements for GQL Implementers

- New Annex identifying the absolute minimal mandatory syntax.
- Documentation of relationships between implementation-defined aspects and optional features.

Bug Fixes and Corrections

- There are errors in GQL:2024
- Corrections are documented in 39075 GQL Technical Corrigendum (TC)
- First GQL TC published July 30, 2026

  https://www.iso.org/standard/93701.html



ISO IEC

International Standard

ISO/IEC 39075

Information technology — Database languages — GQL

TECHNICAL CORRIGENDUM 1

Technologies de l'information — Langages de base de données — GQL

RECTIFICATIF TECHNIQUE 1

First edition
2024-04

TECHNICAL CORRIGENDUM 1
2026-07

Reference number
ISO/IEC 39075:2024/Cor. 1:2026(en)

Anything else?

- There are additional capabilities under discussion
  - Element Type Inheritance
  - Default Values
  - IN predicate
  - Additional AI related capabilities
- Interesting capabilities that have not yet materialized
  - Optimal (Cheapest) paths
  - Schema information graph
  - Partitioned CALL Operator – CALL PER()

Element Type Inheritance

Element Type Inheritance design goals

- Additive element type inheritance
  - Element type definitions to specify required super types and additional property types
  - Inherited labels and property types do not need to be restated.
- Object Oriented-like substitutability for element reference values
- Concise specification of edge type families through endpoint node subtyping
- Minimal, backward-compatible extensions that preserve existing GQL syntax and semantics

Default Values in Graph Types

- Proposal for next WG3 meeting?

```
CREATE GRAPH TYPE MyPrettyGoodGraphType
    AS {
        /* Node types */
        (Person :Person {Name STRING
                        ,Joined DATE DEFAULT CURRENT_DATE}),
        (City :City {Name STRING
                    ,State STRING DEFAULT "Ohio"
                    ,Country STRING DEFAULT "USA"}),
        (Pet :Pet {Name STRING, PetType STRING}),
        /* Relationship types */
        (Person)-[:LIVES_IN {Since DATE}]->(City),
        (Person)-[:KNOWS]->(Person)
        (Person)-
            [:HasPet {since DATE DEFAULT CURRENT_DATE}]->(Pet)
    }
```

IN Predicate

- Proposal for next WG3 meeting?

```
RETURN 1 IN [1, 2, 2]           // TRUE
RETURN 2 IN [1, 2, 2]           // TRUE
RETURN 1 IN [1, 2, 2, NULL]     // TRUE
RETURN 2 IN [1, 2, 2, NULL]     // TRUE
RETURN 4 IN [1, 2, 2]           // FALSE
RETURN 4 IN [1, 2, 2, NULL]     // NULL
RETURN NULL IN [1, 2, 2]        // NULL
RETURN NULL IN [1, 2, 2, NULL] // NULL
```

Potential Additional AI capabilities

- Initial SQL proposal for next WG3 meeting?
- Register an LLM in a database
- Scalar Functions
  - VECTOR_GENERATE
  - AI_GENERATE
  - AI_CLASSIFY
  - AI_EXTRACT
  - AI_MASK
  - AI_SIMILARITY
  - AI_SUMMARIZE
  - AI_ANALYZE_SENTIMENT
- Predicates
  - AI_FILTER
- Aggregate functions
  - AI_AGG
  - AI_SUMMARY_AGG

Timing of the next edition of GQL?

The capabilities already accepted are enough to justify the GQL V2 standard.

- Current target for publication of GQL V2 is late 2027
- Possible timetable (subject to change)
  - Currently resolving Committee Draft (CD) consultation comments
  - Draft International Standard (DIS) Ballot – 20 weeks
    - 8 weeks for ISO central secretariat – June 14, 2027 to August 8, 2027
    - 12 week ballot – August 9, 2027 to November 11, 2027
  - Respond to DIS comments
  - Submit for publication – December, 2027
- Next edition of SQL is on a similar timetable

Summary – What's next?

- GQL V1 Standard – Published April, 2024
  - Full property graph database language
  - GQL Graph Pattern Matching (GPM) identical to SQL/PGQ GPM
  - Modified by Technical Corrigendum – published July 2026
- GQL V2 Standard – Publish late 2027?
  - Major capabilities:
    - Conditional execution of statements
    - VECTOR data type and vector operations
    - Integrity constraints.
    - Abstract element types
  - Other new capabilities?

Questions?

Boston Harbor, October 2024