ORACLE®

Property Graph Extensions for the SQL Standard

Vasileios Trigonakis <vasileios.trigonakis@oracle.com>
Principal Researcher
Oracle Labs, PGX

slides by Oskar van Rest <oskar.van.rest@oracle.com>

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

SQL extensions for Property Graphs (“SQL/PGQ”)

- What?
  - Tabular property graph model: store property graphs as sets of tables
  - Graph pattern matching: fixed-length and variable-length (e.g. shortest path)
  - Possibly more, but not in the first version
- Where?

  Aka the ISO SQL committee

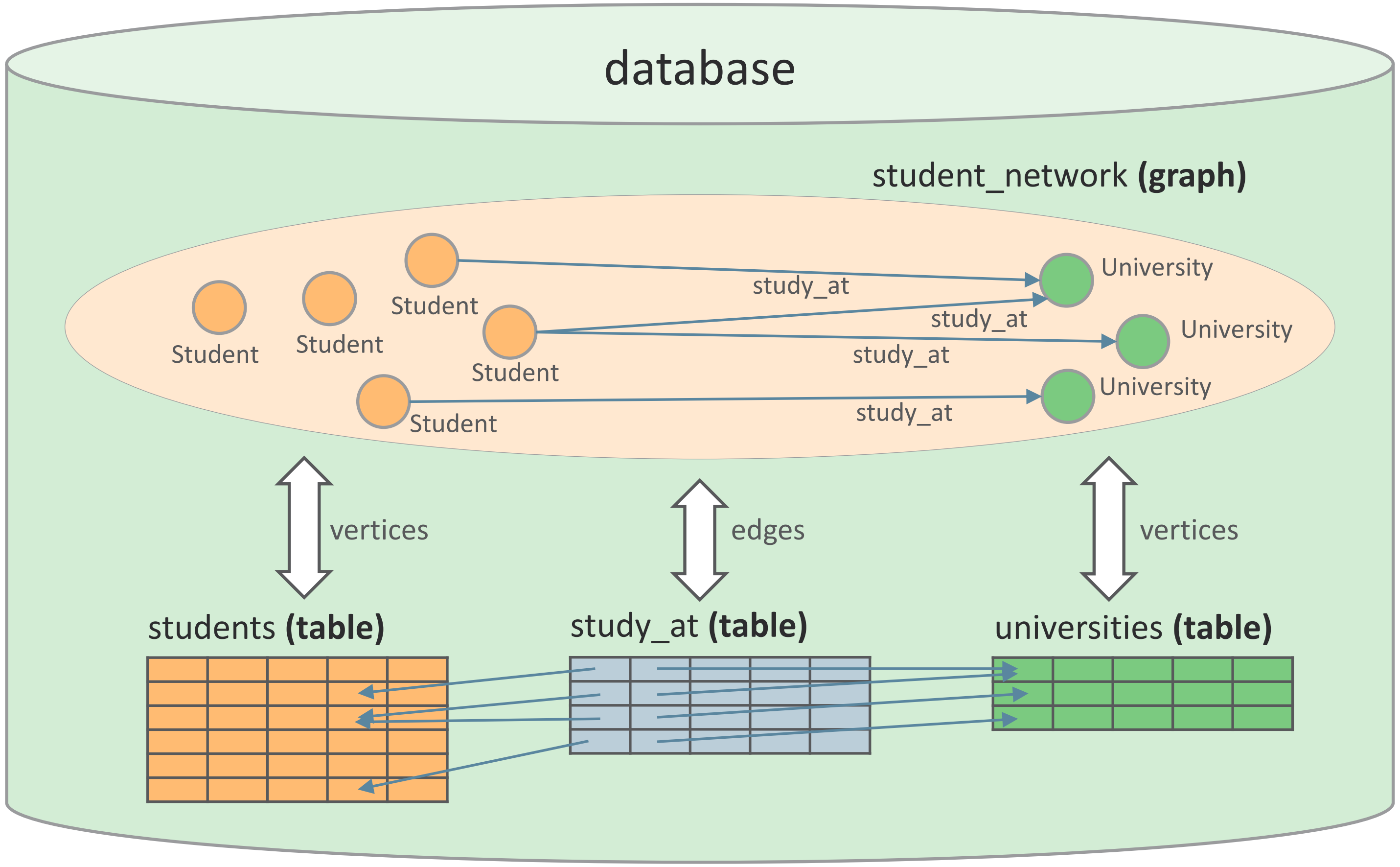
  - ISO: JTC 1/SC32/WG3 (USA, Germany, Japan, UK, Canada, China)

  Aka the US SQL committee

  - ANSI: INCITS / DM32 / DM32.2 / DM32.2 Ad Hoc Group on SQL Extensions for Property Graphs (Oracle, Neo4j, TigerGraph, IBM, SAP/Sybase, JCC Consulting)
- When?
  - Next version of SQL; possibly SQL:2020 or SQL:2021 (current version is SQL:2016)

Property Graphs that are backed by Tables

- A graph is stored as a set of vertex tables and edges tables
- A graph is like a view over existing tables: creating a graph requires no data copying
- There can be multiple graphs per database
- Graphs have a name and live in the same name space as tables



Two Property Graph models in SQL

The **pure property graph** model:

- A set of vertices (or nodes), each having:
  - A unique identifier (its value is implementation-dependent and may not be exposed to users)
  - Zero or more labels
  - Zero or more properties, each having a name and a type, which can be any SQL data type
- A set of edges (or relationships), each having:
  - A unique identifier, zero or more labels, and zero or more properties (like for vertices)
  - A source vertex and a destination vertex

The **tabular property graph** model:

- Store graphs as a set of vertex tables and edge tables
  - Vertex and edge tables are existing SQL tables and do not need to be created by the user
  - DDL maps tables into property graphs (next few slides)
- SQL standard doesn't limit implementations to the tabular property graph model
  - Implementations can provide any concrete models that conform to the pure property graph model

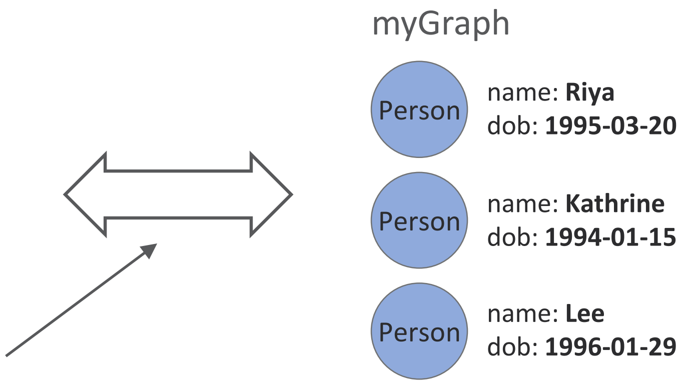
Tables map to sets of vertices and/or edges

- Each row in a vertex/edge table becomes a vertex/edge in the graph
  - By default, table names become labels, but it can be customized
  - By default, all columns become properties, but it can be customized
  - By default, PK-FK relationships are used to create edges, but it can be customized

Example vertex table:

People

id	name	dob
1	Riya	1995-03-20
2	Kathrine	1994-01-15
3	Lee	1996-01-29



SQL DDL statement:

```
CREATE PROPERTY GRAPH myGraph
  VERTEX TABLES (
    People LABEL Person PROPERTIES ( name, dob )
  )
```

PK-FK relationships in tables are used for creating edges

Vertex tables:

Person

id	name	dob
1	Riya	1995-03-20
2	Kathrine	1994-01-15
3	Lee	1996-01-29

University

id	name
1	UC Berkeley

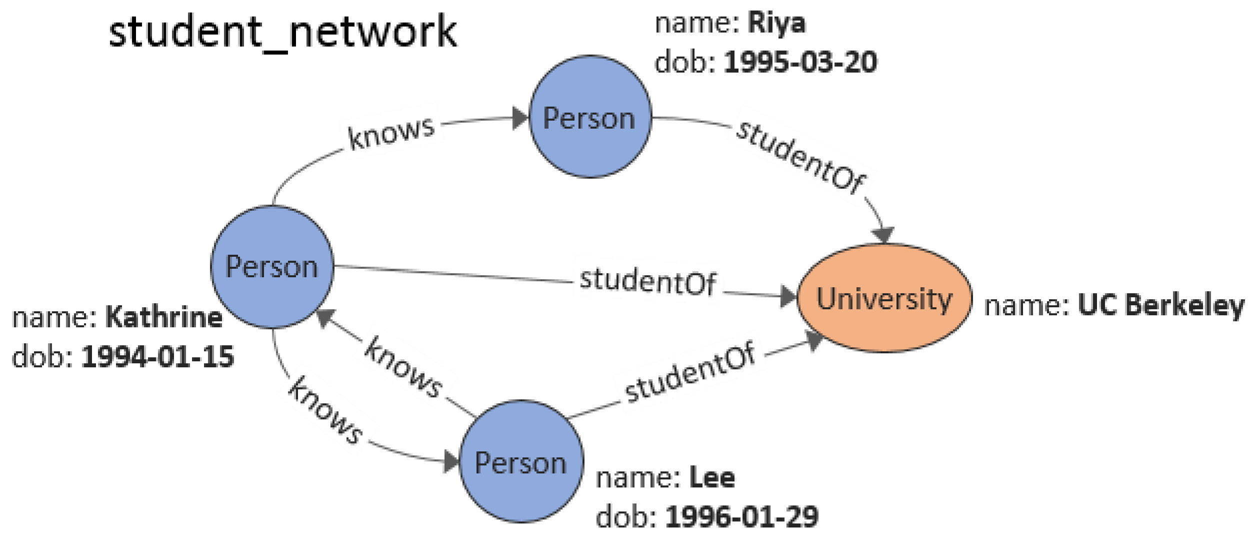
Edge tables:

knows

person1_id	person2_id
2	1
2	3
3	2

studentOf

person_id	university_id
1	1
2	1
3	1



SQL DDL statement:

```
CREATE PROPERTY GRAPH student_network
VERTEX TABLES ( Person PROPERTIES ( name, dob ),
                University PROPERTIES ( name ) )
EDGE TABLES ( knows SOURCE Person DESTINATION Person NO PROPERTIES,
              studentOf SOURCE Person DESTINATION University NO PROPERTIES )
```

Implementation can infer keys from primary/foreign keys of underlying tables

Manually specifying keys for vertices/edges

Multi-column keys supported, e.g., KEY (column1, column2)

Keys need to be manually specified in case the underlying tables (or views) do not already have the necessary keys defined:

SQL DDL statement:

```
CREATE PROPERTY GRAPH student_network
  VERTEX TABLES ( Person KEY ( id ) PROPERTIES ( name, dob ),
                  University KEY ( id ) PROPERTIES ( name ) )
  EDGE TABLES ( knows SOURCE KEY ( person1_id ) REFERENCES Person
                      DESTINATION KEY ( person2_id ) REFERENCES Person
                      NO PROPERTIES,
                studentOf SOURCE KEY ( person_id ) REFERENCES Person
                          DESTINATION KEY ( university_id ) REFERENCES University
                          NO PROPERTIES )
```

Person

id	name	dob
1	Riya	1995-03-20
2	Kathrine	1994-01-15
3	Lee	1996-01-29

knows

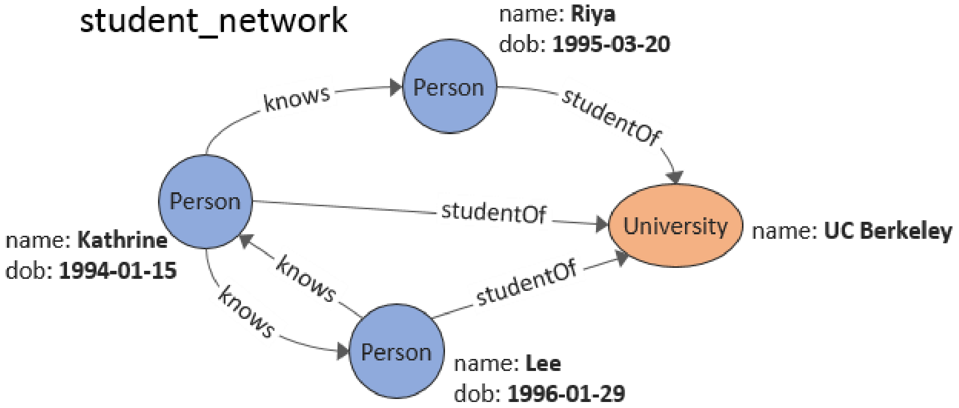
person1_id	person2_id
2	1
2	3
3	2

University

id	name
1	UC Berkeley

studentOf

person_id	university_id
1	1
2	1
3	1



Statically typed properties

Two vertex/edge tables that share a label need to have the same set of properties defined for that label (same property names and compatible data types)

- Each property belongs to a label:
  - Example with two tables with two labels each:

SQL DDL statement:

```
… VERTEX TABLES ( Students LABEL Person PROPERTIES ( first_name, last_name )
                           LABEL Student PROPERTIES ( student_number ),
                Professors LABEL Person PROPERTIES ( fname AS first_name, last_name )
                           LABEL Professor PROPERTIES ( employee_number ) )
```

Does not rename the underlying column

- Static typing provides safety during querying:

```
MATCH (p IS Personn)
```

Error because no label Personn defined

```
MATCH (p IS Professor)
WHERE p.student_number = ...
```

Error because no property student_number for Professor vertices

```
MATCH (p IS Person)
WHERE p.student_number = ...
```

Will give NULL values for professors but not for students

```
MATCH (p)
WHERE p.student_number = ...
```

Will give NULL values for professors but not for students

SQL/PGQ querying – Example

```
SELECT GT.creationDate, GT.content
FROM myGraph GRAPH_TABLE (
  MATCH
   (Creator IS Person WHERE Creator.email = :email1)
      -[ IS Created ]->
   (M IS Message)
      <-[ IS Commented ]-
   (Commenter IS Person WHERE Commenter.email = :email2)
      WHERE ALL_DIFFERENT (Creator, Commenter)
  COLUMNS (
      M.creationDate,
      M.content )
 ) AS GT
```

Get the **creationDate** and **content** of the **messages created by one person** ("email1") and **commented on by another person** ("email2").

ORACLE

Example: table + graph + CHEAPEST path

Given a table with a list of pairs of places called Here and There, for each row in the list, find the cheapest path from Here to There, with a stop at a gas station along the way.

Note: it is possible that some pairs (Here, There) are not connected by a path passing through a gas station; such disconnected pairs must nevertheless be reported in the result. It is possible that Here and There are the same location. It is possible that Here or There or both may be a gas station, in which case it is not necessary to find an additional gas station.

```
SELECT L.Here, GT.GasID, L.There, GT.TotalCost, GT.Eno, GT.Vid GT.Eid
FROM List AS L LEFT OUTER JOIN MyGraph GRAPH_TABLE (
  MATCH CHEAPEST (
      (H IS Place WHERE H.ID = L.Here)
          ( -[R1 IS Route COST R1.Traveltime]-> )*
               (G IS Place WHERE G.HasGas = 1)
           ( -[R2 IS Route COST R2.Traveltime]-> )*
      (T IS Place WHERE T.ID = L.There) )
  ONE ROW PER STEP (V, E)
  COLUMNS ( H.ID AS HID, G.ID AS GasID, T.ID AS TID, TOTAL_COST() AS totalCost,
            ELEMENT_NUMBER (V) AS Eno, V.ID AS Vid, E.ID AS Eid )
) AS GT ON (GT.HID = L.Here AND GT.TID = L.There)
ORDER BY L.Here, L.There, Eno
```

Updating a property graph

Through existing DML:

- Use existing SQL INSERT/UPDATE/DELETE to make changes to tables
  - Changes automatically become visible in all the graphs that depend on the table
  - Existing transactional semantics on tables still apply when tables are used in property graphs
- Example: Add a new person to a table

```
INSERT INTO person
  VALUES (4, 'Camille', 'Duval',
                DATE '1989-03-25')
```

  - This implicitly adds the person to all the graphs that depend on the "person" table

Through DDL:

- New ALTER PROPERTY GRAPH statement allows for:
  - Adding and removing vertex and edge tables
  - Adding and removing labels
  - Adding and removing properties
- Example
  - Add a set of (university) faculties to a graph named "student_network":

```
ALTER PROPERTY GRAPH student_network
  ADD VERTEX TABLE (
    university_faculties LABEL Faculty )
```

Conclusion

- Next version of SQL to support Property Graphs
  - Tabular property graph model
  - Graph pattern matching
- Users will be able to query property graphs and join property graph data with tabular, XML, and JSON data
- Possible future extensions beyond the first version of SQL/PGQ *may* include:
  - Graph DML
  - Graph construction
  - Undirected edges and mixed graphs
  - Graph analytics and procedural extensions

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

ORACLE®