SQL Property Graphs in Oracle Database and Oracle Graph Server (PGX)

Oskar van Rest

Consulting Member of Technical Staff

Product Development – Oracle Property Graph

June 23, 2023

Property Graph Queries are now officially part of the SQL standard

SQL:2023

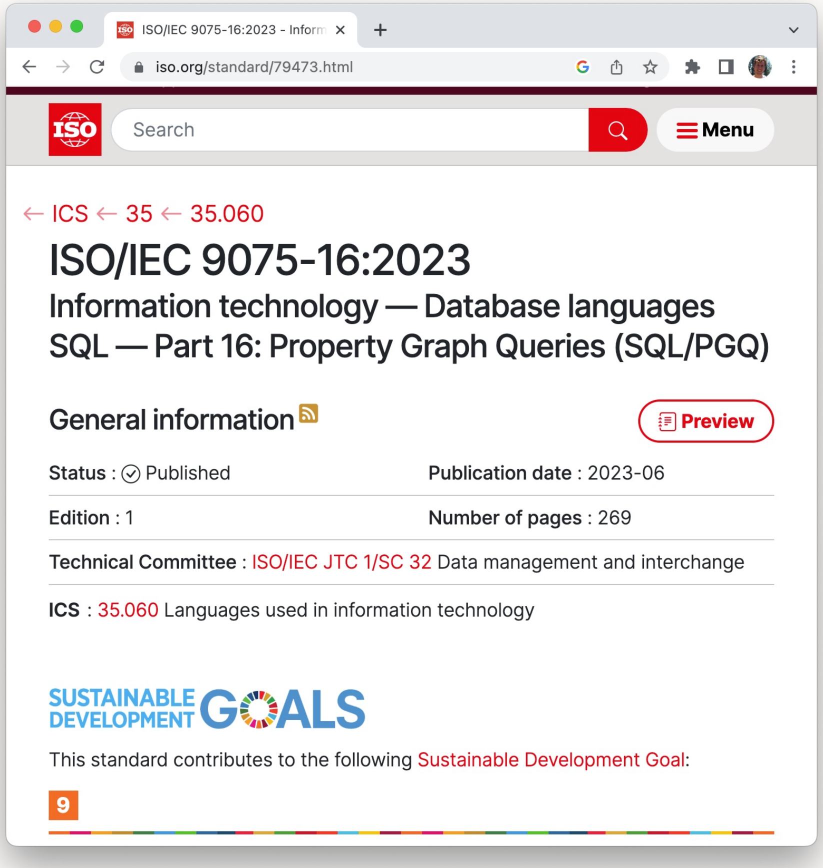
- Latest version of the SQL standard, published on June 1^{st}, 2023
- Includes Part 16: Property Graph Queries (SQL/PGQ)

SQL Property Graphs are defined on top of existing relational or JSON data

- No need to copy or transform data
- Transactional consistency
- Optionally add schemaless data to your graphs

SQL/PGQ is now implemented in Oracle 23c

- Oracle's product documentation[1] refers to the new feature as SQL Property Graphs or Property Graphs in SQL



← ICS ← 35 ← 35.060

ISO/IEC 9075-16:2023

Information technology — Database languages SQL — Part 16: Property Graph Queries (SQL/PGQ)

General information

Status : Published

Publication date : 2023-06

Edition : 1

Number of pages : 269

Technical Committee : ISO/IEC JTC 1/SC 32 Data management and interchange

ICS : 35.060 Languages used in information technology

SUSTAINABLE DEVELOPMENT GOALS

This standard contributes to the following Sustainable Development Goal:

9

https://www.iso.org/standard/79473.html

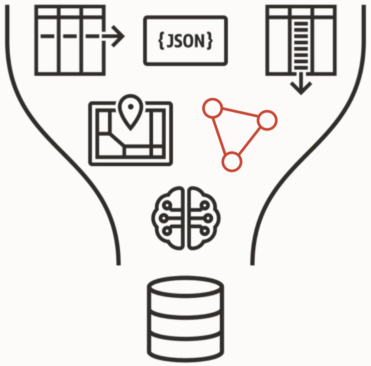
[1] Part II SQL Property Graphs - https://docs.oracle.com/en/database/oracle/property-graph/23.2/spgdg/sql-property-graphs.html

SQL Property Graphs in Oracle Database 23c

Benefits of **property graphs** in the Oracle Database:

- Extreme scalability by leveraging the existing SQL execution engine
- Security: Privileges, DataGuard, DataVault, RAS, Redaction, auditing, etc.
- SQL interoperability:
  - Join property graph data with relational data, JSON data, XML data, spatial data, etc.
  - SQL views, SQL triggers, SQL row pattern matching, SQL window functions, SQL analytics functions
  - PL/SQL, JavaScript Stored Procedures, etc.
- Use existing SQL tools and development environments: APEX, SQL Developer, SQLcl, drivers for Java (JDBC), Python, C, C++, etc.
- Flashback technologies: Undo transactions, Flashback Query, Time Travel, etc.
- Data pump support: Import/export
- Etc.

Complete



Converged Database

Simple

Autonomous Database

Property Graphs as part of a Converged Database

Add a SQL statement (or a REST API call), not another database

Store and Access movie details in **JSON** documents

```
CREATE TABLE movie_details(
         title VARCHAR2(255),
         movie JSON);

SELECT m.title Title,
       m.movie.director   DIR,
       m.movie.Star       STAR
FROM   movie_details m;
```

Find movies that customers have in common using **Graph Pattern Matching**

```
SELECT title
FROM GRAPH_TABLE ( cust_movie
       MATCH
         (c1)-[e1]->(m)<-[e2]-(c2)
       WHERE c1.cust_id = 1246813
         AND c2.cust_id = 1002487
       COLUMNS ( m.title ) )
FETCH FIRST 100 ROWS ONLY;
```

Use **Fuzzy Text Search** to find movie reviews containing “disappointed” or variations of it

```
SELECT title, comments
FROM   movie_reviews
WHERE CONTAINS(
        comment,
        'fuzzy(
          disappointed, 70, 6,
          weight)', 1) > 0;
```

Find theaters within 5km of Jane’s location using built-in **Spatial** functions

```
SELECT theater.name
FROM   theater, customer
WHERE  customer.name = 'Jane'
AND    SDO_WITHIN_DISTANCE(
         theater.location,
         customer.location,
         'distance=5 unit=km')
       = 'TRUE';
```

Store rental transaction in a **Blockchain Table** to prevent fraud

```
CREATE BLOCKCHAIN TABLE rental(
  u_id       number,
  user_name  varchar2(100),
  order_date date, ...);

INSERT INTO rental VALUES
(1,'Dominic','08-FEB-2023',..);
```

Store concession purchases in **XML** and easily retrieve them using standard SQL

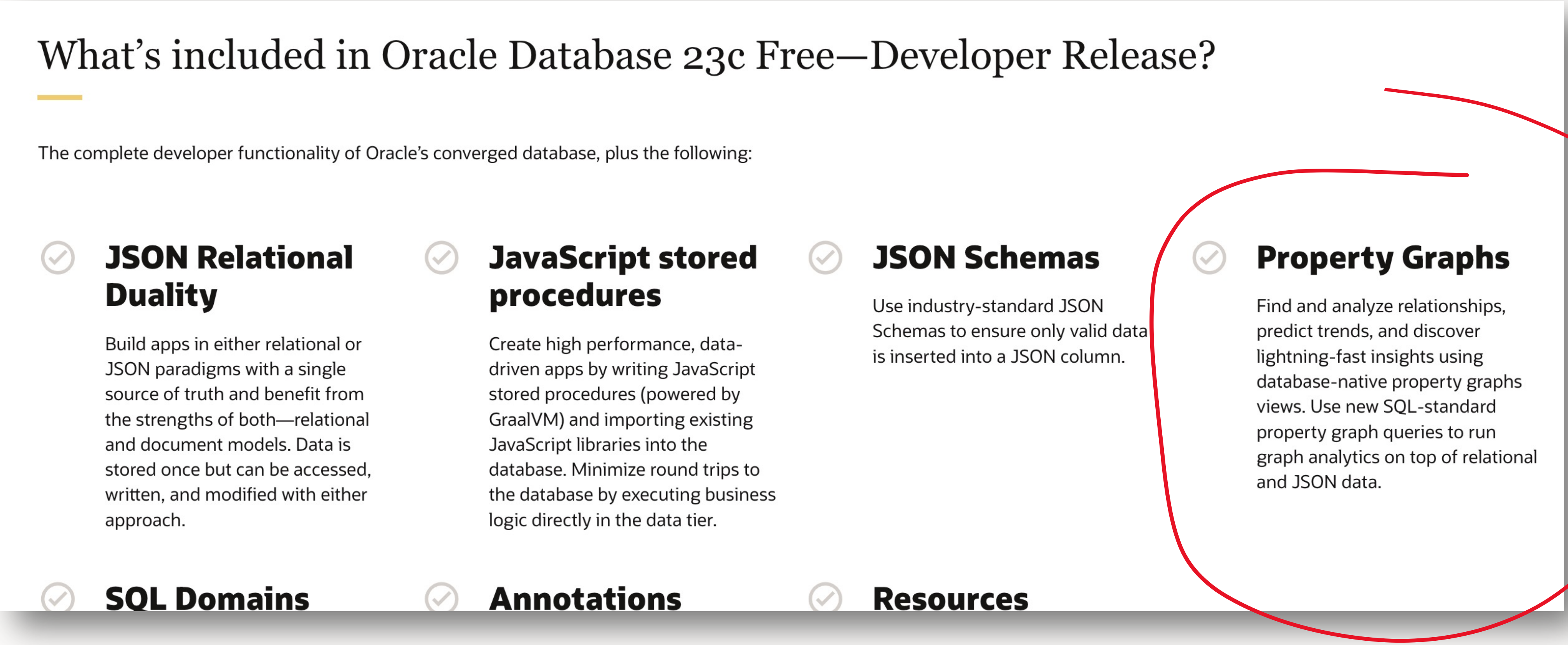
```
CREATE TABLE purchase_orders (
  key_column VARCHAR2(10),
  xml_column XMLType);

SELECT xml_column
FROM purchase_orders;
```

SQL Property Graphs in Oracle Database Free—Developer Release

SQL Property Graphs are part of **Oracle Database Free—Developer Release** (April 2023)

- The same, powerful Oracle Database, packaged for ease of use and simple download

https://www.oracle.com/database/free/



What's included in Oracle Database 23c Free—Developer Release?

The complete developer functionality of Oracle's converged database, plus the following:

JSON Relational Duality

Build apps in either relational or JSON paradigms with a single source of truth and benefit from the strengths of both—relational and document models. Data is stored once but can be accessed, written, and modified with either approach.

JavaScript stored procedures

Create high performance, data-driven apps by writing JavaScript stored procedures (powered by GraalVM) and importing existing JavaScript libraries into the database. Minimize round trips to the database by executing business logic directly in the data tier.

JSON Schemas

Use industry-standard JSON Schemas to ensure only valid data is inserted into a JSON column.

Property Graphs

Find and analyze relationships, predict trends, and discover lightning-fast insights using database-native property graphs views. Use new SQL-standard property graph queries to run graph analytics on top of relational and JSON data.

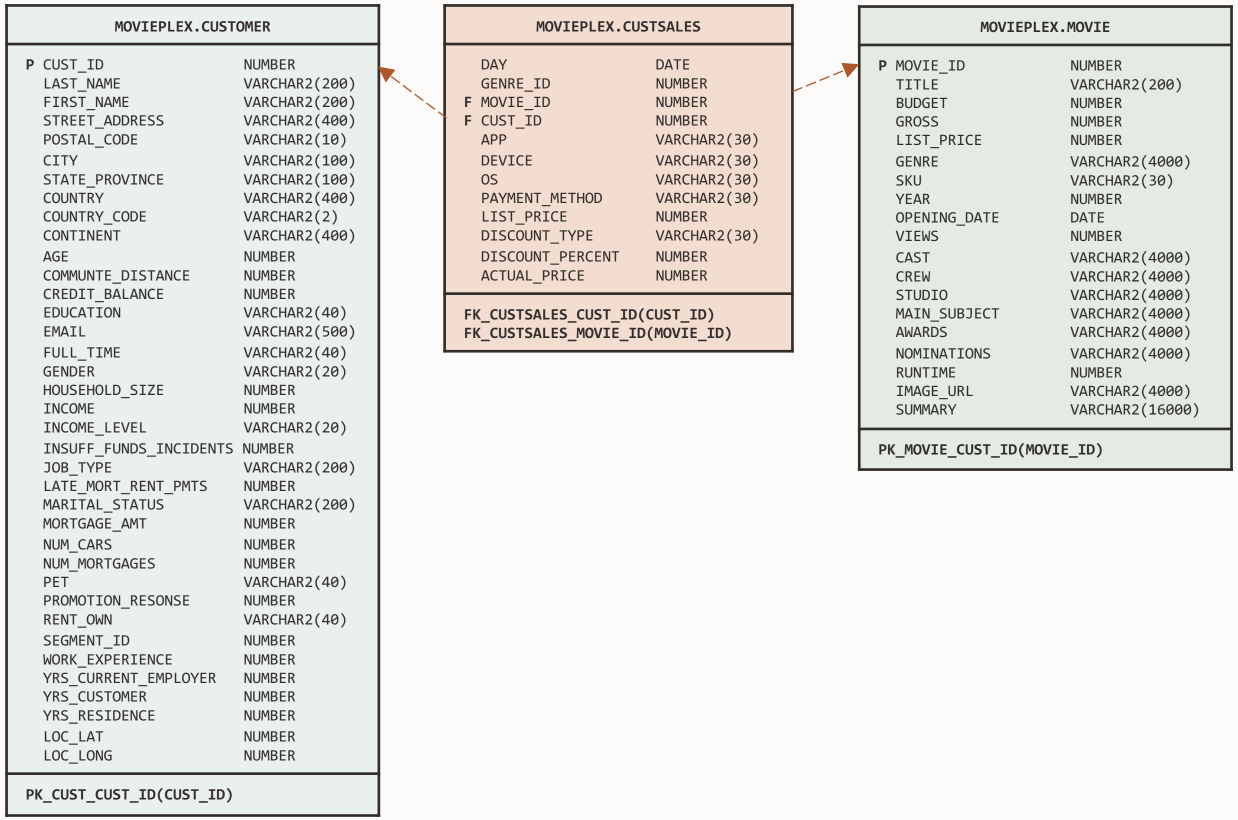
SQL Domains

Annotations

Resources

SQL Property Graph – Let’s start with a simple example

Relational Schema



MOVIEPLEX.CUSTOMER

P	CUST_ID	NUMBER
	LAST_NAME	VARCHAR2(200)
	FIRST_NAME	VARCHAR2(200)
	STREET_ADDRESS	VARCHAR2(400)
	POSTAL_CODE	VARCHAR2(10)
	CITY	VARCHAR2(100)
	STATE_PROVINCE	VARCHAR2(100)
	COUNTRY	VARCHAR2(400)
	COUNTRY_CODE	VARCHAR2(2)
	CONTINENT	VARCHAR2(400)
	AGE	NUMBER
	COMMUNTE_DISTANCE	NUMBER
	CREDIT_BALANCE	NUMBER
	EDUCATION	VARCHAR2(40)
	EMAIL	VARCHAR2(500)
	FULL_TIME	VARCHAR2(40)
	GENDER	VARCHAR2(20)
	HOUSEHOLD_SIZE	NUMBER
	INCOME	NUMBER
	INCOME_LEVEL	VARCHAR2(20)
	INSUFF_FUNDS_INCIDENTS	NUMBER
	JOB_TYPE	VARCHAR2(200)
	LATE_MORT_RENT_PMTS	NUMBER
	MARITAL_STATUS	VARCHAR2(200)
	MORTGAGE_AMT	NUMBER
	NUM_CARS	NUMBER
	NUM_MORTGAGES	NUMBER
	PET	VARCHAR2(40)
	PROMOTION_RESONSE	NUMBER
	RENT_OWN	VARCHAR2(40)
	SEGMENT_ID	NUMBER
	WORK_EXPERIENCE	NUMBER
	YRS_CURRENT_EMPLOYER	NUMBER
	YRS_CUSTOMER	NUMBER
	YRS_RESIDENCE	NUMBER
	LOC_LAT	NUMBER
	LOC_LONG	NUMBER

PK_CUST_CUST_ID(CUST_ID)

MOVIEPLEX.CUSTSALES

	DAY	DATE
	GENRE_ID	NUMBER
F	MOVIE_ID	NUMBER
F	CUST_ID	NUMBER
	APP	VARCHAR2(30)
	DEVICE	VARCHAR2(30)
	OS	VARCHAR2(30)
	PAYMENT_METHOD	VARCHAR2(30)
	LIST_PRICE	NUMBER
	DISCOUNT_TYPE	VARCHAR2(30)
	DISCOUNT_PERCENT	NUMBER
	ACTUAL_PRICE	NUMBER

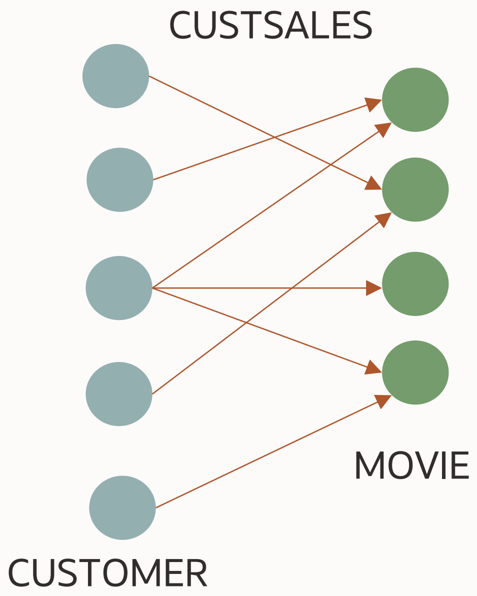
FK_CUSTSALES_CUST_ID(CUST_ID)
FK_CUSTSALES_MOVIE_ID(MOVIE_ID)

MOVIEPLEX.MOVIE

P	MOVIE_ID	NUMBER
	TITLE	VARCHAR2(200)
	BUDGET	NUMBER
	GROSS	NUMBER
	LIST_PRICE	NUMBER
	GENRE	VARCHAR2(4000)
	SKU	VARCHAR2(30)
	YEAR	NUMBER
	OPENING_DATE	DATE
	VIEWS	NUMBER
	CAST	VARCHAR2(4000)
	CREW	VARCHAR2(4000)
	STUDIO	VARCHAR2(4000)
	MAIN_SUBJECT	VARCHAR2(4000)
	AWARDS	VARCHAR2(4000)
	NOMINATIONS	VARCHAR2(4000)
	RUNTIME	NUMBER
	IMAGE_URL	VARCHAR2(4000)
	SUMMARY	VARCHAR2(16000)

PK_MOVIE_CUST_ID(MOVIE_ID)

Graph



SQL Property Graph Creation

Concise syntax when required metadata exists

- i.e. primary and foreign keys, uniqueness constraints

Graph created as a metadata object over original data

- No data copy or transformation
- Transactional consistency

```
CREATE PROPERTY GRAPH MovieRentals
  VERTEX TABLES (
    Customer, Movie
  )
  EDGE TABLES (
    CustSales SOURCE Customer DESTINATION Movie
  );
```

SQL Property Graph Creation – Explicit syntax

Syntax for explicitly defining keys

Syntax for explicitly defining properties

Syntax for exposing expressions as properties

Syntax for explicitly defining edge relationships

```
CREATE PROPERTY GRAPH MovieRentals
  VERTEX TABLES (
   Customer KEY (Cust_ID)
     PROPERTIES (First_Name, Last_Name, Gender),
   Movie KEY (Movie_ID)
     PROPERTIES (Title, Genre, Budget / List_Price AS Cost_Ratio)
 )
 EDGE TABLES (
  Custsales
    SOURCE      KEY(Cust_ID)  REFERENCES Customer (Cust_ID)
    DESTINATION KEY(Movie_ID) REFERENCES Movie (Movie_ID)
    PROPERTIES (Day AS Date_Rented) );
```

SQL Property Graph Creation – Additional Notes

Element (vertex or edge) tables are **existing tables** (base tables, external tables, or materialized views)

User can specify options for

- Labels (1 or more per vertex/edge table)
- Properties (0 or more per label), can rename properties
- Keys (single or multi-column key)

If not specified, **defaults** apply:

- Single label defaults to table name/alias
- All (non-hidden) columns are exposed as properties for a given label
- Keys are inferred from primary/foreign keys of underlying tables.
- PK-FK determines connection between vertices via edges (e.g., customer –[custsales]-> movie)

User can mix and match within a single PG definition:

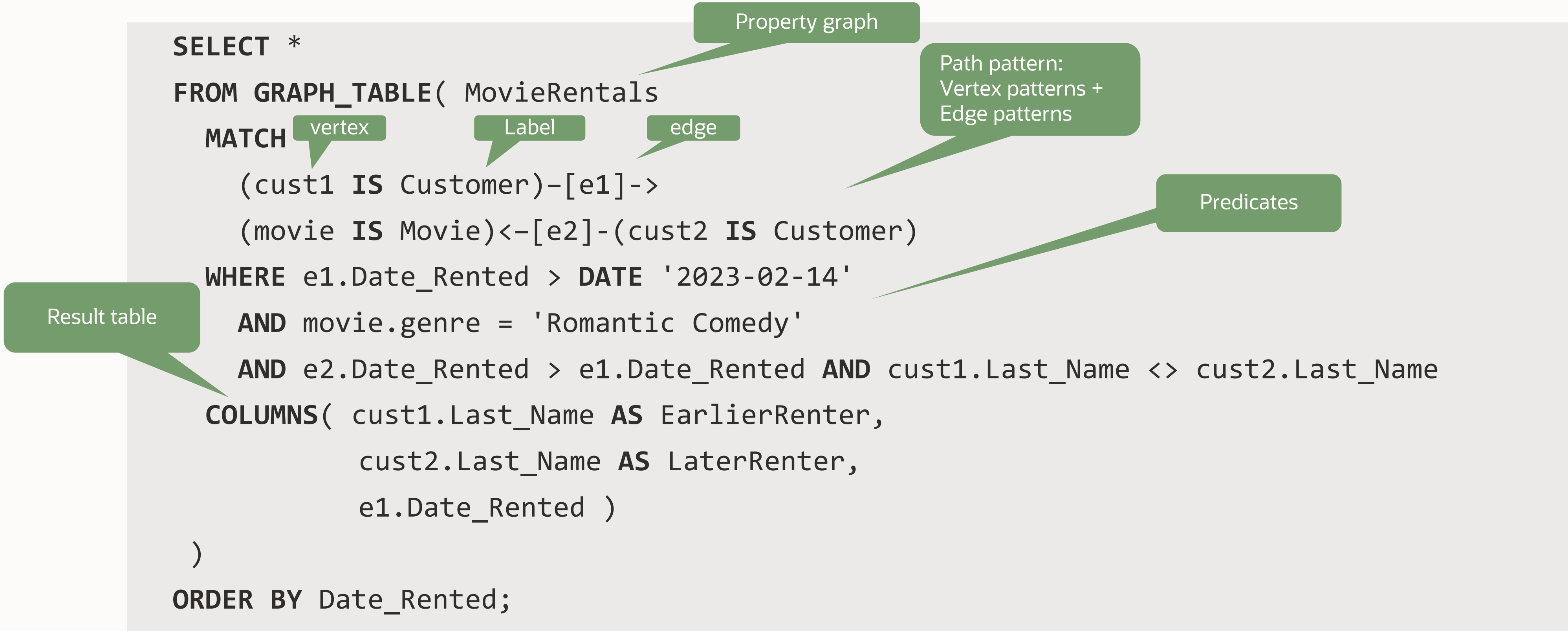
- Explicit options, and
- Implicit defaults

Querying SQL Property Graphs

```
SELECT …
FROM GRAPH_TABLE (
       <graph name>                               -- input graph
       MATCH <graph pattern>                      -- pattern to match
       WHERE <conditions>                         -- conditions to satisfy
       COLUMNS (<columns to return>)              -- return type of result table
     )
WHERE …
GROUP BY …
ORDER BY …
```

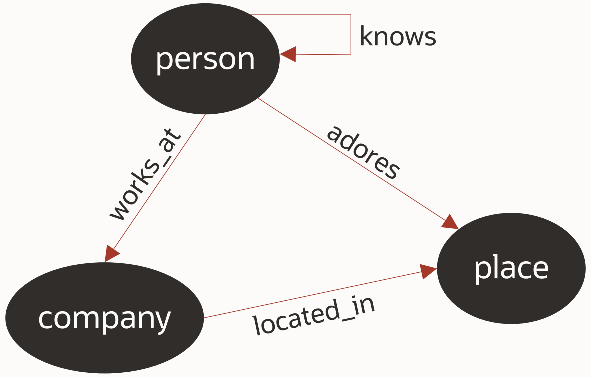
Querying Graphs – GRAPH_TABLE operator example

Find all two customers who rented the same romantic comedy movie one after the other and after February 14th, 2023.



```
SELECT *
FROM GRAPH_TABLE( MovieRentals
  MATCH
    (cust1 IS Customer)-[e1]->
    (movie IS Movie)<-[e2]-(cust2 IS Customer)
  WHERE e1.Date_Rented > DATE '2023-02-14'
    AND movie.genre = 'Romantic Comedy'
    AND e2.Date_Rented > e1.Date_Rented AND cust1.Last_Name <> cust2.Last_Name
  COLUMNS( cust1.Last_Name AS EarlierRenter,
           cust2.Last_Name AS LaterRenter,
           e1.Date_Rented )
 )
ORDER BY Date_Rented;
```

Given the following relational schema



Vertex tables

```
CREATE TABLE person (
  id NUMBER(5) PRIMARY KEY,
  works_at NUMBER(5),
  details JSON,
  CONSTRAINT fk_p FOREIGN KEY (works_at)
    REFERENCES company(id));
```

```
CREATE TABLE company (
  id NUMBER(5) PRIMARY KEY,
  located_in NUMBER(5),
  name VARCHAR2(100) ,
  age NUMBER(5) ,
  size_c NUMBER(10),
  CONSTRAINT fk_c FOREIGN KEY (located_in)
    REFERENCES place(id));
```

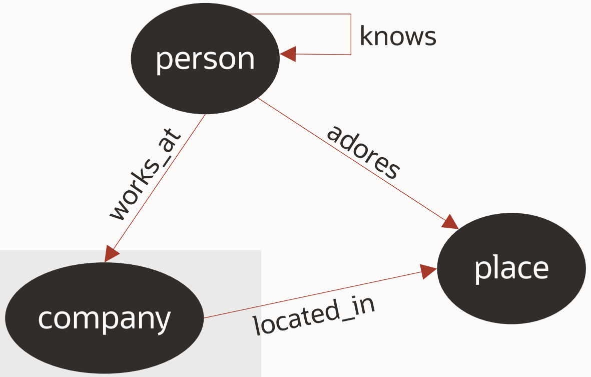
```
CREATE TABLE place (
  id NUMBER(5) PRIMARY KEY,
  name VARCHAR2(100) ,
  size_p NUMBER(10));
```

Edge tables

```
CREATE TABLE knows (
  e_src NUMBER(5) NOT NULL,
  e_dst NUMBER(5) NOT NULL,
  since NUMBER(5),
  CONSTRAINT pk_k PRIMARY KEY (e_src, e_dst),
  CONSTRAINT fk_k1 FOREIGN KEY (e_src) REFERENCES person(id),
  CONSTRAINT fk_k2 FOREIGN KEY (e_dst) REFERENCES person(id));
```

```
CREATE TABLE adores (
  e_src NUMBER (5) NOT NULL ,
  e_dst NUMBER (5) NOT NULL ,
  CONSTRAINT pk_a PRIMARY KEY (e_src, e_dst),
  CONSTRAINT fk_a1 FOREIGN KEY (e_src) REFERENCES person(id),
  CONSTRAINT fk_a2 FOREIGN KEY (e_dst) REFERENCES place(id));
```

Graph creation on top of JSON data



```
CREATE PROPERTY GRAPH MY_GRAPH
  VERTEX TABLES (
    person AS p KEY(id) LABEL person PROPERTIES(
       p.id,
       p.details.name.string() AS name,
       p.details.address.city.string() AS city,
       p.details.address.zip.number() AS zip,
       p.details.birthdate.date() AS birthdate,
       p.details.creditScore[*].avg() AS avg_credit_score),
    company,
    place
  )
  EDGE TABLES (
    knows SOURCE KEY(e_src) REFERENCES p(id) DESTINATION KEY(e_dst) REFERENCES p(id),
    person AS works_at SOURCE KEY(id) REFERENCES p(id)
                       DESTINATION KEY(works_at) REFERENCES company(id),
    adores SOURCE p DESTINATION place,
    company AS located_in SOURCE KEY(id) REFERENCES company(id)
                          DESTINATION KEY(located_in) REFERENCES place(id)
 );
```

JSON simplified syntax used to define properties from schemaless JSON column values

Friends (and friends of friends) working in Seattle

Bob needs a loan to buy a new house in Seattle.

The bank wants to check how many friends and friends of friends of Bob work in Seattle in order to understand the likelihood of his social integration.

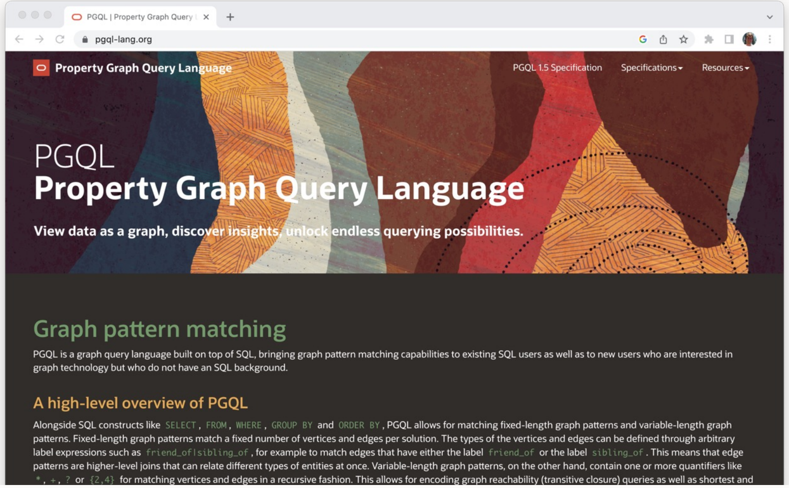
Graph query in SQL

```
SELECT *
FROM GRAPH_TABLE( MY_GRAPH
  MATCH (p)-[IS knows]-{1,2}(f),
        (f)-[IS works_at]->(c IS company),
        (c)-[IS located_in]->(pl IS place)
  WHERE p.name = 'Bob' AND
        pl.name = 'Seattle'
  COLUMNS ( f.name, f.zip AS zip_code ) );
```

Property Graph Query Language (PGQL) and SQL

In Oracle Database 21c and earlier, PGQL is the primary way to query property graphs

- There are two ways to run PGQL queries
  - PGQL on RDBMS: graph queries translated into SQL queries against tables, using Recursive WITH and PL/SQL
  - PGQL in Oracle Graph Server (PGX): graph queries processed in a specialized in-memory graph engine
    - Note: Property graphs in Oracle Database can (optionally) be loaded into Oracle Graph Server (PGX) to accelerate certain types of queries and graph algorithms



PGQL
Property Graph Query Language

View data as a graph, discover insights, unlock endless querying possibilities.

Graph pattern matching

PGQL is a graph query language built on top of SQL, bringing graph pattern matching capabilities to existing SQL users as well as to new users who are interested in graph technology but who do not have an SQL background.

A high-level overview of PGQL

Alongside SQL constructs like `SELECT`, `FROM`, `WHERE`, `GROUP BY` and `ORDER BY`, PGQL allows for matching fixed-length graph patterns and variable-length graph patterns. Fixed-length graph patterns match a fixed number of vertices and edges per solution. The types of the vertices and edges can be defined through arbitrary label expressions such as `friend_of|sibling_of`, for example to match edges that have either the label `friend_of` or the label `sibling_of`. This means that edge patterns are higher-level joins that can relate different types of entities at once. Variable-length graph patterns, on the other hand, contain one or more quantifiers like `*`, `+`, `?` or `{2,4}` for matching vertices and edges in a recursive fashion. This allows for encoding graph reachability (transitive closure) queries as well as shortest and

https://pgql-lang.org/

In Oracle Database 23c the new SQL syntax is introduced

- PGQL will continue to be supported but over time SQL will become the primary way for Oracle customers to query property graphs
- We are adding syntax to PGQL to help customers transition to SQL
  - For example, PGQL now supports SQL's CREATE PROPERTY GRAPH statement and SQL's GRAPH_TABLE operator[1]

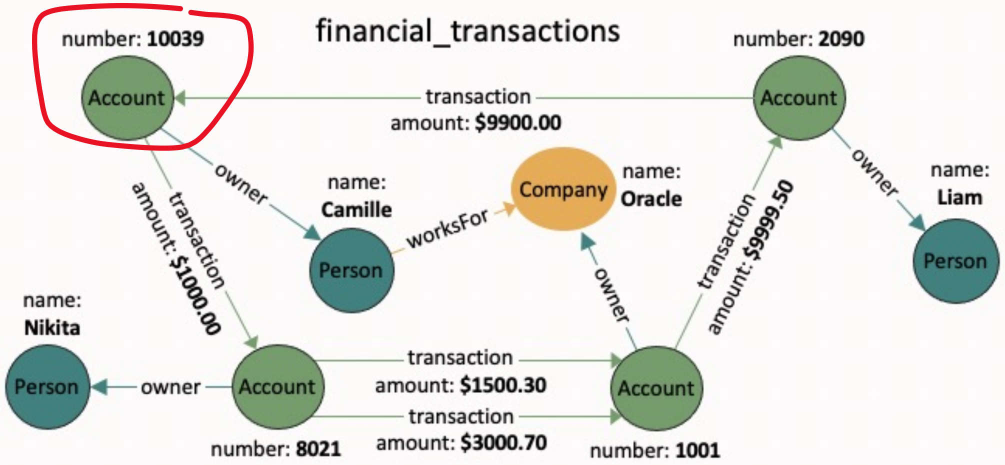
[1] PGQL 2.0 Specification - https://pgql-lang.org/spec/2.0/ (May 2023)

New GRAPH_TABLE operator in PGQL helps to transition to SQL

Through helpful error messages, the GRAPH_TABLE operator in PGQL guides users to use SQL-compatible syntax rather than legacy PGQL syntax

- Increases interoperability between the Oracle Database and the Oracle Graph Server (PGX)

```
SELECT *
FROM GRAPH_TABLE ( financial_network
       MATCH (a IS account) -[e IS transfer]- (b IS Account)
       WHERE a.number = 10039
       COLUMNS ( b.number, e.amount,
         CASE WHEN is_source_of(e, v)
           THEN 'Ougoing transfer' ELSE 'Incoming transfer'
         END AS transfer_type ) )
LIMIT 10
```



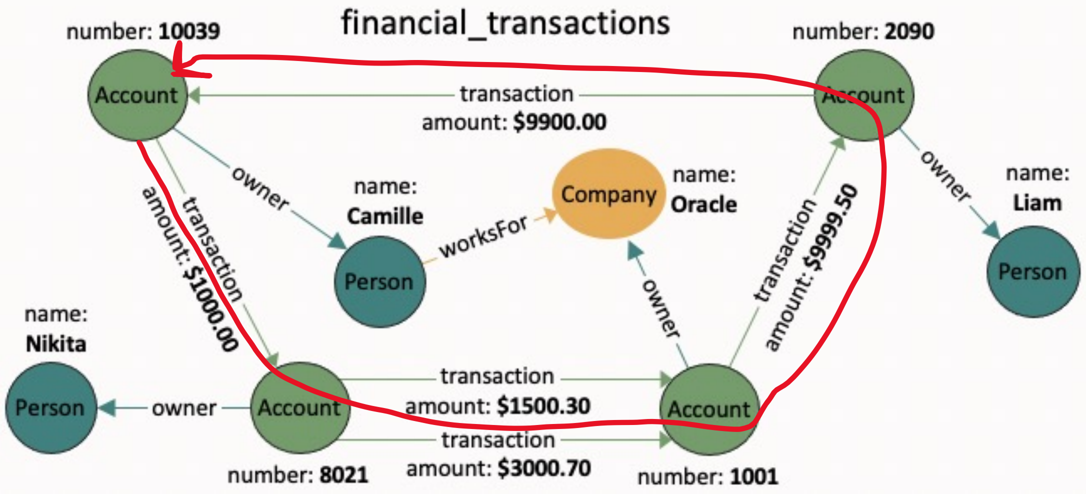
```
    Error(s) in line 6:
            CASE WHEN is_source_of(e, v)
                      ^^^^^^^^^^^^^^^^^^
    GRAPH_TABLE restriction: use v IS SOURCE OF e instead of is_source_of(e, v)

    Error(s) in line 9:
    LIMIT 10
    ^^^^^^^^
    GRAPH_TABLE restriction: use FETCH FIRST 10 ROWS ONLY instead of LIMIT 10
```

New SQL features in PGQL (1/2)

Path modes: ACYCLIC, SIMPLE, TRAIL, WALK

Cycle avoidance in combination with ANY, ALL, SHORTEST or CHEAPEST path finding:

```
SELECT *
FROM GRAPH_TABLE ( financial_transactions
       MATCH SHORTEST 5 SIMPLE PATHS
         (a IS account) -[e IS transaction]->+ (a)
       WHERE a.number = 10039
       COLUMNS (LISTAGG(e.amount, ', ') AS amounts)
     )
ORDER BY amounts
```



```
+-----------------------------------+
| amounts                           |
+-----------------------------------+
| 1000.0, 1500.3, 9999.5, 9900.0 |
| 1000.0, 3000.7, 9999.5, 9900.0 |
+-----------------------------------+
```

(while 5 paths were requested, only 2 valid paths exist in the graph)

SQL's Path Modes explained:

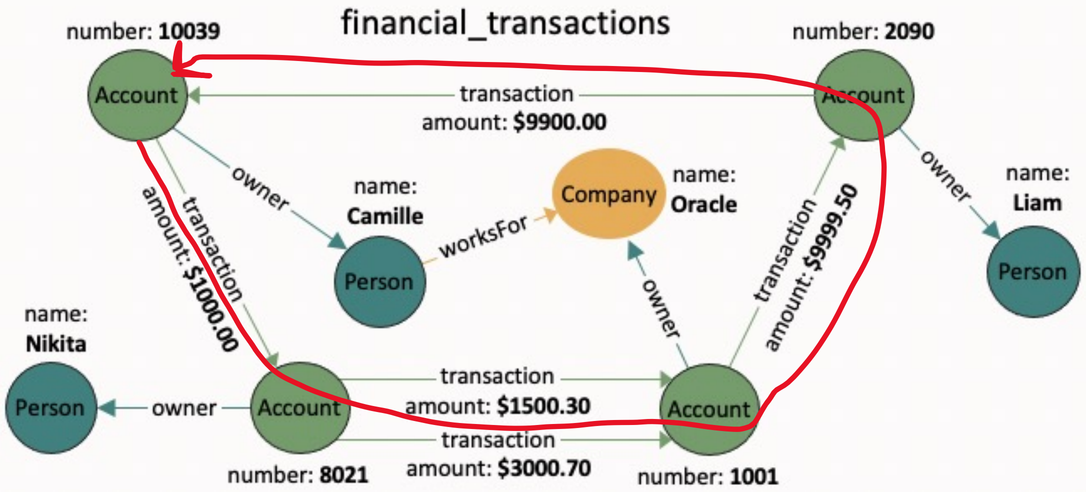
- **WALK (default)**: no filtering of paths happen.
- **TRAIL**: paths with repeated edges are not returned.
- **ACYCLIC**: paths with repeated vertices are not returned.
- **SIMPLE**: paths with repeated vertices are not returned unless the repeated vertex is the first and the last in the path.

New SQL features in PGQL (2/2)

Path unnesting: ONE ROW PER VERTEX / STEP

Graph Table Rows Clause allows for unnesting of paths:

```
SELECT *
FROM GRAPH_TABLE ( financial_transactions
       MATCH SHORTEST 5 SIMPLE PATHS
         (a IS account) -[IS transaction]->+ (a)
       WHERE a.number = 10039
       ONE ROW PER STEP ( v1, e, v2 )
       COLUMNS( MATCHNUM() AS matchnum,
                ELEMENT_NUMBER(e) AS elemnum,
                v1.number AS account1,
                v2.number AS account2, e.amount))
ORDER BY matchnum, elemnum
```



```
+---------------------------------------------------------+
| matchnum | elemnum | account1 | account2 | amount |
+---------------------------------------------------------+
| 0        | 2       | 10039    | 8021     | 1000.0 |
| 0        | 4       | 8021     | 1001     | 1500.3 |
| 0        | 6       | 1001     | 2090     | 9999.5 |
| 0        | 8       | 2090     | 10039    | 9900.0 |
| 1        | 2       | 10039    | 8021     | 1000.0 |
| 1        | 4       | 8021     | 1001     | 3000.7 |
| 1        | 6       | 1001     | 2090     | 9999.5 |
| 1        | 8       | 2090     | 10039    | 9900.0 |
+---------------------------------------------------------+
```

(2 paths with 4 edges each => 8 rows)

SQL's Graph Table Rows Clause explained:

- **ONE ROW PER MATCH (default)**: no unnesting takes place.
- **ONE ROW PER VERTEX**: declares a single iterator vertex variable; produces one row per vertex.
- **ONE ROW PER STEP**: declares an iterator vertex variable, an iterator edge variable, and another iterator vertex variable; produces one row per step (a step is a vertex-edge-vertex triple).

Summary

Graphs can be created and queried in SQL

A converged database like the Oracle Database combines the power of relational, graph, JSON and more

Since graphs are part of the SQL engine all existing tools and programmatic interfaces work with graphs

PGQL (Property Graph Query Language) will help with the transition to SQL, by alignment to SQL

ORACLE