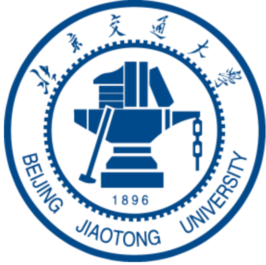


Fusion of Relational and Graph Database Techniques: An Emerging Trend

DASFAA 2023 Tutorial   Apr 19, 2023

Yu Liu, Qingsong Guo, Jiaheng Lu
Beijing Jiaotong University, North University of China, University of Helsinki

OUTLINE

1. Relational and Graph Data Modelling (10 minutes)
2. Multi-Model Queries (20 minutes)
3. Join and Subgraph Matching (15 minutes)
4. Fusion of Query Processing Techniques (40 minutes)
5. Open problems and challenges (5 minutes)

01 Relational and Graph Data Modelling

Data modelling is a never-ending story. We will review the history of relational and graph data modelling.

- The relational model and its extensions
- The graph data models

Big Data Modelling

- Data modelling is a Never-Ending Story
  - Data model enables a user to define the data using **high-level constructs** without worrying about low-level details of how data will be stored on disk

- Many data models proposed to address the variety of big data
  - Structured data (our focus)
    - All data conforms to a predefined schema, e.g., business data
  - Semi-structured data
    - Some structure in the data but implicit and irregular, e.g., XML and JSON
  - Unstructured data
    - No structure in data, e.g., text, sound, images, videos

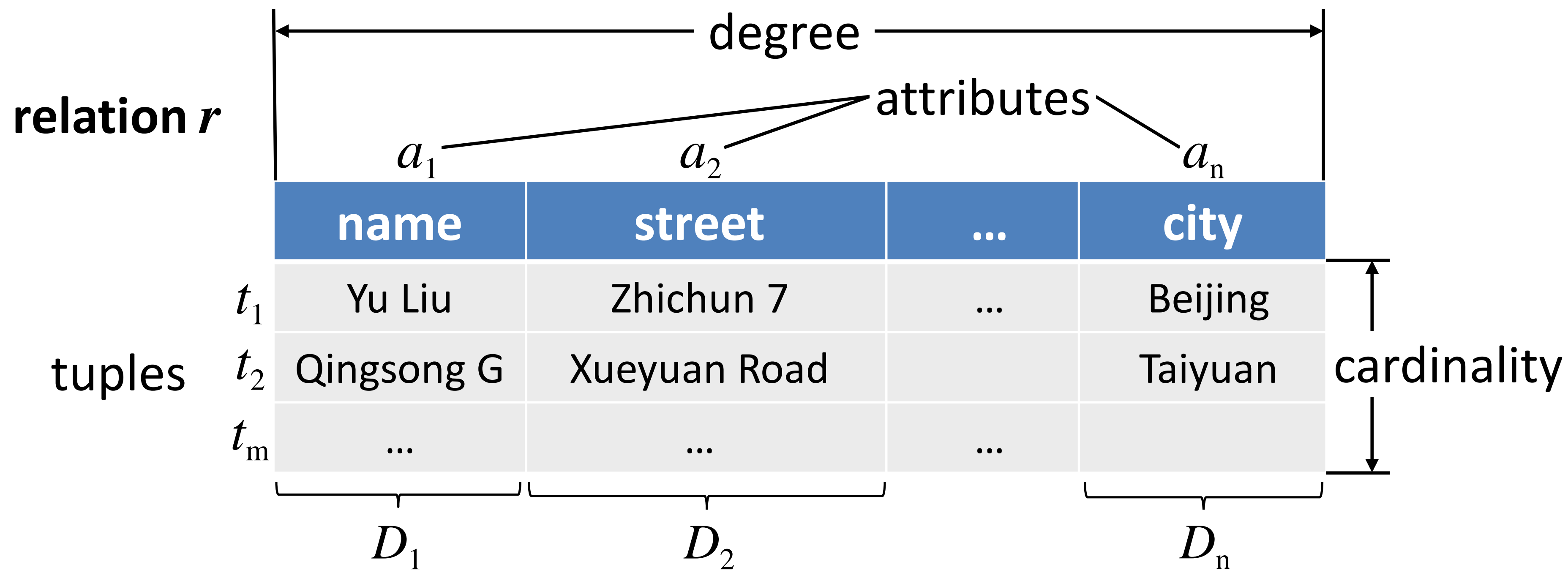
Data Models

- Relational: 1970’s
- Entity-Relationship: 1970’s
  - Successful in logical database design
- Extended Relational: 1980’s
- Semantic: late 1970’s and 1980’s
- Object-oriented: late 1980’s and early 1990’s
  - Address impedance mismatch: relational dbs and OO languages
- Object-relational: late 1980’s and early 1990’s
  - User-defined types, ops, functions, and access methods
- Semi-structured: late 1990’s and 2000’s
- Graph: 1990’s to the present

The Relational Model

The dominant data model over last 5 decades

- A relation is **a subset of Cartesian product** and logically represented as un-ordered tuples and each record is uniquely identified by a key
- Table, columns(attributes), rows (tuples)
- Domain, cardinality, etc.
- Cannot nest one tuple within another



The Relational Extensions

The relational model can be described by 3 components:

- **Primitive types:** number, string, Boolean, Date, null, etc.
- **Relational constructor** used on the primitive types
- A set of **operators** that can be used to each primitive type and type constructor

The relational model can be extended correspondingly

- **Nested relational model**
  - Remove the restriction of 1NF
  - **Nested type constructors** that allow building nested relations from atomic types by using tuple constructors and set constructors
- **Object-relational model**
  - Separates set and tuple of the relational constructor and support object
- **JSON**
  - includes other type constructors such as lists, multisets, arrays, etc.

Semistructured Data

Self-describing by associating semantic tags or markers and enforce hierarchies of records and fields by nesting elements within the data.

- XML, json, protobuf, Parquet, etc.

Can be viewed as relational extensions with restriction removal

- Complex types: arrays, (nested) tuples, maps
- Rigid schema is not necessary

Relational data model

- Rigid flat structure(tables)
- Schema must be fixed in advanced
- Binary representation: good for performance, bad for exchange
- Query language based on Relational Calculus

Semistructured data model

- Flexible, nested structure(trees)
- Schemaless("self-describing")
- Richer types, e.g., text representation: good for exchange, bad for performance
- Query language borrows from automata theory

JSON as an example

Primitive values

- A string, which looks like "Hello"
- A number, which looks like 42 or -3.14159
- true or false
- null

Structured values

- Object: a list of name-value pairs (i.e., fields)
  { "partno": 461,
  "description": "Wrench"
  }
- Array: an ordered list of items
  – [1, 2.5, "Hello", true, null]

Order.json

```
{"Order_no":"0c6df508",
     “Orderlines": [
         { “Product_no”: "2724f”
           “Product_Name”: “Toy",
           "Price":66 },
         { “Product_no”: “3424g”,
           “Product_Name”: “Book",
           "Price":40 } ]
}
```

The items in an array and the values in the fields of an object can be any JSON values, arrays and objects.

Data Viewed as Graphs

A graph consists of a set of vertices V and edges E

- A generalization of the relational model and semi-structured model

Original intuition:

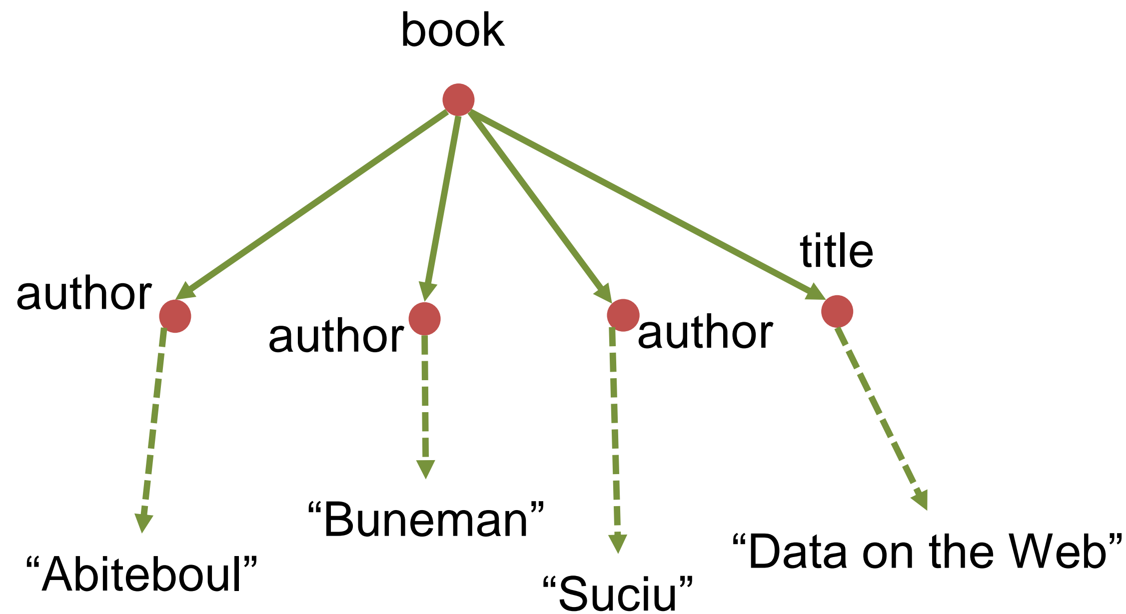
- Entities (objects) are represented as nodes
- Relationships are represented as edges
- Therefore, nodes and edges have associated types, and attributes

Many variations in circulation

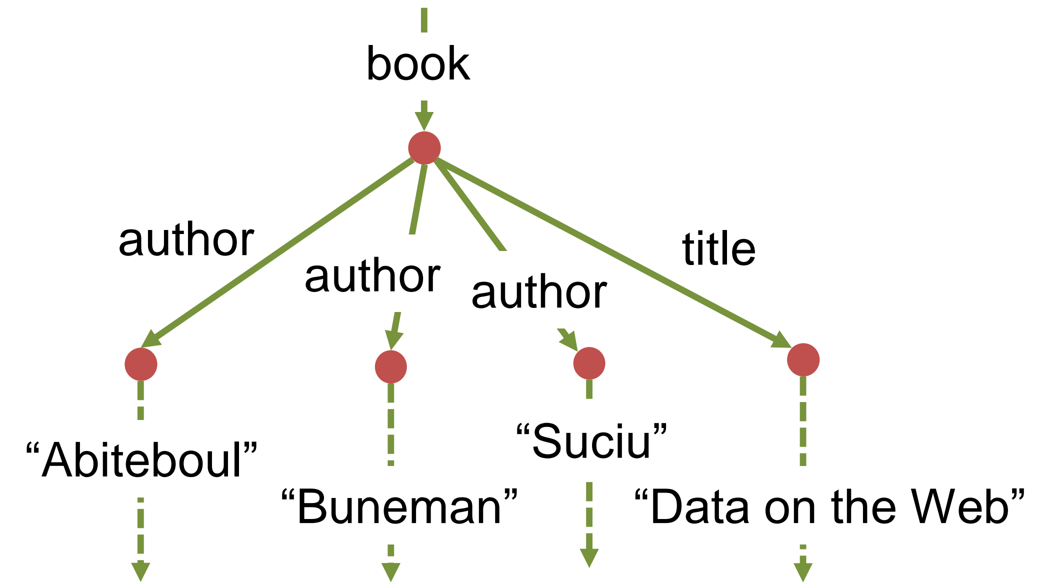
- Kind of edges?
  - Directed, undirected
- Where is data?
  - Only on nodes, only on edges, on both
- Shape of graph?
  - Arbitrary (has cycles), directed acyclic graph (DAG), tree

Two Schemes for Graph Modelling

Node-labeled scheme: nodes are labeled with types (book, author, title) and/or data (strings)



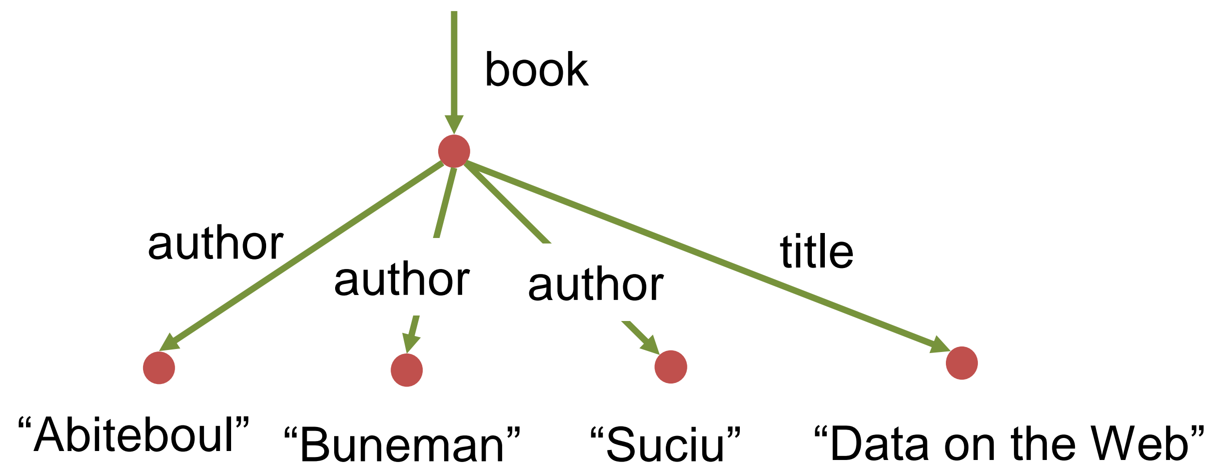
Edge-labeled scheme: edges are labeled with types (book, author, title) and/or data (strings)



Nodes and Edges both Labeled with Data and Type

A combination of **the node-labeled and edge-labeled schemes**:

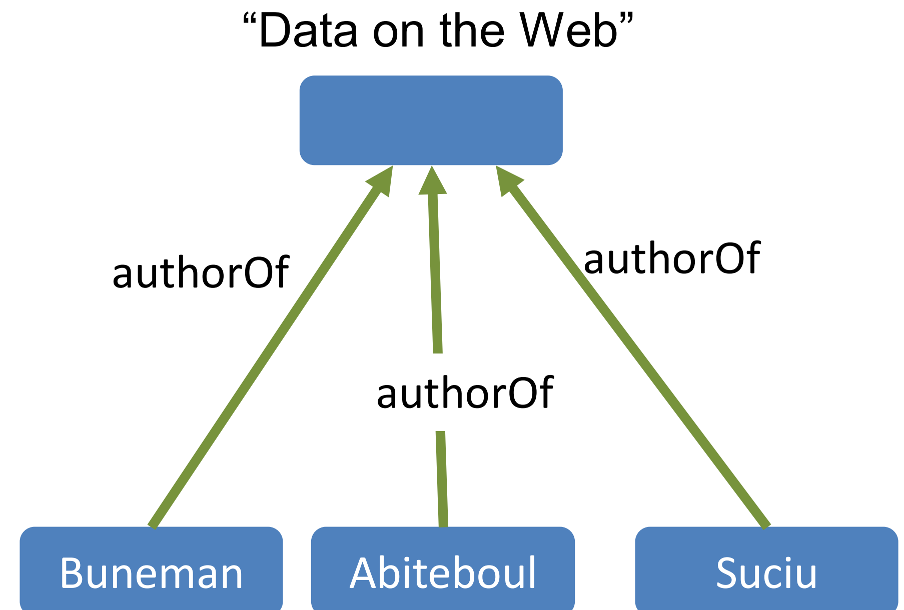
- both nodes and edges are labeled with types (book, author, title) and/or data (strings)
- E.g., node labels: *book*, edge labels: *author* and *title*, data: “Abiteboul”, “Buneman”, “Suciu”, and “Data on the Web”



Edge-Labeled Graph: RDF

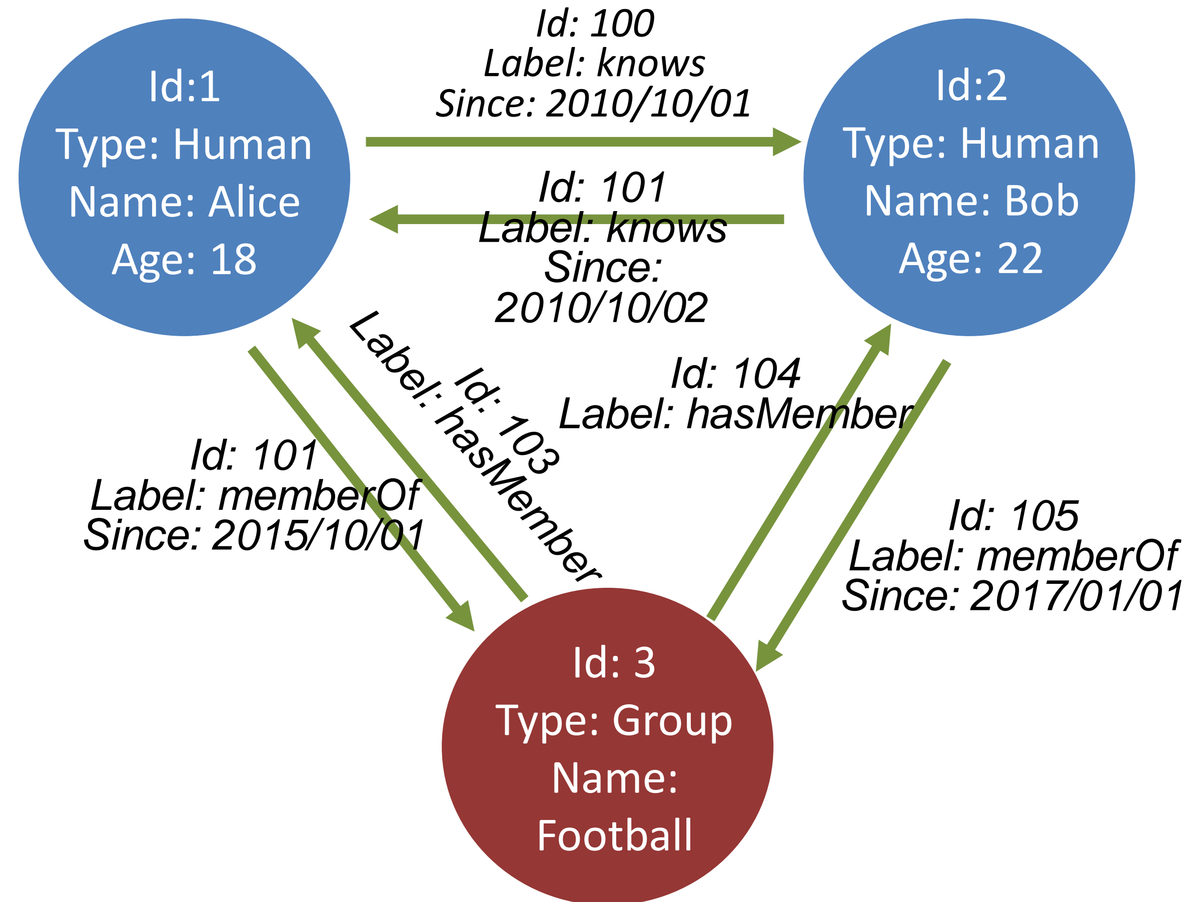
- Edge-labeled graph (N, E, L)
  - RDF triple: <subject, predicate, object>
  - Knowledge graph
  - Query language: SPARQL

RDF triples:
< Abiteboul, authorOf, "Data on the Web">
< Buneman, authorOf, "Data on the Web">
< Suciu, authorOf, "Data on the Web">



Node-Labeled Graph: Property Graph

- Property graph model (PGM)
  - Represents data as a directed, attributed multi-graph.
  - Vertices and edges are rich objects with a set of labels and a set of key-value pairs, so-called properties, e.g., *Type:Human*
  - Semantics of the directions is up to the applications
  - Cypher/openCypher, Gremlin, etc.



Multi-Model Database Systems

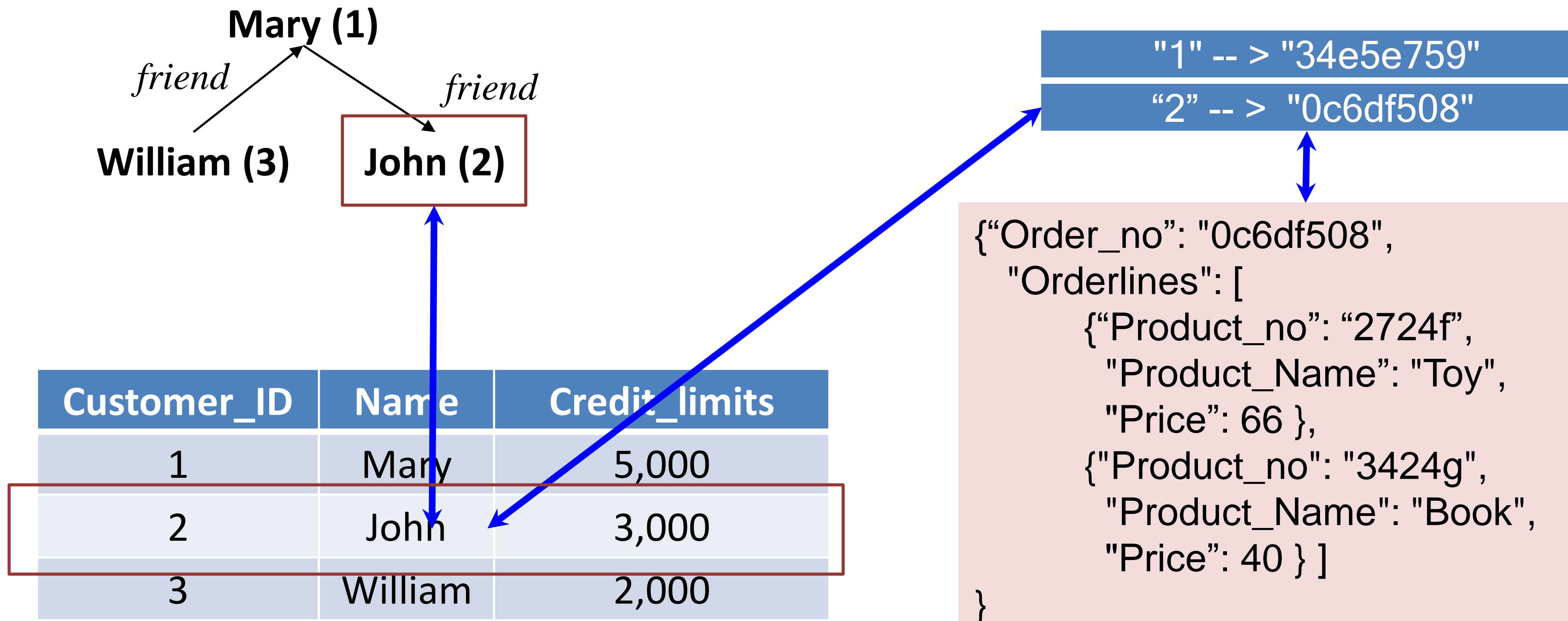
Rank (Apr 2023)	DBMS	Supported Data Models
1.	Oracle	Relational, Document, Graph, RDF, Spatial
2.	MySQL	Relational, Document, Spatial
3.	Microsoft SQL Server	Relational, Document, Graph
4.	PostgreSQL	Relational, Document, Spatial
5.	MongoDB	Documents, Spatial, Time Series, Search Engine
6.	Redis	KV, Document, Graph, Spatial, TS, Search Engine
7.	IBM Db2	Relational, Document, RDF, Spatial
8.	Elasticsearch	Search engine, Document, Spatial
9.	SQLite	Relational
10	Microsoft Access	Relational

By 2017, all leading operational DBMSs offer multiple data models, relational and NoSQL, in a single DBMS platform. **- Gartner report for operational databases 2016**

The **DB-Engines Ranking** ranks DBMSs according to their popularity. The ranking is updated monthly.

- 8 Multi-Model DBMSs in top-10 (124 out of 414 in total)

Multi-Model Data and Query



Recommendation query Q: Return all products which are ordered by a friend of a customer whose credit limit is over 3000!

Multi-Model Query in ArangoDB

ArangoDB is designed as a native multi-model database, supporting key/value, document and graph models.

```
LET CustomerIDs =(
        FOR Customer IN Customers
          FILTER Customer.CreditLimit > 3000
          RETURN Customer.id)
LET FriendIDs=(FOR CustomerID IN CustomerIDs
                    FOR Friend IN 1..1 OUTBOUND CustomerID Knows
                    RETURN Friend.id)

FOR Friend IN FriendIDs
  FOR Order IN 1..1 OUTBOUND Friend Customer2Order
  RETURN Order.orderlines[*].Product_no
```

Q: Return all products which are ordered by a friend of a customer whose credit limit is over 3000!

MMQ in OrientDB

OrientDB

- Supporting graph, document, key/value and object models.
- It supports schema-less, schema-full and schema-mixed modes.

```
SELECT EXPAND(OUT("Knows").Orders.orderlines.Product_no)
FROM   Customers
WHERE CreditLimit > 3000
```

Q: Return all products which are ordered by a friend of a customer whose credit limit is over 3000!

Challenges

Challenges are two-fold:

- Designing a language to express multi-model data queries (MMQs)
  - An MMQ is a mixture of the relational query, path query, graph pattern matching, etc.
- Cross-model query processing strategies
  - The **mediator-wrapper** fashion in Polystores/Multistores
    - Relies heavily on data exchange workflow and hence costly
  - A **holistic evaluation** in MMDB systems
    - In this tutorial, we focus on the techniques dealing with the *relational and graph data*
    - There is an emerging trend that a fusion of relational and graph database techniques

References

- Renzo Angles, Claudio Gutierrez. Survey of graph database models. ACM Comput. Surv. 40(1): 1:1-1:39 (2008)
- Paolo Atzeni, Francesca Bugiotti, Luca Cabibbo, Riccardo Torlone. Data modeling in the NoSQL world. Comput. Stand. Interfaces 67 (2020)
- E. F. Codd. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387, 1970.
- A. Deutsch and Y. Papakonstantinou. Graph data models, query languages and programming paradigms. Proc. VLDB Endow., 11(12):2106–2109, 2018.
- A. Doan, A. Y. Halevy, and Z. G. Ives. Principles of Data Integration. Morgan
- Kaufmann, 2012.
- J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. B. Zdonik. The bigdawg polystore system. SIGMOD Rec., 44(2):11–16, 2015.
- J. Lu and I. Holubová. Multi-model data management: What’s new and what’s next? In Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017, pages 602–605. OpenProceedings.org, 2017.
- J. Lu and I. Holubová. Multi-model Databases: A new journey to handle the variety of data. ACM Computing Surveys, 52(3), 2019.
- J. Lu, I. Holubová, and B. Cautis. Multi-model databases and tightly integrated polystores: Current practices, comparisons, and open challenges. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM ’18, pages 2301–2302, New York, NY, USA, 2018. Association for Computing Machinery.
- M. Saeed, M. Villarroel, A. Reisner, G. Clifford, L.-w. Lehman, G. Moody, T. Heldt, T. Kyaw, B. Moody, and R. Mark. Multiparameter Intelligent Monitoring in Intensive Care II (Mimic-II): A Public-Access Intensive Care Unit Database. Critical care medicine, 39:952–60, 05 2011.
- Marc H Scholl: Extensions to the relational data model. In: Advances in Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development, pp. 163–182 (1992)
- M. Stonebraker and J. Hellerstein. What Goes Around Comes Around. In "Readings in Database Systems" (aka the Red Book). 4th ed.

02 Multi-Model Queries

(20 minutes)

We will briefly present the multi-model queries and languages

- The relational query languages and their extensions
- The semi-structured query languages and their extensions
- The graph query languages and their extensions

Multi-Model Queries

A multi-model query (MMQ) may consist of the following types of fundamental queries

- Relational queries
- Graph pattern matching
- Path queries
- Aggregations
- Key-Value lookups
- …

An MMQ is a mixture of the above types of queries by cross-model joins

- No commonly accepted definition yet.

Relational Queries

- SWF syntax (SELECT-WHERE-FROM)
  - Select, Projection, Join (SPJ)
  - Conjunctive Queries (CQs)
- Aggregation
- Query languages:
  - Relational algebra (RA)
  - Relational calculus (RC)
  - SQL

Conjunctive query (CQ) :

- Written in conjunctive form (without using $\forall$, $\vee$, $\neg$):
  $q(x_1,\ldots,x_n) = \exists y_1.\exists y_2\ldots\exists y_p.(R_1(t_{11},\ldots,t_{1m}) \wedge \ldots \wedge R_k(t_{k1},\ldots,t_{km}))$
- Written in **Datalog** notation:
  $q(x_1,\ldots,x_n)$:- $R_1(t_{11},\ldots,t_{1m}), \ldots, R_k(t_{k1},\ldots,t_{km}))$

Formal Relational Query Languages

A query Language has equivalent expressive power with RA and RC is said to be **Relational Complete**.

- Relational Algebra
  - **Select**, **Project**, **Union**, **Set different**, **Cartesian product**, **Rename**
  - More operational(procedural), and always used as an **internal representation** for query evaluation plans
- Relational Calculus
  - **Tuple Relational Calculus**: filtering variable ranges over tuples {T | Condition}
    - Alpha: proposed by Codd in 1971; QUEL: INGRES 1975
    - { T.name | Author(T) AND T.article = 'database' }
  - **Domain Relational Calculus**: the filtering variable uses the domain of attributes instead of entire tuple values, $\{ a_1, a_2, a_3, ..., a_n \mid P (a_1, a_2, a_3, ... ,a_n)\}$
    - {< article, page, subject > | ∈ TutorialsPoint ∧ subject = 'database'}

SQL(Structured Query Languages)

- SQL is a standard language for querying and manipulating data
  - RA and RC form the basis for "real" languages like SQL
- SQL is a **very high-level (or declarative)** programming language
  - This works because it is optimized well!
- Many standards out there (vendors support various subsets):
  - ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ….
- Query syntax
  - SWF syntax (SELECT-WHERE-FROM)
    - Select, Projection, Join (SPJ)
  - Aggregation
  - Recursion (CTE)

NB: One the world's most successful programming language

CQs and Relational Queries

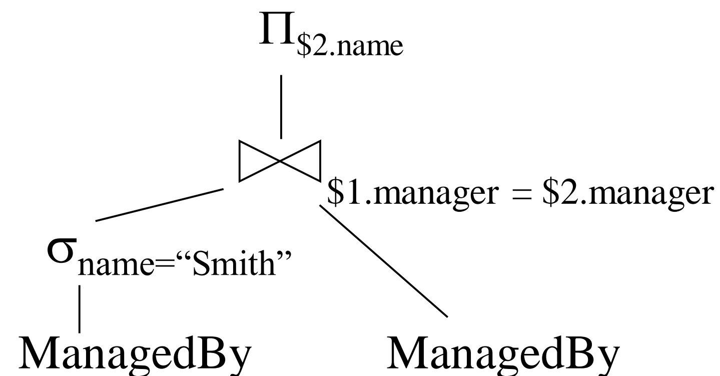
Are CQ queries precisely the SELECT-DISTINCT-FROM-WHERE queries ?

A(x) :- ManagedBy("Smith",y), ManagedBy(x,y)

```
SELECT DISTINCT m2.name
FROM ManagedBy m1, ManagedBy m2
WHERE m1.name="Smith" AND
      m1.manager=m2.manager
```

Relational Algebra:

- CQ correspond precisely to s_C, P_A, $\times$ (missing: $\cup$, $-$)



Graph Queries

Pioneered by academic work on CQ extensions for graphs (in the 90’s)

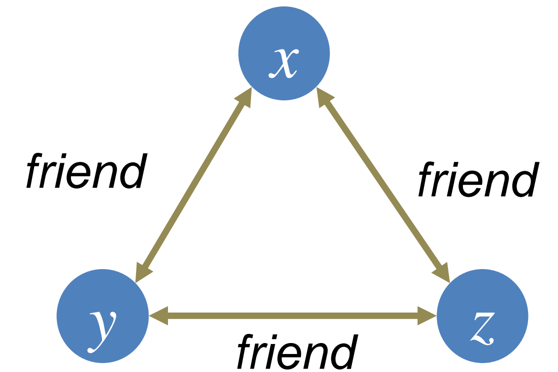
- **Graph pattern**
  - Small subgraph of interests
  - Can be also defined as conjunctive queries over the relational representation of graph data
  - (x, hasWon, Nobel), (x, hasWon, Booker)
- **Path query** for navigating along connected edges
  - x, citizenOf | ((bornIn | livesIn) locatedIn*), y
- **Variables** for manipulating data found during navigation
- **Aggregation** of data encountered during navigation

→ support for bag semantics as prerequisite

Graph Patterns

Graph pattern:

- V={x, y, z, …}, Alphabet Σ = {friend}
- {(x, friend, y), (y, friend, z), (z, friend, x), (y, friend, x), (z, friend, y), (x, friend, y)}
- E.g, in a social network one can match the pattern to look for a clique of three individuals that are all friends with each other



Semantics:

- The semantics of patterns is given using the notion of matching.
- A match of a pattern $P=(V_P, E_P)$ over a graph (V_G, E_G) is a **mapping** π from variables to constants.
- Semantics vary according to the mapping functions, such as **homomorphism** or **isomorphism**.

Graph Patterns as Relational Queries

- Given an alphabet Σ, we define σ(Σ) as the relational schema that consists of one binary predicate symbol E_a, for each symbol a ∈ Σ.
- Each graph database G=(V, E) can be represented as a relational instance D(G) over σ(Σ)
  - The database D(G) consist of all facts of the form Ea(v, v') such that (v, a, v') is an edge in G (we assume that D includes all the nodes in V)
  - CQ Q(x) = ∃y *ϕ(x, y), x and y are tuples of variables and ϕ(x, y) is a conjunction of relational atoms from* σ that use variables from x to y.
  - E.e., Q(x, y, z) = friend(x,y), friend(y,x), friend (x,z), friend(z, x), friend(y,z), friend(z,y)

Path Queries

- Express reachability via constrained paths

- Introduced initially in academic research in early 90s
  - StruQL (AT&T Research, Fernandez, Halevy, Suciu)
  - WebSQL (Mendelzon, Mihaila, Milo)
  - Lorel (Widom et al)

- Today supported by languages of commercial systems
  - XPath/XQuery, SQL++,
  - Cypher, SparQL, Gremlin, GSQL

Path Query Syntax

Various notations to express path queries

- Dot notation, e.g., SQL++, N1QL
- Axes notation, e.g., XPath/XQuery

Adopting here that of SparQL W3C Recommendation.

```
Path expressions  →  Edge label
                     |    _                  // wildcard, any edge label
                     |    ^ edge label       // inverse edge
                     |    path . path        // concatenation
                     |    path | path        // alternation
                     |    path*              // 0 or more reps
                     |    path*(min, max)    // at least min, at most max
                     |    (path)
```

Path Expression Examples

- Pairs of customer and product they bought:

  Bought

- Pairs of customer and product they were involved with (bought or reviewed)

  Bought|Reviewed

- Pairs of customers who bought same product (lists customers with themselves)

  Bought.^Bought

- Pairs of customers involved with same product (like-minded)

  (Bought|Reviewed).(^Bought|^Reviewed)

- Pairs of customers connected via a chain of like-minded customer pairs

  ((Bought|Reviewed).(^Bought|^Reviewed))*

- Bounded-length traversal

  friendOf*(1,3)

Regular Path Queries (RPQ)

The path query can be defined with various grammars, the most widely adopted one is RPQ:

- RQP(*x*, *y*) := (x, R, y), where R is a **regular expression** over the vocabulary of edge labels
- the semantics is defined in terms of *sets of node pairs* (x, y), where there exists a path in G from x to y whose concatenated labels spell out a word in L(PE)
- L(PE) = language accepted by PE when seen as regular expression over alphabet of edge labels

Construction of regular expressions:

- R ::= s | R.R | (R|R) | (R) | R? | R* | R+   // s element from S

Examples:

- Ancestors: isChildOf+
- Cousins: isChildOf, isChildOf, hasChild, hasChild

Conjunctive Regular Path Queries

RQPs can be further extended to **Conjunctive Regular Path Queries (CRPQs)**

- Replace relational atoms appearing in CQs with path expressions.
- Explicitly introduce variables binding to source and target nodes of path expressions.

- **Examples:**
  - Pairs of customers who have bought same product (do not list a customer with herself):

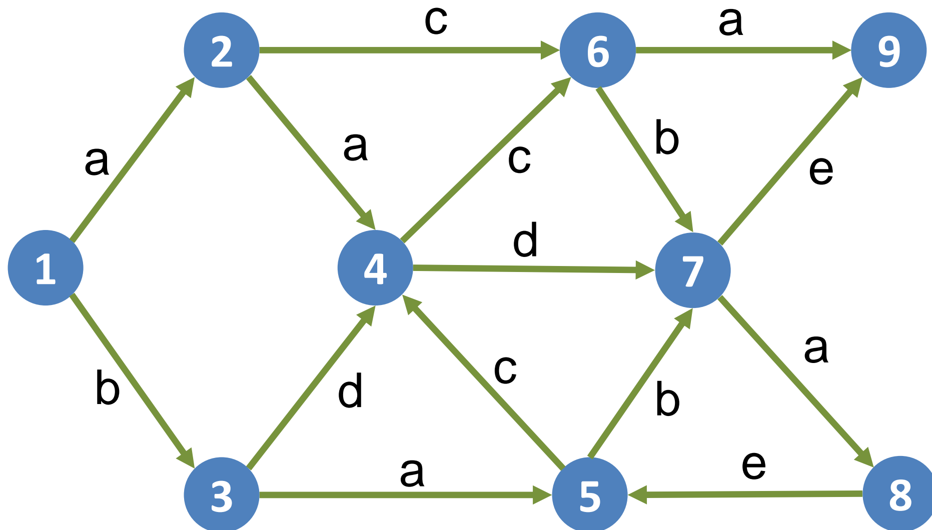
    Q1(c1,c2) :- c1 –Bought.^Bought-> c2, c1 != c2

  - Customers who have bought and also reviewed a product:

    Q2(c) :- c –Bought-> p, c –Reviewed-> p

ANS(x,y) := (x, hasWon, Nobel), (x, hasWon, Booker), (x, (citizenOf | ((bornIn | livesIn) locatedIn*)), y)

RPQ Examples



RPQ = a+(d|c)be

- *acbe: (2,4), (4,5), (5,7), (7,9)*
- *aacbe: (1,2), (2,4), (4,5), (5,7), (7,9)*

Pattern: (x, a, y), (y, e, z), (z, ?, x)

- triangle: (7,8), (8,5), (5,7)

CRPQ: (x, a, y), (y, e, z), (z, ?c+(d|b), x)

- *cycle: (7,8), (8,5), (5,7)*
- *cycle: (7,8), (8,5), (5,4), (4,7)*
- *cycle: (7,8), (8,5), (5,4), (4,6), (6,7)*

Case Study 1: SQL++

- SQL++ : A Backwards-Compatible SQL , which can access a SQL extension with nested and semi-structured data
- Queries exhibit XQuery and OQL abilities, yet backwards compatible with SQL-92
- Supports relation and JSON

- Simpler than XML and the XQuery data model
- Unlike labeled trees (the favorite XML abstraction of XPath and XQuery research) makes the distinction between tuple constructor and list/array/bag constructor

SQL++: http://arxiv.org/abs/1405.3631
http://db.ucsd.edu/wp-content/uploads/pdfs/375.pdf

SQL++ Data Model

Can think of as extension of SQL

- Extend with arrays + nesting + heterogeneity by following JSON's notation

```
{     Heterogeneous tuples in collections
 'location': 'Alpine',
 'readings': [          Array nested inside a tuple
  {
   'time': timestamp('2014-03-12T20:00:00'),
   'ozone': 0.035,
   'no2': 0.0050
  },
  {
   'time': timestamp('2014-03-12T22:00:00'),
   'ozone': 'm',
   'co': 0.4
  } ]
}
```

Arbitrary compositions of array, bag, tuple

Can also think of as extension of JSON

- Use single quotes for literals
- Extended with bags and enriched types

```
{                                  Bags {{ … }}
 'location': 'Alpine',
 'readings': {{
  {
   'time': timestamp('2014-03-12T20:00:00'),
   'ozone': 0.035,
   'no2': 0.0050                   Enriched types
  },
  {
   'time': timestamp('2014-03-12T22:00:00'),
   'ozone': 'm',
   'co': 0.4
  } }}
}
```

SQL++ Query Syntax

BNF Grammar for SQL++ queries

- Semi-structured query
- SFW query*:*
  - **SELECT-FROM-WHERE** (**SFW**)
  - Complex: tuple, collection *or* map
- Expression query:
  - Operator expressions
    - Path expression

<table>
<tr><th>SQL++ QUERY</th><th>→</th><th>SFW QUERY
| EXPRESSION_QUERY</th></tr>
<tr><td>SFW_QUERY</td><td>→</td><td>SELECT [DISTINCT] | [FROM] | [WHERE]
| [GROUP BY] | [HAVING]
| [ORDER BY] | [LIMIT] | [OFFSET]</td></tr>
<tr><td>EXPRESSION</td><td>→</td><td>OperatorExpression
| QuantifiedExpression</td></tr>
<tr><td>Operator Expression</td><td>→</td><td>PathExpression
| Operator OperatorExpression
| OperatorExpression Operator (OperatorExpression)?
|OperatorExpression <BETWEEN> OperatorExpression <AND> OperatorExpression</td></tr>
</table>

Path Navigation in SQL++

Two types path navigations:

1. **Tuple path navigation** *t.a* from the tuple *t* to its **attribute a** returns the value of a
2. **Array path navigation** *a*[*i*] returns the ***i*-th element** of the array *a*

```
<r:{ ci: 1.2, no: [0.5, 2] }>
```

```
@tuple_nav {absent: missing, type_mismatch: null}
@array_nav {absent: missing, type_mismatch: null}

([r.co, r.so, 7.co, r.no[1], r.no[3], r.co[1]])
```

Backwards Compatibility with SQL

Find sensors that recorded a temperature below 50:

```
readings : {{
  { sid: 2, temp: 70.1 },
  { sid: 2, temp: 49.2 },
  { sid: 1, temp: null }
}}
```

sid	temp
2	70.1
2	49.2
1	null

```
SELECT DISTINCT r.sid
FROM readings AS r
WHERE r.temp < 50
```

```
{{
  { sid : 2 }
}}
```

sid
2

Case Study 2: ArangoDB Query Language (AQL)

A native multi-model DBMS that supports

- Graph
- Key-value
- Json

Doing queries with AQL

- Data retrieval with filtering, sorting and more
- Simple graph queries
- Traversing through a graph with different options
- Shortest path queries

SQL	AQL
database	database
table	collection
row	document
column	attribute
table joins	collection joins
primary key	primary key (automatically present on _key attribute)
index	index

AQL query syntax

Query syntax (FOR-FILTER-RETUREN)

- Selecting all rows / documents from a table / collection, with all columns / attributes
- Filtering rows / documents from a table / collection, with projection
- Sorting rows / documents from a table / collection

```
FOR user IN users
  RETURN user
```

```
FOR user IN users
  FILTER user.active == 1
  RETURN {
    name: CONCAT(user.firstName, " ",
                 user.lastName),
    gender: user.gender
  }
```

```
FOR user IN users
  FILTER user.active == 1
  SORT user.name, user.gender
  RETURN user
```

AQL JOINS

ArangoDB has its own implementation of JOINS.

- **Inner join** can be expressed easily in AQL by nesting FOR loops and using FILTER statements:

```
FOR user IN users
  FOR friend IN friends
    FILTER friend.user == user._key
    RETURN MERGE(user, friend)
```

- **Outer join** are not directly supported in AQL, but can be implemented using subqueries:

```
FOR user IN users
  LET friends = (
    FOR friend IN friends
      FILTER friend.user == user._key
      RETURN friend
  )
  FOR friendToJoin IN (
     LENGTH(friends) > 0 ? friends :[ { } ]
            /* no match exists */
   )
   RETURN { user: user, friend: friend }
```

AQL Graph Traversal

- Traverse to the parents
- Traverse to the children
- Traverse to the grandchildren
- Traverse with variable depth

```
FOR v IN 1..1 OUTBOUND “Characters/2901776”
ChildOf
    RETURN v.name
```

```
FOR c IN Characters FILTER c.name == "Ned"
    FOR v IN 1..1 INBOUND c ChildOf
        RETURN v.name
```

```
FOR c IN Characters FILTER c.name == "Tywin"
    FOR v IN 2..2 INBOUND c ChildOf
        RETURN v.name
```

```
FOR c IN Characters FILTER c.name == "Joffrey"
    FOR v IN 1..2 OUTBOUND c ChildOf
        RETURN DISTINCT v.name
```

NB: This FOR loop doesn’t iterate over a collection or an array, it walks the graph and iterates over the connected vertices it finds, with the vertex document assigned to a variable (here: v).

Case Study 3: OrientDB Query Language (OrientQL)

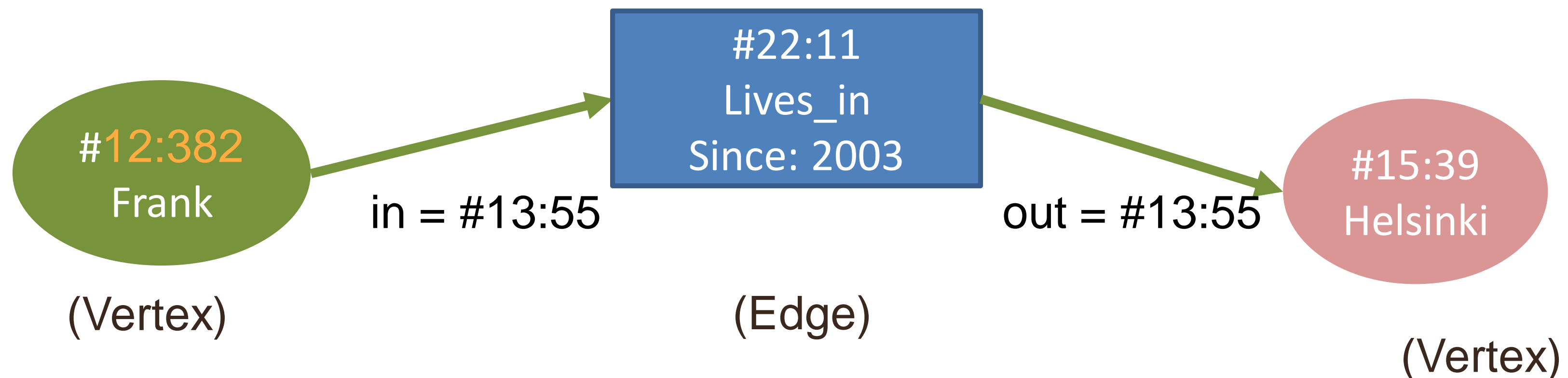
OrientDB is a Multi-Model Database

- Document, Graph, Spatial, FullText
- Tables -> Classes
- Extended SQL

- Each element (vertex and edge) is a JSON document
- Each element in the Graph has own immutable Record ID, such as #13:55, #22:11
- Connections use persistent pointers

{ "@rid": "12:382",
"@class": "Customer",
"name": "Frank",
"surname" : "Raggio",
"phone" : "+358 0402678479",
"details": {"city":"London",
"tags":"millennial" }
}

Data models



OrientQL

OrientDB supports SQL as a query language with some differences:

```
SELECT city, sum(salary) AS salary
FROM Employee
GROUP BY city
HAVING salary > 1000
```

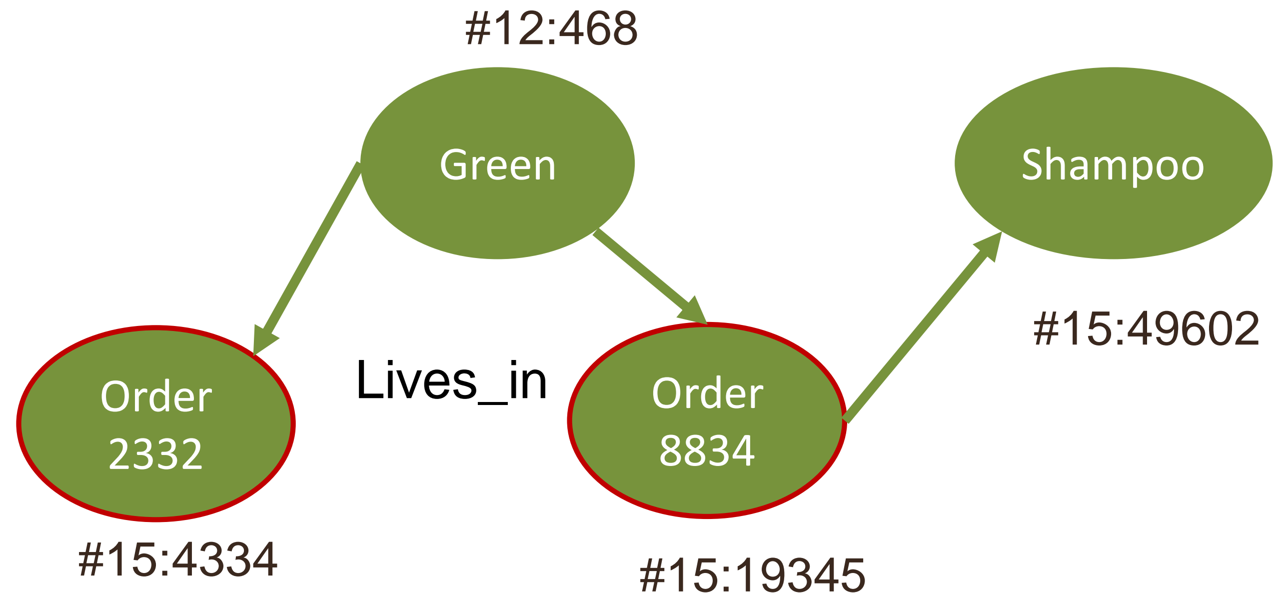
Q: Get all the outgoing vertices connected with edges with label (class) “Eats” and "Favourited" from all the Restaurant vertices in Rome

```
SELECT out('Eats', 'Favorited')
FROM Restaurant
WHERE city = 'Rome'
```

OrientQL Graph Traversal

```
SELECT expand( out() )
FROM #12:468
```

```
SELECT expand( out() )
FROM Customer
WHERE name = 'Green'
```



This uses an index to retrieve the starting vertex (#12:468) vertex

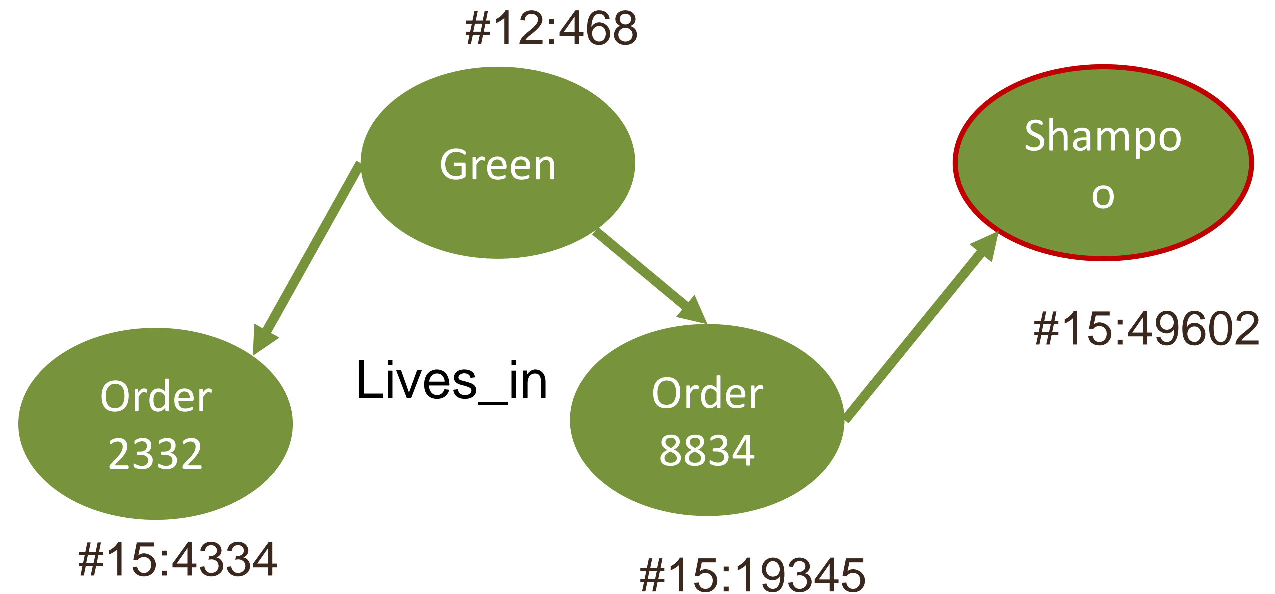
OrientQL: Graph Traversal

```
SELECT expand( out().out() )
FROM #12:468
```

```
SELECT expand( in().in() )
FROM #15:49602
```

```
SELECT expand( out().out() )
FROM Customer
WHERE name = ‘Green’
```

```
SELECT expand( in().in() )
FROM Product
WHERE name = ‘White Soap’
```



OrientDB Graph Traversal and Pattern Matching

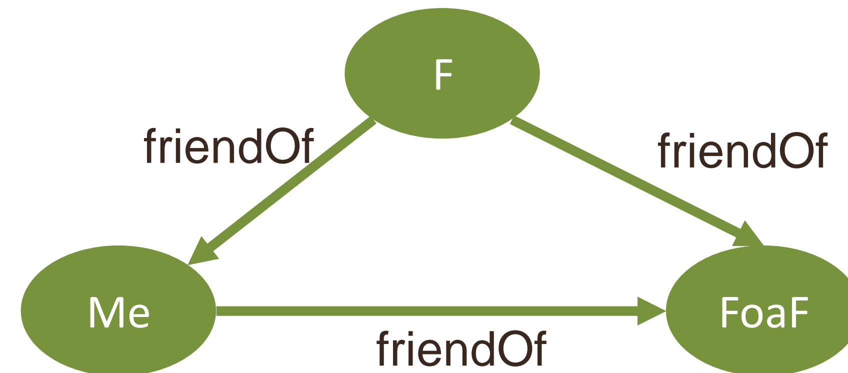
Traversal

In a social network-like domain, a user profile is connected to friends through links.

- TRAVERSE out("Friend")
- FROM #10:1234 WHILE $depth <= 3
- STRATEGY BREADTH_FIRST

Pattern Matching

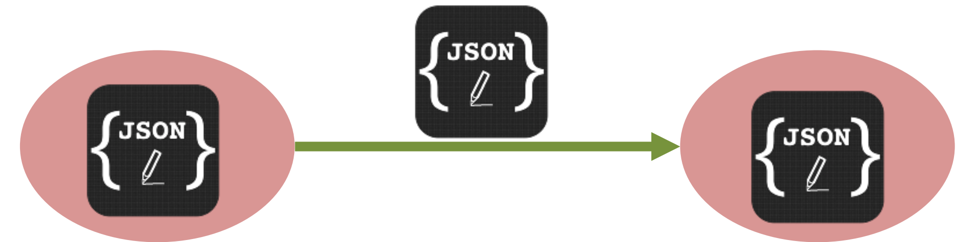
```
MATCH  {class: Person, WHERE: (name = ‘Abel’), AS: me}
      -friendOf->{}-friendOf>{AS: foaf},
          {AS: me}-friendOf->{AS: foaf}
RETURN me.name AS myName, foaf.name AS foafName
```



Case Study 4: AgensGraph Query Language (AgensQL)

A forked project of PostgreSQL (v9.6.2) supports

- Relational data, property graph, and JSON documents
- Integrated querying using SQL (Relational data) and Cypher (Graph data)
- Extended property graph model
- Data objects
  - Graph
  - Vertex and edge
  - Each vertex and edge can have a JSON document as its property



- **Label hierarchy**
  - Vertexes and edges can be grouped into labels (e.g. person, student, teacher, …)
  - Labels are organized as a hierarchy

RPQ with AgensGraph

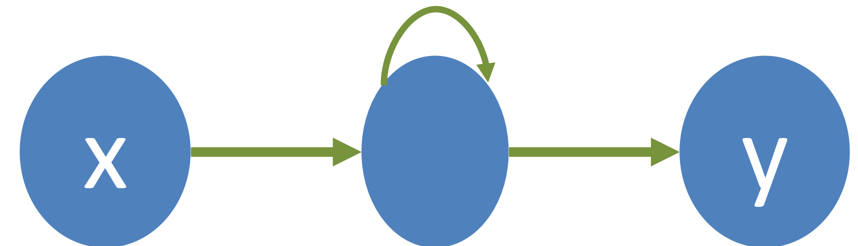
RPQ can be written as Variable-length Edge (VLE) Query

- Can be implemented using recursive common table expression (CTE) in SQL
- But CTE is inefficient for VLE query
  - Using CTE is BFS (Breadth First Search)-style processing
  - BFS processing needs to buffer intermediate results

VLE with Cypher:

```
MATCH    p=(x)-[:Parent*]->(y)
RETURN   (x), (y), length(p)
ORDER BY (y), (x),length(p)
```

```
MATCH (x)-[*1..5]->(y)
RETURN x, y;
```



Reference

- ArangoDB Query Language(AQL). https://www.arangodb.com/docs/stable/aql/index.html.
- C. Zhang and J. Lu. Holistic evaluation in multi-model databases benchmarking. Distributed and Parallel Databases, pages 1–33, 2019.
- C. Zhang, J. Lu, P. Xu, and Y. Chen. UniBench: A Benchmark for Multi-model Database Management Systems. In TPCTC ’18, Rio de Janeiro, Brazil, August 27-31, 2018, Revised Selected Papers, volume 11135 of Lecture Notes in Computer Science, pages 7–23. Springer, 2018.
- S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. R. Borkar, Y. Bu, M. J. Carey, I. Cetindil, M. Cheelangi, K. Faraaz, E. Gabrielova, R. Grover, Z. Heilbron, Y. Kim, C. Li, G. Li, J. M. Ok, N. Onose, P. Pirzadeh, V. J. Tsotras, R. Vernica, J. Wen, and T. Westmann. AsterixDB: A scalable, open source BDMS. Proc. VLDB Endow., 7(14):1905–1916, 2014.
- R. Angles, M. Arenas, P. Barceló, A. Hogan, J. L. Reutter, andD. Vrgoc. Foundations of modern query languages for graph databases. ACM Comput. Surv., 50(5):68:1–68:40, 2017.
- K. W. Ong, Y. Papakonstantinou, and R. Vernoux. The SQL++ semi-structured data model and query language: A capabilities survey of SQL-on-Hadoop, NoSQL and NewSQL databases. CoRR, abs/1405.3631, 2014.
- P. T. Wood. Query languages for graph databases. SIGMOD Rec., 41(1):50–60, 2012.

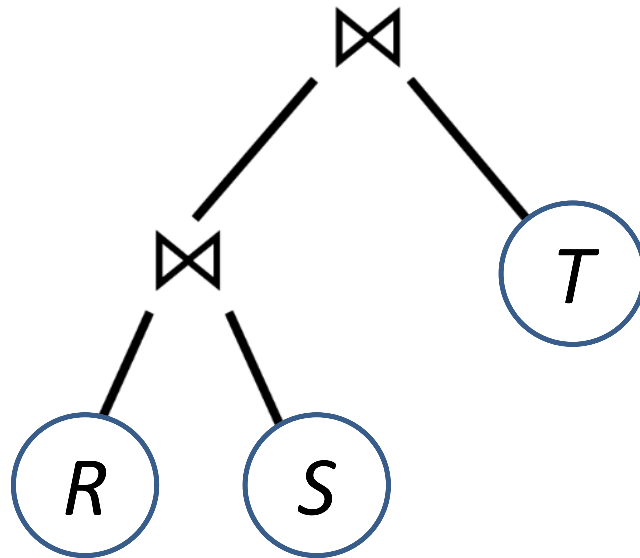
03 Join and Subgraph Matching

We will discuss different types of join algorithms, including:

- Binary joins
- Worst-case optimal joins
- Subgraph matching algorithms

Binary Joins

- Consider R(A, B) ⋈ S(B, C) ⋈ T(A, C)
  - Traditional database systems are typically only able to join two tables at once
  - Pick your two favorite tables and join them to get an intermediate relation, then join that with another table, and so on (until we get a single table)
  - This join process can be represented by a join tree



- Many commercial RDBMSs and GDBMSs adopt binary joins
- It is suboptimal when dealing with queries involving complex "cyclic joins" over many-to-many relationships, since the intermediate results might be unnecessarily large

Adapted from Justin Jaffray, A Gentle(-ish) Introduction to Worst-Case Optimal Joins (https://justinjaffray.com/a-gentle-ish-introduction-to-worst-case-optimal-joins/?try=2)

Joins are Secretly Graph Processing Algorithms

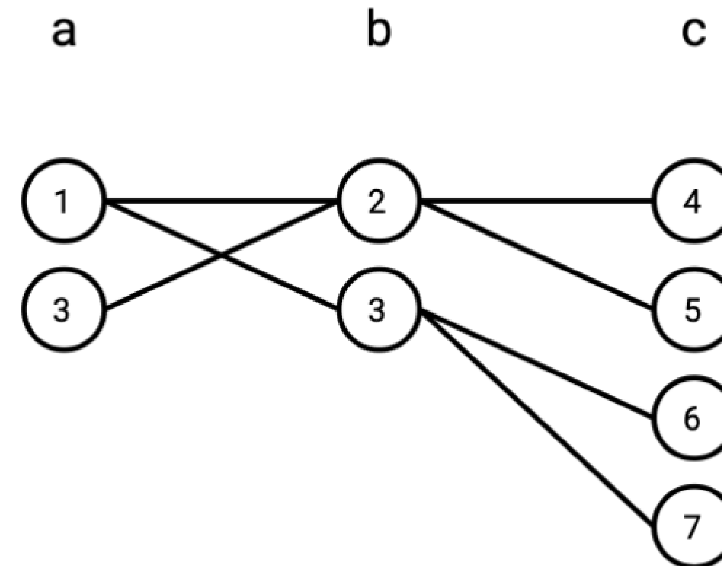
- Consider R(A, B) ⋈ S(B, C)
  - Represent these tables as a graph, where each named column corresponds to a typed set of vertices
  - If you enumerate all the paths that start from a vertex in a, go to a vertex in b, and wind up on a vertex in c, you'll find that set of such paths is precisely the join results (structure finding in graph)

R

a	b
1	2
3	2
1	3

S

b	c
2	4
2	5
3	6
3	7



Adapted from Justin Jaffray, A Gentle(-ish) Introduction to Worst-Case Optimal Joins (https://justinjaffray.com/a-gentle-ish-introduction-to-worst-case-optimal-joins/?try=2)

Worst-Case Optimal (WCO) Joins

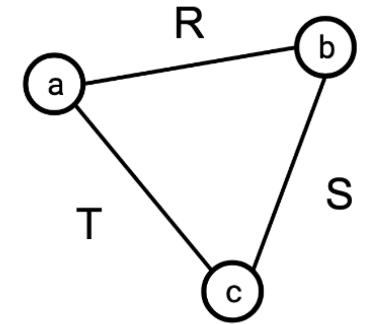
- Let us consider the triangle counting problem in a graph G
- Representing the graph as a table g(from, to)
- And join the table with itself twice (equivalent to $R(A, B) \bowtie S(B, C) \bowtie T(A, C)$)

```
SELECT
    g1.f AS a, g1.t AS b, g2.t AS c
FROM
    g AS g1, g AS g2, g AS g3
WHERE
    g1.t = g2.f AND g2.t = g3.t AND g1.f = g3.f;
```

- It turns out that a graph with O(n) edges will have no more than $O(n^{1.5})$ triangles in it
- For binary joins, there are graphs where that first intermediate join will always have $O(n^2)$ rows in it, no matter which two tables we choose to join first.

Adapted from Justin Jaffray, A Gentle(-ish) Introduction to Worst-Case Optimal Joins (https://justinjaffray.com/a-gentle-ish-introduction-to-worst-case-optimal-joins/?try=2)

Worst-Case Optimal (WCO) Joins

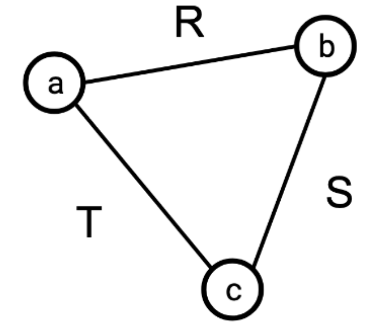


The query graph

- Let us reconsider R(A, B) ⋈ S(B, C) ⋈ T(A, C)
- **Column-at-a-time "Worst-case Optimal" Join Algorithms**
  - Instead of picking a join order for tables, we pick a column order and perform the join column at a time
  - Step 1: Find all a's. Here we will just take all nodes as possible a values.
  - Step 2: For each a value, e.g., a=1, we extend it to find all ab's that can be part of triangles: Here we use the forward index to look up all b values for node with ID 1. This will generate the second intermediate relation.
  - Step 3: For each ab value, e.g., the tuple (a=1 b=0), we will intersect all c's with a=1, and all c's with b=0 (k-way intersections). That is, we will intersect the backward adjacency list of the node with ID 1, and forward adjacency list of the node with ID 0. If the intersection is non-empty, we produce some triangles.

Adapted from Justin Jaffray, A Gentle(-ish) Introduction to Worst-Case Optimal Joins (https://justinjaffray.com/a-gentle-ish-introduction-to-worst-case-optimal-joins/?try=2), and Semih Salihoğlu, Why (Graph) DBMSs Need New Join Algorithms: The Story of Worst-case Optimal Join Algorithms (https://kuzudb.com/blog/wcoj.html)

Worst-Case Optimal (WCO) Joins



- Let us reconsider R(A, B) ⋈ S(B, C) ⋈ T(A, C)
- **Column-at-a-time "Worst-case Optimal" Join Algorithms**
  - Instead of picking a join order for tables, we pick a column order and perform the join column at a time

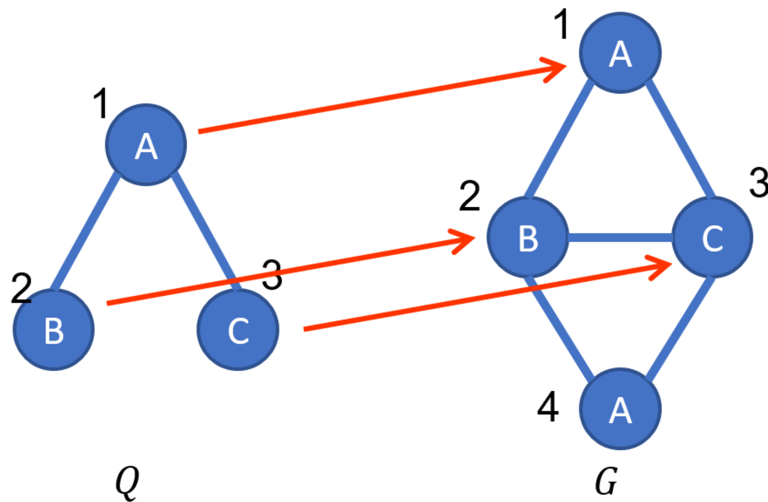
Worst-case optimal:

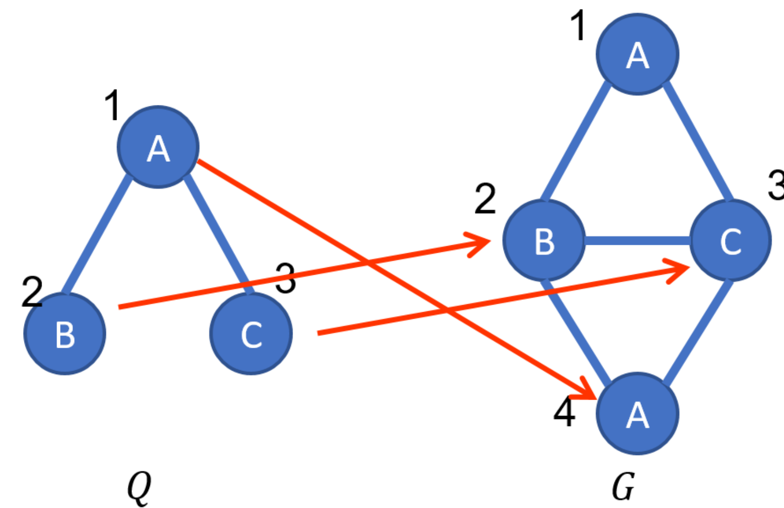
- Let IN denote the input size of the query Q
- The computational cost is IN^{ρ^*}, where ρ^* is the fractional edge cover number of Q (the AGM bound)
- For the above query, the cost is $O(N^{1.5})$

Adapted from Justin Jaffray, A Gentle(-ish) Introduction to Worst-Case Optimal Joins (https://justinjaffray.com/a-gentle-ish-introduction-to-worst-case-optimal-joins/?try=2), and Semih Salihoğlu, Why (Graph) DBMSs Need New Join Algorithms: The Story of Worst-case Optimal Join Algorithms (https://kuzudb.com/blog/wcoj.html)

Subgraph Matching

- Subgraph Isomorphism: Given a query Q and a data graph G, Q is subgraph isomorphism to G, if and only if there exists an injective function $f: V(Q) \rightarrow V(G)$, such that
  - $\forall\, u \in V(Q), f(u) \in V(G), L_V(u) = L_V\big(g(u)\big)$, where $V(Q)$ and $V(G)$ denotes all vertices in Q and G, respectively; and $L_V(\cdot)$ denotes the corresponding vertex label.
  - $\forall \overline{u_1 u_2} \in E(Q), \overline{f(u_1) f(u_2)} \in E(G), L_E(\overline{u_1 u_2}) = L_E\big(\overline{f(u_1) f(u_2)}\big)$





Subgraph Matching

- Subgraph Isomorphism: Given a query Q and a data graph G, Q is subgraph isomorphism to G, if and only if there exists an injective function $f: V(Q) \rightarrow V(G)$, such that
  - $\forall\, u \in V(Q), f(u) \in V(G), L_V(u) = L_V\big(g(u)\big)$, where $V(Q)$ and $V(G)$ denotes all vertices in Q and G, respectively; and $L_V(\cdot)$ denotes the corresponding vertex label.

- Subgraph Isomorphism Testing is NP-complete
  - Decide whether there is a subgraph of G that is isomorphic to Q
- Enumerating all subgraph isomorphic embeddings is NP-hard
  - Many techniques have been developed for efficient enumeration in practice

4 A

Q G

Subgraph Matching – *Ullman Algorithm*

- Given two graphs Q and G, their corresponding matrices are $MA_{n\times n}$ and $MB_{m\times m}$.
- Goal: 1) Find matrix $M'_{n\times m}$ such that $MC{=}M'(M' \cdot MB)^T$ $\forall i,j,\ MA[i][j]=1 \rightarrow MC[i][j]=1$
  2) or report no such matrix M'.

	1	2	3
1	0	1	1
2	1	0	0
3	1	0	0

MA

	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	1
4	0	1	1	0

MB

	1	2	3	4
1	0	0	0	1
2	0	1	0	0
3	0	0	1	0

M'

	4	2	3
4	0	1	1
2	1	0	1
3	1	1	0

$MC = M'(M' \bullet MB)^T$

MA: the adjacency matrix of query Q
MB: the adjacency matrix of graph G
M': the matching matrix, which specifies the isomorphism from Q to a subgraph of G if it exists.
(M' specifies an subgraph isomorphism from Q to G.)

1) M'[i][j] = 1 means that the i-th vertex in Q corresponds to j-th vertex in query G;
2) Each row in M' contains exactly one 1;
3) No column contains more than one 1.

Julian R. Ullmann: An Algorithm for Subgraph Isomorphism. J. ACM 23(1): 31-42 (1976)

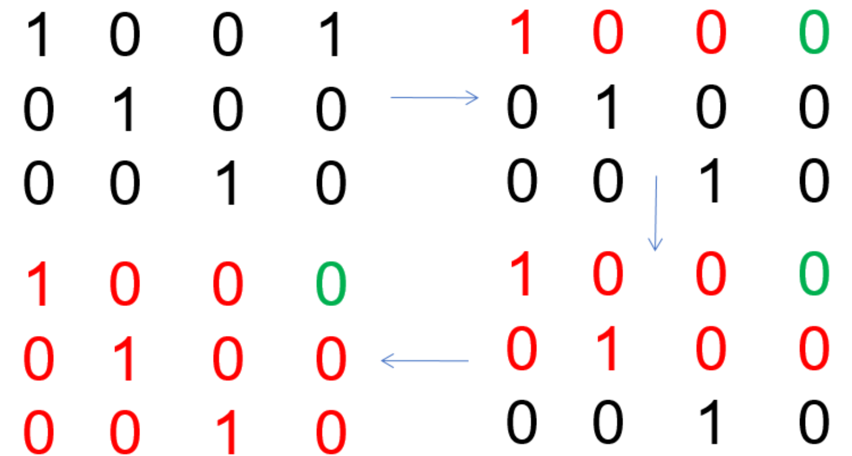
Subgraph Matching – *Ullman Algorithm*

- Step 1. Set up matrix $M_{n\times m}$, such that M [i][j]=1, if 1) the i-th vertex in Q has the same label as the j-th vertex in G; and 2) the i-th vertex in Q has smaller vertex degree than the j-th vertex in G.
- Step 2. Matrices M' are generated by systematically changing to 0 all but one of the 1's in each of the rows of M, subject to the definitory condition that no column of a matrix M' may contain more than one 1 (the maximal depth is |MA|).
- Step 3. Verify matrix M' by the following equation:

$$MC = M'(M' \cdot MB)^T$$

$$\forall i,j\ MA[i][j] = 1 \rightarrow MC[i][j] = 1$$

- Iterate the above steps and enumerate all possible matrixes M'.



Julian R. Ullmann: An Algorithm for Subgraph Isomorphism. J. ACM 23(1): 31-42 (1976)

Subgraph Matching – *Ullman Algorithm*

- Step 1. Set up matrix $M_{n \times m}$, such that M [i][j]=1, if 1) the i-th vertex in Q has the same label as the j-th vertex in G; and 2) the i-th vertex in Q has smaller vertex degree than the j-th vertex in G.
- Step 2. Matrices M' are generated by systematically changing to 0 all but one of the 1's in each of the rows of M, subject to the definitory condition that no column of a matrix M' may contain more than one 1 (the maximal depth is |MA|).

- Neighborhood Connection Pruning
  - Let the i-th vertex *v* in *Q* corresponds to the j-th vertex *u* in *G*. Each neighbor vertex of *v* in *Q* must correspond to some neighbor vertex of *u* in *G*. Otherwise, v cannot correspond to u.

Julian R. Ullmann: An Algorithm for Subgraph Isomorphism. J. ACM 23(1): 31-42 (1976)

Subgraph Matching – *VF2 Algorithm*

- Considering two graph Q and G, the (sub)graph isomorphism from Q to G is expressed as the set of pairs (n, m) (with $n \in Q$ and $m \in G$)
- Let s be an intermediate state. Actually, s denotes a partial mapping from Q to G, namely, a mapping from a subgraph of Q to a subgraph of G. These two subgraphs are denoted as $Q(s)$ and $G(s)$, respectively.
- All neighbor vertices to $Q(s)$ in graph Q are denoted as $NQ(s)$, and all neighbor vertices to $G(s)$ in graph G are denoted as $NG(s)$. Candidate pair sets are a subset of $NQ(s) \times NG(s)$. Apply structural feasibility rules to prune unpromising candidate pairs.
  - E.g., neighbor connection

$$F(s,n,m) = F_{structure}(s,n,m) \wedge F_{label}(s,n,m)$$

$$F(s,n,m) \Leftrightarrow (\forall n' \in (V_1(s) \cap N_1(n,Q)))$$

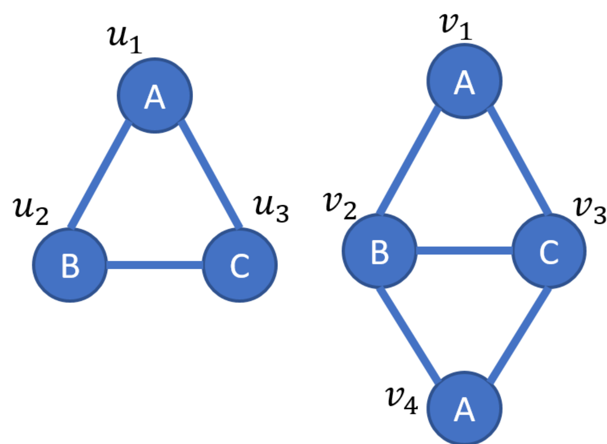
$$\exists m' \in (V_2(s) \cap N_2(m,G))$$

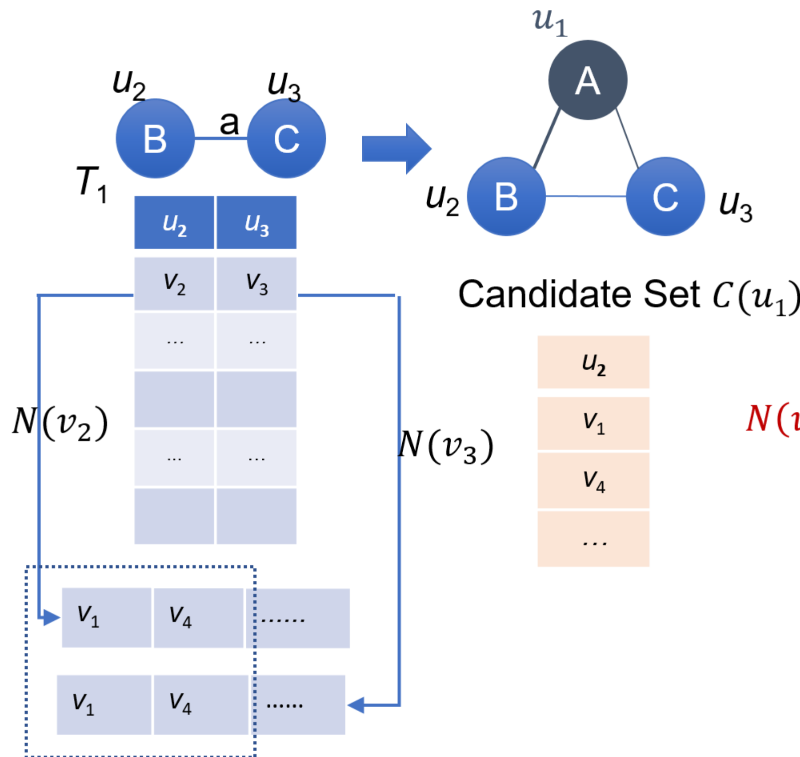
$N_1(n, Q)$: The neighbors of vertex n in graph Q;
$N_2(m, G)$: The neighbors of vertex m in graph G;

Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, Mario Vento: A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. IEEE Trans. Pattern Anal. Mach. Intell. 26(10): 1367-1372 (2004)

Subgraph Matching – *Multi-Way Join*

- Recall that a subgraph query Q is equivalent to a multiway self-join query over edge tables
- Worst-case optimal join





set intersection

$N(v_2) \cap N(v_3) \cap C(u_1) = \{v_1, v_4\}$

u_1	u_2	u_3
v_1	v_2	v_3
v_4	v_2	v_3

Its running time complexity is $O(N^{1.5})$, matching the worst case output size.

A. ATSERIAS, M. GROHE and D. MARX, “Size bounds and query plans for relational joins,” FOCS 2008.

Subgraph Matching

- A Summary of representative subgraph matching algorithms

<table>
<tr><th colspan="2" rowspan="2">Methodology</th><th colspan="2">Algorithms and Systems</th></tr>
<tr><th>Sequential</th><th>Parallel</th></tr>
<tr><td colspan="2">Backtracking Search</td><td>Ullman, VF2, QuickSI, GADDI,
SPath, GraphQL, TurboISO, BoostISO, CFL, SGMatch, CECI, DP-iso</td><td>PGX, PSM, STwig</td></tr>
<tr><td rowspan="2">Multi-way Join</td><td>Pair-wise Join</td><td>PostgreSQL, MonetDB, Neo4J</td><td>GpSM, GSI</td></tr>
<tr><td>Worst-Case Optimal Join</td><td>LogicalBlox, gStore</td><td>EmptyHeaded, GraphFlow</td></tr>
</table>

Equivalence between Join and Subgraph Matching

- We have discussed the equivalence in previous slides…
- The equivalence has been observed in a bunch of studies…
  - …by using the standard relational algebra, a graph traversal has to be represented as a sequence of joins. [EDBT/ICDT 2016 Workshops]
  - …we discuss the alternative approach of using graph exploration, instead of substructure joins, to answer subgraph matching queries. [VLDB 2012]
  - The execution process of join operations can be considered as explorations over links in an entity-relationship graph. [VLDB 2016]
  - …subgraph matching is equivalent to multi-way joins between base Vertex and base Edge tables on ID attributes. [SIGMOD-GRADES&NDA 2021]
  - …a subgraph query Q is equivalent to a multi-way self-join query that contains one E(ai,aj) (for Edge) relation for each ai→aj ∈ E_Q. [VLDB 2019]

Reference

- Justin Jaffray, A Gentle(-ish) Introduction to Worst-Case Optimal Joins (https://justinjaffray.com/a-gentle-ish-introduction-to-worst-case-optimal-joins/?try=2)
- Semih Salihoğlu, Why (Graph) DBMSs Need New Join Algorithms: The Story of Worst-case Optimal Join Algorithms (https://kuzudb.com/blog/wcoj.html)
- Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, Mario Vento: A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. IEEE Trans. Pattern Anal. Mach. Intell. 26(10): 1367-1372 (2004)
- A. ATSERIAS, M. GROHE and D. MARX, "Size bounds and query plans for relational joins," FOCS 2008.
- Julian R. Ullmann: An Algorithm for Subgraph Isomorphism. J. ACM 23(1): 31-42 (1976)
- Jiang, Chuntao, Frans Coenen, and Michele Zito. "A survey of frequent subgraph mining algorithms." The Knowledge Engineering Review 28.1 (2013): 75-105.
- Hölsch, Jürgen, and Michael Grossniklaus. "An algebra and equivalences to transform graph patterns in neo4j." EDBT/ICDT 2016 Workshops: EDBT Workshop on Querying Graph Structured Data (GraphQ). 2016.
- Sun Z, Wang H, Wang H, et al. Efficient subgraph matching on billion node graphs[J]. VLDB 2012.
- Ma, Hongbin, et al. "G-SQL: Fast query processing via graph exploration." Proceedings of the VLDB Endowment 9.12 (2016): 900-911.
- Mhedhbi, Amine, et al. "LSQB: a large-scale subgraph query benchmark." Proceedings of the 4th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). 2021.
- Amine Mhedhbi, Semih Salihoglu: Optimizing Subgraph Queries by Combining Binary and Worst-Case Optimal Joins. Proc. VLDB Endow. 12(11): 1692-1704 (2019)

04 Fusion of Query Processing Techniques

- Relational database techniques for graph queries
- Graph techniques for relational queries

Motivation of RDBMSs Supporting Graph Processing

- ***Graph processing (e.g., various analytics) is getting increasingly popular!***
  - Social networks, transportation networks, ad networks, e-commerce, web search, …
- In many real-world scenarios, data is collected and stored in a relational database
  - Using specialized graph engines -> First need to dump data from RDBMSs with pre- and post-processing
- Limited capacity of specialized graph processing systems compared to RDBMSs
  - Transactions, checkpointing and recovery, fault tolerance, durability, integrity constraints
- "Relational" vs "graph" distinction is blurry
  - Most structured data can be modeled as relations or graph
- Advances of relational data analytics
  - E.g., column-oriented databases

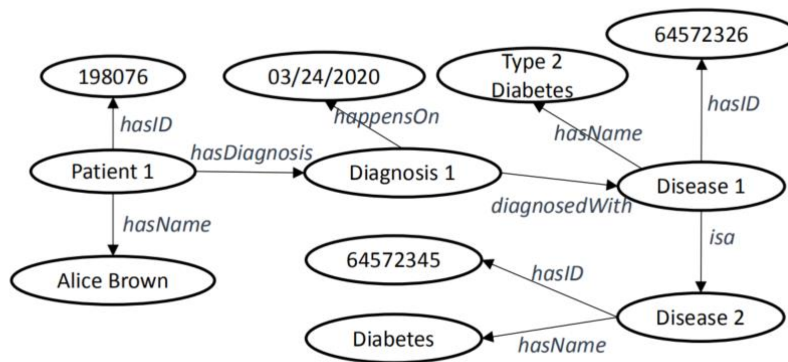
The World of Graph Databases from An Industry Perspective

- Two different types of graph workloads
  - –Graph queries/Graph OLTP
    - Low-latency graph traversal and pattern matching; typically only touch small local regions of a graph
    - E.g., 2-hop neighbors, single-pair shortest path
  - –Graph algorithms/Graph OLAP/Graph analytics
    - Typically iterative, long running processing on the entire graph
    - Graph ML, e.g., Graph Neural Networks (GNNs)

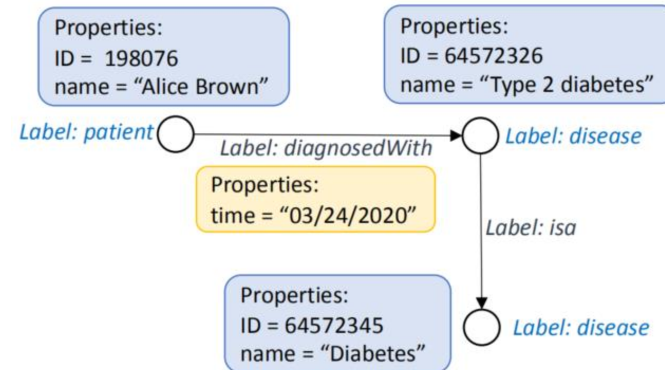
Tian, Yuanyuan. "The World of Graph Databases from An Industry Perspective." *ACM SIGMOD Record* 51.4 (2023): 60-67.

The World of Graph Databases from An Industry Perspective

- Two prominent graph models
  - RDF Model (W3C standard)
    - Directed edge-labeled graph, represented by the subject–predicate–object (s, p, o) triples
  - Property Graph Model
    - Vertex and edge can have arbitrary number of properties and can also be tagged with labels



(a) RDF Model



(b) Property Graph Model

The World of Graph Databases from An Industry Perspective

- Query languages for graph OLTP
  - RDF Model: SPARQL
  - Property graphs: Tinkerpop Gremlin, Cypher/openCypher (Neo4j), PGQL (Oracle), GSQL (TigerGraph), G-Core (LDBC), GQL (ISO/IEC)
  - Imperative vs. declarative: Gremlin is the only imperative query language
  - Turing complete? (Gremlin, GSQL)
- Query languages for graph OLAP
  - No standard language or API
  - Most vendors support Pregel-like API
  - A library of build-in graph algorithms is acceptable

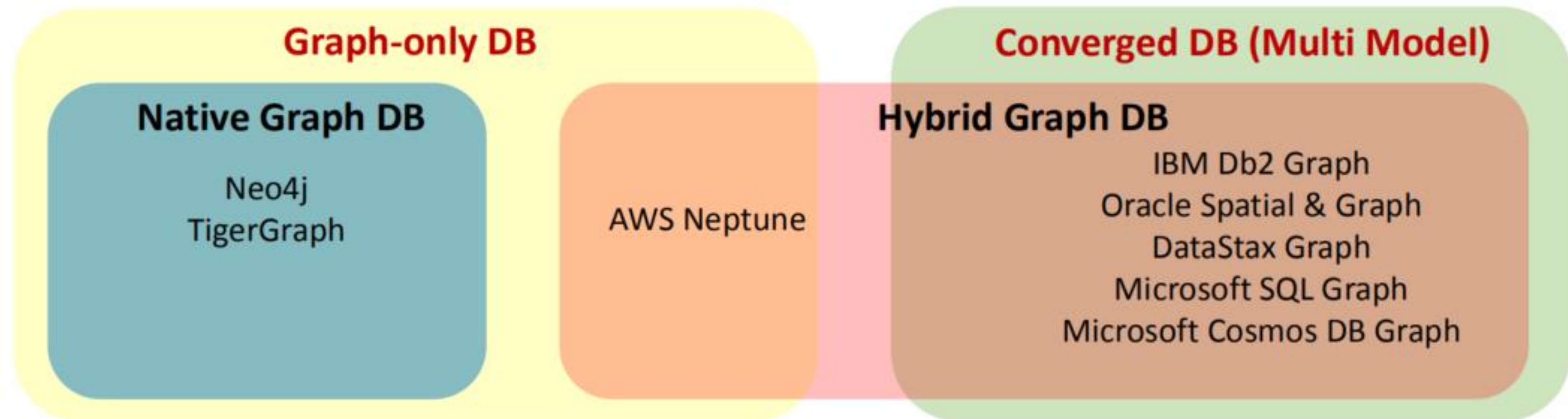
The World of Graph Databases from An Industry Perspective

- Graph Databases

		Deployment	Graph Model	Graph OLTP			Graph OLAP	Scale-Out
				Query Language	Visualization tools	Transaction		
Graph Only Companies	**TigerGraph**	On-prem / AWS, Azure, GCP	PG	GSQL	Graph Studio	ACID	GSQL, 23 built-in algorithms	Yes
	Neo4J	On-prem / AWS, Azure, GCP	PG	Cypher	Studio	Non-repeatable reads may occur	Pregel API, 48 built-in algorithms (including Graph ML)	Yes
Data Companies	**DataStax** Enterprise Graph	On-prem / AWS, Azure, GCP	PG	Gremlin	Studio	Row-level (Cassandra)	SparkGraphComputer API	Yes
	Databricks GraphX & GraphFrames	On-prem / AWS, Azure, GCP	PG	Motif Finding DSL	-	-	Pregel API, 7 built-in algorithms	Yes
Enterprise Cloud Companies	**Amazon** Neptune	AWS	PG, RDF	Gremlin, SPARQL	Neptune Workbench	ACID	-	Yes
	Microsoft SQL Graph	On-prem / Azure	PG	SQL Extension	Power BI plugin, 3rd party tools	ACID	Python/R scripts via Machine Learning Services	Yes (Read-Only Queries)
	Microsoft Cosmos DB Graph	Azure	PG	Gremlin	Azure Portal, 3rd party tools	-	-	Yes
	Oracle Spatial and Graph	On-prem / OCI AWS, Azure, GCP	PG, RDF	PGQL, SPARQL	Graph Studio	ACID	Green Marl DSL, 50+ built-in algorithms (including Graph ML)	Yes
	IBM Db2 Graph	On-prem / CP4D	PG	Gremlin	Graph UI	ACID	-	Yes

The World of Graph Databases from An Industry Perspective

- Graph database solution space
  - Native graph DB vs. hybrid graph DB
  - Graph-only DB vs converged (i.e., multi-model) DB
- Advantages of native/graph-only DB: efficiency, Graph OLAP, …
- Advantages of hybrid/converged DB come from the backend data store (transactions, access control, high availability, disaster recovery, …)



The World of Graph Databases from An Industry Perspective

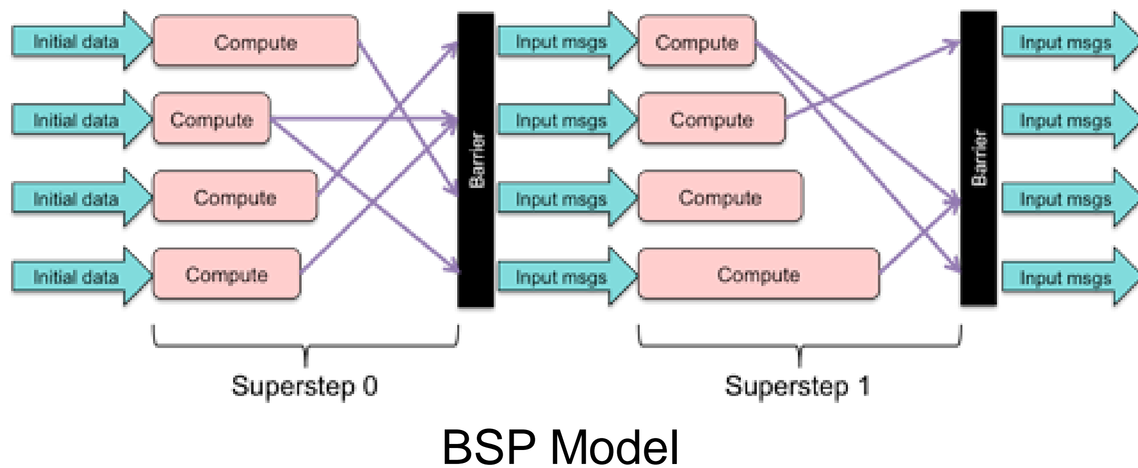
- Graph benchmarks
  - No standard benchmarks like TPC-C/H/DS
  - Linked Data Benchmark Council (LDBC), e.g., SNB
  - Linkbench from Facebook
  - Graph500
  - Open Graph Benchmark (OGB) for graph ML

Overview of Relational Database Techniques for Graph Processing

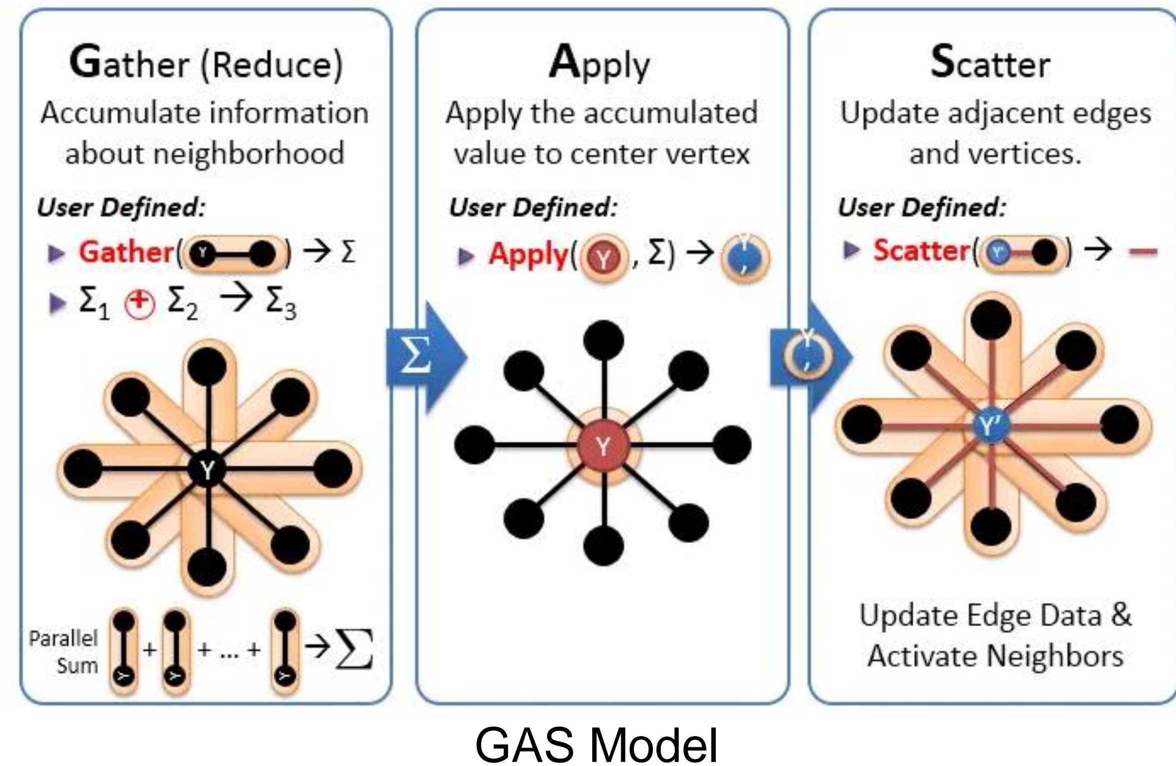
- Allow users to think in terms of a graph with an (unmodified) relational database
  - E.g., with the vertex-centric programming interface
- Support graph analytic processing by SQL and relational algebra
- Improve graph queries (i.e., subgraph matching) via more efficient join algorithms (e.g., worst-case optimal join)

Vertex-Centric Graph Processing

- Popular for graph analytics
- Thinking like a vertex: processing logic applies on a vertex level and communicate via message passing
  - Programmer only specifies a vertex program
  - System takes care of running it in parallel
- Bulk Synchronous Parallel (BSP) model
- Gather-Apply-Scatter (GAS) model



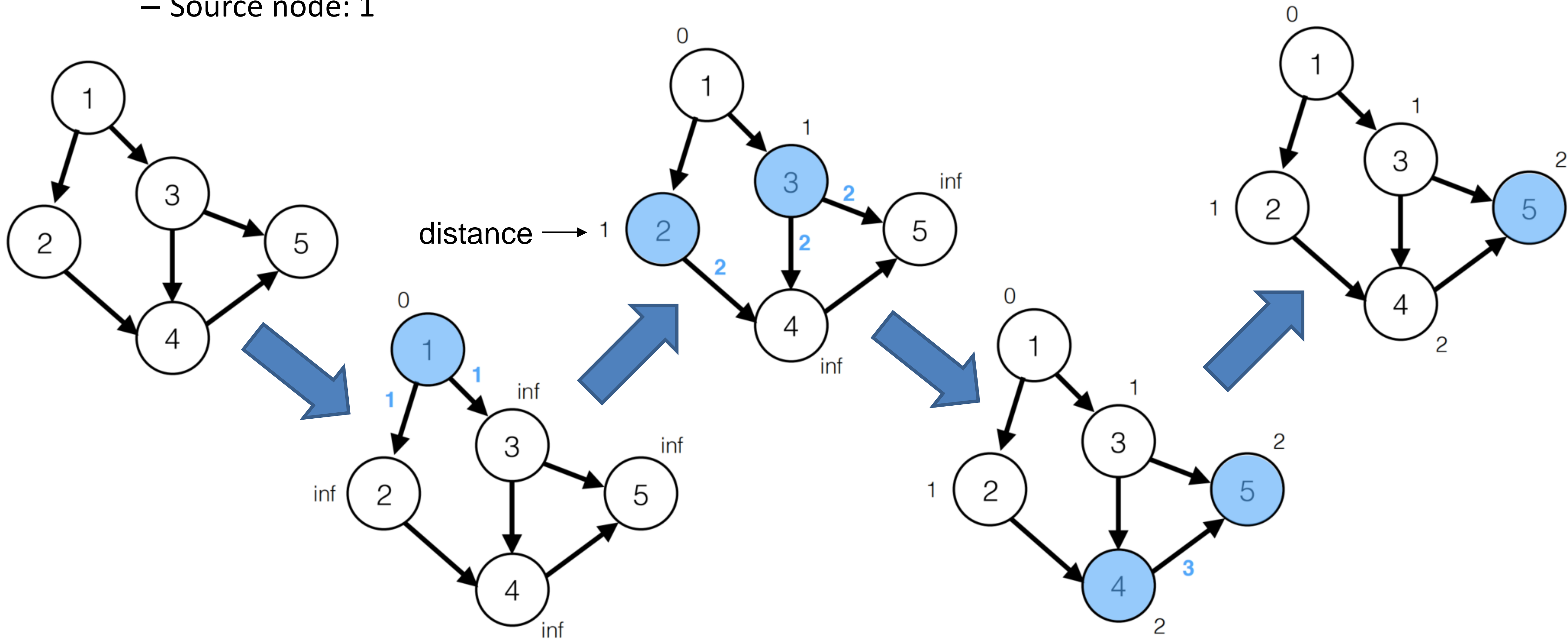
BSP Model



GAS Model

Vertex-Centric Graph Processing

- Vertex-centric BSP computation of the Single-Source Shortest Path (SSSP) algorithm:
  - Source node: 1

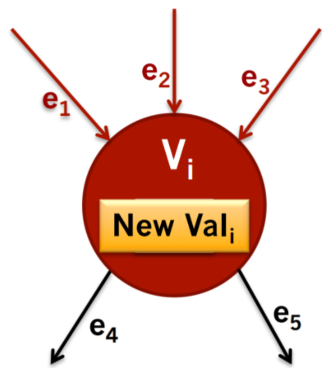


Grail: The Case Against Specialized Graph Analytics Engines [CIDR 2015]

- Motivation: Is graph processing that different from other types of data processing?
  - Answer: No. Can be subsumed by "traditional" relational processing
- **Vertex-centric programming** adopted by specialized graph engines

Bulk Synchronous Parallel (BSP)

(e.g., Giraph)

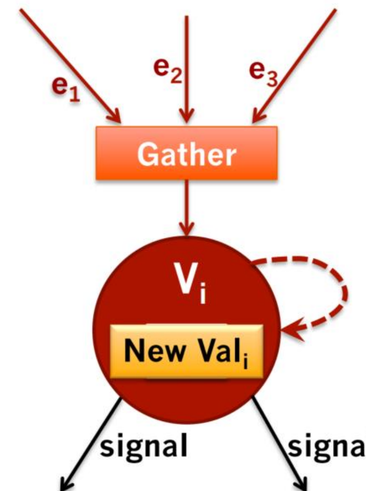


Vertex Centric:

```
do {
  foreach vertex in the graph {
    receive_messages();
    mutate_vertex_value();
    if (send_to_neighbors()) {
      send_messages_to_neighbors();
    }
  }
} until (has_converged() || reached_limit())
```

Gather-Apply-Scatter (GAS)

(e.g., GraphLab)

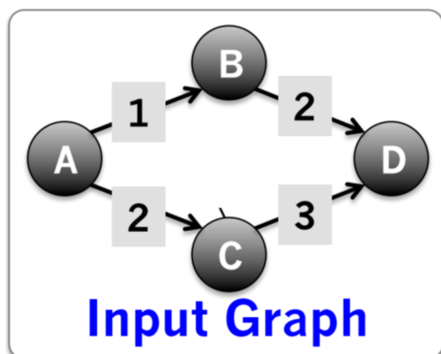


1. **Gather** values (from neighbors)
2. **Apply** updates to local state
3. **Scatter** signals to your neighbors

Fan, Jing, Adalbert Gerald Soosai Raj, and Jignesh M. Patel. "The Case Against Specialized Graph Analytics Engines." *CIDR*. 2015.

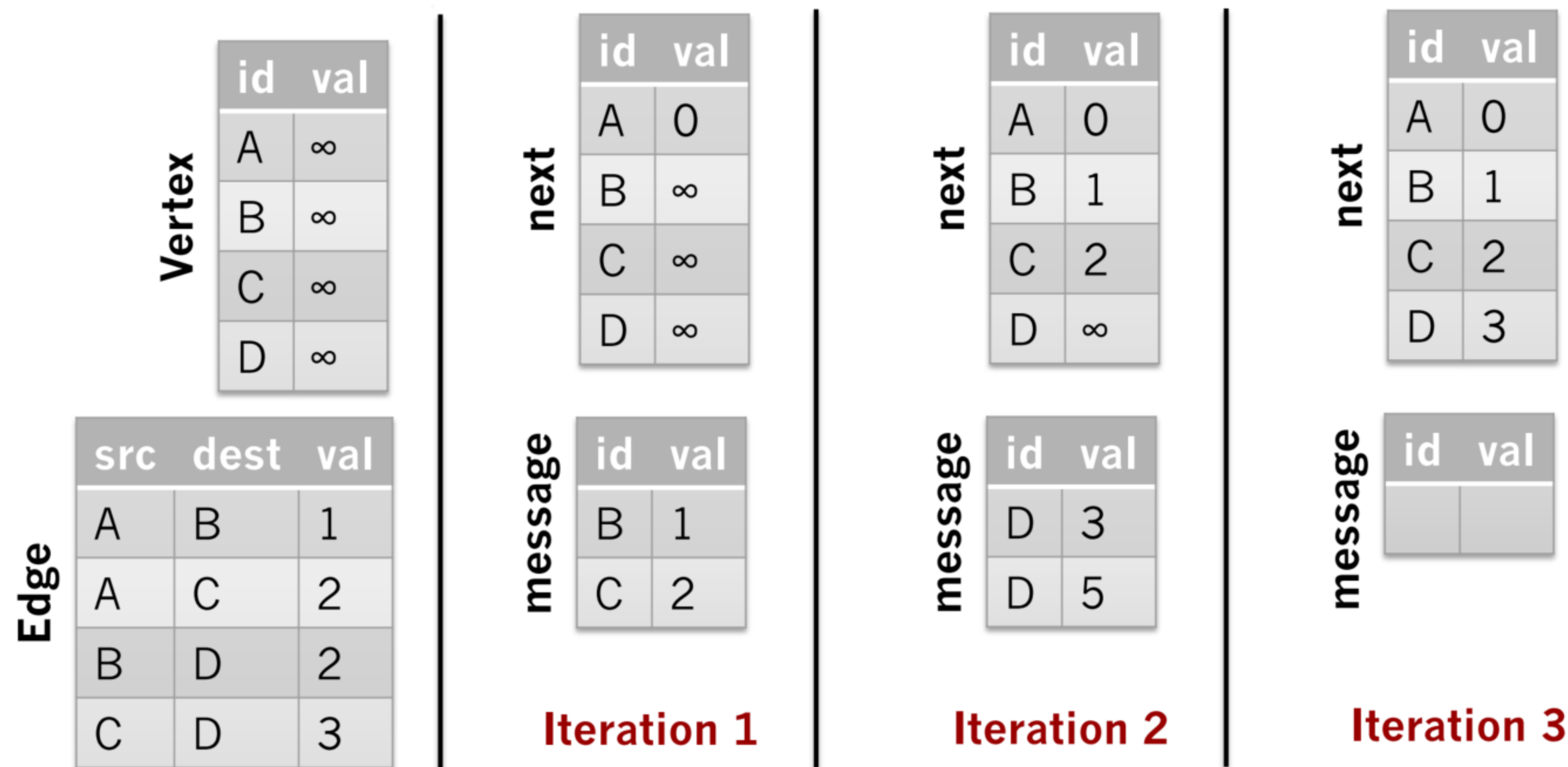
Grail: The Case Against Specialized Graph Analytics Engines [CIDR 2015] - *Schema Definitions*

- Basic idea: Build a similar vertex-centric simple API and then map it to SQL (with good performance)
- An example of the single-source shortest path algorithm:



Permanent:
- edge(src, dst, data, val)
- vertex(id, data, val)

Intermediate:
- next(id, val)
- cur(id, val)
- message(id, val)

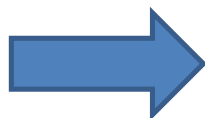


Grail: The Case Against Specialized Graph Analytics Engines [CIDR 2015]
- *From Grail API to SQL*

- An example of the **single-source shortest path** algorithm:

The Grail API

```
1  VertexValType: INT
2  MessageValType: INT
3  InitiateVal : INT_MAX
4  InitialMessage : (1, 0)
5  CombineMessage: MIN(message)
6  UpdateAndSend: update=cur.val<getVal()
7          if (update) {
8            setVal(cur.val)
9            send(out, cur.val+1)
10         }
11 End: NO_MESSAGE
```



T-SQL Code

```
1  DECLARE @flag int;
2  SET @flag = 1;
3  SELECT vertex.id, 2147483647 AS val
4  INTO next
5  FROM vertex;
6  CREATE TABLE message(
7     id int,
8     val int
9  );
10 INSERT INTO message values(1,0);
11 WHILE (@flag != 0)
12 BEGIN
13    SELECT message.id AS id, MIN(message.val) AS val
14    INTO cur
15    FROM message
16    GROUP BY message.id;
17    DROP TABLE message;
18    SELECT cur.id AS id, cur.val AS val
19    INTO update
20    FROM cur, next
21    WHERE cur.id = next.id AND cur.val <  next.val;
22    UPDATE next
23    SET next.val = update.val
24    FROM update, next
25    WHERE  next.id = update.id;
26    SELECT edge.dest AS id, update.val + 1 AS val
27    INTO message
28    FROM update, edge
29    WHERE edge.src = update.id;
30    DROP TABLE cur;
31    DROP TABLE update;
32    SELECT @flag = COUNT(*) FROM message;
33 END
```

- Lines 3–5: **Initialize**
- Lines 6–10: **Initialize the message table**
- Lines 11–16: **Aggregate the messages**
- Lines 18–21: **Create an update table and only consider updated vertices**
- Lines 22–25: **Update the next table**
- Lines 26–29: **Generate the message table for the next iteration**
- Line 32: **Stop when there are no new messages**

Grail: The Case Against Specialized Graph Analytics Engines [CIDR 2015]

- *The Role of Relational Algebra*

- Vertex-centric operators -> relational algebra -> SQL

Vertex Centric	Relational Algebra
Receive messages	cur $\longleftarrow \gamma_{id,F_0(val)}(message)$
Mutate value	$next \stackrel{u}{\longleftarrow} \pi_{next.id,F_1(other.val)} other \bowtie_{id} next$
Send messages	$\pi_{edge.B,F_2(other.val,edge.val)} other \bowtie_{other.id=edge.A} edge$

Aggregate function (can be a UDAF) | Scalar computation (can be a UDF) | Scalar computation (can be a UDF) | Join attributes control the direction

For single source shortest path: min | sum | identity | Outgoing edges

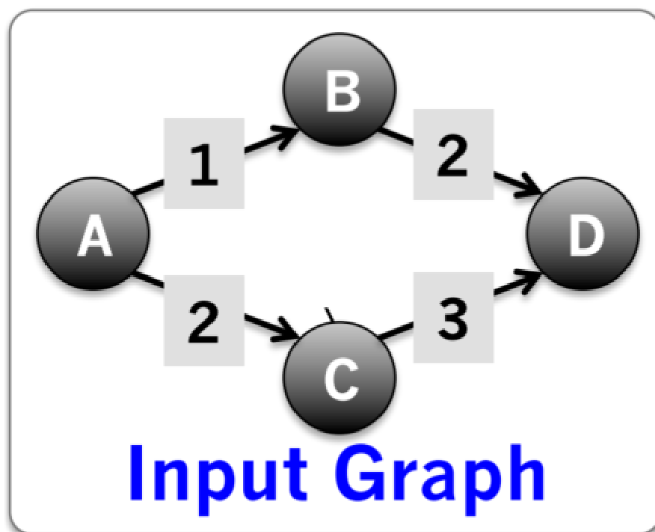
Grail: The Case Against Specialized Graph Analytics Engines [CIDR 2015]

- *The Role of Relational Algebra*

- Vertex-centric operators -> relational algebra -> SQL

Listing 2: Relational Algebra for SSSP in Grail

$$cur \leftarrow \gamma_{id,MIN(val)}(message)$$
$$update \leftarrow \pi_{cur.id,cur.val}(cur \bowtie_{cur.id=next.id \ AND \ cur.val<next.val} next)$$
$$next \xleftarrow{u} \pi_{next.id,update.val} update \bowtie_{id} next$$
$$message \leftarrow \pi_{edge.dest,update.val+edge.val}(update \bowtie_{update.id=edge.src} edge)$$



Vertex

id	val
A	∞
B	∞
C	∞
D	∞

Edge

src	dest	val
A	B	1
A	C	2
B	D	2
C	D	3

Iteration 1

next

id	val
A	0
B	∞
C	∞
D	∞

message

id	val
B	1
C	2

Iteration 2

next

id	val
A	0
B	1
C	2
D	∞

message

id	val
D	3
D	5

Iteration 3

next

id	val
A	0
B	1
C	2
D	3

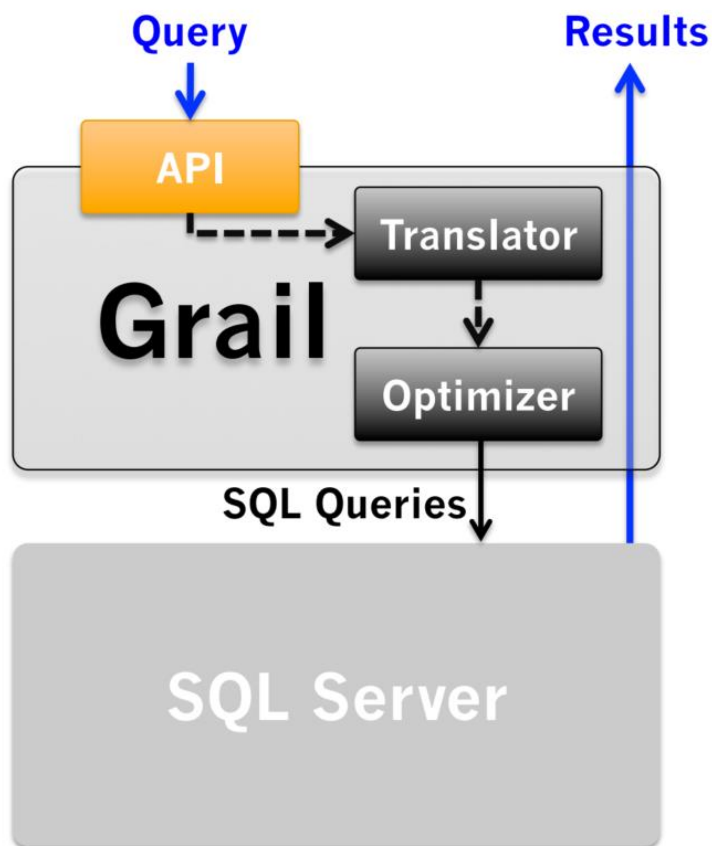
message

id	val

Grail: The Case Against Specialized Graph Analytics Engines [CIDR 2015]

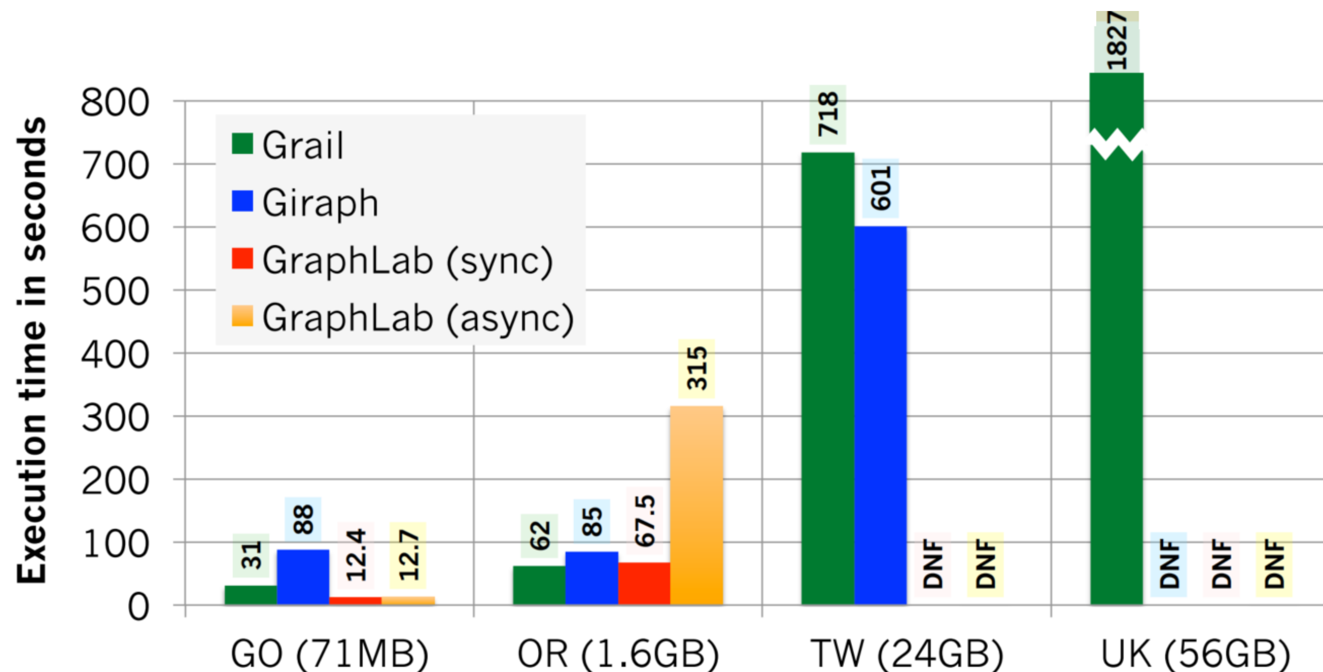
- *Implementation and Performance*

Implementation:



- Queries
  - Single-source shortest path (SSSP)
  - PageRank
  - Weakly connected components (WCC)

Dataset	#nodes	#edges	size
web-google (GO)	9K	5M	71MB
com-Orkut (OR)	3M	117M	1.6GB
Twitter-10 (TW)	41.6M	1.5B	24GB
uk-2007-05 (UK)	100M	3.3B	56GB



Graph Analytics using Vertica [VLDB 2014, BigData 2015]

- Vertex-centric processing -> query execution plan (e.g., Giraph)
- -> logical query plan -> query optimization -> SQL on standard relational databases

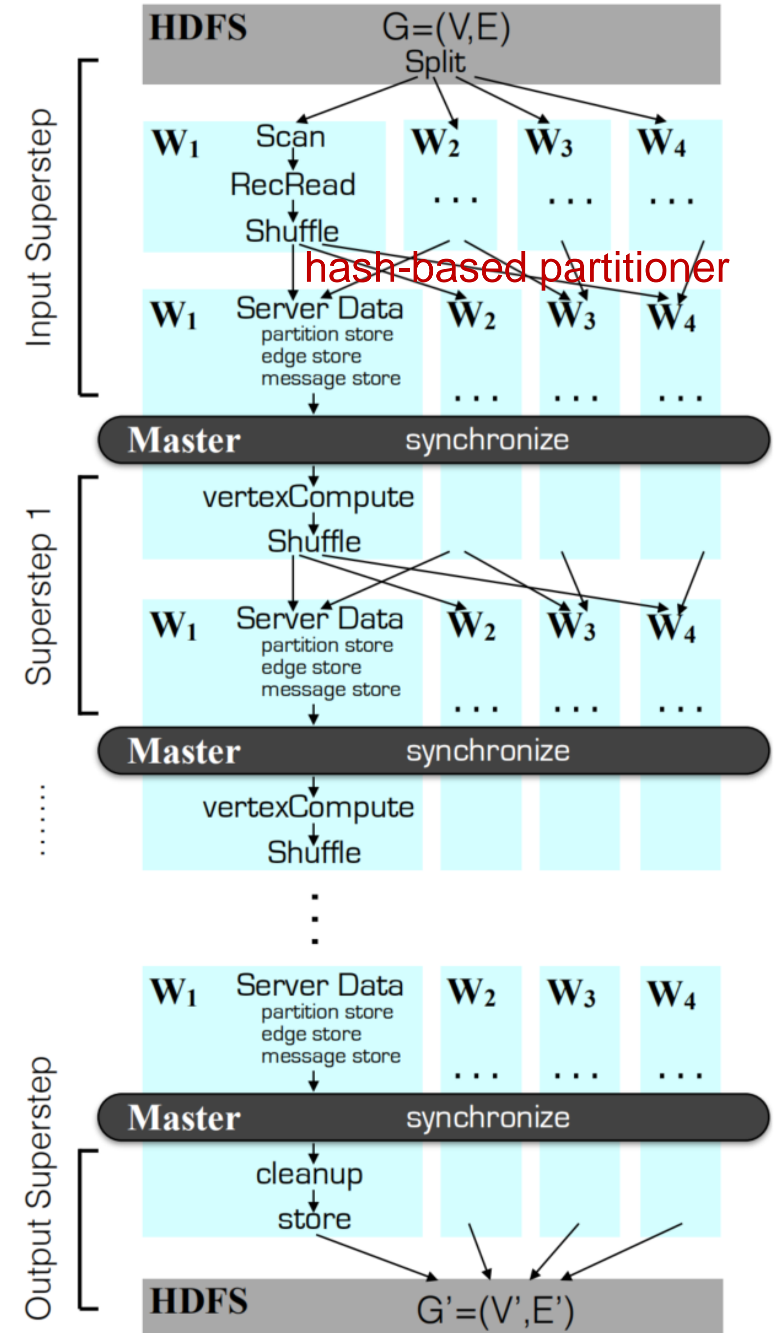
```
public void compute( Iterable <IntWritable> messages) {

  // get the minimum distance
  if ( getSuperstep () == 0)
    setValue (new DoubleWritable( Integer .MAX_VALUE));
  int minDist = isSource () ? 0 : Integer .MAX_VALUE;
  for ( IntWritable message : messages)
    minDist = Math.min(minDist, message.get ());

  // send messages to all edges if new minimum is found
  if (minDist < getValue().get()) {
    setValue (new IntWritable (minDist));
    for (Edge<?, ?> edge : getEdges()) {
      int distance = minDist + edge.getValue () . get ();
      sendMessage(edge. getTargetVertexId () , new IntWritable ( distance ));
    }
  }
  voteToHalt (); // halt
}
```

Listing 1: Single Source Shortest Path in Giraph.

Alekh Jindal, Praynaa Rawlani, Eugene Wu, Samuel Madden, Amol Deshpande, Mike Stonebraker: VERTEXICA: Your Relational Friend for Graph Analytics! Proc. VLDB Endow. 7(13): 1669-1672 (2014)
Jindal, Alekh, et al. "Graph analytics using vertica relational database." *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015.

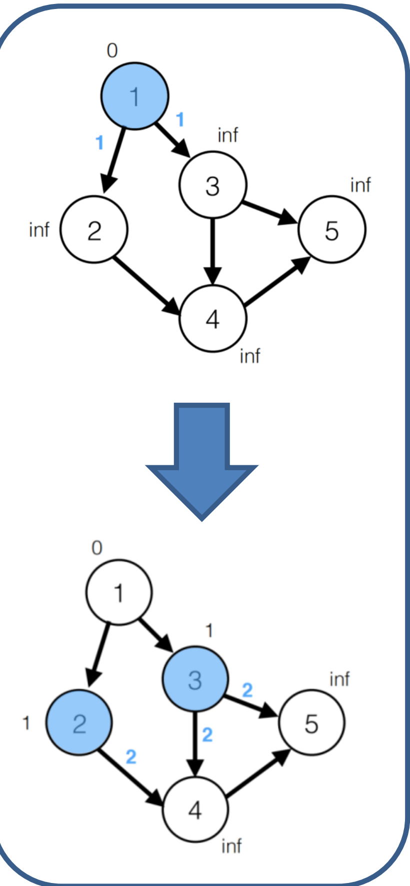
Giraph Physical Plan

- Giraph: a popular, open-source graph analytics system on Hadoop
- The Giraph physical plan: hard coded physical execution pipeline
- Server Data
  - Partition store: partition vertices and related metadata
  - Edge store: partition edges and related metadata
  - Message store: incoming messages for this partition
- In each superstep, the workers run the **vertexCompute** UDF



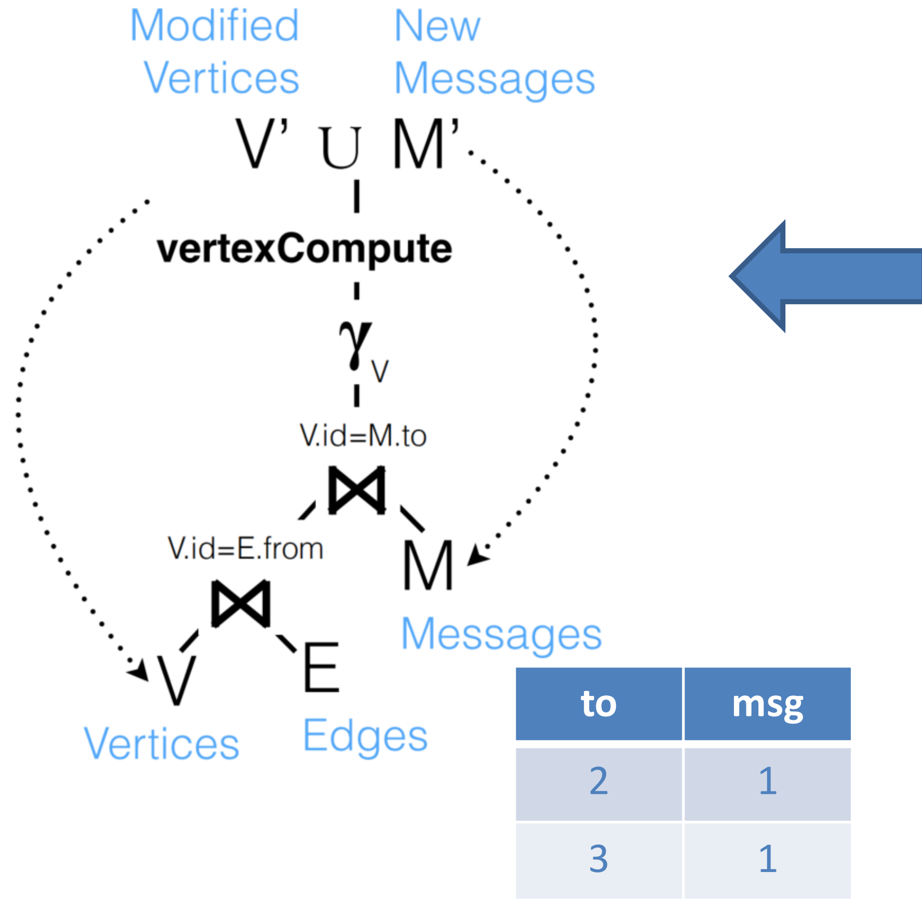
Giraph Physical Plan

- The Giraph physical plan: hard coded physical execution pipeline

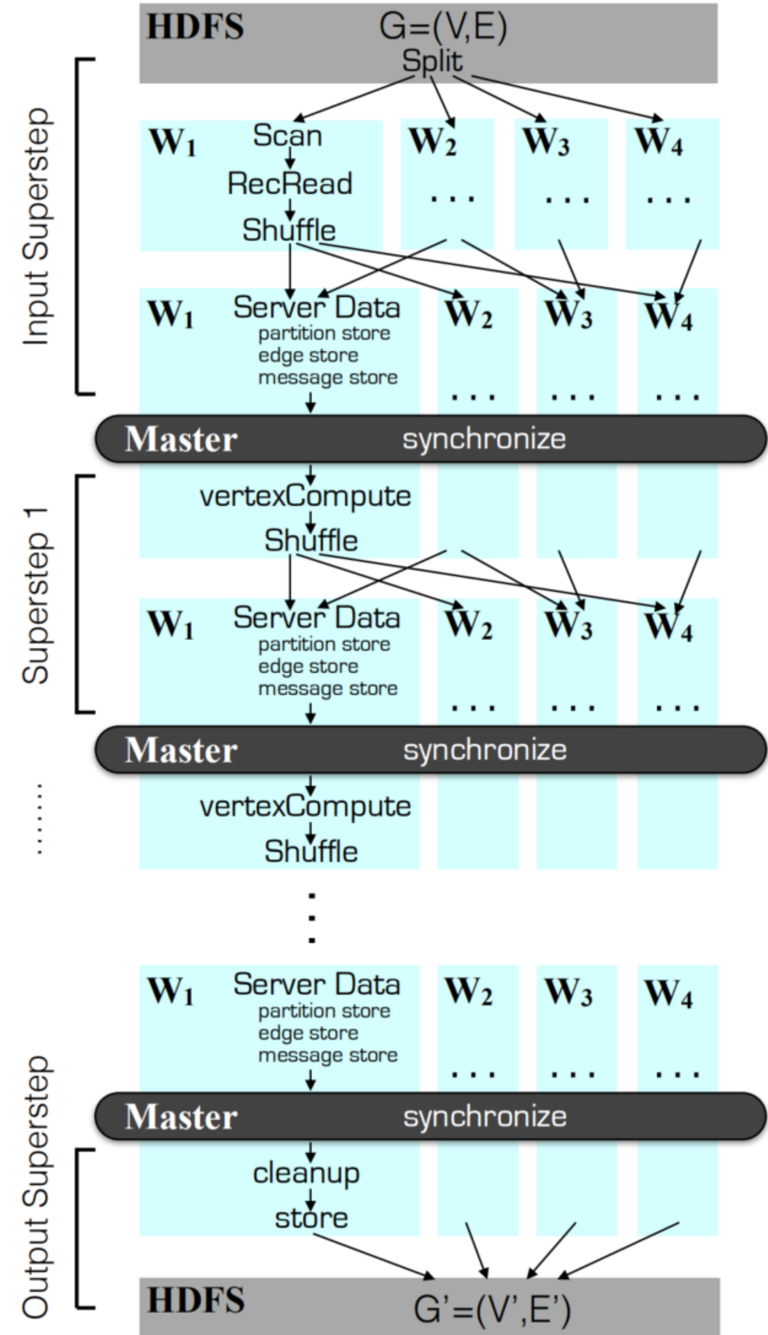


id	value
1	0
2	inf
…	…

from	to
1	2
1	3
…	…



to	msg
2	1
3	1

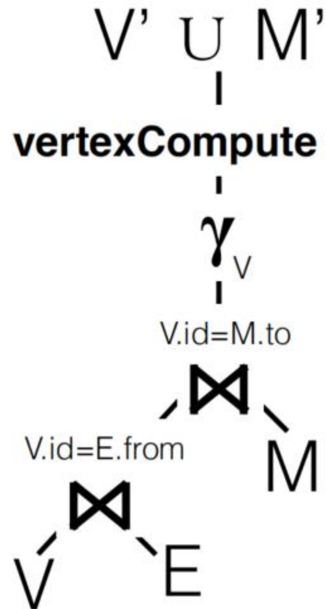


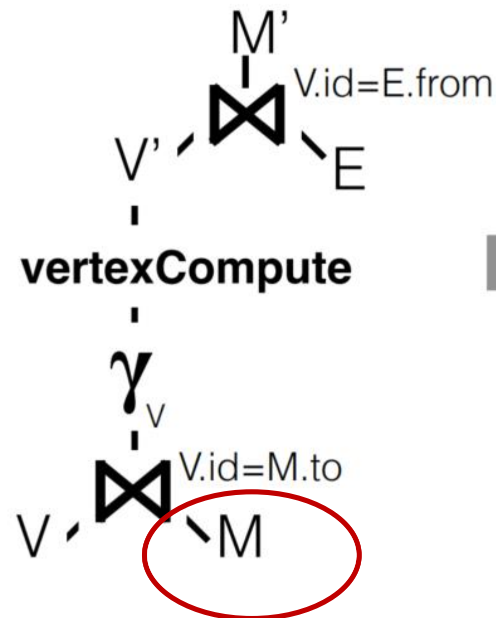
Graph Analytics using Vertica [VLDB 2014, BigData 2015]

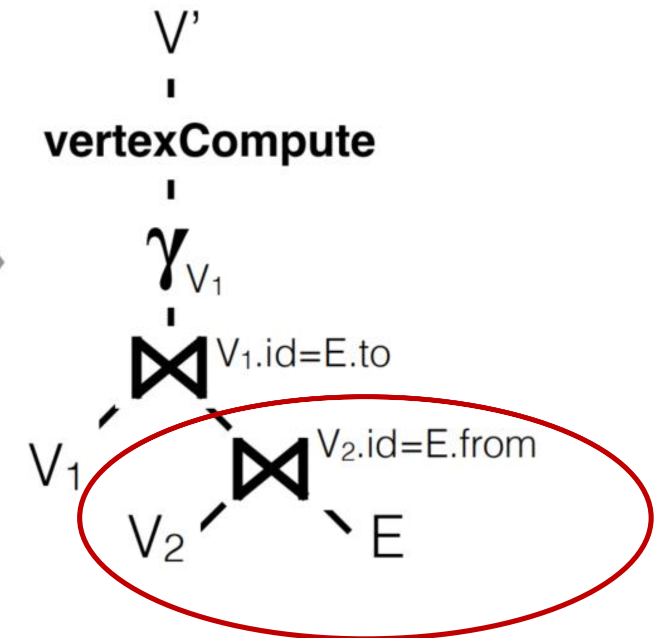
- *Rewriting Logical Giraph Plan*

Eliminating the message table (by directly update V in RDBMS):

1. Giraph logical query plan
2. Pushing down the vertexCompute UDF
3. Replacing M by V ⋈ E







Graph Analytics using Vertica [VLDB 2014, BigData 2015]

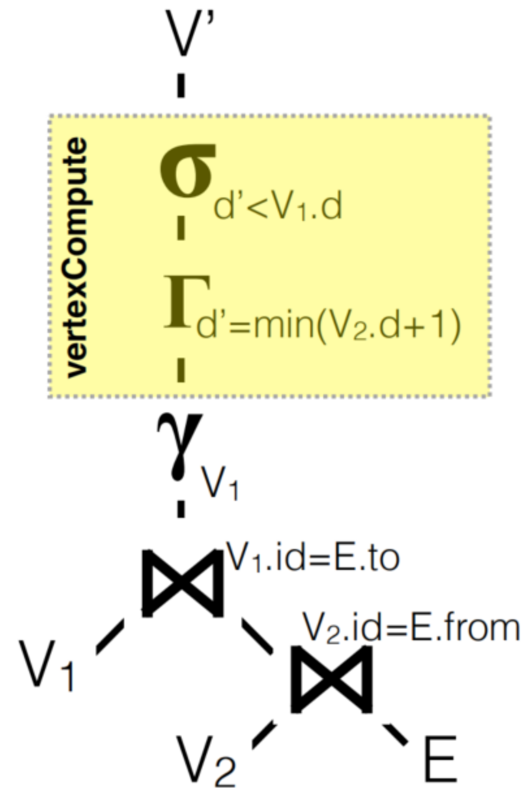
- *Rewriting Logical Giraph Plan*

Translating **vertexCompute** to **relational algebra/SQL**:

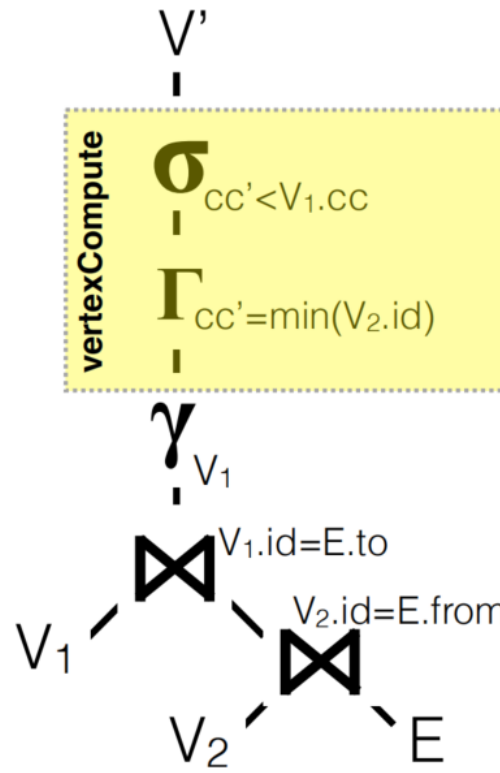
$vertexCompute \longmapsto \sigma_{d'<V_1.d}(\Gamma_{d'=\min(V_2.d+1)})$

$\sigma_{cc'<V_1.cc}(\Gamma_{cc'=\min(V_2.id)})$

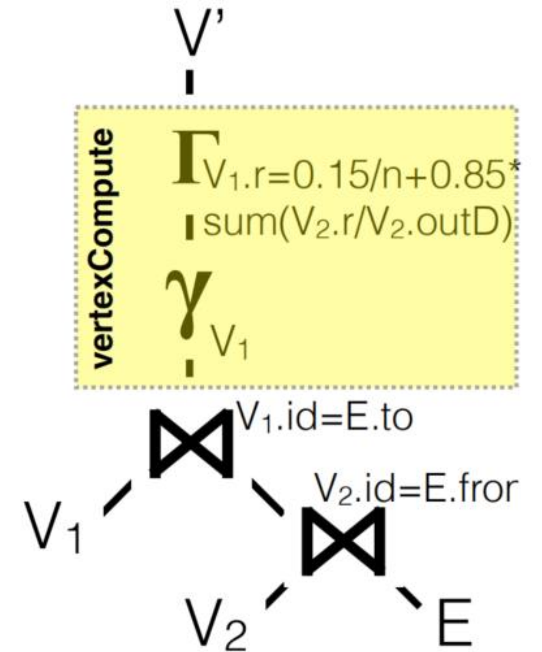
$\Gamma_{V_1.r=\frac{0.15}{n}+0.85*sum(\frac{V_2.r}{V_2.outD})}$



Single-Source Shortest Path



Connected Components



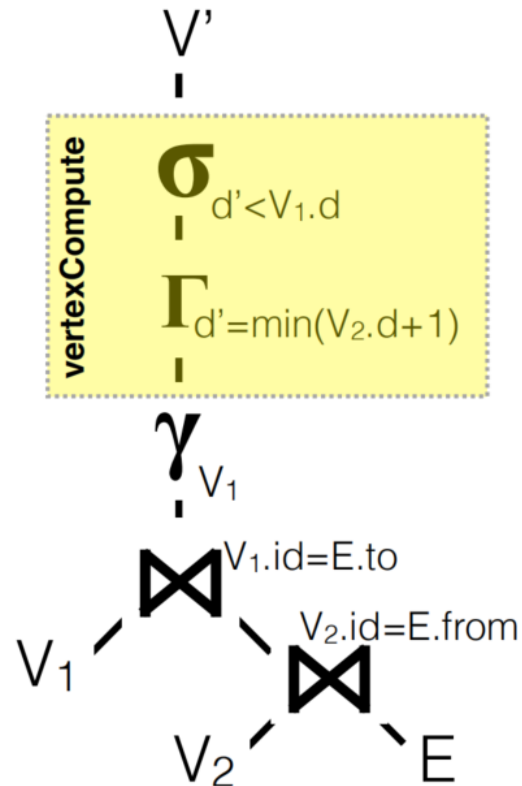
PageRank

Graph Analytics using Vertica [VLDB 2014, BigData 2015]

- *SSSP as an example*

Translating **vertexCompute** to **relational algebra/SQL**:

$$vertexCompute \longmapsto \sigma_{d' < V_1.d}(\Gamma_{d'=\min(V_2.d+1)})$$

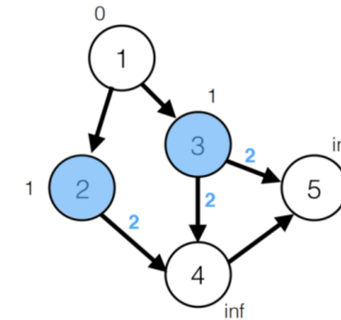


```
UPDATE vertex AS v SET v.d=v'.d
  FROM (
    SELECT v1.id, MIN(v2.d+1) AS d
      FROM vertex AS v1, edge AS e,  vertex  AS v2
      WHERE v2.id = e.from_node AND v1.id = e.to_node
      GROUP BY e.to_node, v1.d
      HAVING MIN(v2.d+1) < v1.d
    ) AS v'
    WHERE v.id=v'.id;
```

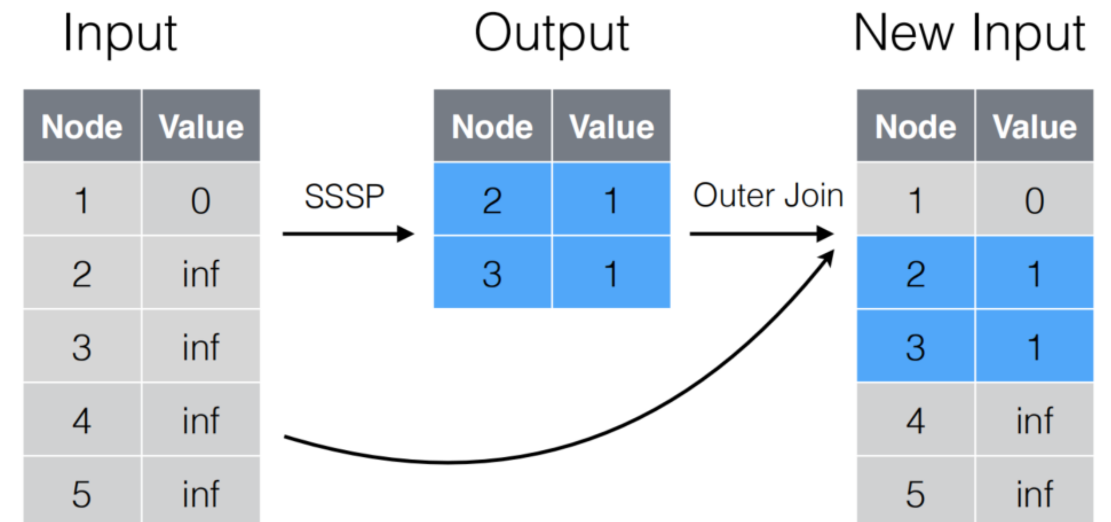
Single-Source Shortest Path

Graph Analytics using Vertica [VLDB 2014, BigData 2015]
- Query Optimization: Update vs. Replace

- For large number of updates:
  - Create a new vertex relation (vertex_prime) by joining the updated vertices with the non-updated vertices
  - Replace vertex with vertex_prime




```
CREATE TABLE vertex_prime AS
  SELECT v.id, ISNULL(v'.d, v.d) AS d
    FROM vertex AS v LEFT JOIN (
      SELECT v1.id AS id, MIN(v2.d+1) AS d
        FROM vertex AS v1, edge AS e,  vertex  AS v2
        WHERE v2.id=e.from_node AND v1.id=e.to_node
        GROUP BY e.to_node, v1.d
        HAVING MIN(v2.d+1) < v1.d
    ) AS v'
    ON v.id = v'.Id;
```



Input

Node	Value
1	0
2	inf
3	inf
4	inf
5	inf

Output

Node	Value
2	1
3	1

New Input

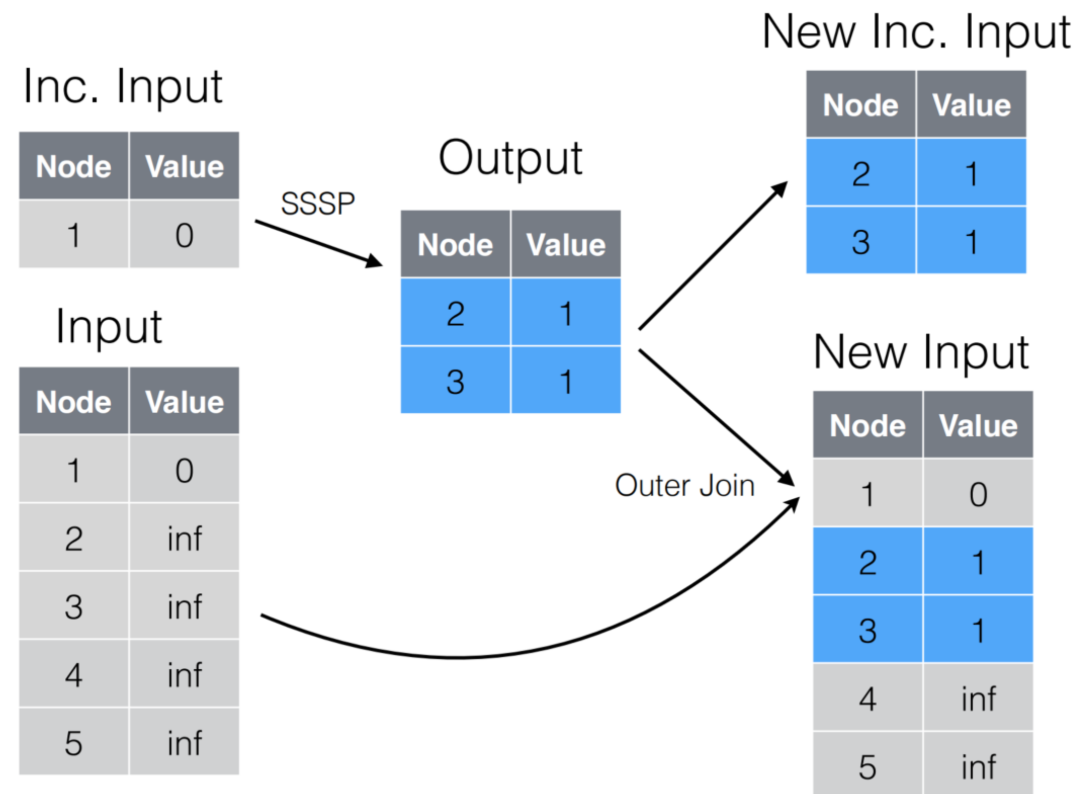
Node	Value
1	0
2	1
3	1
4	inf
5	inf

Graph Analytics using Vertica [VLDB 2014, BigData 2015]

- *Query Optimization: Incremental Evaluation*

- In single-source shortest path (SSSP)
  - only need to explore the neighbors of vertices that found a smaller distance in the previous iteration, i.e., the updated vertices table v_update

```
CREATE TABLE v_update_prime AS
  SELECT v1.id, MIN(v2.d+1) AS d
    FROM v_update AS v2, edge AS e, vertex AS v1
    WHERE v2.id=e.from_node AND v1.id=e.to_node
    GROUP BY e.to_node, v1.d
    HAVING MIN(v2.d+1) < v1.d;

DROP TABLE v_update;
ALTER TABLE v_update_prime RENAME TO v_update ;

CREATE TABLE vertex_prime AS
  SELECT v.id, ISNULL(v_update.d, v.d) AS value
    FROM vertex AS v LEFT JOIN v_update
    ON v.id = v_update . id ;

DROP TABLE vertex; ALTER TABLE vertex_prime RENAME TO vertex;
```

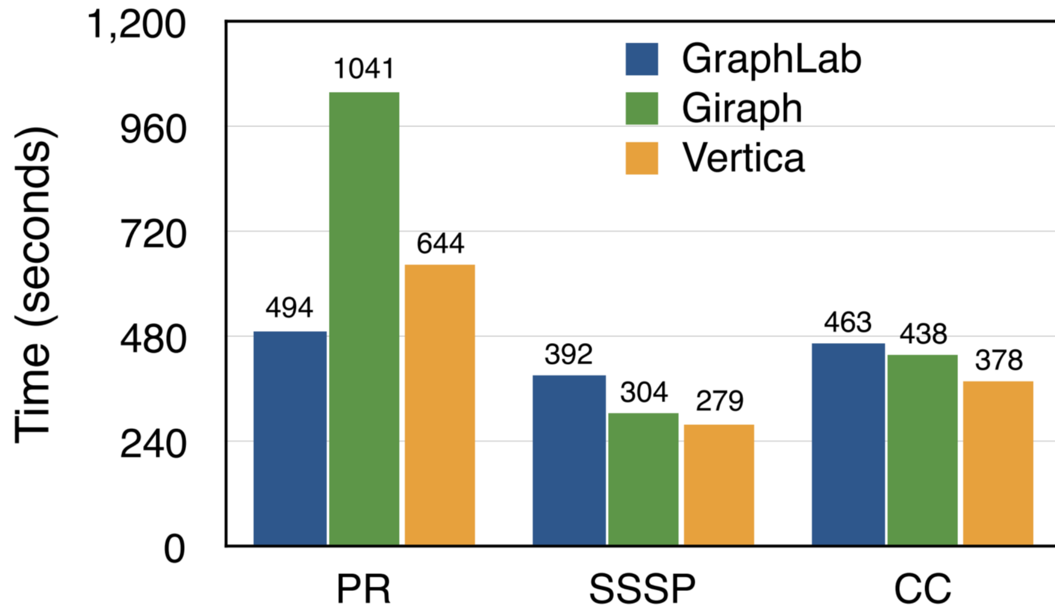


Graph Analytics using Vertica [VLDB 2014, BigData 2015]
- Comparison with Specialized Graph Systems

- Typical graph analytics
- Advanced graph analytics (e.g., multi-hop neighborhood queries)

Typical Graph Analytics

Twitter graph: 1.4 billion edges, 41.6 million nodes



- Multi-hop neighborhood queries

Query	Dataset	Vertica	Giraph
Strong Overlap	Youtube	259.56	230.01
	LiveJournal-undir	381.05	out of memory
Weak Ties	Youtube	746.14	out of memory
	LiveJournal-undir	1,475.99	out of memory

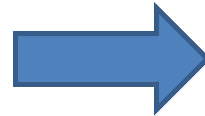
- **Strong overlap**: Find all pairs of nodes having a large number of common neighbors (i.e., above the threshold)
- **Weak ties**: Find all nodes that act as a bridge between two otherwise disconnected node-pairs, i.e., connect at least a threshold number of node pairs

All-in-One: Graph Processing in RDBMSs Revisited [SIGMOD 2017]

A large number of graph algorithms

- *Breadth-First Search (BFS)*
- *Connected Component*
- *Shortest Distance*
- *Topological Sorting*
- *PageRank*
- *Random Walk with Restart*
- *SimRank*
- *Label Propagation*
- *Maximum Independent Set*

…



4 new relational algebra operations

- *MM-join*
- *MV-join*
- *Anti-join*
- *Union-by-update*

Recursive SQL

- *The with clause*

Zhao, Kangfei, and Jeffrey Xu Yu. "All-in-one: graph processing in RDBMSs revisited." *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017.

All-in-One: Graph Processing in RDBMSs Revisited [SIGMOD 2017]

- *Four New Relational Algebra Operations*

- Let V and M be the relation representation of vector V and matrix M
  - Schema: $V(ID, vw)$, $M(F, T, ew)$
- Matrix-matrix / matrix-vector multiplication

$$\mathsf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad \mathsf{B} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}, \quad \mathsf{C} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

$$\mathsf{A} \cdot \mathsf{B} = \begin{pmatrix} a_{11} \odot b_{11} \oplus a_{12} \odot b_{21} & a_{11} \odot b_{12} \oplus a_{12} \odot b_{22} \\ a_{21} \odot b_{11} \oplus a_{22} \odot b_{21} & a_{21} \odot b_{12} \oplus a_{22} \odot b_{22} \end{pmatrix}$$

$$\mathsf{A} + \mathsf{B} = \begin{pmatrix} a_{11} \oplus b_{11} & a_{12} \oplus b_{12} \\ a_{21} \oplus b_{21} & a_{22} \oplus b_{22} \end{pmatrix}$$

$$\mathsf{A} \cdot \mathsf{C} = \begin{pmatrix} a_{11} \odot c_1 \oplus a_{12} \odot c_2 \\ a_{21} \odot c_1 \oplus a_{22} \odot c_2 \end{pmatrix}$$

All-in-One: Graph Processing in RDBMSs Revisited [SIGMOD 2017]

- *Four New Relational Algebra Operations*

- New relational algebra (RA) operations

group-by & aggregation operation

A, B: (F, T, ew), C: (ID, vw)

Operation	Definition	Expression
MM-join	$A \overset{\oplus(\odot)}{\underset{A.T=B.F}{\bowtie}} B$	${}_{A.F,B.T}\mathcal{G}_{\oplus(\odot)}(A \underset{A.T=B.F}{\bowtie} B)$
MV-join	$A \overset{\oplus(\odot)}{\underset{T=ID}{\bowtie}} C$	${}_{F}\mathcal{G}_{\oplus(\odot)}(A \underset{T=ID}{\bowtie} C)$
Anti-join	$R \mathbin{\bar{\ltimes}} S$	$R - (R \ltimes S)$
Union-by-update	$R \uplus_A S$	Update the B attributes values of r by the B attributes values of s if r.A = s.A (multiple s matching a single r is not allowed)

R, S: (A, B)

All-in-One: Graph Processing in RDBMSs Revisited [SIGMOD 2017]

- Graph Processing with New Relational Algebra Operations

- Let V and E be the relation representation of vector V and matrix E
  - Schema: $V(ID, vw)$, $E(F, T, ew)$
- Breadth-First Search (BFS)
  - Initially, only the source node has $vw=1$, $E_{ij}=1$ if there exists an edge from v_i to v_j
  - The traversal operation of BFS (expressed in matrix formation): $E^T \cdot V$

Generalized addition $\oplus$

Generalized multiplication $\odot$

$$V \leftarrow \rho_V(E \overset{max(vw*ew)}{\underset{F=ID}{\bowtie}} V)$$

Union-by-update

All-in-One: Graph Processing in RDBMSs Revisited [SIGMOD 2017]

- Graph Processing with New Relational Algebra Operations

- Let V and E be the relation representation of vector V and matrix E
  - Schema: $V(ID, vw)$, $E(F, T, ew)$
- Breadth-First Search (BFS)
  - Initially, only the source node has $vw=1$, $E_{ij}=1$ if there exists an edge from v_i to v_j
  - The traversal operation of BFS (expressed in matrix formation): $E^T \cdot V$

Generalized addition $\oplus$

Generalized multiplication $\odot$

$$V \leftarrow \rho_V(E \underset{F=ID}{\overset{max(vw*ew)}{\bowtie}} V)$$

Union-by-update

  - The control structure: Relational algebra plus while

initialize R

while (R changes) { $\cdots$; $R \leftarrow \boxed{\cdots}$ }

All-in-One: Graph Processing in RDBMSs Revisited [SIGMOD 2017]

- Graph Processing with New Relational Algebra Operations

- **Representative graph algorithms:**

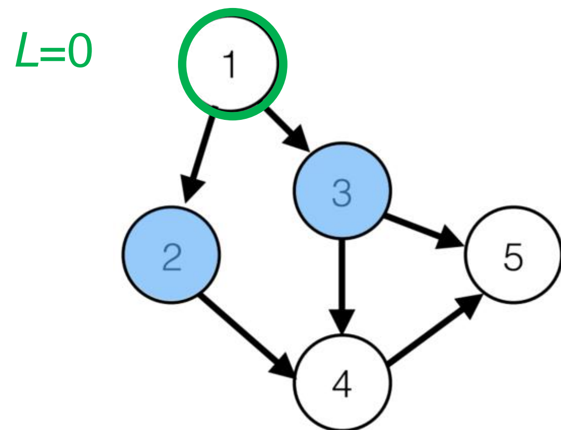
Breadth-First Search (BFS)	$V \leftarrow \rho_V(E \overset{max(vw*ew)}{\underset{F=ID}{\bowtie}} V)$
Connected Component	$V \leftarrow \rho_V(E \overset{min(vw*ew)}{\underset{F=ID}{\bowtie}} V)$
Bellman-Ford for SSSP	$V \leftarrow \rho_V(E \overset{min(vw+ew)}{\underset{F=ID}{\bowtie}} V)$
Floyd-Warshall for APSP	$E \leftarrow \rho_E((E \rightarrow E_1) \overset{min(E_1.ew+E_2.ew)}{\underset{E_1.T=E_2.F}{\bowtie}} (E \rightarrow E_2))$
PageRank	$V \leftarrow \rho_V(E \overset{f_1(\cdot)}{\underset{T=ID}{\bowtie}} V)$ $f_1(\cdot) = c * sum(vw * ew) + (1-c)/n$
Random Walk with Restart	$V \leftarrow \rho_V(\Pi_{V.ID, f_2(\cdot)+(1-c)*P.vw}(E \overset{f_2(\cdot))}{\underset{S.T=ID}{\bowtie}} V)$ $f_2(\cdot) = c * sum(vw * ew)$ $P(ID, vw)$ denotes the restart probability

All-in-One: Graph Processing in RDBMSs Revisited [SIGMOD 2017]

- *Graph Processing with New Relational Algebra Operations*

- **Representative graph algorithms:**

Topological Sorting	Let $Topo(ID, L)$ be a relation that contains a set of nodes having no incoming edges with initial L value 0 ($\Pi_{ID,0}(V \bar{\ltimes}_{ID=E.T} E)$) ① $L_n \leftarrow \rho_L(\mathcal{G}_{max(L)+1} Topo)$ ② $V_1 \leftarrow V \bar{\ltimes}_{V.ID=T.ID} Topo$ ③ $E_1 \leftarrow \Pi_{E.F,E.T}(V_1 \bowtie_{ID=E.F} E)$ ④ $T_n \leftarrow \Pi_{ID,L}(V_1 \bar{\ltimes}_{V_1.ID=E_1.T} E_1) \times L_n$ ⑤ $Topo \leftarrow Topo \cup T_n$



① L_1

Lvl
1

Topo

ID	L
1	0

② V_1 contains {2 ,3, 4, 5}

③ E_1 contains {(2, 4), (3, 4), (3, 5), (4, 5)}

④ T_2 contains {(2, 1), (3, 1)}

All-in-One: Graph Processing in RDBMSs Revisited [SIGMOD 2017]

- The With Clause

- Enhance the with clause in SQL'99
- Implemented by SQL/Persistent Stored Model (PSM) procedure
- The recursive queries defined by the 4 RA operators have fixpoint

```
with R as
    select ··· from R_{1,j}, ··· computed by ··· (Q_1)
    union all
    ···
    union all
    select ··· from R_{i,j}, ··· computed by ··· (Q_i)
    union all
    ···
    union all
    select ··· from R_{n,j}, ··· computed by ··· (Q_n)
```

Figure 4: The general form of the enhanced recursive with

```
1.  with
2.  Topo(ID, L) as (
3.        (select ID, 0 from V
4.        where ID not in select E.T from E)
5.        union all
6.        (select ID, L from T_n
7.        computed by
8.              L_n(L) as select max(L) + 1 from Topo;
9.              V_1 as
10.                   select V.ID from V
11.                   where ID not in select ID from Topo;
12.             E_1 as
13.                   select E.F, E.T from V_1, E
14.                   where V_1.ID = E.F;
15.             T_n as
16.                   select ID, L from V_1, L_n
17.                   where ID not in select T from E_1;))
18. select from Topo;
```

Figure 5: The recursive with **for** *TopoSort*

IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020]

- Build graph query support inside Db2 that is synergistic with other analytics and retrofittable to existing data
- Db2 Graph is a layer inside Db2 specialized for graph queries
  - With the property graph model

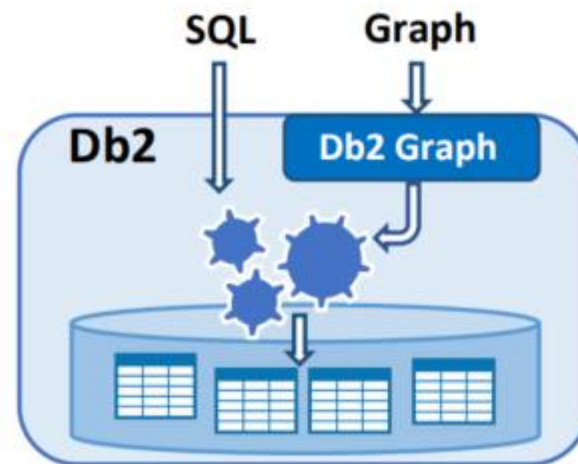


Figure 1: Synergistic graph queries inside Db2

Tian, Yuanyuan, et al. "Synergistic graph and SQL analytics inside IBM Db2." *Proceedings of the VLDB Endowment* 12.12 (2019): 1782-1785.
Tian, Yuanyuan, et al. "IBM db2 graph: Supporting synergistic and retrofittable graph queries inside IBM db2." *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020.

IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020]

- Creating Graph View on Tables

- Use **graphQuery** (i.e., the polymorphic table function) based on Gremlin
  - The returned result is a table
- However, the graph is not actually built

Patient Table

patientID	name	address	subscriptionID
1	Alice	...	115
...			...

HasDisease Table

patientID	diseaseID	description
1	64572326	...
...	...	...

Disease Table

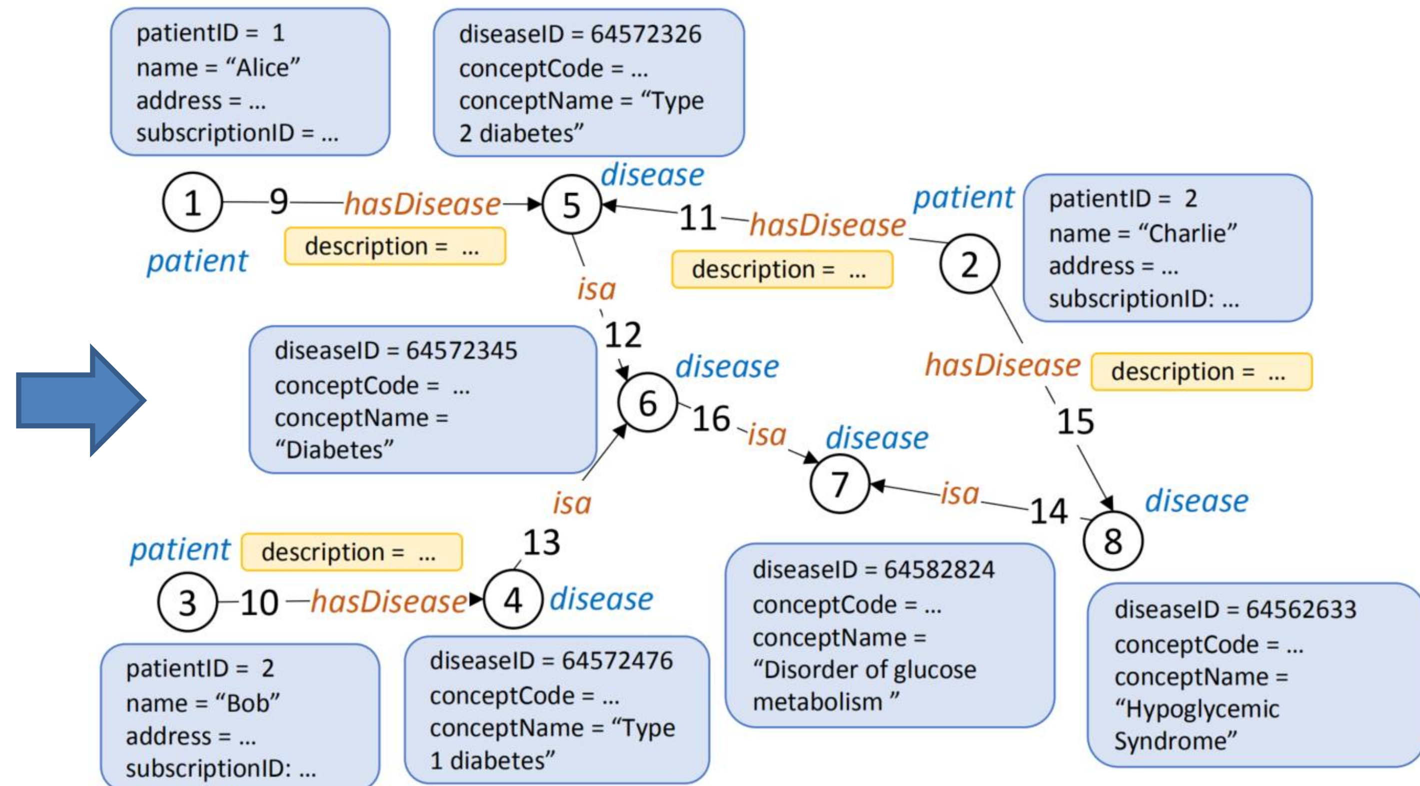
diseaseID	conceptCode	conceptName
64572326	44054006	“Type 2 diabetes”
...	...	

DiseaseOntology Table

sourceID	targetID	type
64572326	73211009	“isa”
...	...	

DeviceData Table

subscriptionID	date	steps	execiseMinutes	activeEnergy
115	11/15/2018	9039	25	208
...	...	...		...

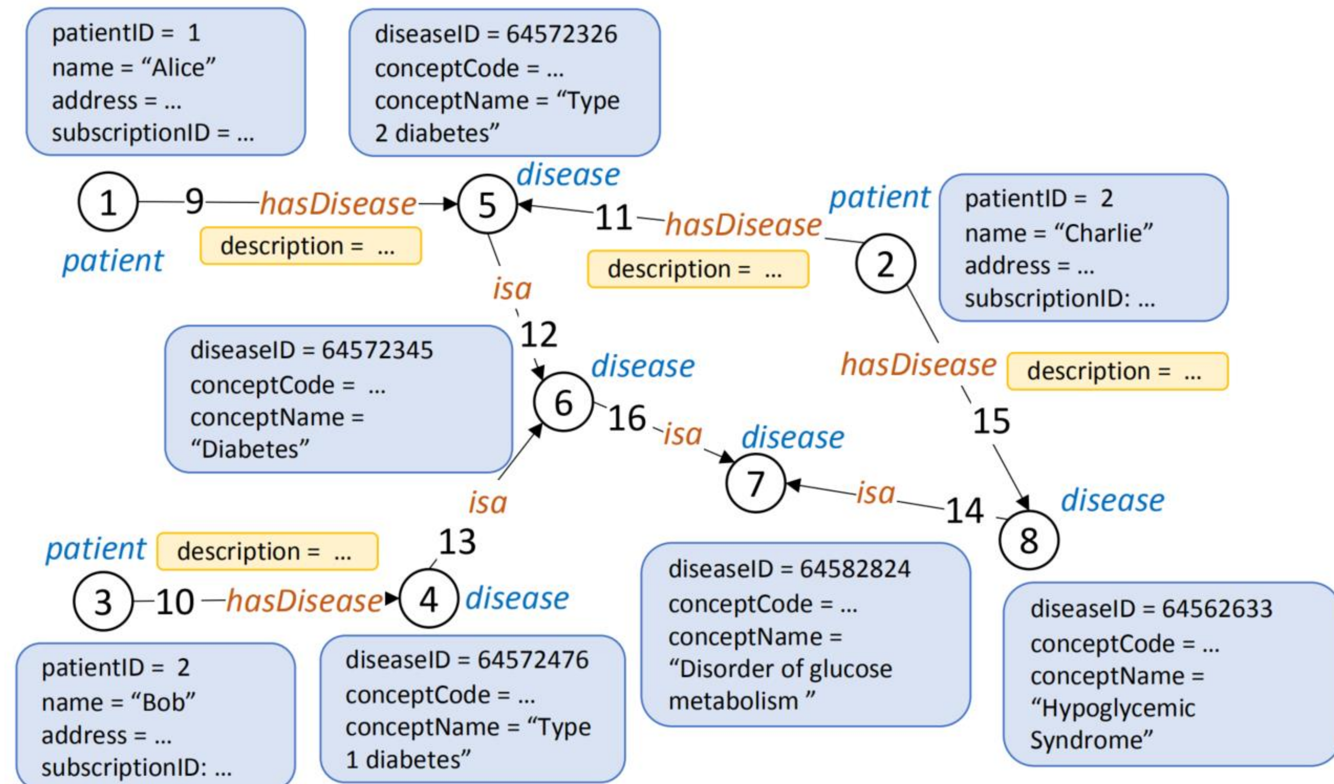


IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020] *- Creating Graph View on Tables*

- Use **graphQuery** (i.e., the polymorphic table function) based on Gremlin
  - The returned result is a table
- However, the graph is not actually built

```
SELECT patientID, AVG(steps), AVG(execiseMinutes)
FROM DeviceData AS D,
 TABLE (graphQuery('gremlin', 'similar_diseases = g.V()
  .hasLabel(\'patient\').has(\'patientID\', \'1\').out(\'hasDisease\')
  .repeat(out(\'isa\').dedup().store(\'x\')).times(2)
  .repeat(in(\'isa\').dedup().store(\'x\')).times(2).cap(\'x\').next();
  g.V(similar_diseases).in(\'hasDisease\').dedup()
  .values(\'patientID\', \'subscriptionID\')'))
  AS P (patientID long, subscriptionID long)
WHERE D.subscriptionID = P.subscriptionID
GROUP BY patientID
```

Finds patients that have similar diseases as those of a particular patient (with patientID=1), and compares their daily exercise patterns



IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020] - *Creating Graph View on Tables*

- Specify the relation-graph mapping via the **overlay configuration file**
  - What table(s) store the vertex information? What table column(s) are mapped to the required id field? What is the label for each vertex? …

```
1   "v_tables": [
2   {
3       "table_name": "Patient",
4       "prefixed_id": true,
5       "id": "'patient'::patientID",
6       "fix_label": true,
7       "label": "'patient'",
8       "properties": ["patientID", "name", "address", "
        subscriptionID"]
9   },
10  {
11      "table_name": "Disease",
12      "id": "diseaseID",
13      "fix_label": true,
14      "label": "'disease'",
15      "properties": ["diseaseID", "conceptCode", "
        conceptName"]
16  }],
```

```
17  "e_tables": [
18  {
19      "table_name": "DiseaseOntology",
20      "src_v_table": "Disease",
21      "src_v": "sourceID",
22      "dst_v_table": "Disease",
23      "dst_v": "targetID",
24      "prefixed_edge_id": true,
25      "id": "'ontology'::sourceID::targetID",
26      "label": "type"
27  },
28  {
29      "table_name": "HasDisease",
30      "src_v_table": "Patient",
31      "src_v": "'patient'::patientID",
32      "dst_v_table": "Disease",
33      "dst_v": "diseaseID",
34      "implicit_edge_id": true,
35      "fix_label": true,
36      "label": "'hasDisease'"
37  }]
```

IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020]

- Creating Graph View on Tables

- Automatically generation of the overlay configuration file (AutoOverlay)
  - Step 1. First queries Db2 catalog to get all the metadata information for each table such as table schema, and primary key/foreign key constraints
  - Step 2. If a table has primary key, map it to a vertex table; if it has foreign key(s), also map it to an edge table
  - Step 3. Maps the required fields in the property graph model to columns in the vertex/edge tables
- Note that
  - Heavily rely on the primary and foreign key constraints!
  - One can manually specify the configuration
  - Machine learning techniques to infer the constraints (as future work)

IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020] - *Architecture*

g.V().has('name', 'Alice').outE()

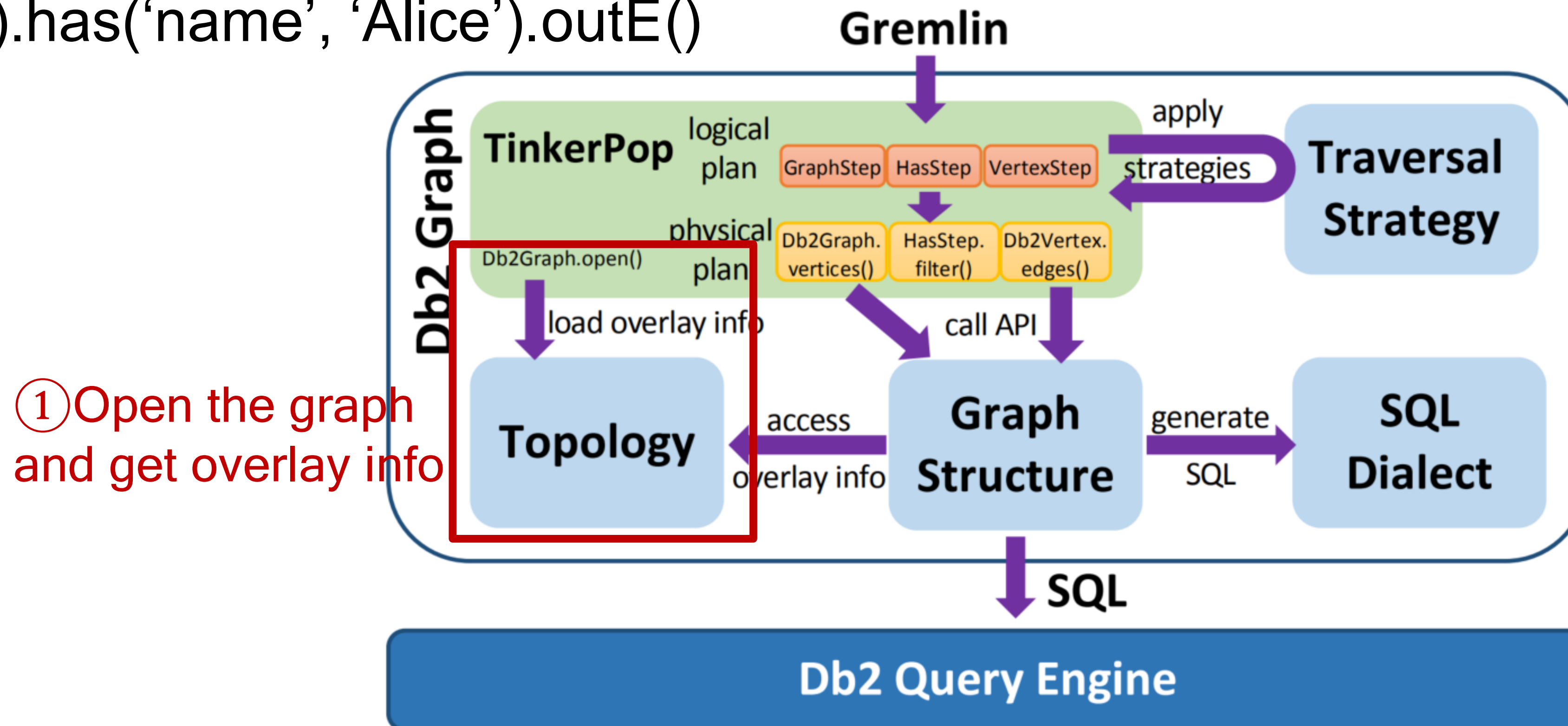


Figure 3: Db2 Graph architecture

IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020]

- Architecture

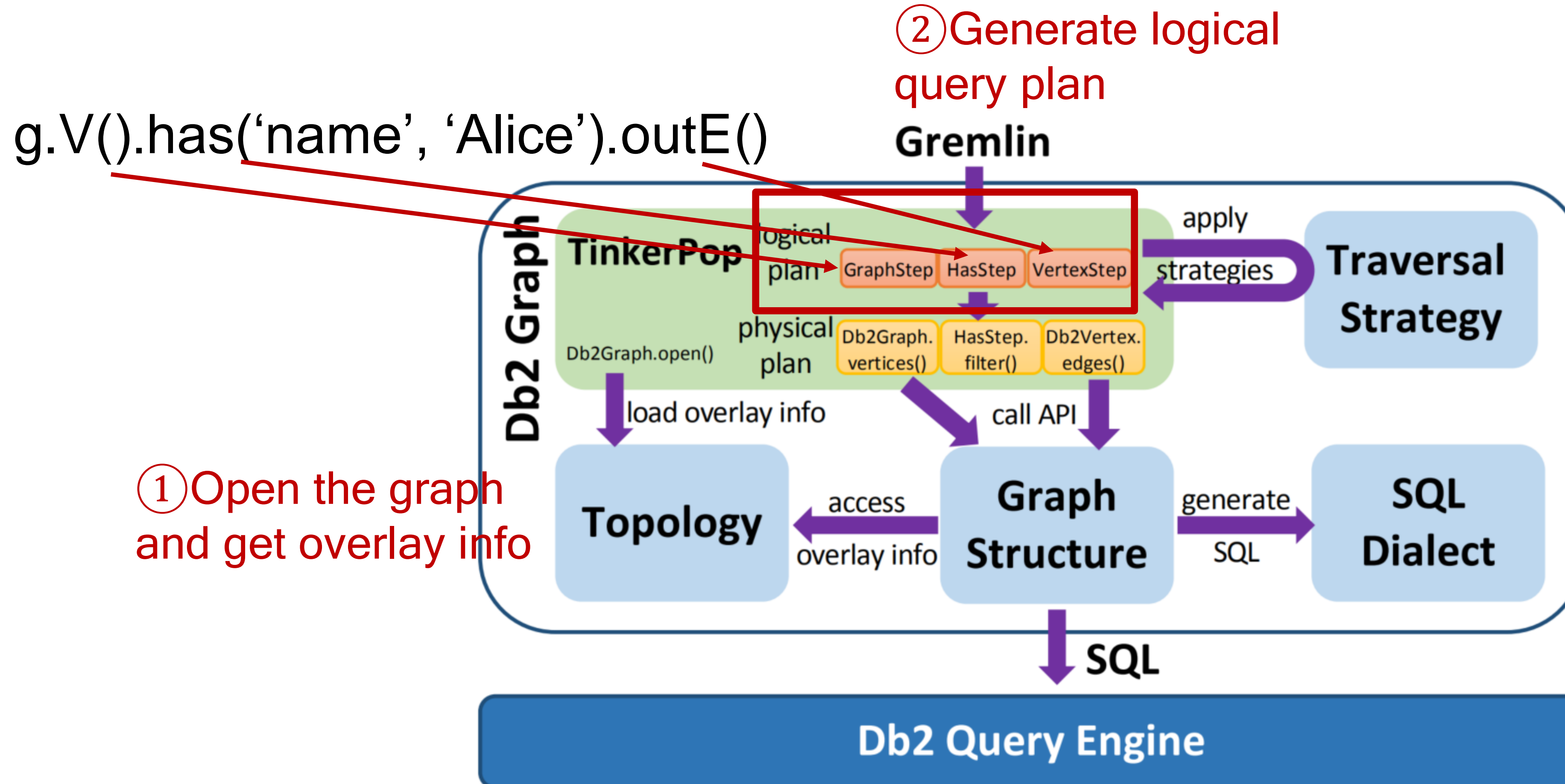


Figure 3: Db2 Graph architecture

IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020]

- Architecture

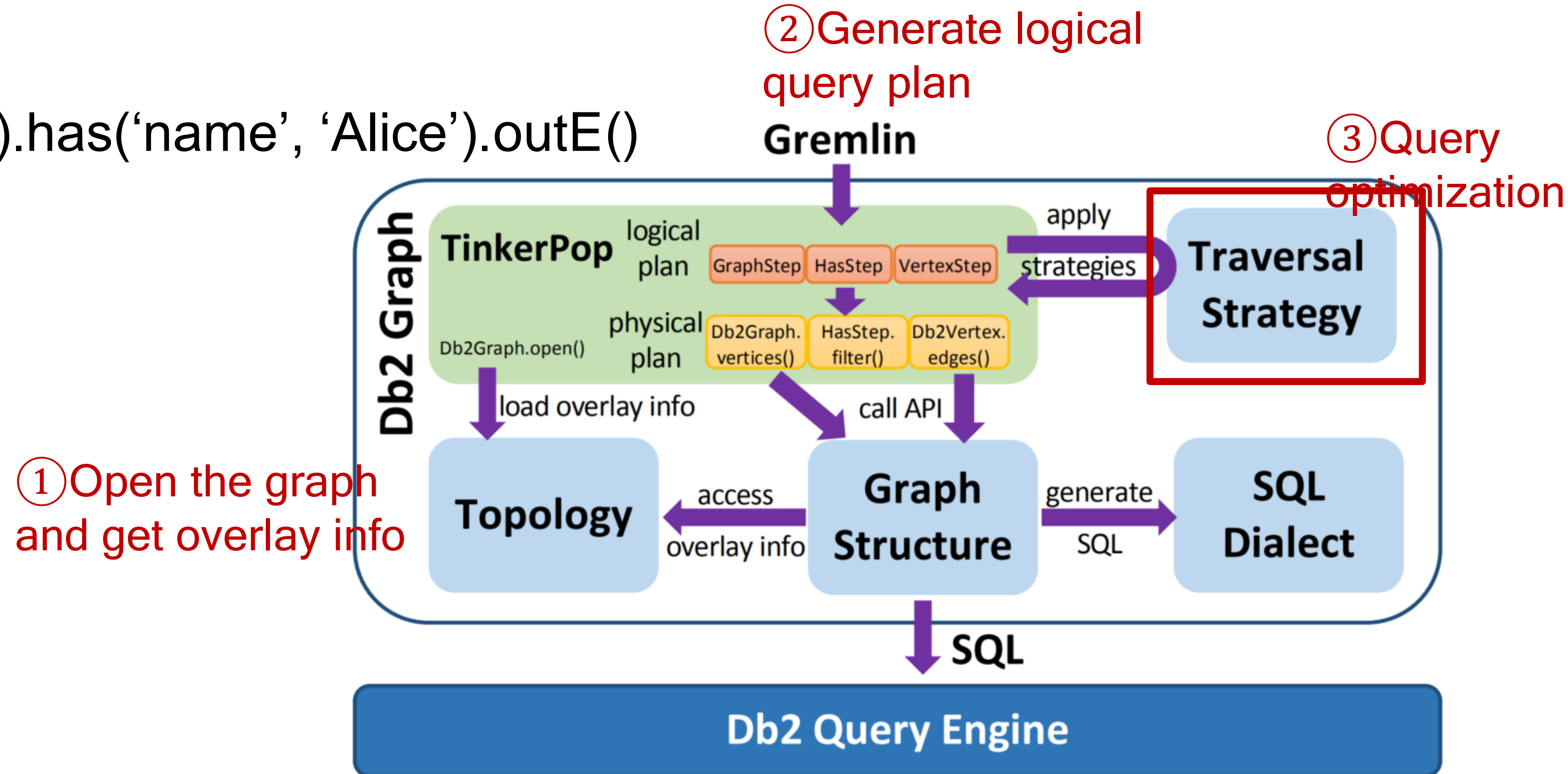


Figure 3: Db2 Graph architecture

IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020] - *Architecture*

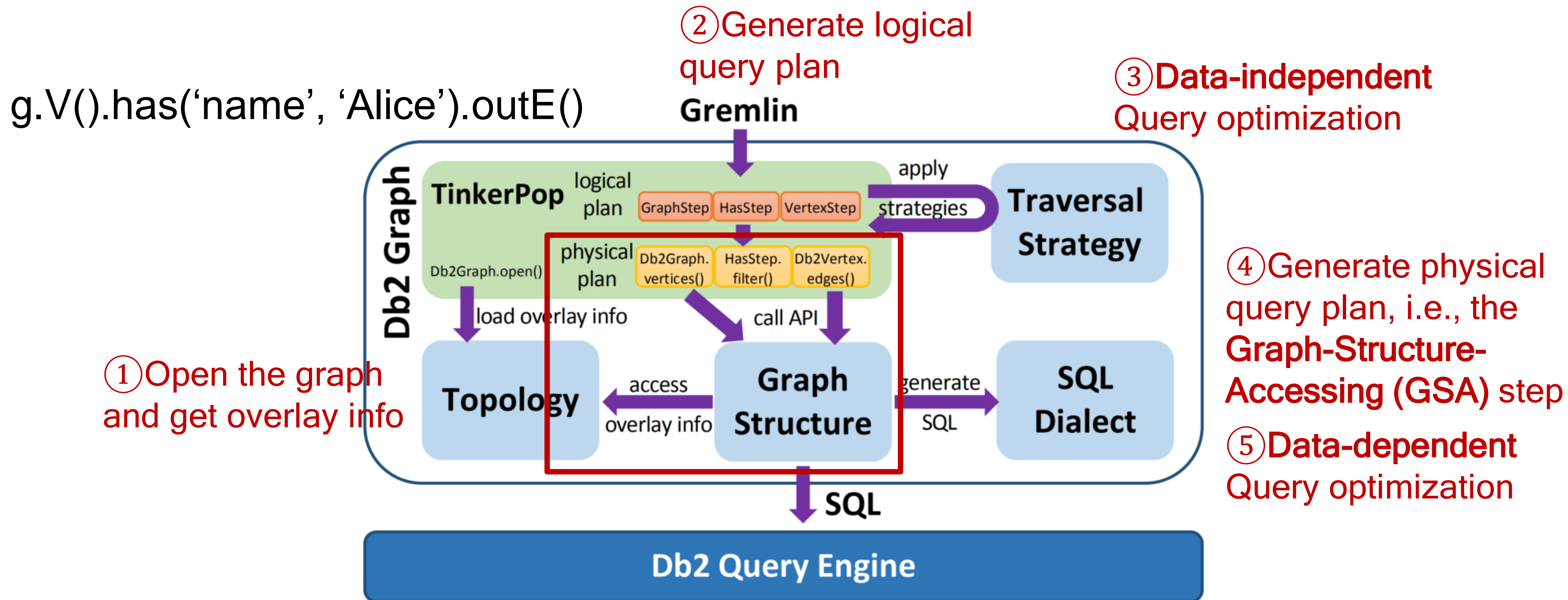


Figure 3: Db2 Graph architecture

IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020] - *Architecture*

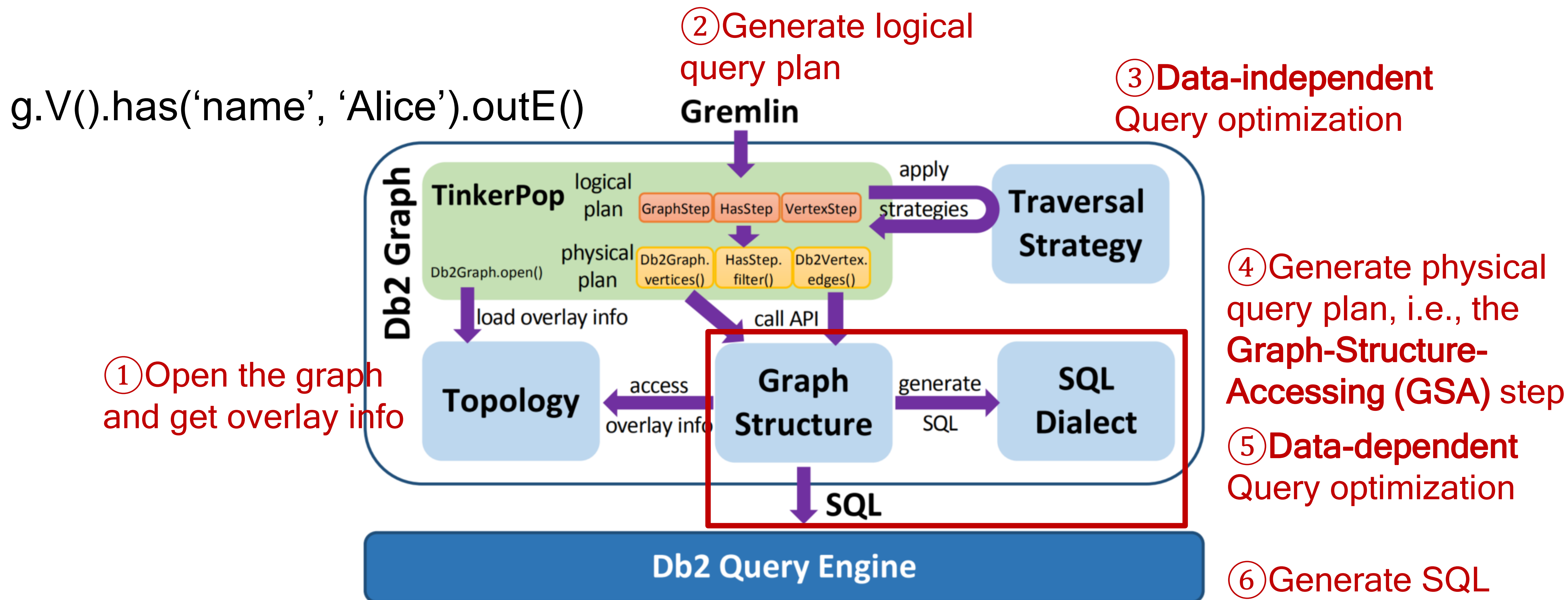


Figure 3: Db2 Graph architecture

IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020]
- *Query Optimization*

- Data-independent strategies
  - Predicate Pushdown with Filter Steps
    - E.g., for g.V().has('name', 'Alice'), fold the HasStep into the GraphStep
  - Projection Pushdown with Properties Steps
    - E.g., for g.V().values('name', 'address'), the GraphStep is "SELECT id, label, name, address FROM …"
  - Aggregate Pushdown with Aggregation Steps
  - …
- Data-dependent strategies
  - Use src_v_table/dst_v_table to record from which relational table the nodes/edges are mapped
  - Using properties of the graph
    - Using Property Names in Pushdown Information
    - Using Label/Prefix ID Values/…

IBM Db2 Graph: Graph Queries Inside IBM Db2 [VLDB 2019, SIGMOD 2020] - *Experimental Study*

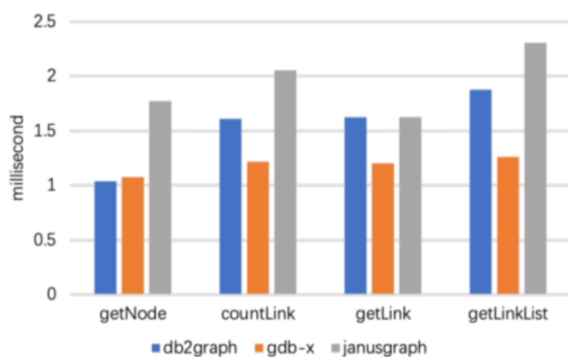
- Graph loading time matters!
- IBM Db2 Graph achieves satisfactory query efficiency on LinkBench (simple queries)

Table 1: LinkBench Queries

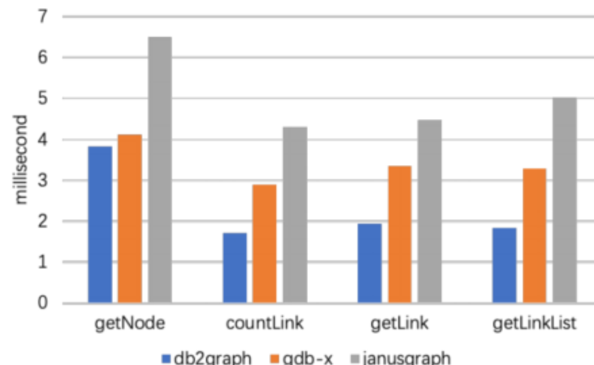
LinkBench Query	Gremlin
getNode(id, lbl)	g.V(id).hasLabel(lbl)
countLinks(id1,lbl)	g.V(id1).outE(lbl).count()
getLink(id1,lbl,id2)	g.V(id1).outE(lbl).filter(outV().id() == id2)
getLinkList(id1,lbl)	g.V(id1).outE(lbl)

Table 3: Graph loading time for different graph databases

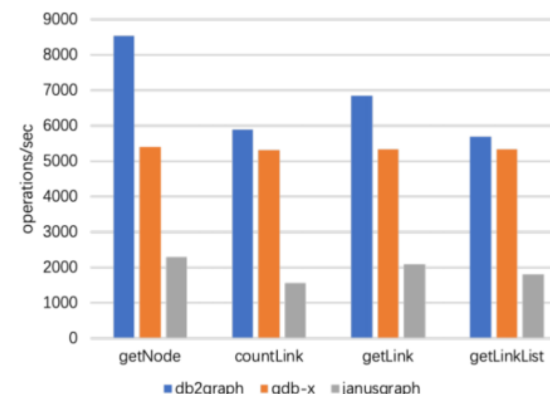
	Db2 Graph		Export	GDB-X			JanusGraph		
Linkbench Dataset	Disk Usage	Open Graph	From DB	Disk Usage	Load Data	Open Graph	Disk Usage	Load Data	Open Graph
10M	4.6GB	1.4 sec	5 min	28GB	42 min	14 sec	29GB	65 min	15 sec
100M	45.8GB	2.1 sec	32 min	327GB	8 hr	15 sec	326GB	13.5 hr	17 sec



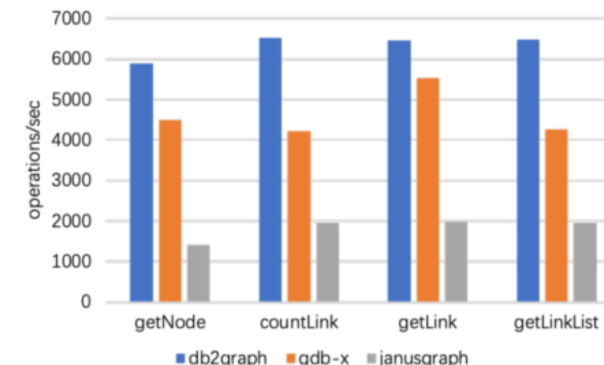
(a) Linkbench-10M



(b) Linkbench-100M



(a) Linkbench-10M



(b) Linkbench-100M

Latency

Throughputs

Overview of Graph Techniques for Relational Queries

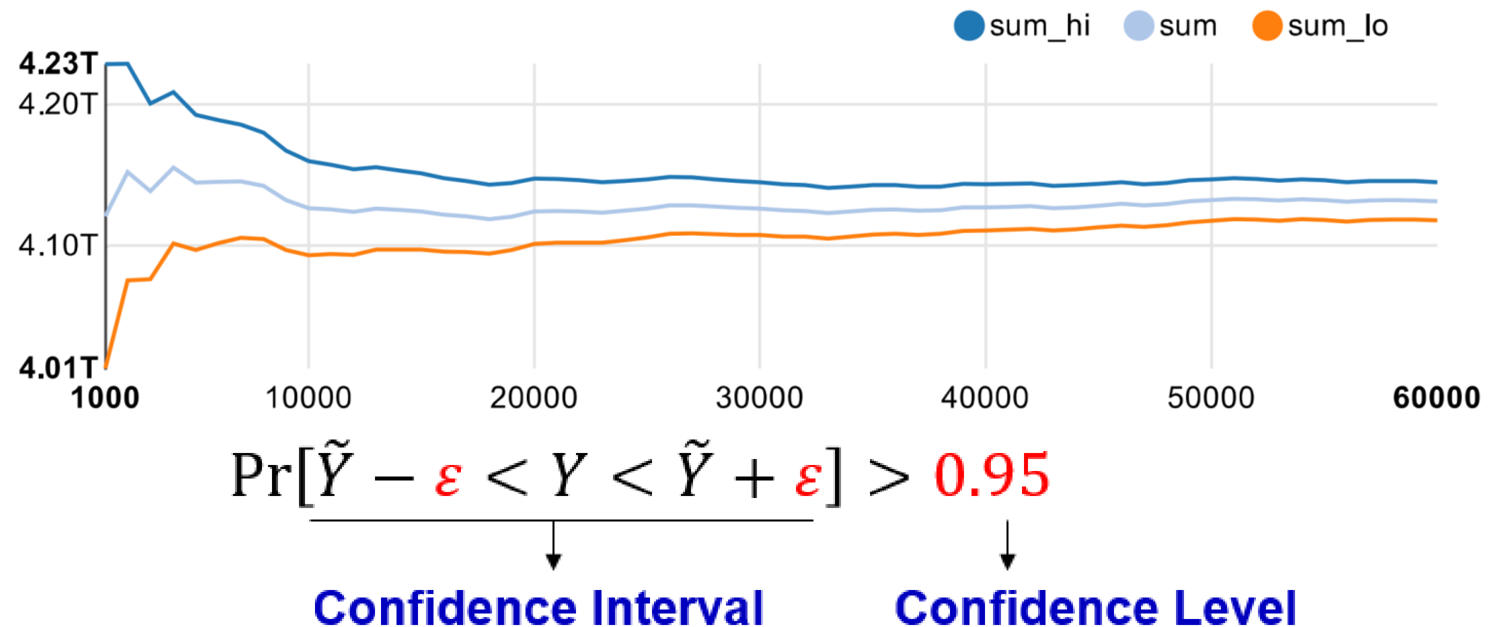
- Think in terms of graph processing when dealing with joins
- Understanding the advantages and disadvantages of GDBMSs over RDBMSs
- Improving analytical queries (OLAP) such as TPC-H/DS using GDBMSs

Wander Join: Online Aggregation via Random Walks [SIGMOD 2016]

- Online aggregation
  - Analytical queries do not always need 100% accuracy
  - Return an approximate answer with improving 'quality' guarantee
- How do we estimate an aggregate query that involves multiple joins?
- Notion of quality: express in form of confidence intervals

```
SELECT SUM(l_extendedprice * (1 - l_discount))
FROM customer, lineitem, orders, nation, region
WHERE c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND l_returnflag = 'R'
  AND c_nationkey = n_nationkey
  AND n_regionkey = r_regionkey
  AND r_name = 'ASIA'
```

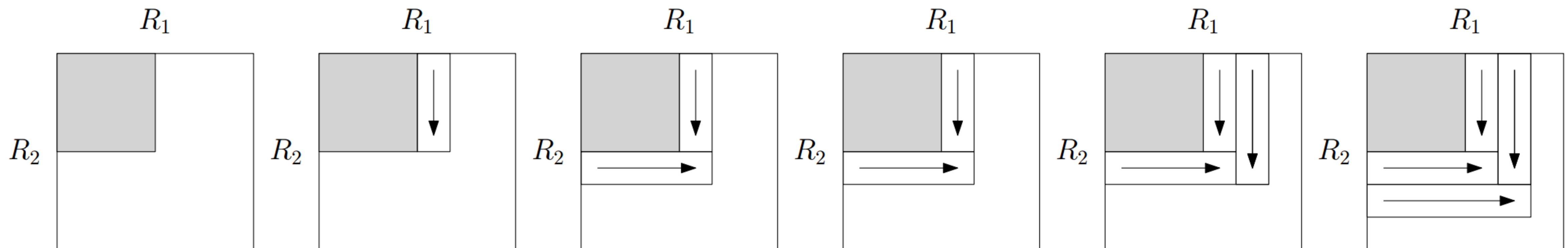
(This query finds the total revenue loss due to returned orders in a given region)



Li, Feifei, et al. "Wander join: Online aggregation via random walks." *Proceedings of the 2016 International Conference on Management of Data*. 2016.

Ripple Join

- Store tuples in each table in random order
- In each step
  - Reads the next tuple from a table in a round-robin fashion
  - Join with sampled tuples from other tables
  - Estimate the aggregation value from samples, calculate confidence interval from estimator (using the central limit theorem)
- Works well for full Cartesian product
  - But most joins are sparse



Peter J. Haas, Joseph M. Hellerstein: Ripple Joins for Online Aggregation. SIGMOD Conference 1999: 287-298

Ripple Join

What's the total revenue of all orders from customers in China?

Nation	CID
US	1
US	2
China	3
UK	4
China	5
US	6
China	7
UK	8
Japan	9
UK	10

BuyerID	OrderID
4	1
3	2
1	3
5	4
5	5
5	6
3	7
5	8
3	9
7	10

OrderID	ItemID	Price
4	301	$2100
2	304	$100
3	201	$300
4	306	$500
3	401	$230
1	101	$800
2	201	$300
5	101	$200
4	301	$100
2	201	$600

N: size of each table, e.g., 10^9
n: # tuples taken from each table
s: # estimators, e.g., 10^3

$$n^3 \cdot \frac{1}{N^2} = s$$

$$n = N^{2/3} s^{1/3} = 10^7$$

Peter J. Haas, Joseph M. Hellerstein: Ripple Joins for Online Aggregation. SIGMOD Conference 1999: 287-298

Ripple Join

What's the total revenue of all orders from customers in China?

Nation	CID
US	1
US	2
China	3
UK	4
China	5

BuyerID	OrderID
4	1
3	2
1	3
5	4
5	5

OrderID	ItemID	Price
4	301	$2100
2	304	$100
3	201	$300
4	306	$500
3	401	$230

N: size of each table, e.g., 10^9

n: # tuples taken from each table

s: # estimators, e.g., 10^3

$$n^3 \cdot \frac{1}{N^2} = s$$

$$n = N^{2/3} s^{1/3} = 10^7$$

Estimator for sum:

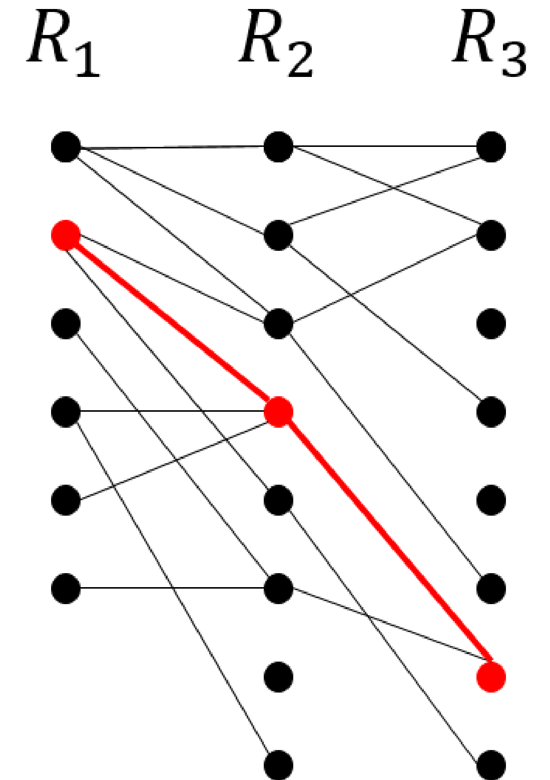
$$SUM(expression(R,S)) = \frac{|R| \times |S|}{|R_n| \times |S_n|} \sum_{(r,s) \in R_n \times S_n} expression_p(r,s)$$

$$expression_p(r,s) = \begin{cases} 0, & \text{if fails WHERE} \\ expression(r,s), & \text{otherwise} \end{cases}$$

Wander Join: Online Aggregation via Random Walks [SIGMOD 2016]

-Join as a Graph

- Take a randomly sampled tuple from ONLY one table
- Conduct a random walk from that tuple to the neighbors (join tuples)
  - For queries with many join relations, there may be different *walk paths*
  - Can handle cyclical queries
  - Assumes indexes on other tables
- Provide an unbiased estimator for each aggregator
- Does not provide consistent result: must run full join in conjunction with wander join



Conceptual only
Never materialized

Wander Join: Online Aggregation via Random Walks [SIGMOD 2016]

-Join as a Graph

```
SELECT SUM(Price)
FROM Customers C,
     Orders O,
     Items I
WHERE
 C.Nation = 'China'
 C.CID = O.BuyerID
 O.OrderID =
       I.OrderID
```

Nation	CID
US	1
US	2
China	3
UK	4
China	5
US	6
China	7
UK	8
Japan	9
UK	10

BuyerID	OrderID
4	1
3	2
1	3
5	4
5	5
5	6
3	7
5	8
3	9
7	10

OrderID	ItemID	Price
4	301	$2100
2	304	$100
3	201	$300
4	306	$500
3	401	$230
1	101	$800
2	201	$300
5	101	$200
4	301	$100
2	201	$600

Wander Join: Online Aggregation via Random Walks [SIGMOD 2016]

-Join as a Graph

```
SELECT SUM(Price)
FROM Customers C,
     Orders O,
     Items I
WHERE
 C.Nation = 'China'
 C.CID = O.BuyerID
 O.OrderID =
       I.OrderID
```

Nation	CID
US	1
US	2
China	3
UK	4
China	5
US	6
China	7
UK	8
Japan	9
UK	10

BuyerID	OrderID
4	1
3	2
1	3
5	4
5	5
5	6
3	7
5	8
3	9
7	10

OrderID	ItemID	Price
4	301	$2100
2	304	$100
3	201	$300
4	306	$500
3	401	$230
1	101	$800
2	201	$300
5	101	$200
4	301	$100
2	201	$600

Wander Join: Online Aggregation via Random Walks [SIGMOD 2016]

-Join as a Graph

```
SELECT SUM(Price)
FROM Customers C,
     Orders O,
     Items I
WHERE
 C.Nation = 'China'
 C.CID = O.BuyerID
 O.OrderID =
       I.OrderID
```

Nation	CID
US	1
US	2
China	3
UK	4
China	5
US	6
China	7
UK	8
Japan	9
UK	10

BuyerID	OrderID
4	1
3	2
1	3
5	4
5	5
5	6
3	7
5	8
3	9
7	10

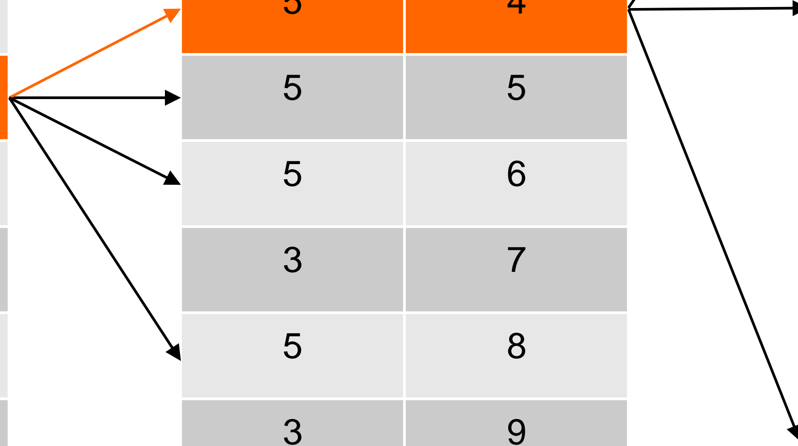
OrderID	ItemID	Price
4	301	$2100
2	304	$100
3	201	$300
4	306	$500
3	401	$230
1	101	$800
2	201	$300
5	101	$200
4	301	$100
2	201	$600

Wander Join: Online Aggregation via Random Walks [SIGMOD 2016]

-Join as a Graph

```
SELECT SUM(Price)
FROM Customers C,
     Orders O,
     Items I
WHERE
 C.Nation = 'China'
 C.CID = O.BuyerID
 O.OrderID =
       I.OrderID
```

Nation	CID
US	1
US	2
China	3
UK	4
China	5
US	6
China	7
UK	8
Japan	9
UK	10



BuyerID	OrderID
4	1
3	2
1	3
5	4
5	5
5	6
3	7
5	8
3	9
7	10

OrderID	ItemID	Price
4	301	$2100
2	304	$100
3	201	$300
4	306	$500
3	401	$230
1	101	$800
2	201	$300
5	101	$200
4	301	$100
2	201	$600

Wander Join: Online Aggregation via Random Walks [SIGMOD 2016]

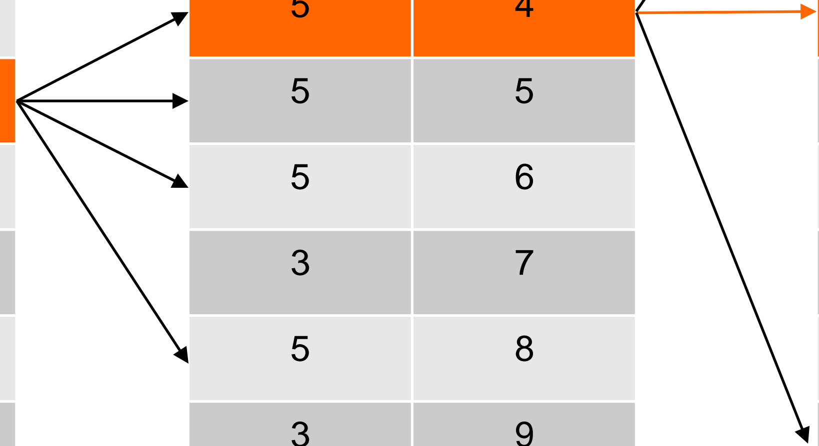
-Join as a Graph

```
SELECT SUM(Price)
FROM Customers C,
     Orders O,
     Items I
WHERE
 C.Nation = 'China'
 C.CID = O.BuyerID
 O.OrderID =
       I.OrderID
```

Nation	CID
US	1
US	2
China	3
UK	4
China	5
US	6
China	7
UK	8
Japan	9
UK	10

BuyerID	OrderID
4	1
3	2
1	3
5	4
5	5
5	6
3	7
5	8
3	9
7	10

OrderID	ItemID	Price
4	301	$2100
2	304	$100
3	201	$300
4	306	$500
3	401	$230
1	101	$800
2	201	$300
5	101	$200
4	301	$100
2	201	$600



Wander Join: Online Aggregation via Random Walks [SIGMOD 2016]

-Join as a Graph

```
SELECT SUM(Price)
FROM Customers C,
     Orders O,
     Items I
WHERE
 C.Nation = 'China'
 C.CID = O.BuyerID
 O.OrderID =
       I.OrderID
```

Nation	CID
US	1
US	6
China	7
UK	
Japar	
UK	

BuyerID	OrderID
4	1
5	6
3	7

OrderID	ItemID	Price
4	301	$2100
		00
		0
		0
		0
1	101	$800
2	201	$300
		$200
		$100
		$600

N: size of each table size, e.g., 10^9

n: # tuples taken from each table = # random walks

s: # estimators, e.g., 10^3

$$n = s = 10^3$$

Unbiased estimator: $\frac{\$500}{\text{sampling prob.}} = \frac{\$500}{1/3 \cdot 1/4 \cdot 1/3}$

Wander Join: Online Aggregation via Random Walks [SIGMOD 2016]

- Sampling by Random Walks

- Estimator of aggregate might be biased
  - Penalize paths that are sampled with higher probability proportionally
- Unbiased estimator
  - Walk plan optimization

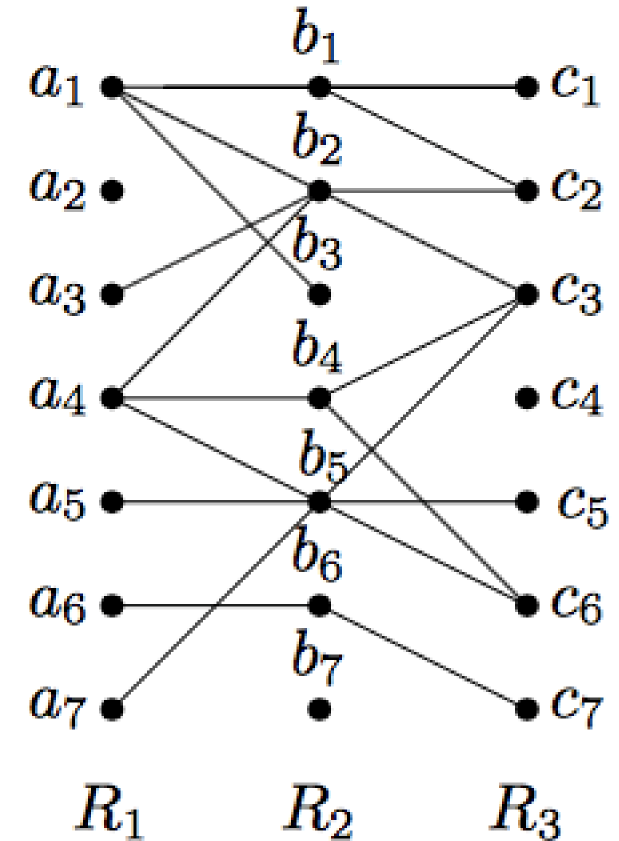
$$\gamma = \text{path}$$

$$v(\gamma) = \text{aggregate on } \gamma$$

$$p(\gamma) = \text{probability of } \gamma$$

$$\sum_{\gamma} v(\gamma) = \text{SUM(expression) from ripple}$$

$$\text{Then } \frac{v(\gamma)}{p(\gamma)} \text{ is unbiased estimator}$$



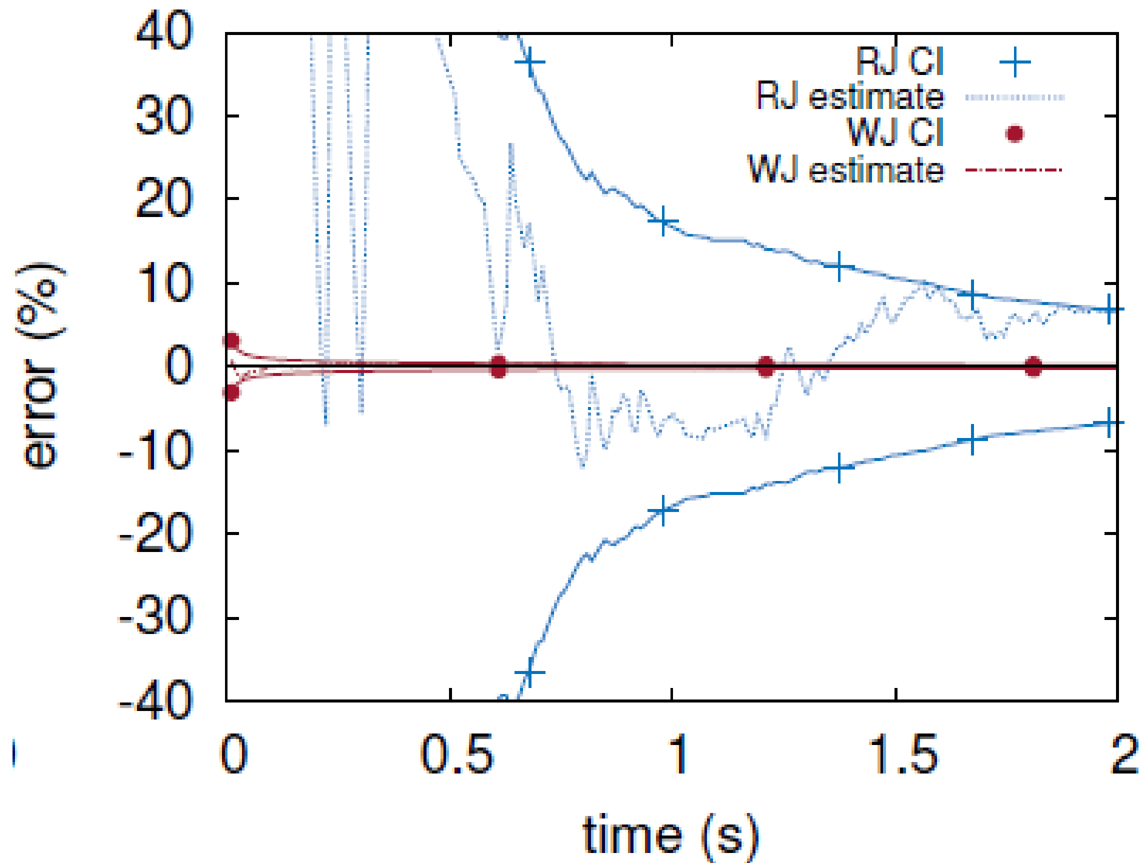
$$p(a_1, b_1, c_1) = 1/7 \times 1/3 \times 1/2$$

$$p(a_6, b_6, c_7) = 1/7 \times 1 \times 1$$

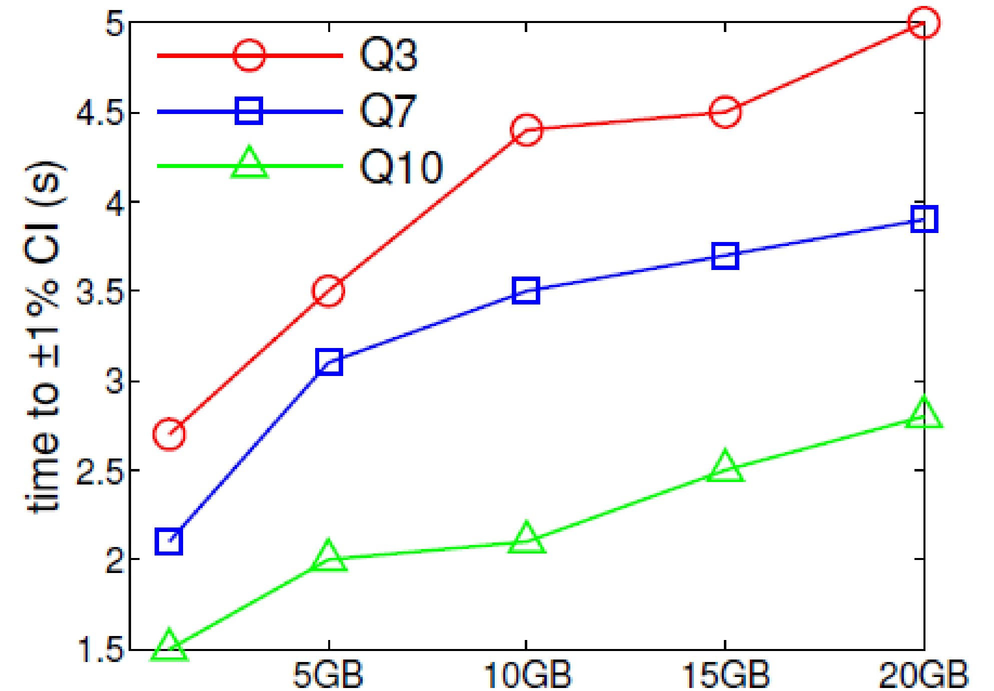
Wander Join: Online Aggregation via Random Walks [SIGMOD 2016]

- Experimental Study

Convergence Comparison



Wander Join in PostgreSQL



(b) Wander join in PostgreSQL

Logarithmic growth due to B-tree lookup to find random neighbours

Which Category Is Better: Benchmarking Relational and Graph Database Management Systems [DSE 2019]

- A Unified Benchmark

- Evaluate RDBMSs and GDBMSs on the same datasets
  - Extend TPC-H to evaluate GDBMSs
  - Extend LDBC to evaluate RDBMSs

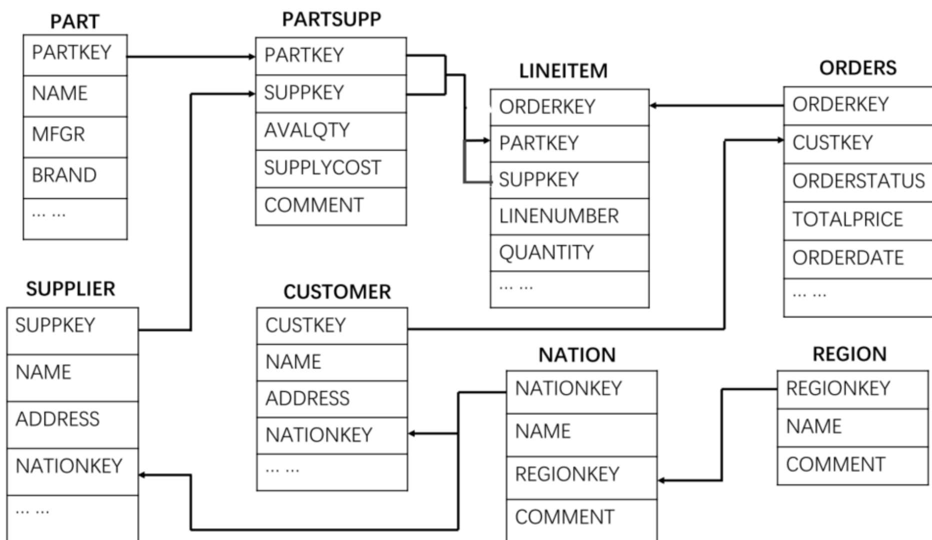


Fig. 1 The database schema for TPC-H benchmark

records -> vertices
PK-FK relationship -> edges

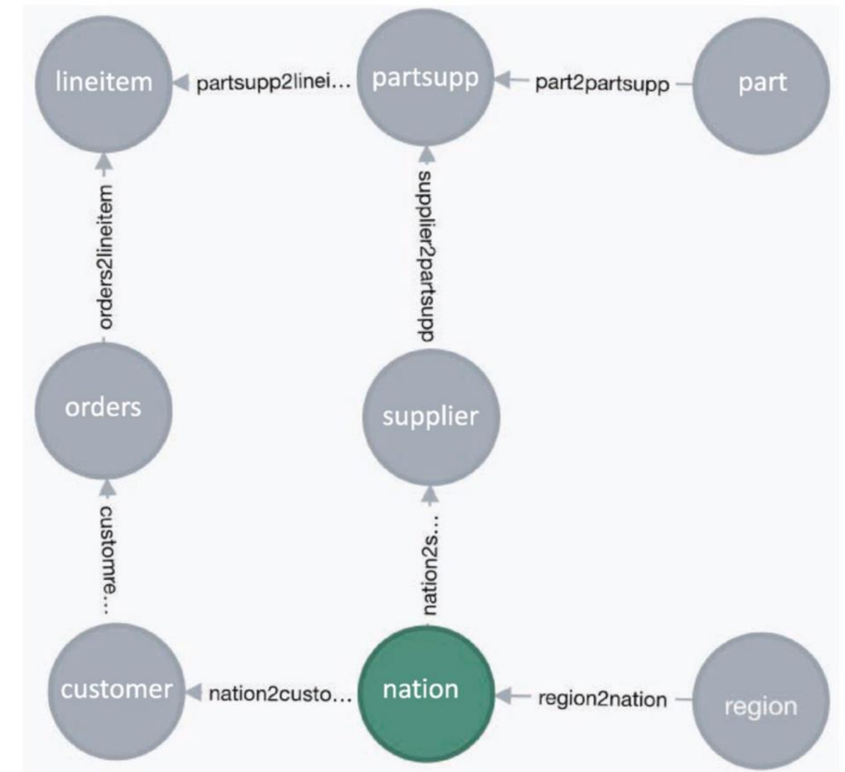


Fig. 2 The graph schema for TPC-H benchmark

Yijian Cheng, Pengjie Ding, Tongtong Wang, Wei Lu, Xiaoyong Du: Which Category Is Better: Benchmarking Relational and Graph Database Management Systems. Data Sci. Eng. 4(4): 309-322 (2019)

Which Category Is Better: Benchmarking Relational and Graph Database Management Systems [DSE 2019]

- A Unified Benchmark

- Evaluate RDBMSs and GDBMSs on the same datasets
  – Extend TPC-H to evaluate GDBMSs
  – Extend LDBC to evaluate RDBMSs
- Graph-to-relation mapping
  – Simply store the directed edges as triples (*fromVertex, edgeLabel, toVertex*)
- Datasets

Table 3 TPC-H datasets

ID	Size	Vertices	Edges
tpch-0.05	50 MB	432,844	2,261,723
tpch-0.1	100 MB	866,602	4,530,029
tpch-0.5	500 MB	4,330,622	22,634,256
tpch-1	1 GB	8,661,245	45,268,530

Table 4 The real graph datasets

Graphs	Vertices	Edges	Size	Domain
Wiki-Vote	7115	103,689	S	Social
Cit-HepTh	27,770	352,807	M	Citation
Web-Stanford	281,903	2,312,497	L	Web graphs
Wiki-Talk	2,394,385	5,021,410	XL	Communication

Which Category Is Better: Benchmarking Relational and Graph Database Management Systems [DSE 2019]

- *Query Workloads*

- **Atomic relational queries**, including Projection, Aggregation, Join, and Order by
- **TPC-H query workloads** (22 queries)
- **Graph query workloads**, including BFS, Community Detection using Label Propagation (CDLP), PageRank (PR), Local Clustering Coefficient (LCC), and Weakly Connected Components (WCC)

Algorithm 2 Cypher for TPC-H Query 2

1: **MATCH**$(ps : Partsupp) - [] - > (s : Supplier) - [] - > (n : Nation) - [] - > (r : Region)$
2: **WHERE**
3: $r.rName =' EUROPE'$
4: **WITH** $min(ps.psSupplycost)$ **as** $minvalue$
5: **MATCH**$(ps : Partsupp) - [] - > (p : Part), (ps : Partsupp) - [] - > (s : Supplier) - [] - > (n : Nation) - [] - > (r : Region)$
6: **WHERE**
7: $p.pSize = 13$ **and** $p.pType = \ '.*SMALL.*'$ **and** $r.rName =' EUROPE'$ **and** $ps.psSupplycost = minvalue$
8: **RETURN**
9: $s.sAcctbal,$
10: $s.sName,$
11: *and other elements*
12: **ORDER BY**
13: $s.sAcctbal\ desc, n.nName, s.sName, p.pPartkey$

Transform TPC-H into equivalent SQL-like graph query statements

Algorithm 3 Bread-First Search in SQL

1: **with RECURSIVE** $BFS(toID, level, fromid, paths)$
2: **as**(
3: **select** $toID, 0, fromID, ARRAY[null, toID]$ **from** R_{rel}
4: **where** $toID = m$ **and** $fromID$ is $NULL$
5: **union all**
6: **select** $R_{rel}.toID, level + 1, BFS.toID, paths || R_{rel}.toID$
7: **from** R_{rel}, BFS
8: **where** $R_{rel}.fromID = BFS.toID$
9: **and** $level < n$
10:)
11: **select** $level, paths$ **from** BFS

Implement the 5 graph algorithms in SQL using the procedure with While loop

Which Category Is Better: Benchmarking Relational and Graph Database Management Systems [DSE 2019]

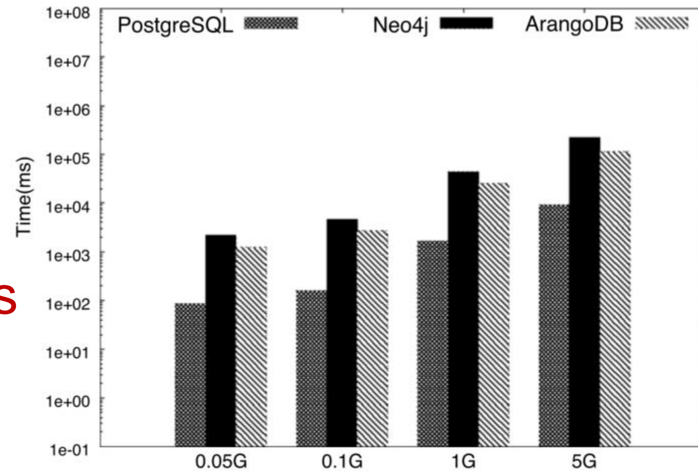
- Experiments

- Tested databases
  - RDBMSs: PostgreSQL (v9.5), Oracle (11g), MS SQL Server (2017)
  - GDBMSs: Neo4j (v3.4.6), ArangoDB (v3.3.19)
    - With varied back-end storage engines
- Metrics
  - Query processing time
  - Memory usage ratio
  - CPU usage ratio

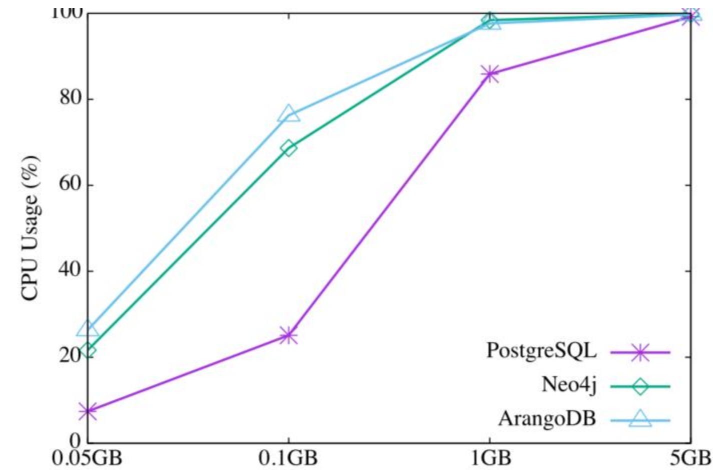
Which Category Is Better: Benchmarking Relational and Graph Database Management Systems [DSE 2019]

- Experiments on TPC-H Workloads

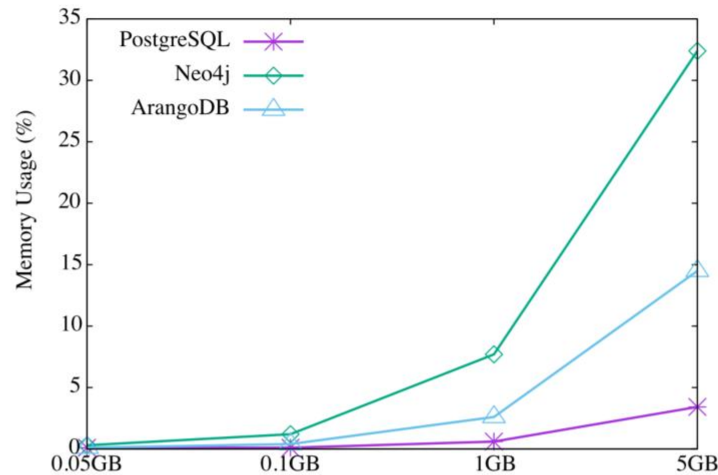
The GDBMSs show their inefficiency when dealing with TPC-H datasets



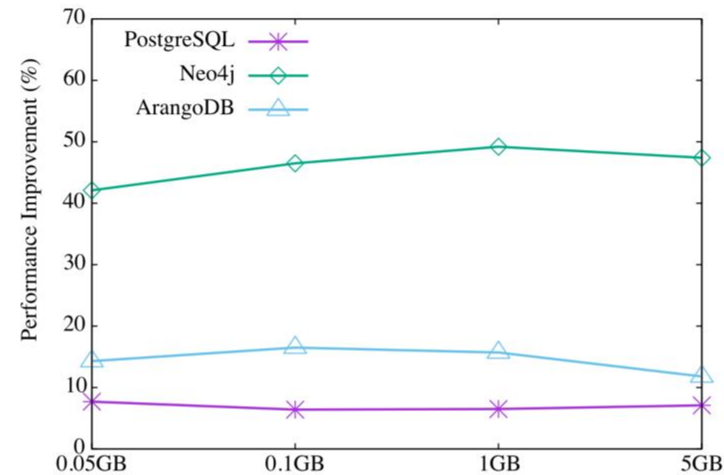
(a) Query Processing Time



(b) CPU Usage



(c) Memory Usage

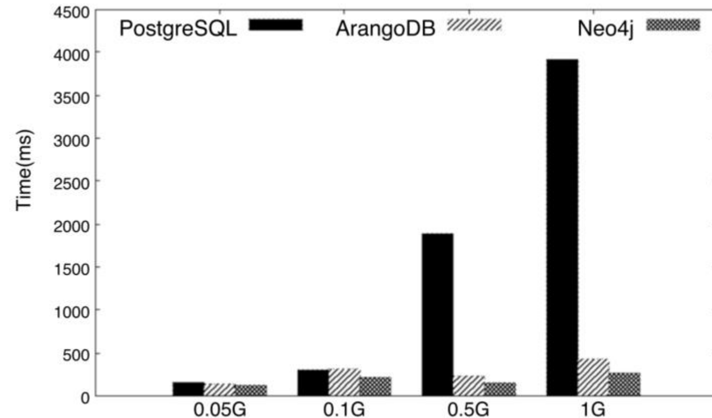


(d) Query optimization Results

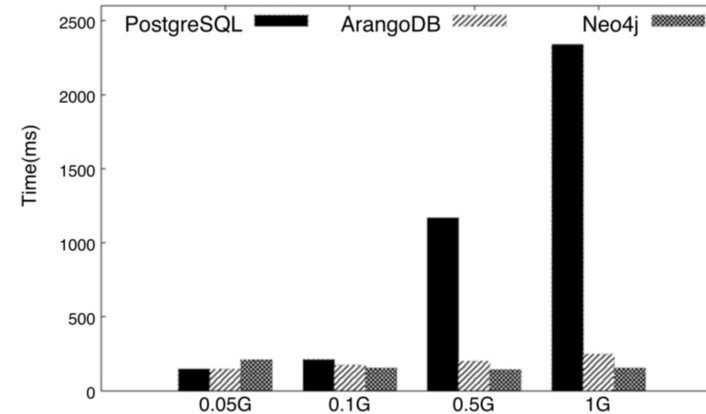
But can be further optimized for complex operations (Aggregation, Order By) via creating indices

Which Category Is Better: Benchmarking Relational and Graph Database Management Systems [DSE 2019]

- Experiments on Relational Operations

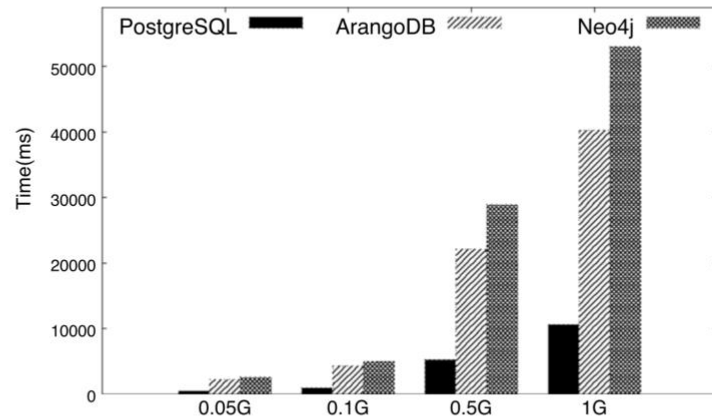


(a) PROJECTION

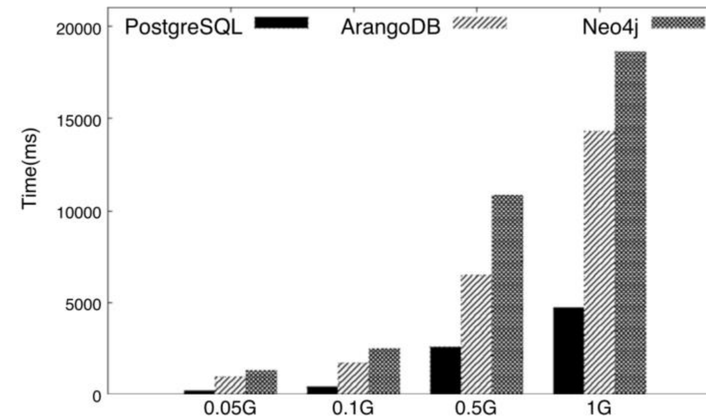


(b) JOIN

GDBMSs achieve better performance for Projection and Join operations



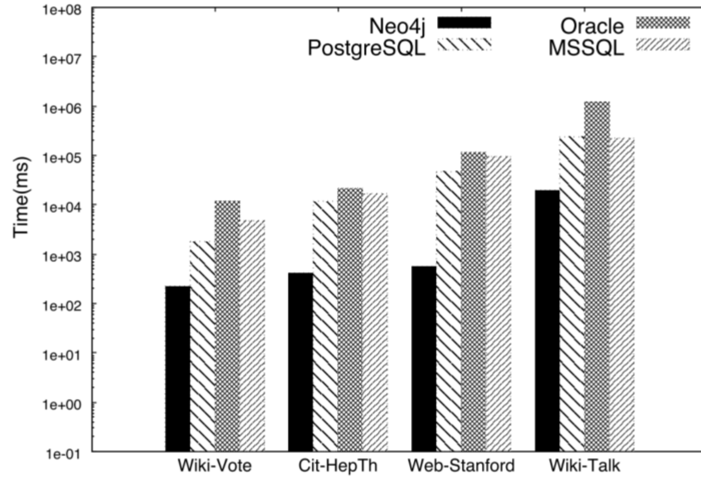
(c) AGGREGATION



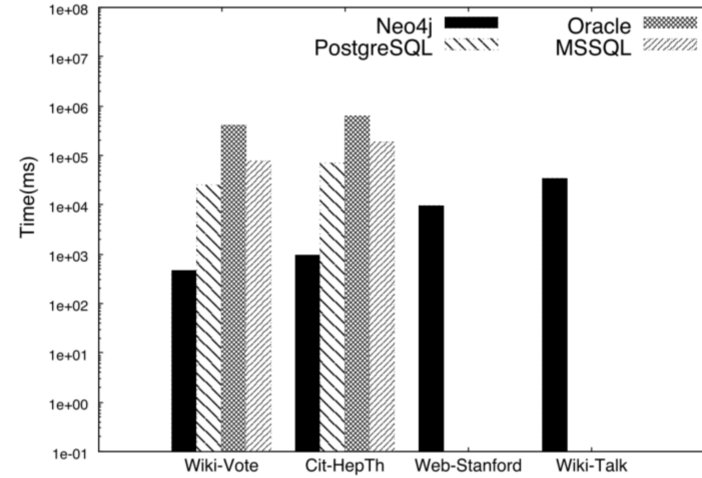
(d) ORDER BY

Which Category Is Better: Benchmarking Relational and Graph Database Management Systems [DSE 2019]

- Experiments on Graph Algorithms

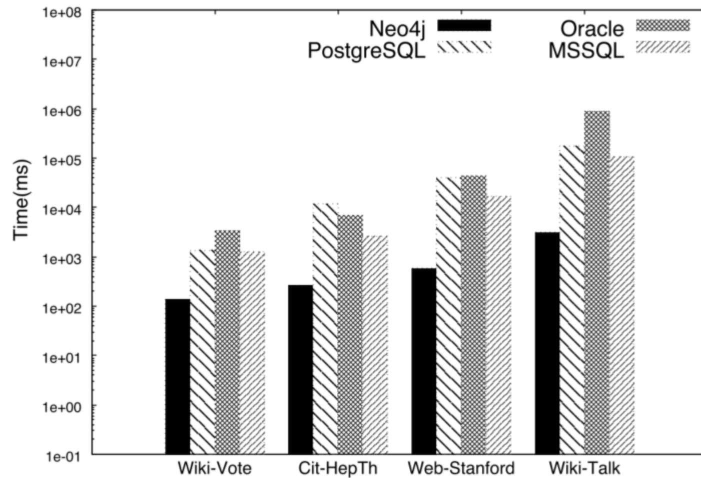


CDLP

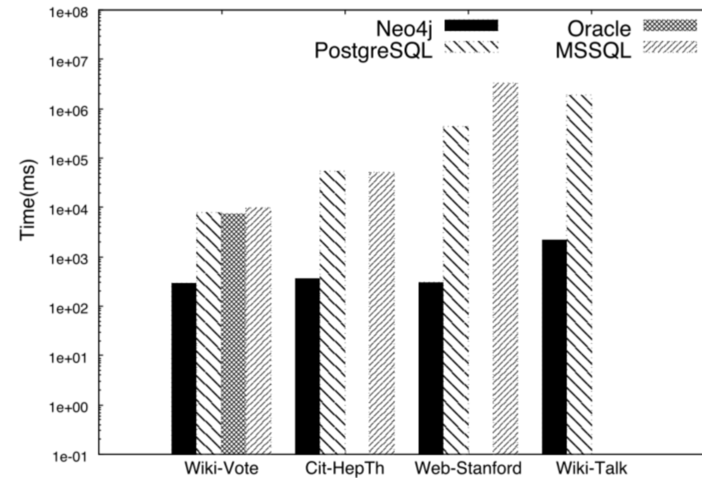


LCC

The intermediate results are of tremendous scale for LCC



PR



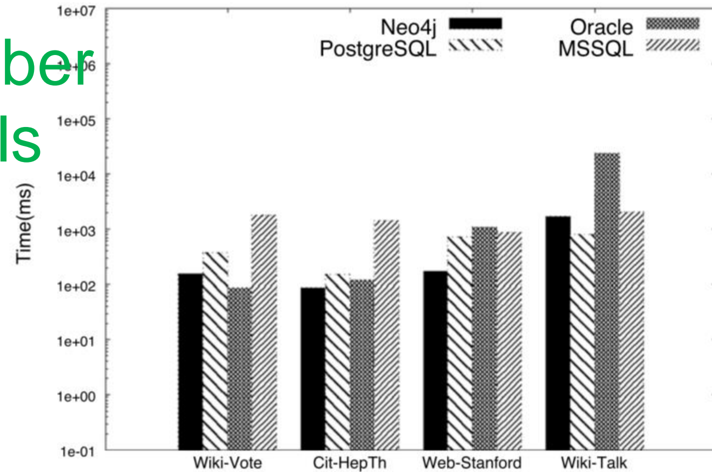
WCC

Multi-level recursive joins for WCC

Which Category Is Better: Benchmarking Relational and Graph Database Management Systems [DSE 2019]

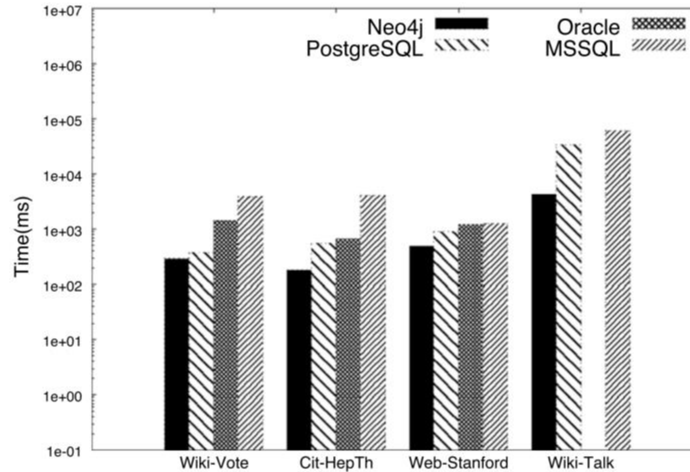
- Experiments on Graph Algorithms

Testing BFS by varying the number of traversal levels



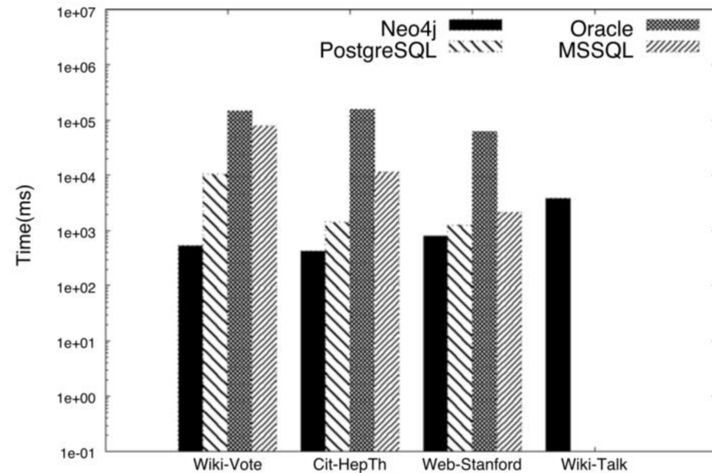
Similar performance at level 1

(a) level = 1

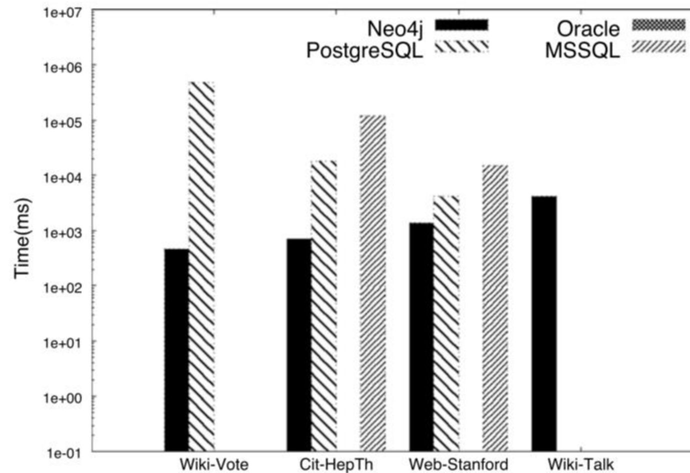


(b) level = 2

Self-joins are expensive for RDBMSs



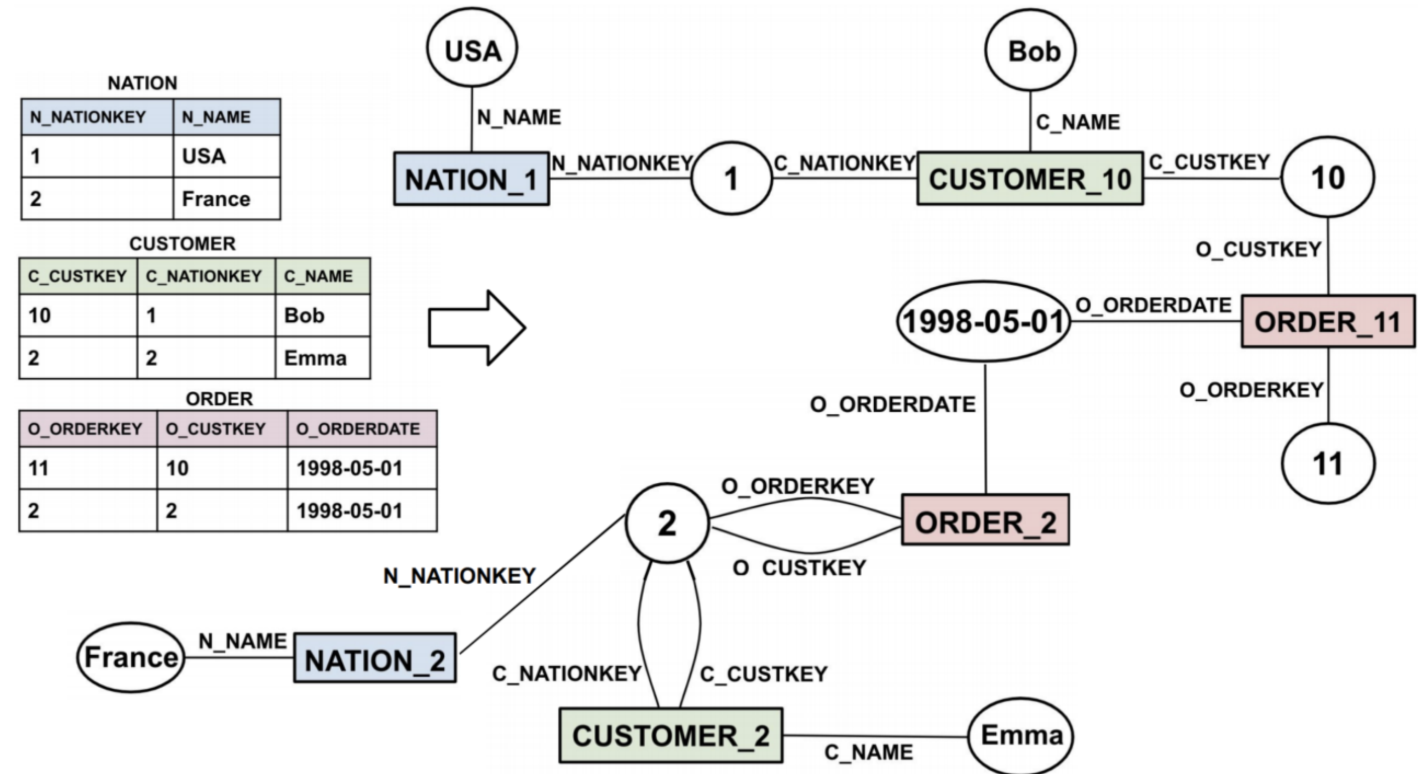
(c) level = 3



(d) level = 4

Vertex-centric Parallel Computation of SQL Queries [SIGMOD 2021]
- Tuple-Attribute Graph (TAG) Encoding

- Each vertex and edge has
  - A label, i.e., node/edge type
  - A collection of attributes (key-value pairs)
- Create exactly one vertex per value regardless of how many times the value occurs in the database
  - Essentially an RDF graph
- Attribute vertices acts as an indexing scheme for joins



NATION

N_NATIONKEY	N_NAME
1	USA
2	France

CUSTOMER

C_CUSTKEY	C_NATIONKEY	C_NAME
10	1	Bob
2	2	Emma

ORDER

O_ORDERKEY	O_CUSTKEY	O_ORDERDATE
11	10	1998-05-01
2	2	1998-05-01

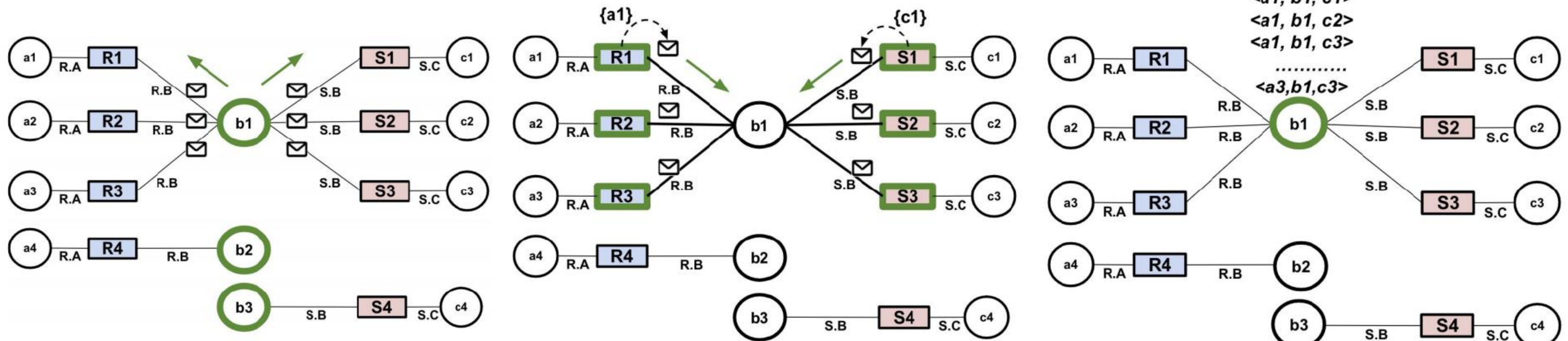
Figure 1: Encoding relational data in a TAG representation. Tuple vertices are depicted as rectangles, and attribute vertices as circles.

Ainur Smagulova, Alin Deutsch: Vertex-centric Parallel Computation of SQL Queries. SIGMOD Conference 2021: 1664-1677

Vertex-centric Parallel Computation of SQL Queries [SIGMOD 2021]

- Vertex-Centric Two-Way Join

- Vertex-centric computation based on Yannakakis' algorithm
  - First compute two semi-joins: $J_1 := R \ltimes S \quad J_2 := S \ltimes R.$
  - Conduct join on the reduced relations: $J_1 \bowtie J_2$



Computational cost：	IN=|R|+ |S|	|R⋉S|+|S⋉R|	|R⋉S|+|S⋉R|
Communication cost：	|R⋉S|+|S⋉R|=min(IN, OUT)	|R⋉S|+|S⋉R|	|R⋉S|+|S⋉R|

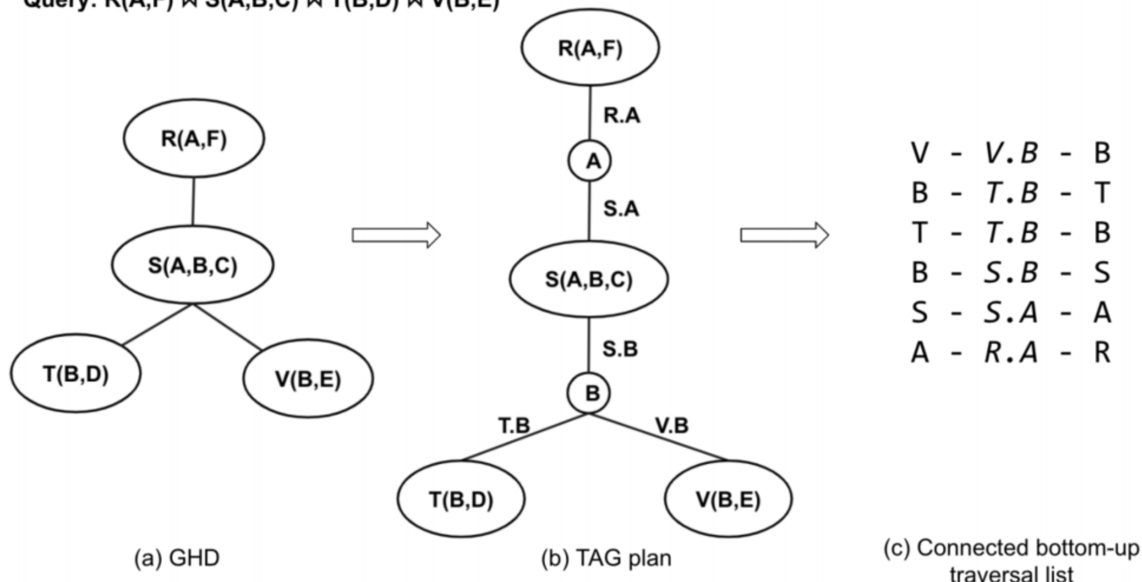
(Centralized & Factorized)

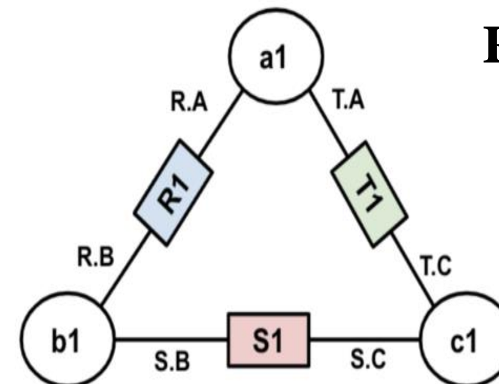
$R(A, B) \bowtie S(B, C)$

Vertex-centric Parallel Computation of SQL Queries [SIGMOD 2021]

- Acyclic Multi-Way Joins & Cyclic Joins

- TAG traversal plan generation
  - Generalized hypertree decomposition (GHD) of the query
  - Connected bottom-up traversal
- Vertex-centric algorithm
  - Reduction phase (O(IN) cost) and collection phase (O(OUT) cost)





R(A, B) ⋈ S(B, C) ⋈ T(A, C)

Can be improved to worst-case optimal (by the strategy of the NPRR algorithm)

$$(R_{heavy} \bowtie S) \ltimes T)) \cup ((R_{light} \bowtie T) \ltimes S)$$

if $|\sigma_{A=a}R| > \theta$ then (a,b) $\in R_{heavy}$,

otherwise (a,b) $\in R_{light}$

Vertex-centric Parallel Computation of SQL Queries [SIGMOD 2021]

- Experimental Study

- Compared with commercial RDBMSs on TPC-H/DS

Table 2: Number of TPC-DS queries where TAG-join approach outperforms, shows competitive or worse performance against each of the relational systems at SF-75. Total number of queries is 84.

#queries	outperforms	competitive	worse
psql	84	-	-
rdbX	74	4	4
rdbX_im	64	3	17
rdbY	53	22	9
rdbY_non	64	12	8
spark_sql	73	5	6

rdbX: leading commercial RDBMS/row store
rdbX_im: in-memory column store

rdbY: commercial RDBMS with row store support
rdbY_non: non-clustered primary key

Table 3: Runtime (in seconds) of TPC-DS workload at SF-75 broken down by aggregation type

SF-75	No agg	LA	GA	Scalar GA
psql	0.58	1913.7	7788.433	391.592
rdbX	2.42	72.3	1375.739	438.137
rdbX_im	1.71	43.9	1279.353	286.183
rdbY	0.52	42.8	1771.947	302.384
rdbY_non	0.32	138.3	2736.952	355.361
spark_sql	13.6	160.4	1549.5	231.5
TAG_tg	0.16	8.37	231.044	117.734

Table 4: Peak RAM usage percentage at SF-75.

%	psql	rdbX	rdbX_im	rdbY	spark_sql	TAG_tg
TPC-H	65.9	57.1	51.2	55.1	57.4	53.8
TPC-DS	61.7	49.8	43.5	54.3	68.1	52.9

Reference

- Tian, Yuanyuan. "The World of Graph Databases from An Industry Perspective." ACM SIGMOD Record 51.4 (2023): 60-67.
- Fan, Jing, Adalbert Gerald Soosai Raj, and Jignesh M. Patel. "The Case Against Specialized Graph Analytics Engines." CIDR. 2015.
- Alekh Jindal, Praynaa Rawlani, Eugene Wu, Samuel Madden, Amol Deshpande, Mike Stonebraker: VERTEXICA: Your Relational Friend for Graph Analytics! Proc. VLDB Endow. 7(13): 1669-1672 (2014)
- Jindal, Alekh, et al. "Graph analytics using vertica relational database." 2015 IEEE International Conference on Big Data (Big Data). IEEE, 2015.
- Zhao, Kangfei, and Jeffrey Xu Yu. "All-in-one: graph processing in RDBMSs revisited." Proceedings of the 2017 ACM International Conference on Management of Data. 2017.
- Tian, Yuanyuan, et al. "Synergistic graph and SQL analytics inside IBM Db2." Proceedings of the VLDB Endowment 12.12 (2019): 1782-1785.
- Tian, Yuanyuan, et al. "IBM db2 graph: Supporting synergistic and retrofittable graph queries inside IBM db2." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.
- Li, Feifei, et al. "Wander join: Online aggregation via random walks." Proceedings of the 2016 International Conference on Management of Data. 2016.
- Peter J. Haas, Joseph M. Hellerstein: Ripple Joins for Online Aggregation. SIGMOD Conference 1999: 287-298
- Yijian Cheng, Pengjie Ding, Tongtong Wang, Wei Lu, Xiaoyong Du: Which Category Is Better: Benchmarking Relational and Graph Database Management Systems. Data Sci. Eng. 4(4): 309-322 (2019)
- Ainur Smagulova, Alin Deutsch: Vertex-centric Parallel Computation of SQL Queries. SIGMOD Conference 2021: 1664-1677
- Mhedhbi, Amine, and Semih Salihoğlu. "Modern techniques for querying graph-structured relations: foundations, system implementations, and open challenges." Proceedings of the VLDB Endowment 15.12 (2022): 3762-3765.

05 Open problems and challenges

Open Problems and Challenges

- For designing multi-model data query languages
  - Design an algebra for a multi-model query language
  - General approaches for cross-model query optimization

- For RDBMS techniques supporting graph query and analytics
  - Leverage the vast amount of efficient graph algorithms
  - Achieve a balance between generality and efficiency of graph analytics

- For graph techniques/GDBMSs supporting relational queries
  - Improve GDBMSs in transactions, checkpointing and recovery, fault tolerance, durability, integrity constraints, …
  - Hybrid OLTP and OLAP graph processing systems

THANKS

Acknowledgement

- We would like to thank all the organizers and attendees of DASFAA 2023.
- We would also like to thank all the authors of the referenced papers for their contributions to the papers as the slides available on the Internet. We have borrowed generously from their papers and slides for this tutorial.
- This work is partially sponsored by CCF-Tencent Open Fund and Kingbase.