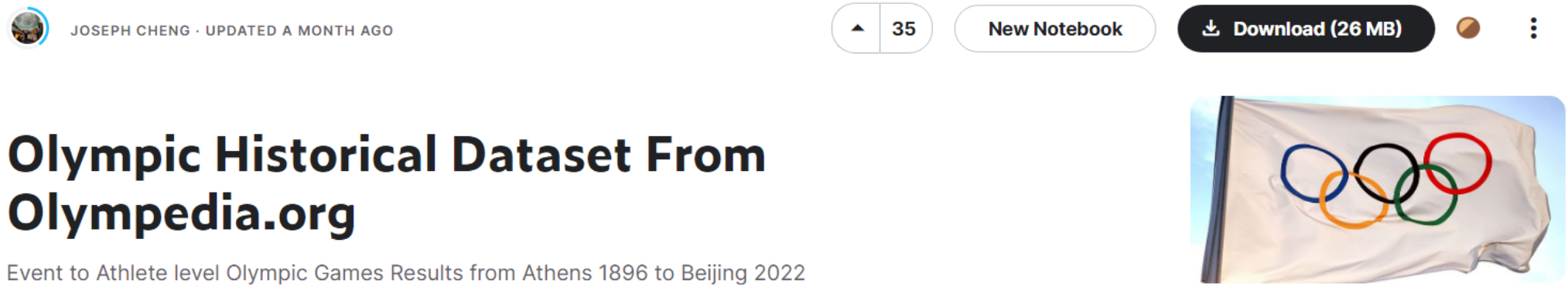
Exploring Graph Database Management Systems

Dataset



This dataset is as an attempt to create up to date Olympic Event datasets (from 1986 to 2022 Olympics Game) for any sports/data enthusiast to use to visualise and create some insights on the Olympic Event dataset.

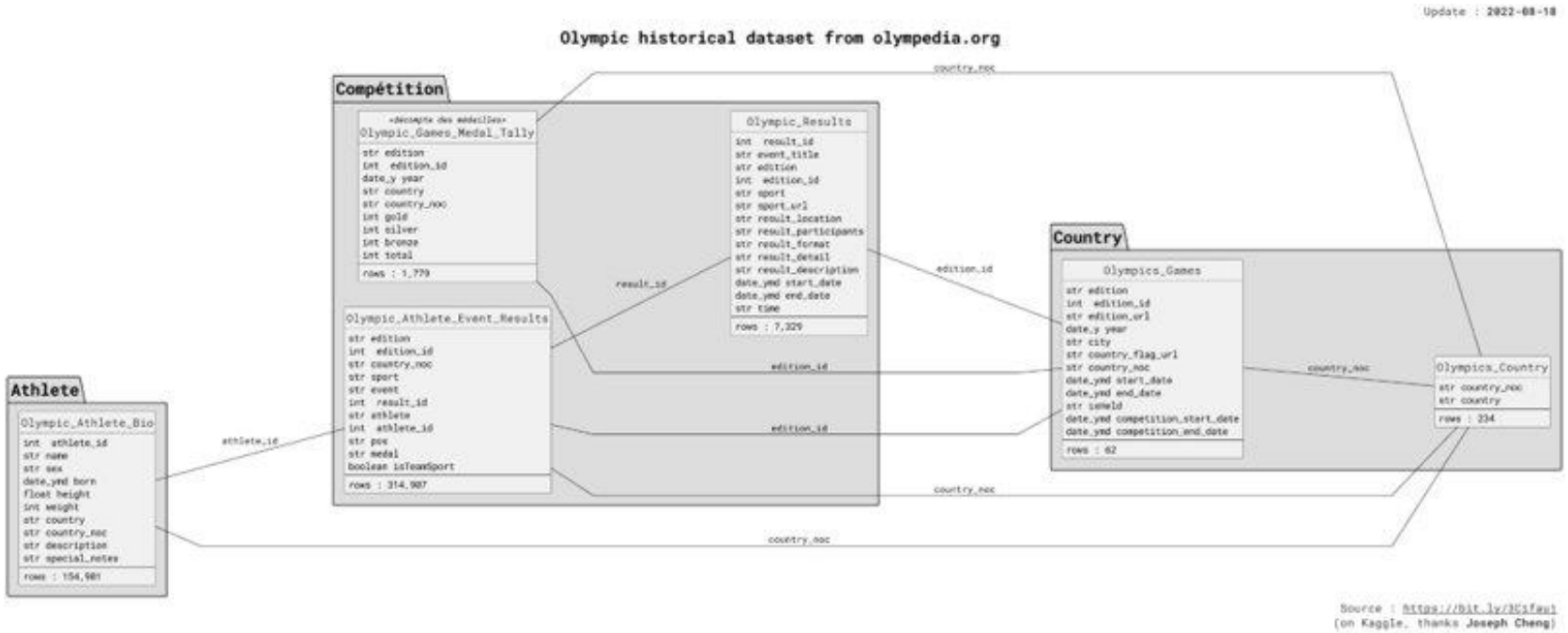
It contains ranking of each sporting event linked to specific country / athlete, which could be used to for any performance related analytics, and information about the athlete's bio, which could be useful in understanding more about the athlete.

The data is scrapped from www.olympedia.org which has the latest up to date olympic data set. Web Scrapping Project is provided via the source code in github using Python's BeautifulSoup.

Dataset

This dataset contains:

- 154,902 unique athletes and their biological information i.e. height, weight, date of birth
- All Winter / Summer Olympic games from 1896 to 2022
- 7326 unique results (result for a specific event played at an Olympic game)
- 314,726 rows of athlete to result data which includes both team sports and individual sports
- 235 distinct countries (some existing from the past)



From relations to the graph

ATHLETE_BIO(athlete_id, name, sex, born, height, weight, country, country_noc, description, notes)

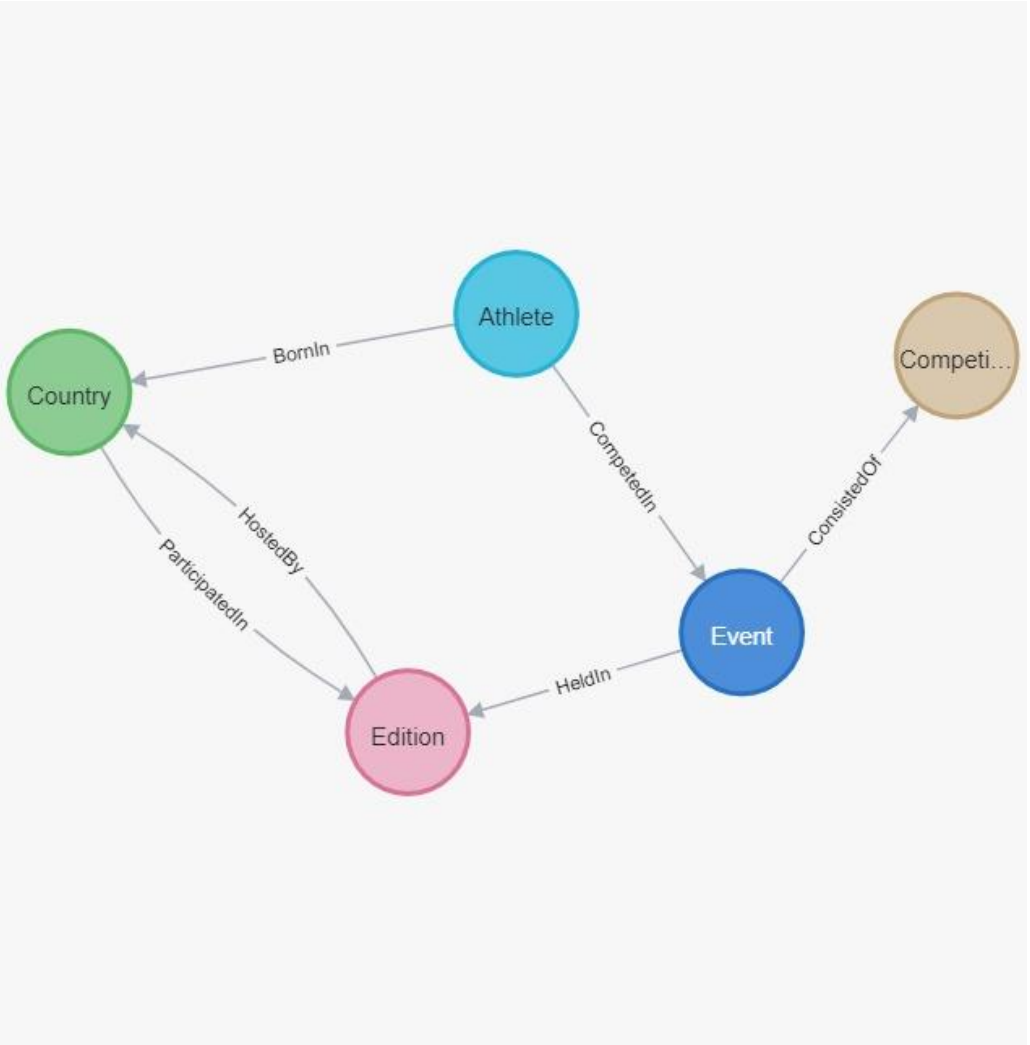
ATHLETE_EVENT_RESULTS(edition, edition_id, country, sport, event, result_id, athlete, athlete_id, pos, medal, isTeamSport)

GAMES_MEDAL_TALLY(edition, edition_id, year, country, country_noc, gold, silver, bronze, total)

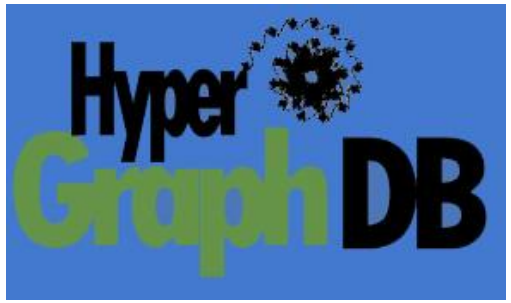
RESULTS(result_id, event, edition, edition_id, sport, sport_url, location, participants, format, detail, description, start_date, end_date, time)

COUNTRY(country_noc, country)

GAMES(edition, edition_id, edition_url, season, year, city, flag_url, country_noc, start_date, end_date, isHeld, competition_start_date, competition_end_date)



Tools



Tools

	Model		Repr.		Data Organization						Data Distribution & Query Execution								Query Language				
	lpg	rdf	al	am	fs	vs	dp	se	sv	lw	ms	rp	sh	ce	pe	tr	oltp	olap	SPARQL	Cypher	SQL	GraphQL	Prog. API
	NATIVE GRAPH DATABASES (LPG model based)																						
Neo4j	■	□	■	□	■	□	■	■	□	□	■	■	□	■	□	■	■	■	□	■	□	■	◧ (Java)
	DOCUMENT STORES																						
OrientDB	■	□	■	□	□	■	■	■	□	■	■	■	■	■	■	■	■	□	■	■	■*	□	■ (Java)
	KEY-VALUE STORES																						
HyperGraphDB	□	□	□	□	□	■	□	■	□	□	■	■	■	■	■	◧	■	■	□	□	□	□	■ (Java)

- lpg: the system supports the Labeled Property Graph model without prior data transformation
- rdf: the system supports the RDF model without prior data transformation
- am: the structure is represented as the adjacency matrix
- al: the structure is represented as the adjacency list
- fs: data records are fixed size
- vs: data records are variable size
- dp: the system can use direct pointers to link records
- se: edges can be stored in a separate edge record
- sv: edges can be stored in a vertex record
- lw: edges can be lightweight
- ms: the system can operate in a multi server (distributed) mode
- rp: the system enables Replication of datasets
- sh: the system enables Sharding of datasets
- ce: the system enables Concurrent Execution of multiple queries
- pe: the system enables Parallel Execution of single queries on multiple nodes/CPUs
- tr: support for ACID Transactions

M. Besta et al. Demystifying Graph Databases. 2019

Neo4j – creating the database

Creating nodes:

```
LOAD CSV WITH HEADERS FROM 'file:///<path>.csv' AS row
MERGE (athlete:Athlete {athleteID: row.athlete_id})
  ON CREATE SET athlete.name = row.name
```

Creating relationships:

```
LOAD CSV WITH HEADERS FROM 'file:///<path>.csv' AS row
MATCH (edition:Edition {editionID: row.edition_id})
MATCH (country:Country {countryNOC: row.country_noc})
MERGE (edition)-[hb:HostedBy]->(country)
  ON CREATE SET hb.city = row.city
```

Creating constraints:

```
CREATE CONSTRAINT athlete_id ON (ath:Athlete) ASSERT ath.athleteID IS UNIQUE
```

Neo4j – querying the graph

Get the schema:

```
CALL db.schema.visualization()
```

Find the countries (and the related city) that have won more than 30 medals in the editions they hosted:

```
MATCH (c:Country)-[pi:ParticipatedIn]->(e:Edition)-[hb:HostedBy]->(c)
WHERE toInteger(pi.total) > 30
RETURN e.name, c.name, hb.city, pi.total
ORDER BY e.name
```

Find the swimmers who have won at least one medal and the competitions in which they competed:

```
MATCH (a:Athlete)-[ci:CompetedIn]->(:Event)-[:ConsistedOf]->(c:Competition)
WHERE ci.medal is not Null and c.sport="Swimming"
RETURN distinct a.name, c.name
ORDER BY a.name
```

Neo4j – querying the graph

Find the italian athletes who won a medal in 2008, with the related competition and sport:

```
MATCH (noc:Country)<-[:BornIn]-(a:Athlete)-[ci:CompetedIn]->(ev:Event)-
[:ConsistedOf]->(c:Competition), (ev)-[:HeldIn]->(ed:Edition)
WHERE ci.medal is not Null and toInteger(ed.year)=2008 and noc.name="Italy"
RETURN distinct a.name, c.sport, c.name, ci.medal
ORDER BY c.sport
```

Find the athletes who competed in the country where they were born:

```
MATCH (c:Country)<--(a:Athlete)-[*2]->(ed:Edition)-->(c)
RETURN distinct ed.name, a.name, c.name
ORDER BY ed.name
```

Find the italian athletes who have competed in an edition hosted by Italy and have won at least one medal:

```
MATCH (c:Country)<--(a:Athlete)-[ci:CompetedIn|HeldIn*2]->(ed:Edition)-->(c)
WHERE ci[0].medal is not Null and c.name="Italy"
RETURN distinct ed.name, a.name, ci[0].medal
ORDER BY ed.name
```

Neo4j – querying the graph

Compute the age of the athletes in the year in which they won a medal, with the related edition:

```
MATCH (a:Athlete)-[ci:CompetedIn|HeldIn*2]->(ed:Edition)
CALL { WITH a
       RETURN
       CASE
           WHEN right(a.born, 4)=~'.*-.*' THEN left(a.born, 4)
           ELSE right(a.born, 4)
       END AS born }
WITH ed, a, ci[0].medal AS medal, toInteger(ed.year) - toInteger(born) AS age
WHERE age IS NOT NULL and medal IS NOT NULL
RETURN distinct ed.name, a.name, age, medal
```

Find the sports included in a single edition of the Olympics:

```
MATCH (ed:Edition)<--()-->(c:Competition)
WITH DISTINCT ed.name AS edition, c.sport AS sport
WITH sport, count(edition) AS editions
WHERE editions = 1
RETURN sport
```

Neo4j – querying the graph

Find the youngest athletes in each edition who have won a gold medal:

```
MATCH (a:Athlete)-[ci:CompetedIn|HeldIn*2]->(ed:Edition)
CALL {
    WITH a
    RETURN
    CASE
        WHEN right(a.born, 4)=~'.*-.*' THEN left(a.born, 4)
        ELSE right(a.born, 4)
    END AS born
}
WITH ed, a, ci[0].medal AS medal, toInteger(ed.year) - toInteger(born) AS age
WHERE age IS NOT NULL and medal="Gold"
WITH ed.name AS edition, collect(distinct [a.name, age]) AS ath_lst,
     min(age) AS min_age
UNWIND ath_lst as athlete
WITH edition, min_age, athlete
WHERE min_age = athlete[1]
RETURN edition, athlete[0] AS athlete, athlete[1] AS age
```

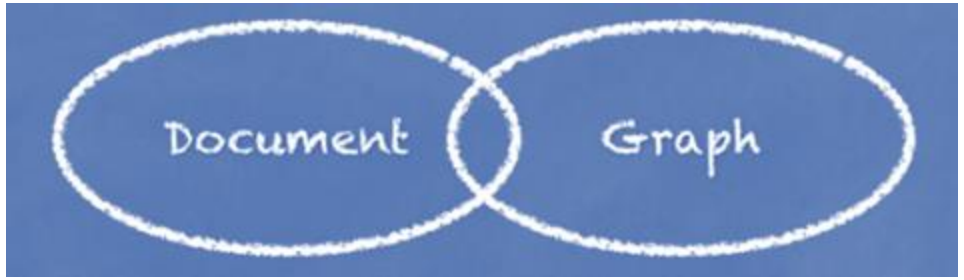
Neo4j – querying the graph

Compute the athletes medal tally:

```
CALL {
    MATCH (a:Athlete)-[:CompetedIn {medal: "Gold"}]->()
    RETURN a.name AS athlete, count(*) AS gold, 0 AS silver, 0 AS bronze
    UNION
    MATCH (a:Athlete)-[:CompetedIn {medal: "Silver"}]->()
    RETURN a.name AS athlete, 0 AS gold, count(*) AS silver, 0 AS bronze
    UNION
    MATCH (a:Athlete)-[:CompetedIn {medal: "Bronze"}]->()
    RETURN a.name AS athlete, 0 AS gold, 0 AS silver, count(*) AS bronze
}
WITH athlete, sum(gold) AS gold, sum(silver) AS silver, sum(bronze) AS bronze
RETURN athlete, gold, silver, bronze, gold+silver+bronze AS total
ORDER BY total DESC
```

OrientDB

OrientDB is a Multi-Model Open Source NoSQL DBMS that combines the power of graphs and the flexibility of documents into one scalable, high-performance operational database.



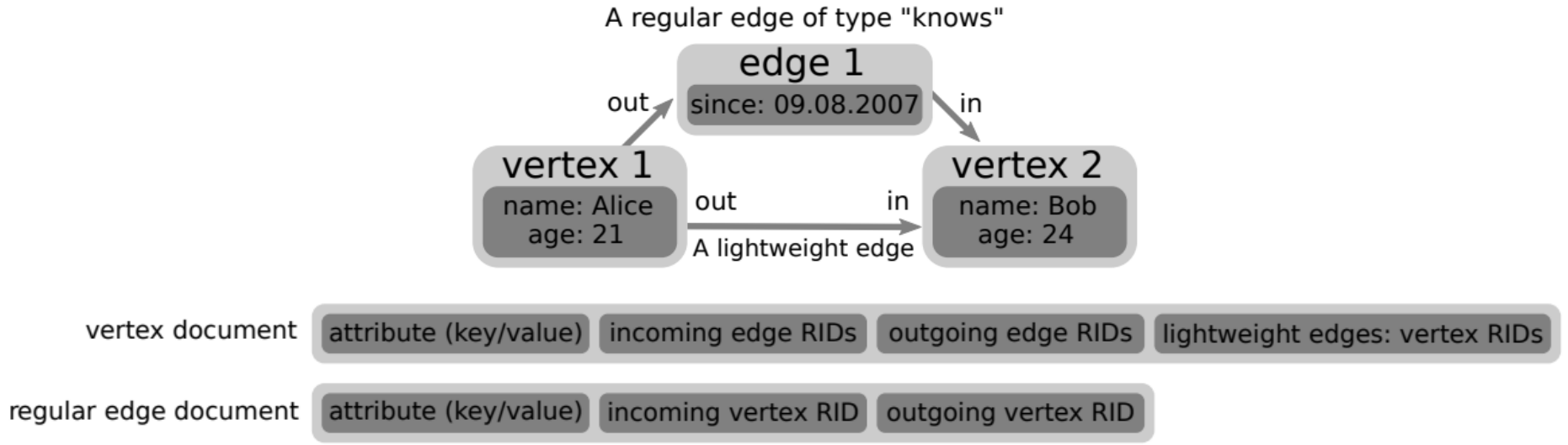
OrientDB features

FEATURES	ORIENTDB	MONGODB	NEO4J	MYSQL (RDBMS)
Operational Database	X	X		X
Graph Database	X		X	
Document Database	X	X		
Object-Oriented Concepts	X			
Schema-full, Schema-less, Schema mix	X			
User and Role & Record Level Security	X			
Record Level Locking	X		X	X
SQL	X			X
ACID Transaction	X		X	X
Relationships (Linked Documents)	X		X	X
Custom Data Types	X	X		X
Embedded Documents	X	X		
Multi-Master Zero Configuration Replication	X			
Sharding	X	X		
Server Side Functions	X	X		X
Native HTTP Rest/ JSON	X	X		
Embeddable with No Restrictions	X			

OrientDB

OrientDB

In OrientDB, every document d has a Record ID (RID), consisting of the ID of the collection of documents where d is stored, and the position (also referred to as the offset) within this collection.
OrientDB introduces regular edges and lightweight edges. Regular edges are stored in an edge document and can have their own associated key/value pairs (e.g., to encode edge properties or labels). Lightweight edges, on the other hand, are stored directly in the document of the adjacent source vertex. Such edges do not have any associated key/value pairs. Thus, a vertex document not only stores the labels and properties of the vertex, but also a list of lightweight edges and a list of pointers to the adjacent regular edges.



M. Besta et al. Demystifying Graph Databases. 2019

OrientDB – creating the database

Creating nodes:

```
CREATE CLASS Athlete EXTENDS V
```

Creating edges:

```
CREATE CLASS BornIn EXTENDS E
```

Creating properties:

```
CREATE PROPERTY Athlete.name STRING
```

Populate DB:

```
INSERT INTO Athlete (identity, athleteID, name, sex, born, weight, height)
VALUES ("4", "58758", "Hossein Mollaghasemi", "M", "15/03/1933", "63", "173.0")

CREATE EDGE BornIn FROM (SELECT FROM Athlete WHERE identity=4)
TO (SELECT FROM Country WHERE identity=78443) SET identity=1842
```

OrientDB – querying the graph

Find the countries (and the related city) that have won more than 30 medals in the editions they hosted:

```
MATCH {class: Country, as: c}
      .outE('ParticipatedIn'){as: pi, where: (total > 30)}
      .inV('ParticipatedIn'){class: Edition, as: ed}
      .outE('HostedBy'){as: hb}
      .inV('HostedBy'){class: Country, where: ($matched.c = $currentMatch)}
RETURN ed.name as edition, c.name as country, hb.city as city, pi.total as total
```

Find the swimmers who have won at least one medal and the competitions in which they competed:

```
MATCH {class: Athlete, as: a}
      .outE('CompetedIn'){as: ci, where: (medal is not Null)}
      .inV('CompetedIn'){class: Event}
      .out('ConsistedOf'){as: c, where: (sport = "Swimming")}
RETURN DISTINCT a.name as athlete, c.name as competition
```

OrientDB – querying the graph

Find the italian athletes who won a medal in 2008, with the related competition and sport:

```
MATCH {class: Country, as: noc, where: (name = "Italy")} <--
      {class: Athlete, as: a}
      .outE('CompetedIn'){as: ci, where: (medal is not NULL)}
      .inV('CompetedIn'){class: Event, as: ev} -->
      {class: Competition, as: c},
      {as: ev} -->
      {class: Edition, as: e, where: (year = 2008)}
RETURN a.name as athlete, c.sport as sport, c.name as competition,
       ci.medal as medal, noc.name as country
```

Find the athletes who competed in the country where they were born:

```
MATCH {class: Country, as: c} <--
      {class: Athlete, as: a} -->
      {} -->
      {class: Edition, as: e} -->
      {class: Country, where: ($matched.c = $currentMatch)}
RETURN e.name as edition, a.name as athlete, c.name as country
```

OrientDB – querying the graph

Compute the age of the athletes in the year in which they won a medal, with the related edition:

```
SELECT ath, ed, medal, year -
if(eval('born.right(4).indexOf("-") > -1'), born.left(4), born.right(4)) as age
FROM (
  MATCH {class: Athlete, as: a, where: (born is not NULL)}
        .outE('CompetedIn'){as: ci, where: (medal is not NULL)}
        .inV('CompetedIn'){}.out('HeldIn'){class: Edition, as: e}
  RETURN a.name as ath, e.name as ed, a.born as born,
         e.year as year, ci.medal as medal )
```

Get the athletes medal tally:

```
SELECT ath, sum(g) as gold, sum(s) as silver,
       sum(b) as bronze, sum(g)+sum(s)+sum(b) as total
FROM (
  SELECT ath, if(eval("medal='Gold'"), 1, 0) as g,
  if(eval("medal='Silver'"), 1, 0) as s, if(eval("medal='Bronze'"), 1, 0) as b
  FROM ( … ))
GROUP BY ath
```

OrientDB – querying the graph

Find the youngest athletes in each edition who have won a gold medal:

```
SELECT ed as edition, ath as athlete, age
FROM (
  SELECT *, $temp as temp
  FROM (
    SELECT ed, ath, … as age
    FROM (
      MATCH {class: Athlete, as: a, where: (born is not NULL)}
            .outE('CompetedIn'){as: ci, where: (medal="Gold")}
            .inV('CompetedIn'){}.out('HeldIn'){class: Edition, as: e}
      RETURN e.name as ed, a.name as ath, a.born as born, e.year as year )
  )
  LET $temp = ( SELECT ed, min(age) as min_age
                FROM ( … )
                GROUP BY ed )
  UNWIND temp
)
WHERE ed = temp.ed and age = temp.min_age
```

OrientDB – querying the graph

Get the athletes medal tally: (alternative version)

```
SELECT athlete, sum(gold) as gold, sum(silver) as silver,
       sum(bronze) as bronze, sum(gold) + sum(silver) + sum(bronze) as total
FROM (
  SELECT expand($union)
  LET $gold = (SELECT athlete, count(*) as gold, 0 as silver, 0 as bronze
               FROM ( … )
               GROUP BY athlete),
      $silver = (SELECT athlete, 0 as gold, count(*) as silver, 0 as bronze
                 FROM ( … )
                 GROUP BY athlete),
      $bronze = (SELECT athlete, 0 as gold, 0 as silver, count(*) as bronze
                 FROM ( … )
                 GROUP BY athlete),
      $union = unionAll($gold, $silver, $bronze)
)
GROUP BY athlete
```

HyperGraphDB

HyperGraphDB is a general purpose, extensible, portable, distributed, embeddable, open-source data storage mechanism. It is a graph database designed specifically for artificial intelligence and semantic web projects; it can also be used as an embedded object-oriented database for projects of all sizes.

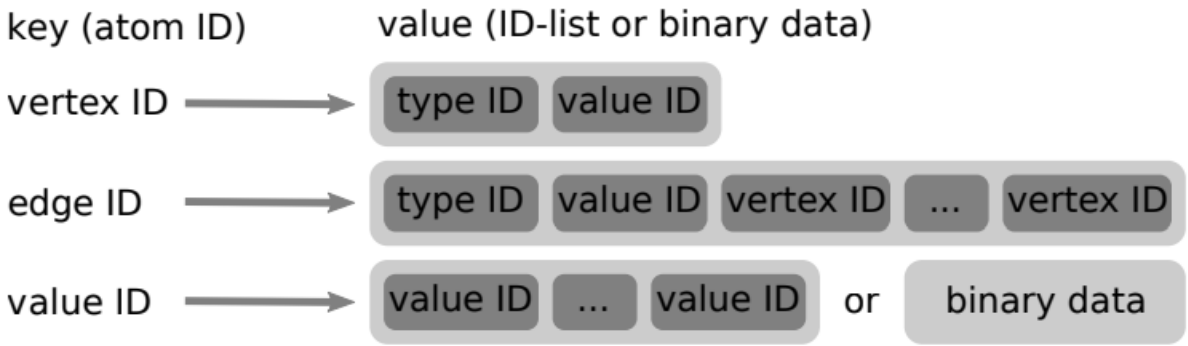
HyperGraphDB is primarily what its carefully chosen name implies: a database for storing hypergraphs. A hypergraph is an extension to the standard graph concept that allows an edge to point to more than two nodes. HyperGraphDB extends this even further by allowing edges to point to other edges as well and making every node or edge carry an arbitrary value as payload. The basic unit of storage in HyperGraphDB is called an atom. Each atom is typed, has an arbitrary value and can point to zero or more other atoms.

The current implementation is solely Java based and it offers an automatic mapping of idiomatic Java types to a HyperGraphDB data schema. While it falls into the general family of graph databases, it is hard to categorize HyperGraphDB as another database because much of its design evolves around providing the means to manage structure-rich information with arbitrary layers of complexity. The design is minimalistic at its core and the end-goal is to evolve a set of concepts and practices, combining structure and interpretation in such a way as to allow future software to meet the complexities of the real-world better that now.

from HyperGraphDB website

HyperGraphDB

The basic building blocks of HyperGraphDB are atoms, the values of the KV store. Every atom has a cryptographically strong ID. Both hypergraph vertices and hyperedges are atoms. Thus, they have their own unique IDs.
An atom of a hyperedge stores a list of IDs corresponding to the vertices connected by this hyperedge. Vertices and hyperedges also have a type ID (i.e., a label ID) and they can store additional data (such as properties) in a recursive structure (referenced by a value ID). This recursive structure contains value IDs identifying other atoms (with other recursive structures) or binary data.



M. Besta et al. Demystifying Graph Databases. 2019

HyperGraphDB – creating the database

Creating nodes:

```
HyperGraph graph = HGEnvironment.get(db_path);
JSONArray ja = (JSONArray) new JSONParser().parse(new FileReader(athletes_json_file));
Iterator it = ja.iterator();

while(it.hasNext()) {
    JSONObject jo = (JSONObject) it.next();

    Long identity = (Long) jo.get("identity");
    Long athleteID = (Long) jo.get("athleteID");
    String name = (String) jo.get("name");
    String sex = (String) jo.get("sex");
    …

    Athlete athlete = new Athlete(identity, athleteID, name, sex, born, weight, height);
    graph.add(athlete);
}
```

HyperGraphDB – creating the database

Defining edges:

```
public class BornIn extends HGPlainLink {
   public BornIn() { super(); }

   public BornIn(HGHandle nameHandle, HGHandle idHandle,
                 HGHandle athleteHandle, HGHandle countryHandle) {
      super(nameHandle, idHandle, athleteHandle, countryHandle);
   }

   public HGHandle getName() { return this.getTargetAt(0); }

   public HGHandle getIdentity() { return this.getTargetAt(1); }

   public HGHandle getHead() { return this.getTargetAt(2); }

   public HGHandle getTail() { return this.getTargetAt(3); }
}
```

HyperGraphDB – querying the graph

Find the athletes who won a medal in 2008, with the related competition and sport:

```
HGQueryCondition ci_condition = hg.type(CompetedIn.class);
HGSearchResult<HGHandle> ci_rs = graph.find(ci_condition);

while (ci_rs.hasNext()) {
   HGHandle ci_handle = ci_rs.next();
   CompetedIn ci = graph.get(ci_handle);

   Athlete athlete = graph.get(ci.getHead());
   List<HGHandle> tail = ci.getTail();
   Competition competition = graph.get(tail.get(0));
   Edition edition = graph.get(tail.get(1));
   String medal = graph.get(ci.getMedal());

   if (!medal.equals("None") && edition.getYear() == 2008) {
      System.out.println("Edition:" + edition.getName()); … }
}
```

HyperGraphDB – querying the graph

Find the athletes who competed in the country where they were born:

```
HGQueryCondition bi_condition = hg.type(BornIn.class);
HGSearchResult<HGHandle> bi_rs = graph.find(bi_condition);

while (bi_rs.hasNext()) {
    HGHandle bi_handle = bi_rs.next();
    BornIn bi = graph.get(bi_handle);
    HGHandle a_handle = bi.getHead();
    HGHandle c_handle = bi.getTail();

    HGQueryCondition hb_condition = hg.type(HostedBy.class);
    HGSearchResult<HGHandle> hb_rs = graph.find(hb_condition);

    while (hb_rs.hasNext()) {
        HGHandle hb_handle = hb_rs.next();
        HostedBy hb = graph.get(hb_handle);
        HGHandle e_handle = hb.getHead();
```

HyperGraphDB – querying the graph

```
if (c_handle.equals(hb.getTail())) {
    HGQueryCondition ci_condition = hg.link(a_handle, e_handle);
    HGSearchResult<HGHandle> ci_rs = graph.find(ci_condition);

    while (ci_rs.hasNext()) {
        HGHandle ci_handle = ci_rs.next();
        CompetedIn ci = graph.get(ci_handle);
        Athlete athlete = graph.get(a_handle);
        List<HGHandle> tail = ci.getTail();
        Competition competition = graph.get(tail.get(0));
        Edition edition = graph.get(e_handle);
        Country country = graph.get(c_handle);
        String medal = graph.get(ci.getMedal());

        if (!medal.equals("None") && country.getCountryNOC().equals("ITA")) {
            System.out.println("Edition: " + edition.getName());
            …
} … }
```

HyperGraphDB – querying the graph

Find the youngest athletes in each edition who have won at least one medal:

```
Map<Edition, Long> age_map = new HashMap<>();
Map<Edition, List<Athlete>> ath_map = new HashMap<>();

HGQueryCondition ci_condition = hg.type(CompetedIn.class);
HGSearchResult<HGHandle> ci_rs = graph.find(ci_condition);

while (ci_rs.hasNext()) {
    HGHandle ci_handle = ci_rs.next();
    CompetedIn ci = graph.get(ci_handle);
    Athlete athlete = graph.get(ci.getHead());
    List<HGHandle> tail = ci.getTail();
    Edition edition = graph.get(tail.get(1));
    String medal = graph.get(ci.getMedal());
    String born = athlete.getBorn();
    Long year = edition.getYear();
    Long age;
```

HyperGraphDB – querying the graph

```
if (born.substring(born.length()-4).indexOf('-') > -1) { age = year - Long.parseLong(born.substring(0, 4)); }
else { age = year - Long.parseLong(born.substring(born.length()-4)); }
if (age_map.get(edition) == null) { … }
if (age < age_map.get(edition)) {
    List<Athlete> ath_lst = new ArrayList<>();
    ath_lst.add(athlete);
    ath_map.put(edition, ath_lst);
    age_map.put(edition, age); }
if (age == age_map.get(edition) && !ath_map.get(edition).contains(athlete)) {
    List<Athlete> ath_lst = ath_map.get(edition);
    ath_lst.add(athlete);
    ath_map.put(edition, ath_lst); }
 } ... }

for (Edition ed : age_map.keySet()) {
  System.out.println("Edition: " + ed.getName() + " - Min age: " + age_map.get(ed));
  for (Athlete ath : ath_map.get(ed)) { System.out.println("\tAthlete: " + ath.getName()); }
}
```

HyperGraphDB – querying the graph

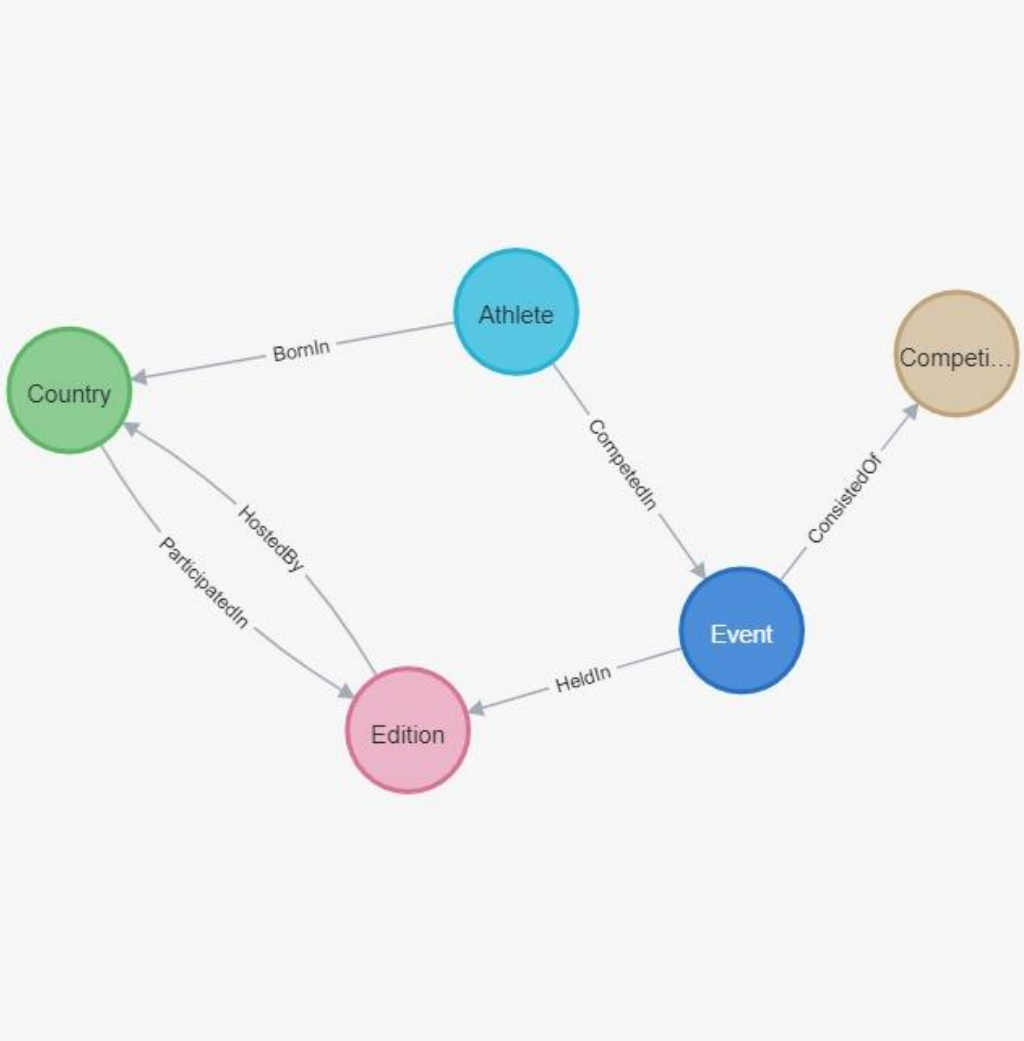
Get the athletes medal tally:

```
Map<Athlete, List<Integer>> map = new HashMap<>();

HGQueryCondition ci_condition = hg.type(CompetedIn.class);
HGSearchResult<HGHandle> ci_rs = graph.find(ci_condition);

while (ci_rs.hasNext()) {
    HGHandle ci_handle = ci_rs.next();
    CompetedIn ci = graph.get(ci_handle);

    Athlete athlete = graph.get(ci.getHead());
    String medal = graph.get(ci.getMedal());

    if (map.get(athlete) == null) {
        List<Integer> medals = new ArrayList<>();
        medals.add(0); ...
        map.put(athlete, medals);
    }
```

HyperGraphDB – querying the graph

```
    if (medal.equals("Gold")) {
        List<Integer> medals = map.get(athlete);
        medals.set(0, medals.get(0) + 1);
        medals.set(3, medals.get(3) + 1);
        map.put(athlete, medals);
    }
    if (medal.equals("Silver")) { … }
    if (medal.equals("Bronze")) { … }
}

for (Athlete ath : map.keySet()) {
    System.out.println("Athlete: " + ath.getName());
    System.out.println("\tGold: " + map.get(ath).get(0));
    System.out.println("\tSilver: " + map.get(ath).get(1));
    System.out.println("\tBronze: " + map.get(ath).get(2));
    System.out.println("\tTotal: " + map.get(ath).get(3));
    System.out.println("---------------");
}
```

MongoDB – from the graph to the documents



DATABASES: 1 COLLECTIONS: 11

+ Create Database

Search Namespaces

OlympicsDB

- Athletes
- BornIn
- CompetedIn
- Competitions
- ConsistedOf
- Countries
- Editions
- Events
- HeldIn
- HostedBy
- ParticipatedIn

OlympicsDB.BornIn

STORAGE SIZE: 4.52MB LOGICAL DATA SIZE: 8.22MB TOTAL DC

Find Indexes Schema Anti-Patterns 0

FILTER { field: 'value' }

QUERY RESULTS: 1-20 OF MANY

```
_id: ObjectId('63176d146f642da8d7d5e8f5')
identity: 1845
start: 7
end: 78422
```

```
_id: ObjectId('63176d146f642da8d7d5e8f6')
identity: 1846
```

MongoDB – creating the database

Defining the documents (nodes):


```
[{
  "identity": 4,
  "athleteID": 58758,
  "sex": "Male",
  "born": "15/03/1933",
  "name": "Hossein Mollaghasemi",
  "weight": "63",
  "height": 173.0
}, {
  "identity": 20,
  "athleteID": 28096,
  "born": "19/01/1929",
  "sex": "Female",
  "name": "Gertrude Winnige Barosch"
}, ...]
```


Defining the documents (edges):


```
[{
  "identity": 0,
  "start": 102074,
  "end": 78426,
  "city": "Athina"
}, {
  "identity": 1,
  "start": 102075,
  "end": 78420,
  "city": "Paris"
}, {
  "identity": 2,
  "start": 102076,
  "end": 78568,
  "city": "St. Louis"
}, ... ]
```

MongoDB – querying the database

Get the medals won by countries in the editions they hosted, and the related city:

```
db.Editions.aggregate([{
  $lookup: {
    from: "HostedBy",
    localField: "identity",
    foreignField: "start",
    as: "hb"
  }
}, { $unwind: "$hb" }, {
  $lookup: {
    from: "Countries",
    localField: "hb.end",
    foreignField: "identity",
    as: "countries"
  }
}, { $unwind: "$countries" }, {
  $sort: { name: 1 }
}, {
  $project: {
    _id: 0,
    ed_id: "$identity",
    edition: "$name",
    c_id: "$countries.identity",
    country: "$countries.name",
    city: "$hb.city"
  }
}, {
  $lookup: {
    from: "ParticipatedIn",
    let: { ed: "$ed_id", c: "$c_id" },
    pipeline: [{
      $match: {
        $expr: {
          $and: [
            {$eq: ["$start", "$$c"]},
            {$eq: ["$end", "$$ed"]}
          ]
        }
      }
    }],
    as: "pi"
  }
}, { $unwind: "$pi" }, {
  $project: {
    edition: 1,
    country: 1,
    city: 1,
    gold: "$pi.gold",
    silver: "$pi.silver",
    bronze: "$pi.bronze",
    total: "$pi.total"
}}])
```