DB Management Systems
Graph: Arango

Joel Klein – jdk514@gwmail.gwu.edu

Introduction to Arango

Arango

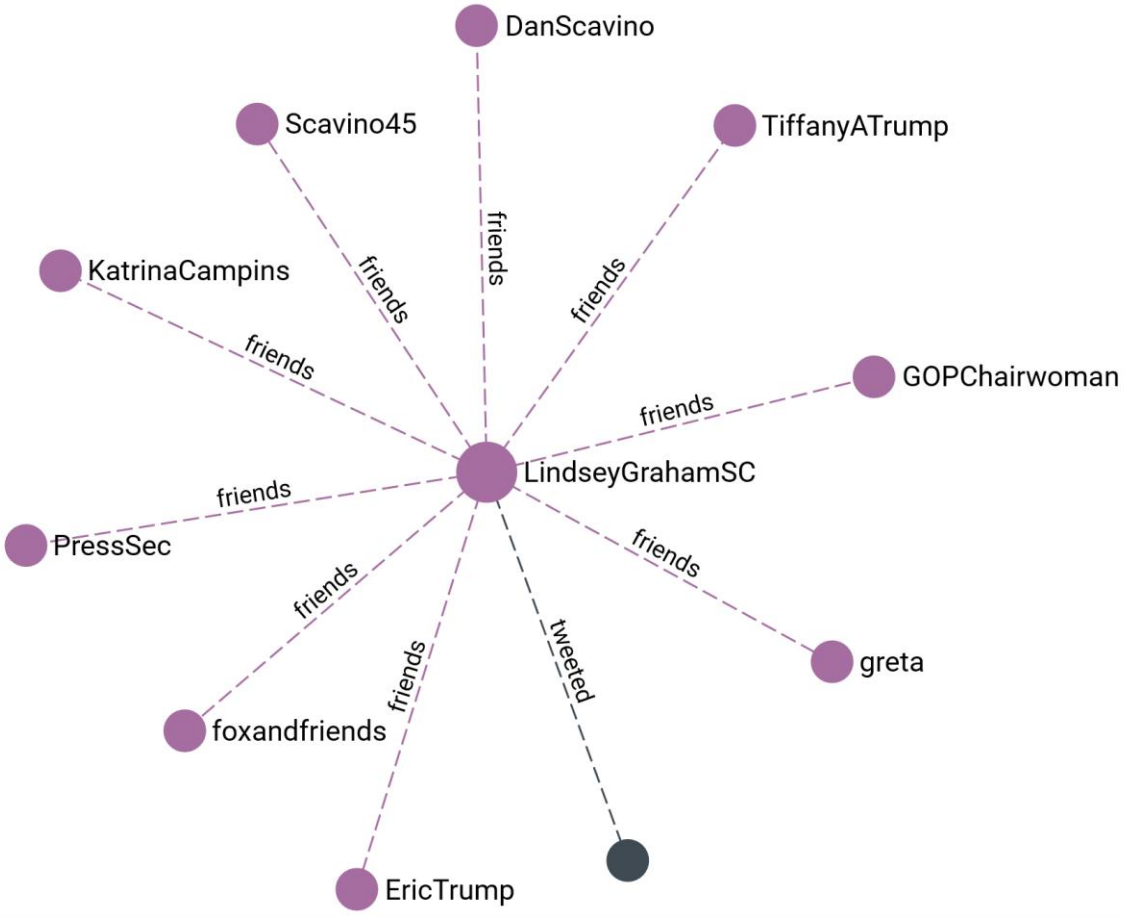
- Arango is a Graph based Database
  - Graph Databases primarily store information about relationships between nodes/entities
    - These nodes/entities may be represented as flat information or a dictionary of information
- The key design behind Arango (Graph Databases) is to enable people to query data based on its relationships, rather than its underlying information

Terminology

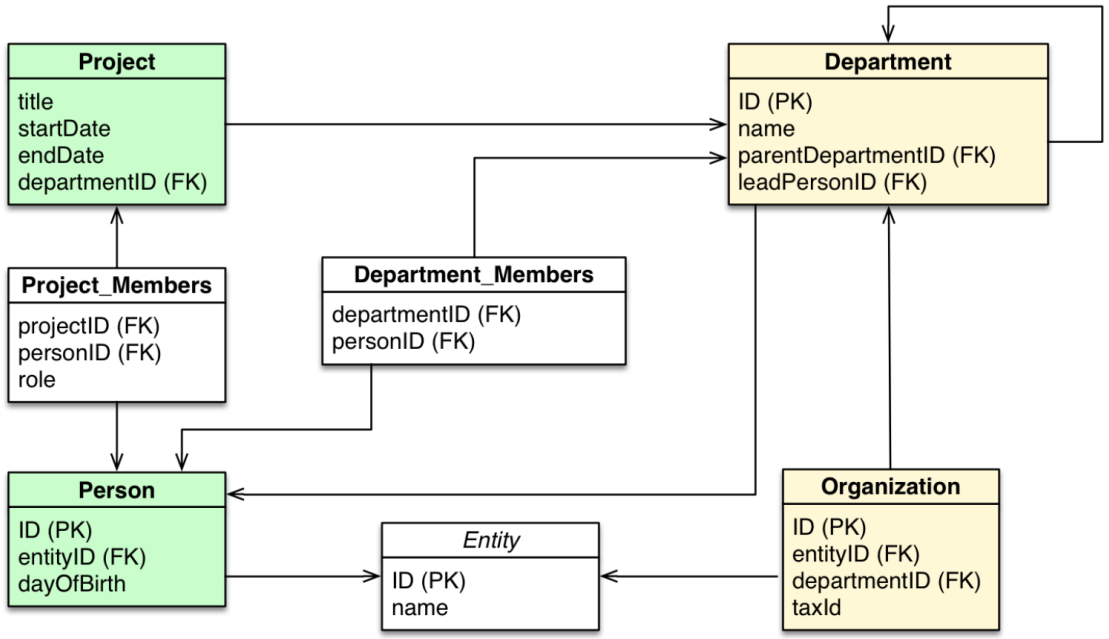
- Database: A group of collections
- Vertex Collection: Collection of documents defining graph nodes
  - Think mongo documents
- Edge Collection: Collection of node pairings
  - Similar to Join logic used in SQL
- Graph: Joint representation of vertex and edge collections
- Node (think document): Data element, containing a number of properties and labels
- Primary Key: Document attribute that identifies the node
- AQL: Query language for Arango

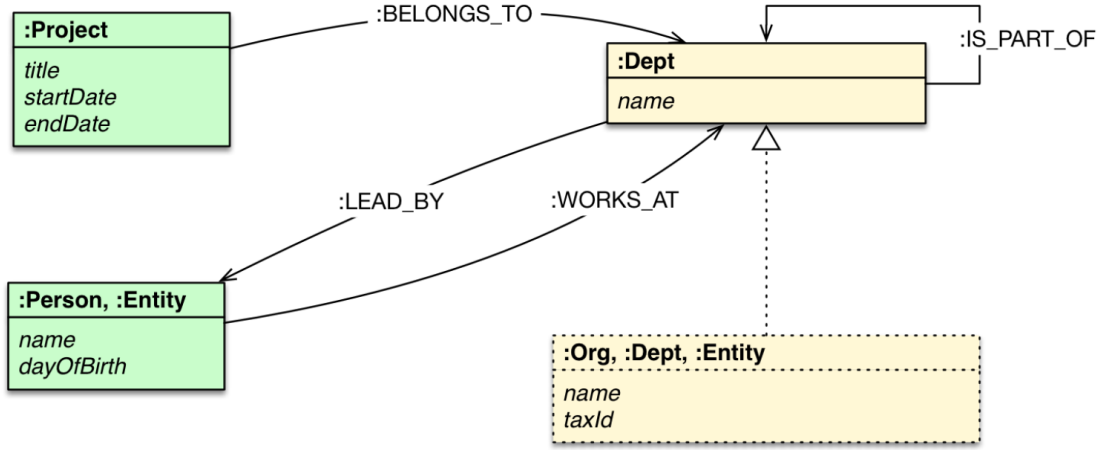
Arango Graph Layout

- When we visualize a graph, we see how different vertices (nodes) are connected via their edges (relationships)
- As mentioned earlier, each vertex in Arango is comprised of labels and properties
- In addition to vertices, Arango also has edges. These are on a similar level as vertices, as they can house data too.
  - Both vertices and edges store data in JSON-like documents



Relational vs Graph





Acceptable Arango Data Types

- Basic Types:
  - Boolean
  - Numerical
  - String
  - Null
- Compound Types:
  - Array
  - Object/Document
- Graph Structure Types
  - Node
  - Relationship
  - Path

Querying Arango - Browser

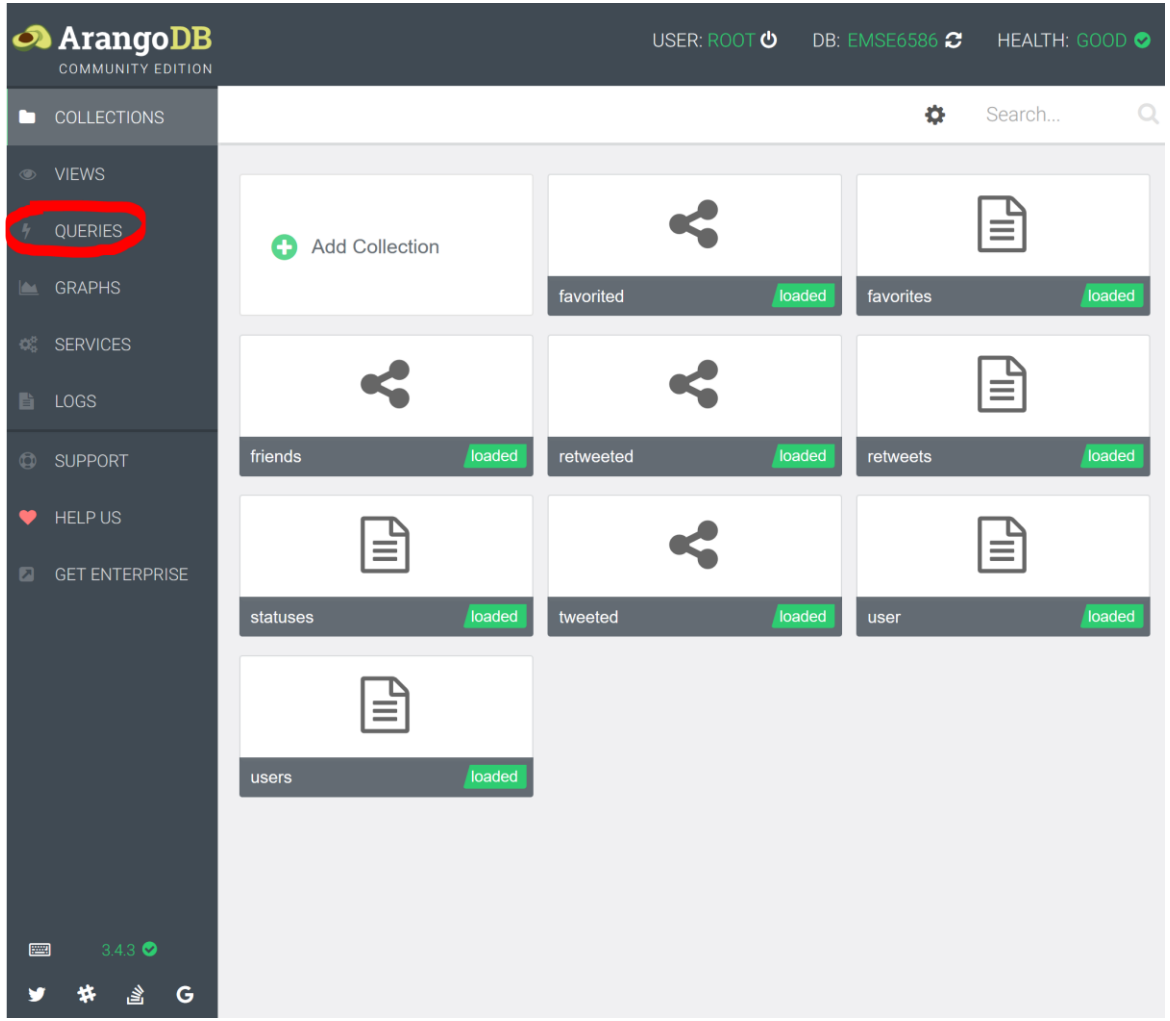
Arango Browser

- We can query information from Arango directly from a browser
  - This allows us to both query and visualize information

- You simply need to navigate to http://18.219.151.47:8529 to access the browser based GUI
  - I'll provide the username and password in class

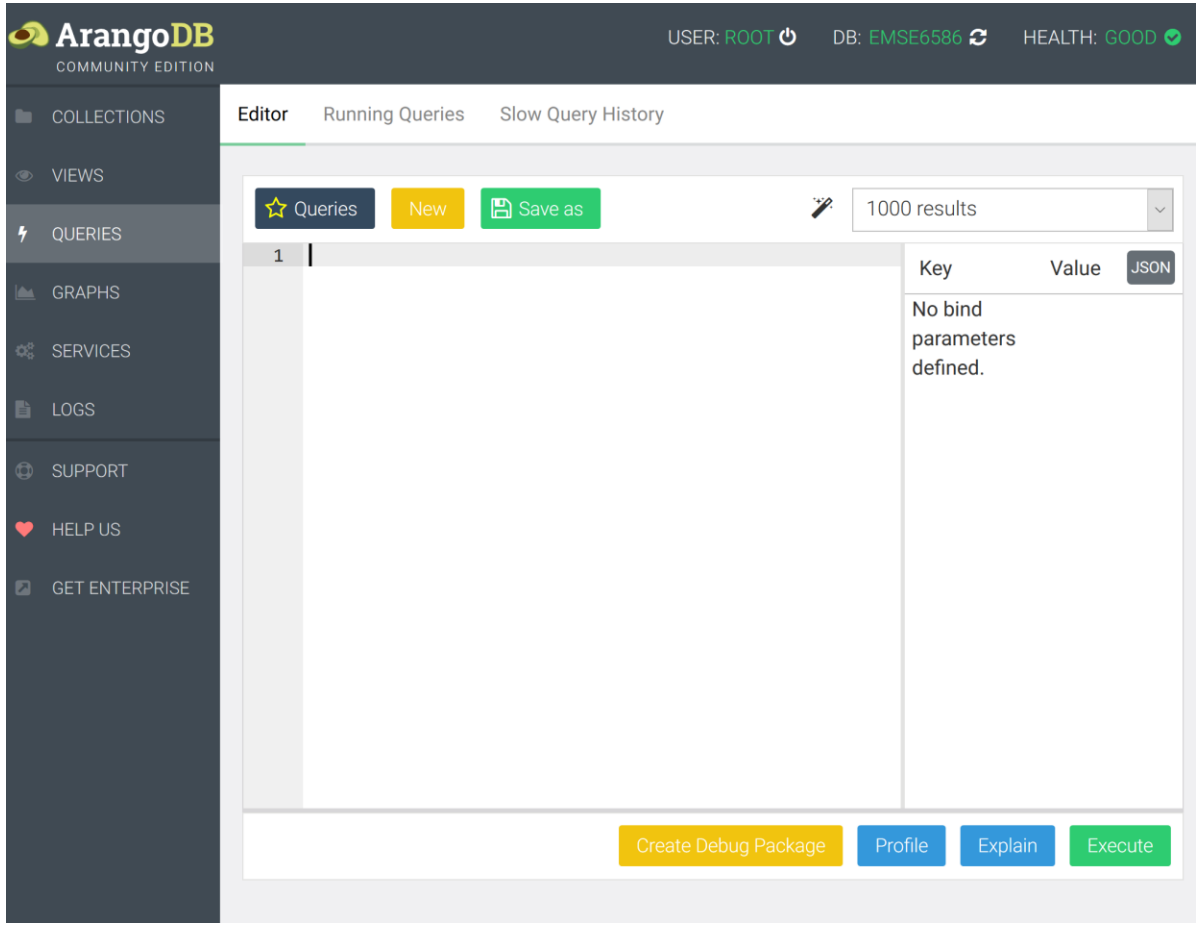
Arango Browser Cont. 1

- Once logged in, the page shown on the right should appear
- From here you can view the collections of data, but more importantly we can run queries from the **QUERIES** tab



Arango Browser Cont. 2

- From here we have an editor to write and view our queries
- Certain queries will even return a graph-based visualization along with the data



ArangoDB Query Language

AQL

- AQL is very different than any of the previous languages we have worked with
  - It is focused on telling Arango what to return, not necessarily how.

- It also is unique in that it works by defining the filter/relationships, and then defining what should be returned from the resulting set
  - It is also unique in the fact that it currently there is not a default or dominant graph query language (Gremlin API may be a front runner)

Before Querying

- Before writing anything in AQL, it is good to understand the structure of the nodes and relationships in the graph. This requires understanding the vertex and edge collections

Vertex Collections

- Users – information about users in our data
  - screen_name, location, etc.
- Statuses – information about the tweets made by our users
  - created_date, user_id, text, hashtags, etc.
- Retweets – information about what status a user retweeted
  - created_date, retweeted_status, text, etc.

Edge Collections

- Friends – details whether two users are friends
- Tweeted – what statuses where tweeted by what users
- Favorited – what statuses where favorited by what users
- Retweeted – what statuses are retweets of what statuses

Reading Documents

- Reading documents from a collection is similar to working with an iterable in python
  - **FOR variableName IN collectionName RETURN variableName**

- Example:
  - FOR user IN users RETURN user

Formatting the Return

- The **Return** statement in an AQL query can be formatted in several different manners. For **FOR user IN users RETURN** we can return the following:
  - **user** – this returns each document that exists in the **users** table
  - **user.{attr}** – This returns the attribute for each user in the users table
  - **{attr_name_1: user.attr_1, attr_name_2: user.attr_2, …}** – This returns a new document based on the provided attributes and names

- **In Class:**
  - Write a query to get all the screen_name's and their created_date in the **users** collection

Filtering

- When asking for documents (or other values) to be **RETURNED**, we can filter the results using the **FILTER** keyword.
  - **FOR {var} IN {collection} FILTER {var}.{attr_name} == {cond} RETURN user**

- Example:
  - **FOR user IN users FILTER user.statuses_count > 50000 RETURN user**

Filtering Cont.

- We can **Filter** on a number of different aspects/attr:
  - Equality: ==
  - Range: >, <, >=, <=
  - Logical Operators: AND, OR, &&, ||
  - Other: LIKE, IN, NOT IN, REGEX

- Note: If logical operators aren't used to join **Filters**, then they follow a linear order of operation
  - Example in notes

- In Class:
  - Write a query that returns the status_id, favorites_count, and retweet_count for statuses that contain the word "AI"

Limit

- Like Mongo and SQL, we can **LIMIT** the number of results returned
  - **FOR {var} IN {coll} LIMIT {num} RETURN {var}**

- Example:
  - **FOR user IN users LIMIT 5 RETURN user**

- The location of the **LIMIT** statement can drastically affect the query, as it will limit the current resultset wherever it is placed

Sort

- Just like **LIMIT**, AQL also provides a means to **SORT** resultsets based on certain parameters
  - **FOR {var} IN {coll} SORT {var}.{attr_name} [ASC|DESC] RETURN {var}**

- Example:
  - **FOR stat IN statuses SORT stat.favorites_count DESC RETURN stat**
  - Note: **SORT** defaults to ASC ordering

Sort Cont.

- Similar to SQL, we can **SORT** on multiple attributes. This will sort on the first attribute, and resolve any equivalencies using the second attribute

  - **FOR {var} IN {coll} SORT {var}.{attr_name_1}, {var}.{attr_name_2} [ASC|DESC] RETURN {var}**

Graph Traversals

Traversals

- When "merging" data between collections in arango, we don't join or concat information, rather we traverse our graph.

- With a traversal we define an origin point (document) and the nature/depth of the edges we wish to traverse
  - **FOR v, e, p [min_depth]..[max_depth] IN [ANY|OUTBOUND|INBOUND] {start_node} {edge_coll} RETURN {var}**

- Example:
  - **FOR v, e, p IN 1..1 OUTBOUND "users/44196397" tweeted LIMIT 25 RETURN p**

Traversals Edge Relations

- The edge relationship **[ANY|OUTBOUND|INBOUND]** dictates how we traverse relationships
  - **ANY** – Follow any edge in the graph
  - **OUTBOUND** – Only follow edges where the starting node is the **FROM** node
  - **INBOUND** – The inverse of **OUTBOUND**

Traversals Depth

- The numeric values in our query dictate the depth of the search that we perform.
  - Example – **1..1**

- In this situation the first number is dictating the starting depth, while the second number dictates how deep we are willing to traverse
  - 1..1 -> only go one level deep
  - 2..3 -> go two-three levels deep into the graph

Traversals Options

- When running traversal based queries, we can define the options for traversing the relationships

- Options:
  - uniqueVertices: [path|global|none]
    - Should we terminate at a previously seen vertex
  - uniqueEdges: [path|none]
    - Should we terminate at a previously seen edge
  - bfs: [true|false]
    - This dictates if we want to use bfs or dfs

Filtering Traversals

- We can filter our traversals based on vertex, edge, or path values.
  - FOR v, e, p IN 1..1 ANY {start_node} GRAPH {graph_name}
    FILTER v.nodes[{index}].{attribute} == {value}
    RETURN v

- This essentially enables us to focus our traversal to nodes/edges/paths of interest

6 Degrees of Kevin Bacon

- 6 Degrees of Kevin Bacon is a game trying to link Kevin Bacon to another actor using a maximum of 6 co-stars

- We can implement this same concept in Arango with the following:

```
FOR v, e, p IN 1..6 OUTBOUND "users/44196397" friends
OPTIONS {bfs: True, uniqueEdges : 'path', uniqueVertices : 'path'}
RETURN DISTINCT v.user_id
```

- In Class:
  - Implement the same logic for either Mongo or MySQL

Tracking Retweet Groups

- Graph traversals like this also enable us to explore how subsets of our data aggregate together.

- Find all groups of retweet users:

```
For status IN statuses
  FILTER status.user_id == "44196397"
        FOR v, e, p IN 1..2 ANY status retweeted, tweeted
        FILTER p.vertices[1].user_id != "44196397"
  Return p
```

Advanced Features

Grouping Results

- Given that we can RETURN any formatted JSON, Arango provides a COLLECT keyword to define lists of returned elements
- FOR var1 IN coll COLLECT x = var1.attr INTO var2 RETURN var2
- Example:
  - FOR tweet IN statuses
    - LIMIT 100
    - COLLECT favs = tweet.favorites_count INTO sid = tweet.status_id
    - RETURN {fav_count: favs, text: sid}

Counting Records

- Counting documents in Arango actually requires the use of a number of different special functions
  - FOR var IN coll
    - COLLECT WITH COUNT INTO len
    - RETURN len

- Note: Recall that all functions occur sequentially, so filtering the results prior to the COLLECT enables us to count filtered subsets

End Slide

EMSE 6992 – DBMS for Data Analytics