

NoSQL III: base de datos en grafos y AQL

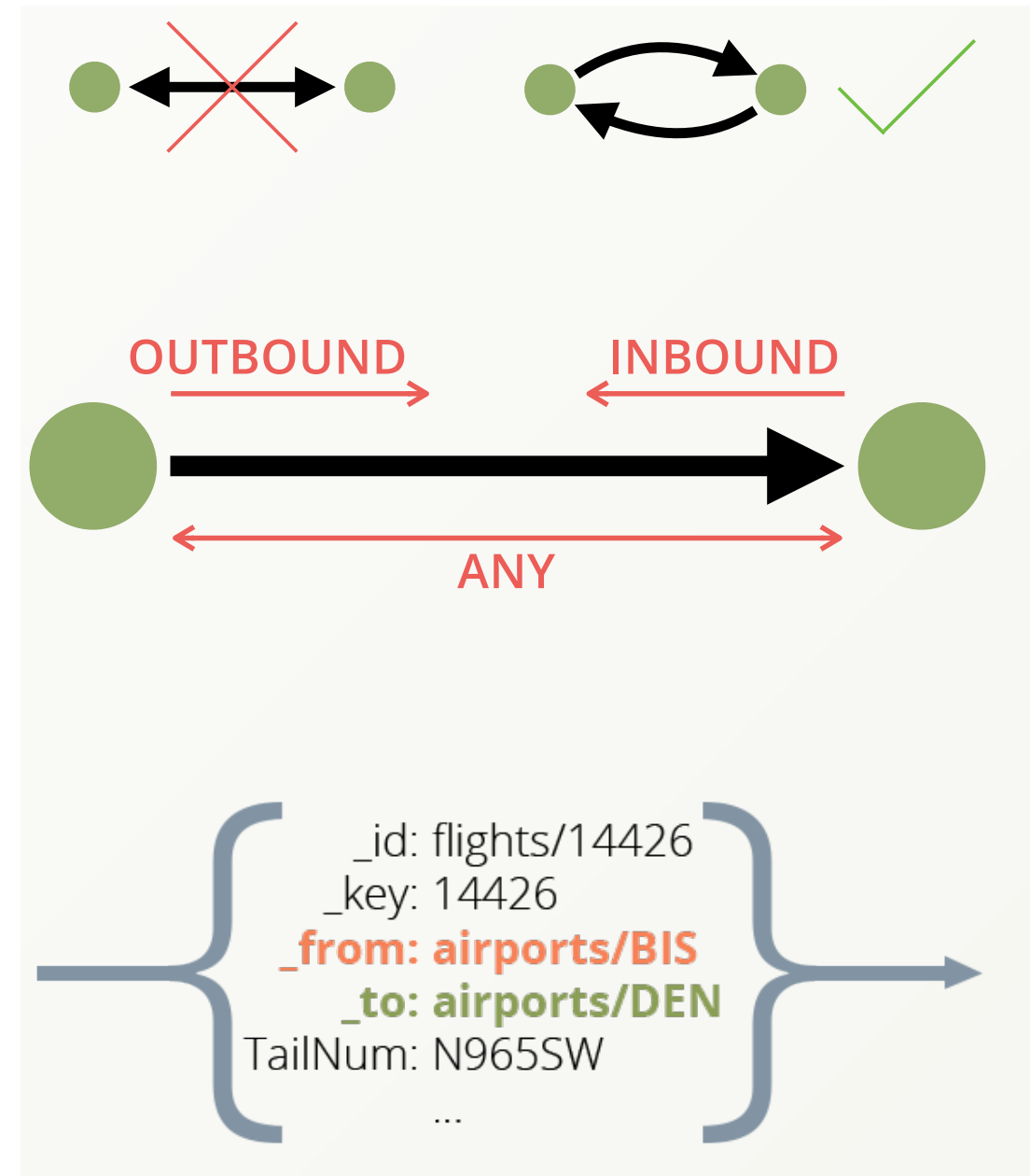
Alex Di Genova

10/11/2022

DB en Grafos

ArangoDB

- En ArangoDB, cada arista tiene una sola dirección, no puede apuntar en ambos sentidos a la vez. Este modelo también se conoce como **grafo dirigido**.
- Pero la dirección se puede ignorar (seguir en CUALQUIER dirección) cuando movemos en el grafo, o seguir las aristas en dirección inversa (INBOUND) en lugar de ir en la dirección a la que realmente apuntan (OUTBOUND). Moverse en un grafo se llama recorrido.
- ArangoDB permite almacenar todo tipo de grafos en diferentes formas y tamaños, con y sin ciclos. Podemos guardar uno o más aristas entre dos vértices o incluso con el mismo vértice.
- Las aristas son documentos JSON completos, por lo tanto podemos almacenar tanta información como deseemos/necesitemos.

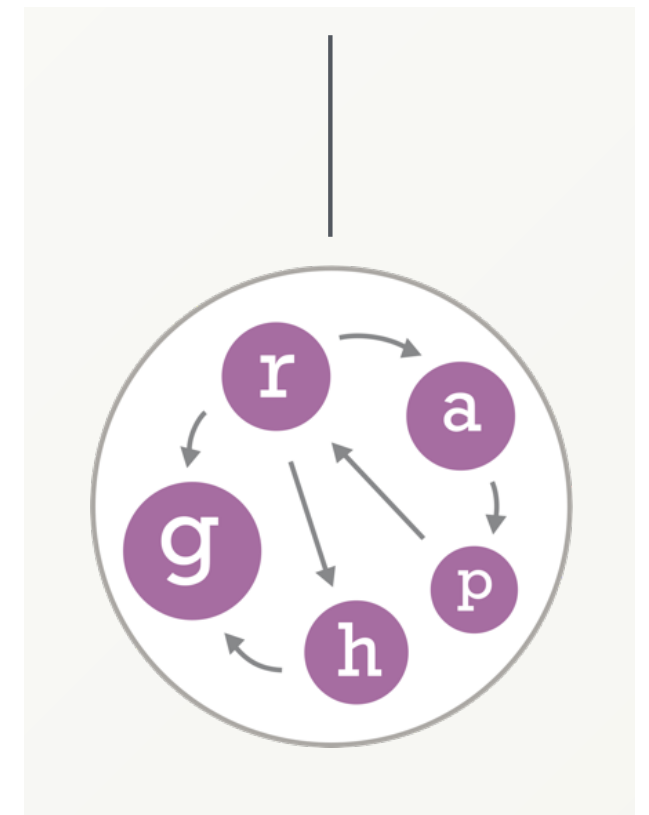


ArangoDB

Collecciones de aristas

- En resumen:
- Lugar para representar/manipular relaciones
- Comparable con relaciones de N:M en sistemas SQL
- Documentos, pero con atributos especiales
 - `_from`: `_id` valor del vértice origen
 - `_to`: `_id` del vértice destino
- Índice de aristas incorporado para cada colección de aristas
- Elementos de construcción de grafos en ArangoDB.

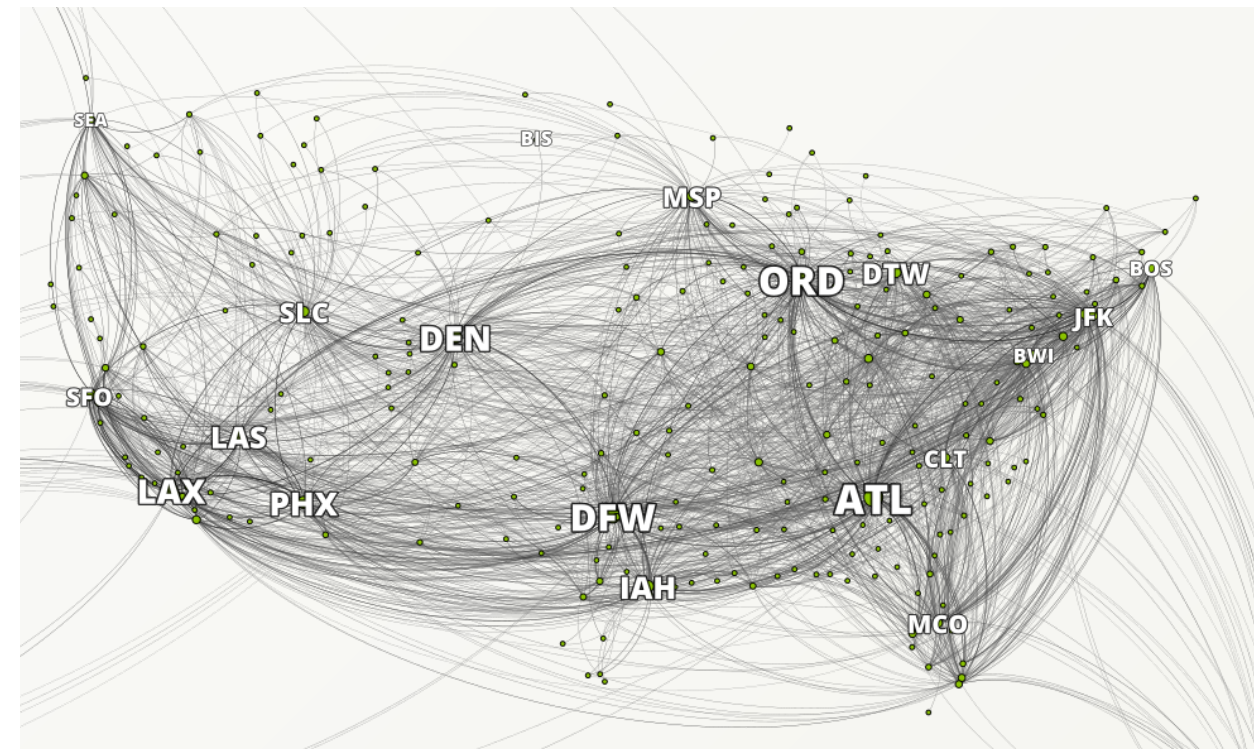
Los atributos especiales `_from` y `_to` en documentos de aristas que apuntan a otros documentos conforman un grafo en ArangoDB



DB en grafos

Aeropuertos y vuelos

- Aeropuertos : 3,375
- Vuelos : 286,463
- Consultas:
 - Listar todos los vuelos que salen de **JFK** (aeropuerto de Nueva York)
 - Listar todos los vuelos que aterrizan en LAX (aeropuerto de Los Ángeles) el 5 de enero.
 - ¿Cuál es la cantidad mínima de escalas para volar desde BIS (Aeropuerto Municipal de Bismarck en Dakota del Norte) a LAX?



Vuelos&Aeropuertos DB

Ejemplos de Documentos en JSON

Aeropuertos

```
{
  "_key": "JFK",
  "_id": "airports/JFK",
  "_rev": "_Y0008KG--T",
  "name": "John F Kennedy Intl",
  "city": "New York",
  "state": "NY",
  "country": "USA",
  "lat": 40.63975111,
  "long": -73.77892556,
  "vip": true
}
```

```
{
  "_key": "BIS",
  "_id": "airports/BIS",
  "_rev": "_Y0SrLBe--r",
  "name": "Bismarck Municipal",
  "city": "Bismarck",
  "state": "ND",
  "country": "USA",
  "lat": 46.77411111,
  "long": -100.7467222,
  "vip": false
}
```

Vuelos

```
{
  "_key": "25471",
  "_id": "flights/25471",
  "_from": "airports/BIS",
  "_to": "airports/MSP",
  "_rev": "_Y008JXG--f",
  "Year": 2008,
  "Month": 1,
  "Day": 2,
  "DayOfWeek": 3,
  "DepTime": 1055,
  "ArrTime": 1224,
  "DepTimeUTC": "2008-01-02T16:55:00.000Z",
  "ArrTimeUTC": "2008-01-02T18:24:00.000Z",
  "UniqueCarrier": "9E",
  "FlightNum": 5660,
  "TailNum": "85069E",
  "Distance": 386
}
```

```
{
  "_key": "71374",
  "_id": "flights/71374",
  "_from": "airports/JFK",
  "_to": "airports/DCA",
  "_rev": "_Y008LYG--N",
  "Year": 2008,
  "Month": 1,
  "Day": 4,
  "DayOfWeek": 5,
  "DepTime": 1604,
  "ArrTime": 1724,
  "DepTimeUTC": "2008-01-04T21:04:00.000Z",
  "ArrTimeUTC": "2008-01-04T22:24:00.000Z",
  "UniqueCarrier": "MQ",
  "FlightNum": 4755,
  "TailNum": "N854AE",
  "Distance": 213
}
```

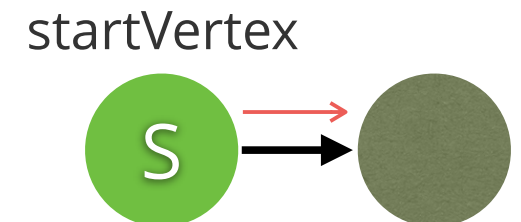

Recorridos en ArangoDB

AQL

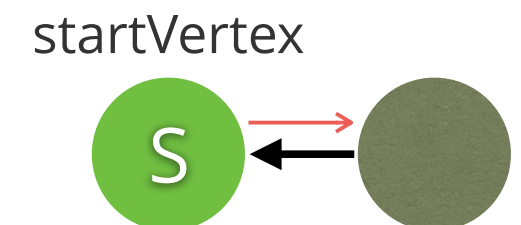
```
FOR vertex[, edge[, path]]  
IN [min[.max]] OUTBOUND |  
INBOUND | ANY startVertex  
edgeCollection[, more...]
```

- **FOR** : vertices, aristas, caminos
- **IN** : define la profundidad minima y maxima.
- **OUTBOUND** : El recorrido sigue las aristas salientes
- **INBOUND** : El recorrido sigue las aristas salientes
- **ANY** : El recorrido sigue las aristas en cualquier dirección.

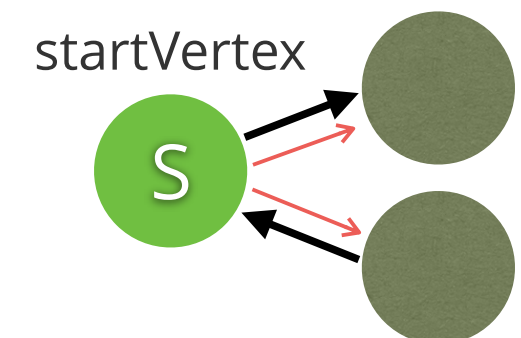
OUTBOUND



INBOUND



ANY

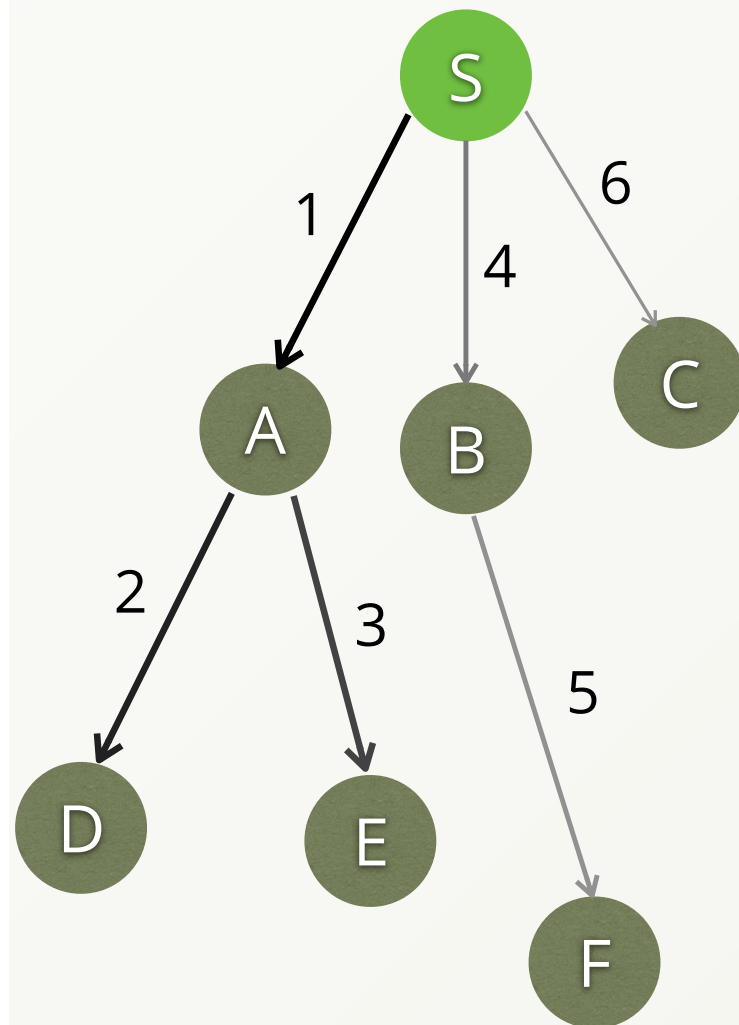


Recorridos en ArangoDB

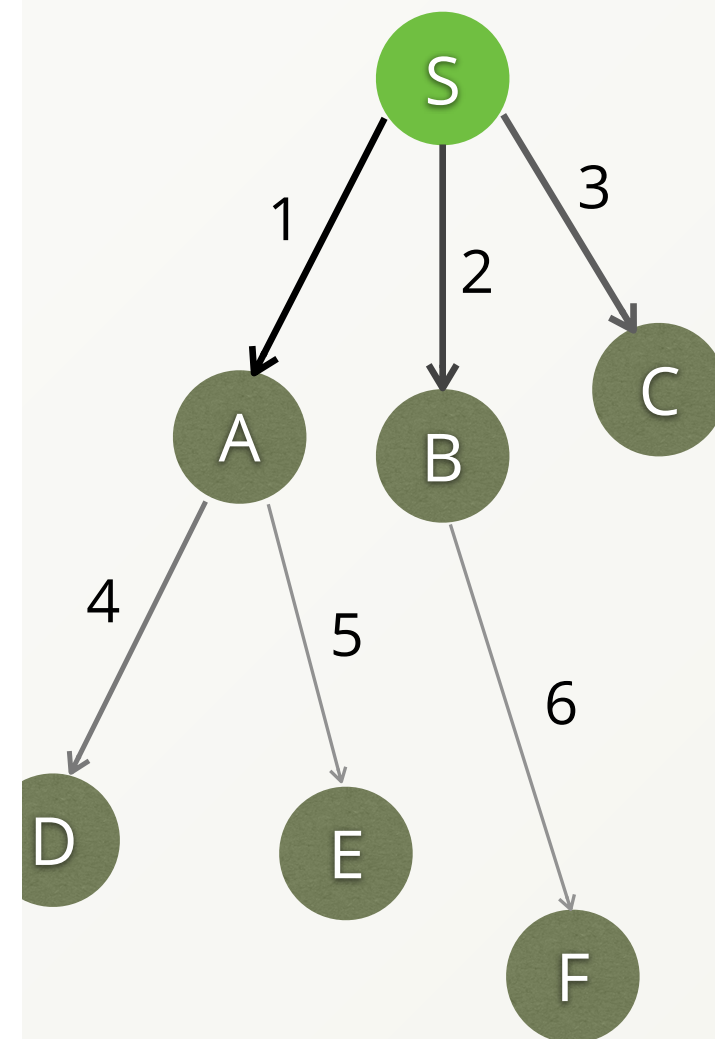
Depth vs. Breadth-First Search

- Depth (defecto): Continúe hacia abajo por las aristas desde el vértice de inicio hasta el último vértice en ese camino o hasta alcanzar la profundidad de recorrido máxima, luego camine por los otros caminos.
- Breadth (opcional) : Siga todas las aristas desde el vértice de inicio hasta el siguiente nivel, luego siga todas las aristas de sus vecinos por otro nivel y continúe este patrón hasta que no haya más aristas para seguir o se alcance la profundidad máxima.

Depth-first search



Breadth-first search

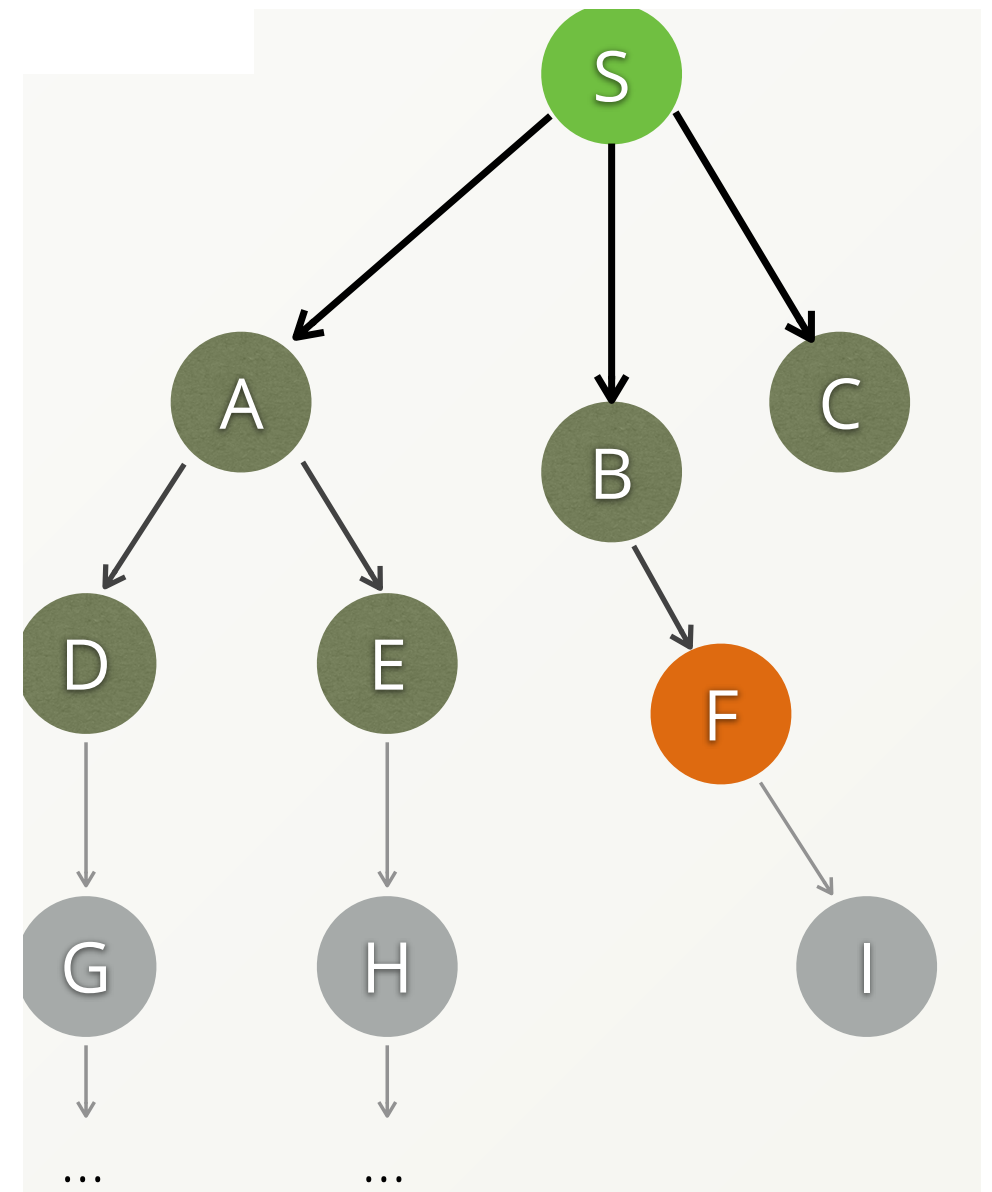


Depth/Breath First search

DFS o BFS?

- Las búsquedas pueden ser significativamente más rápidas si se usan filtros y límites (profundidad máxima).
- Por ejemplo:
 - Recorrer un G desde S con profundidad de 1 .. 10
 - Encontrar un vertice (F) que cumpla algún criterio.
- DFS: visitaría primero A luego exploraría hasta profundidad 10 y volvería.
- BFS: encontraría F a profundidad 2 y no exploraría las demás profundidades.

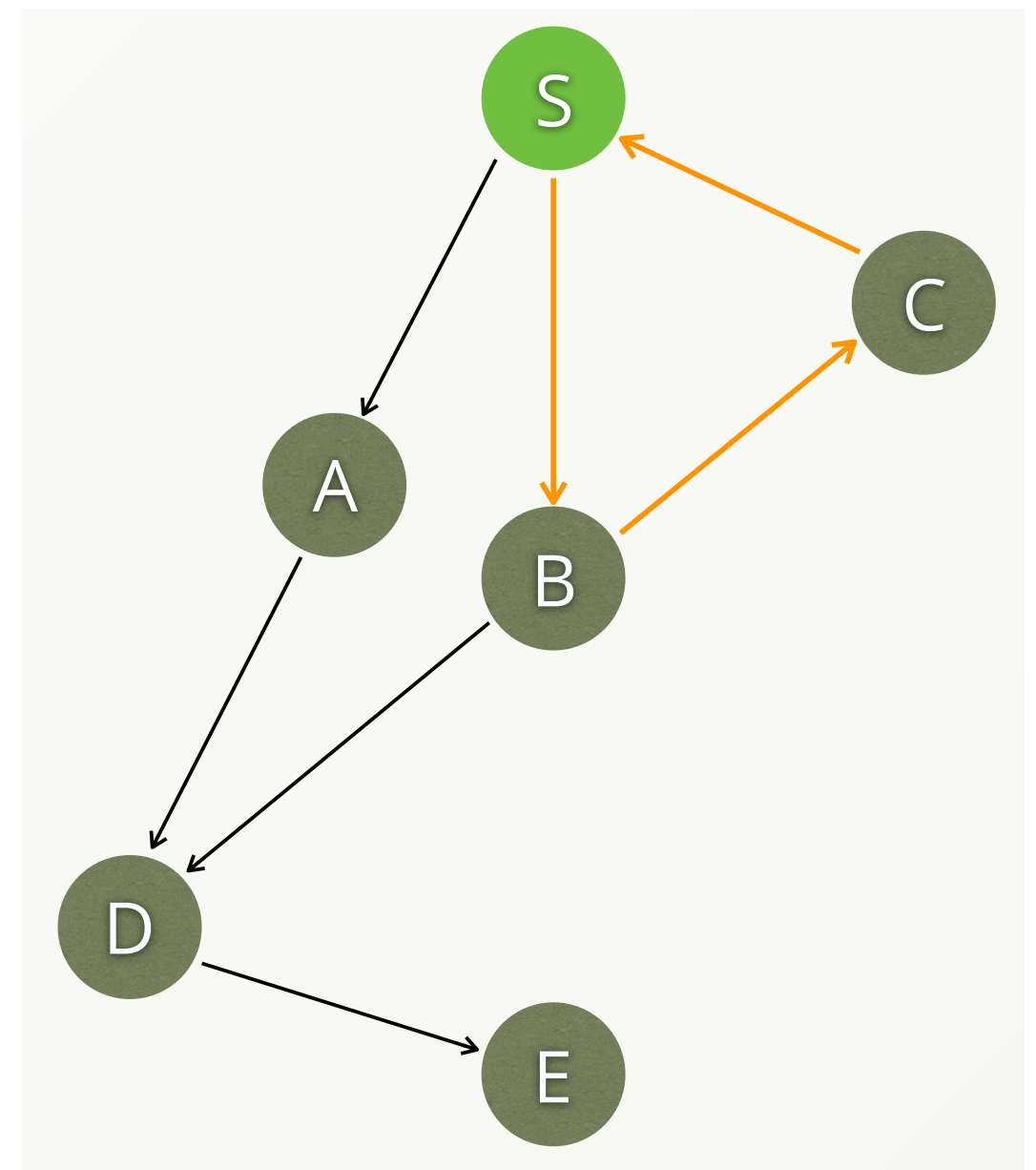
```
FOR v IN 1..10 OUTBOUND  
'verts/S' edges OPTIONS  
{bfs: true}  
FILTER v._key == 'F'  
LIMIT 1  
RETURN v
```



Depth/Breath First search

Controlando recorridos

- Los Grafos pueden ser complejos.
 - Múltiples rutas entre dos vertices
 - Ciclos
- Las aristas de un camino no pueden estar duplicadas.
- Se permiten vértices duplicados en una ruta a menos que el recorrido esté configurado de otra manera.



S -> E

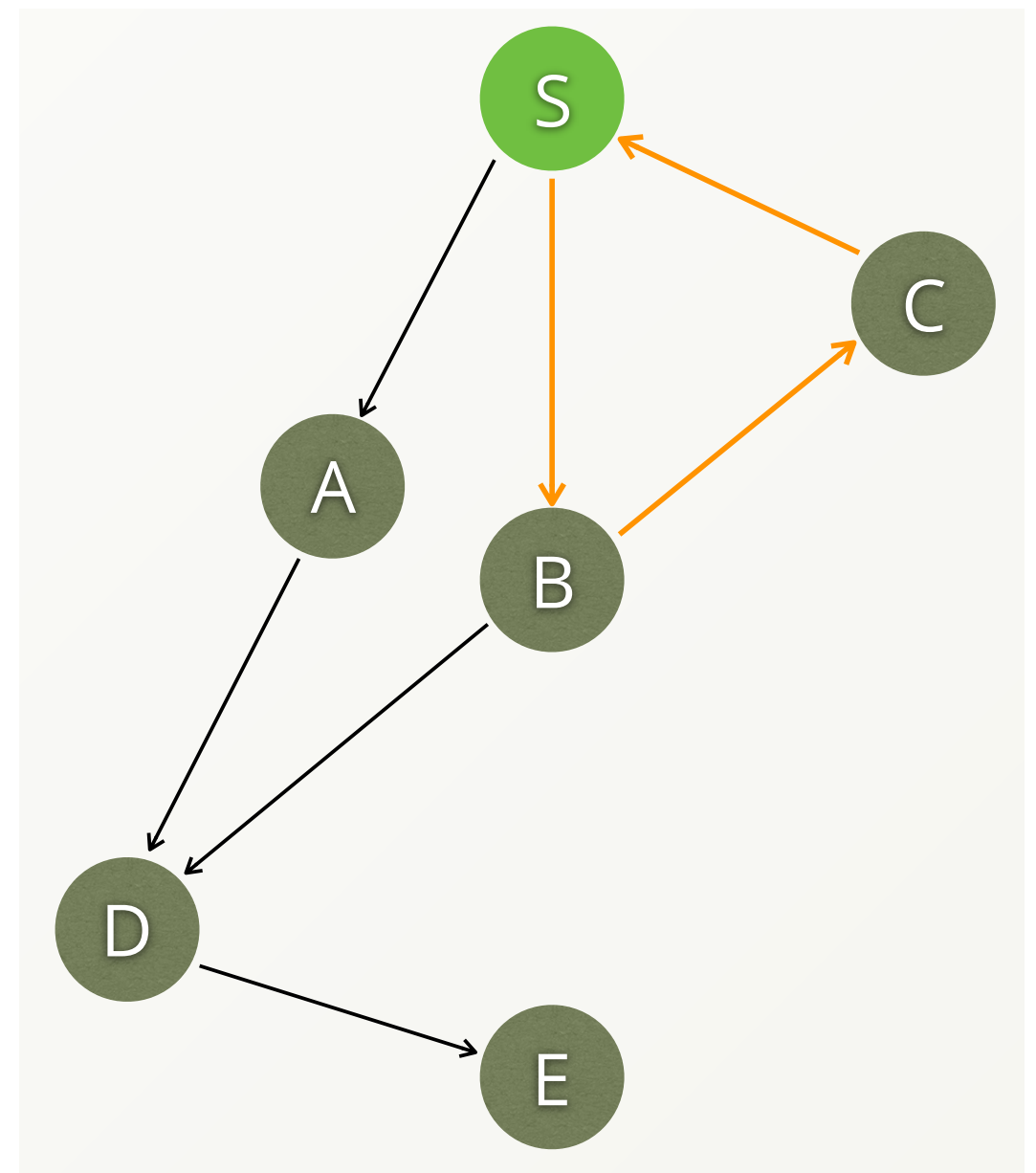
Depth/Breath First search

Controlando recorridos

```
FOR v, e, p IN 1..5 OUTBOUND  
'verts/S' edges OPTIONS {  
    uniqueVertices: 'none',  
    uniqueEdges: 'path'  
}  
RETURN CONCAT_SEPARATOR( '->',  
p.vertices[*]._key)
```

"S->A->D->E"

- uniqueVertices: 'path' asegura que no existen vertices duplicados en un camino.
- uniqueVertices: 'global' asegura que todos los vertices alcanzables son visitados solo una vez (BFS:true).



Depth/Breath First search

Ejemplo

- Listar todos los aeropuertos accesibles desde un aeropuerto particular.

```
FOR airport IN OUTBOUND 'airports/LAX' flights  
RETURN DISTINCT airport
```

```
FOR airport IN OUTBOUND 'airports/LAX' flights  
OPTIONS { bfs: true, uniqueVertices: 'global' }  
RETURN airport
```

Cual es más eficiente?

Combinando documentos y grafos en una misma consulta

ArangoDB

```
FOR orig IN airports
  FILTER orig._key IN [ "JFK", "PBI" ]
  FOR dest, flight IN
    OUTBOUND orig flights
  FILTER dest.FlightNum IN [ 859, 860 ]
  RETURN { from: orig.name,
    to: dest.name, number: f.FlightNum,
    day: f.Day }
```

<https://www.arangodb.com/docs/stable/aql/fundamentals-syntax.html>

```
FOR u IN users
  FILTER u.age < 39
  RETURN u
```

```
FOR u IN users
  FILTER u.status == "not active"
  UPDATE u WITH { status:
    "inactive" } IN users
```

AQL

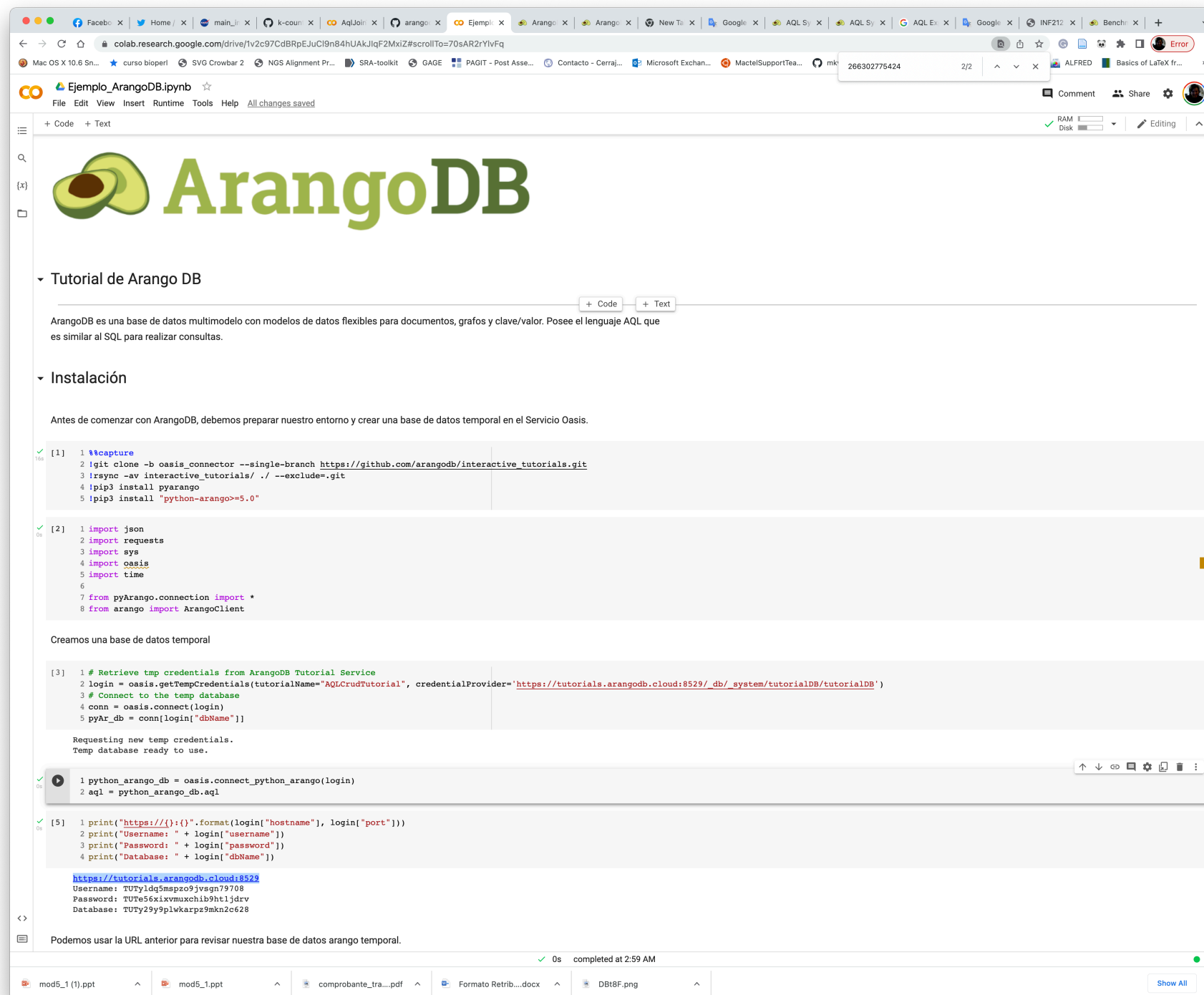
ArangoDB

- **FOR**: Iterate over a collection or View, all elements of an array or traverse a graph
- **RETURN**: Produce the result of a query.
- **FILTER**: Restrict the results to elements that match arbitrary logical conditions.
- **SEARCH**: Query an `arangosearch` or `search-alias` View.
- **SORT**: Force a sort of the array of already produced intermediate results.
- **LIMIT**: Reduce the number of elements in the result to at most the specified number, optionally skip elements (pagination).
- **LET**: Assign an arbitrary value to a variable.
- **COLLECT**: Group an array by one or multiple group criteria. Can also count and aggregate.
- **WINDOW**: Perform aggregations over related rows.
- **REMOVE**: Remove documents from a collection.
- **UPDATE**: Partially update documents in a collection.
- **REPLACE**: Completely replace documents in a collection.
- **INSERT**: Insert new documents into a collection.
- **UPSERT**: Update/replace an existing document, or create it in the case it does not exist.
- **WITH**: Specify collections used in a query (at query begin only)

<https://www.arangodb.com/docs/stable/aql/fundamentals-syntax.html>

ArangoDB

ArangoDB Google Colab



The screenshot shows a Google Colab notebook titled "Ejemplo_ArangoDB.ipynb". The notebook content includes the ArangoDB logo, a tutorial introduction, and installation instructions. The code cells are as follows:

```
[1] 1 %capture
2 git clone -b oasis_connector --single-branch https://github.com/arangodb/interactive_tutorials.git
3 %sync -av interactive_tutorials/ ./ --exclude=.git
4 %pip3 install pyarango
5 %pip3 install "python-arango==5.0"
```

```
[2] 1 import json
2 import requests
3 import sys
4 import oasis
5 import time
6
7 from pyArango.connection import *
8 from arango import ArangoClient
```

Creemos una base de datos temporal

```
[3] 1 # Retrieve tmp credentials from ArangoDB Tutorial Service
2 login = oasis.getTempCredentials(tutorialName="AQLCrudTutorial", credentialProvider='https://tutorials.arangodb.cloud:8529/_db/_system/tutorialDB/tutorialDB')
3 # Connect to the temp database
4 conn = oasis.connect(login)
5 pyAr_db = conn[login["dbName"]]
```

Requesting new temp credentials.
Temp database ready to use.

```
[4] 1 python_arango_db = oasis.connect_python_arango(login)
2 aql = python_arango_db.aql
```

```
[5] 1 print("https://{}:{:}".format(login["hostname"], login["port"]))
2 print("Username: " + login["username"])
3 print("Password: " + login["password"])
4 print("Database: " + login["dbName"])
```

<https://tutorials.arangodb.cloud:8529>
Username: TUTyldq5m5pz09jvsgn79708
Password: TUTe56xixvmuxchib9htljdvr
Database: TUTy29y9plwkarpz9mkn2c628

Podemos usar la URL anterior para revisar nuestra base de datos arango temporal.

https://github.com/adigenova/uohpmd/blob/main/code/Arango_GraphDB.ipynb
<https://www.arangodb.com/docs/stable/aql/tutorial.html>

Consultas?

Consultas o comentarios?

Muchas gracias