

# NoSQL III: ArangoDB y AQL

**Alex Di Genova**

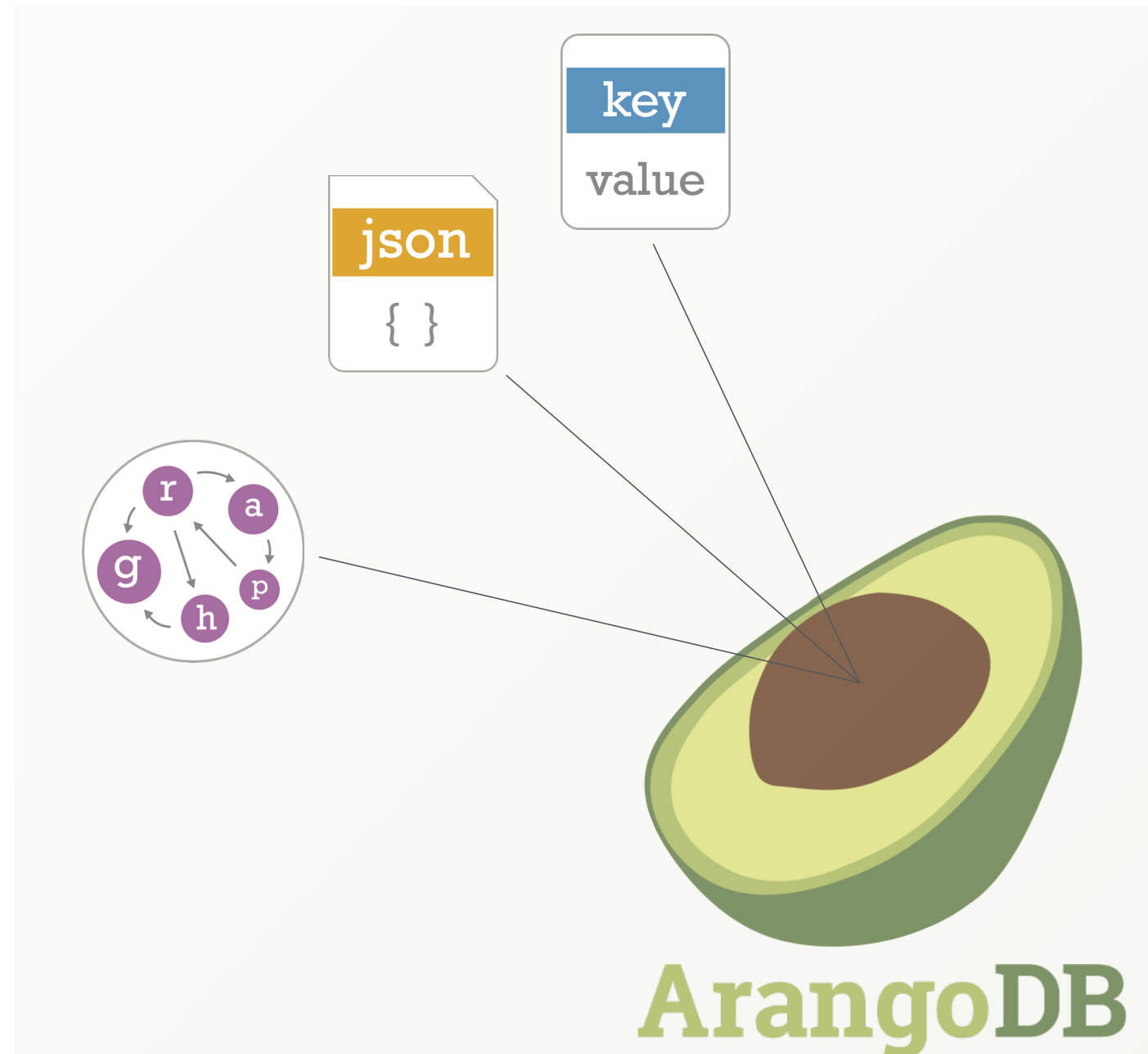
**07/07/2023**

# Repaso

# ArangoDB

## Motor multi-modelo

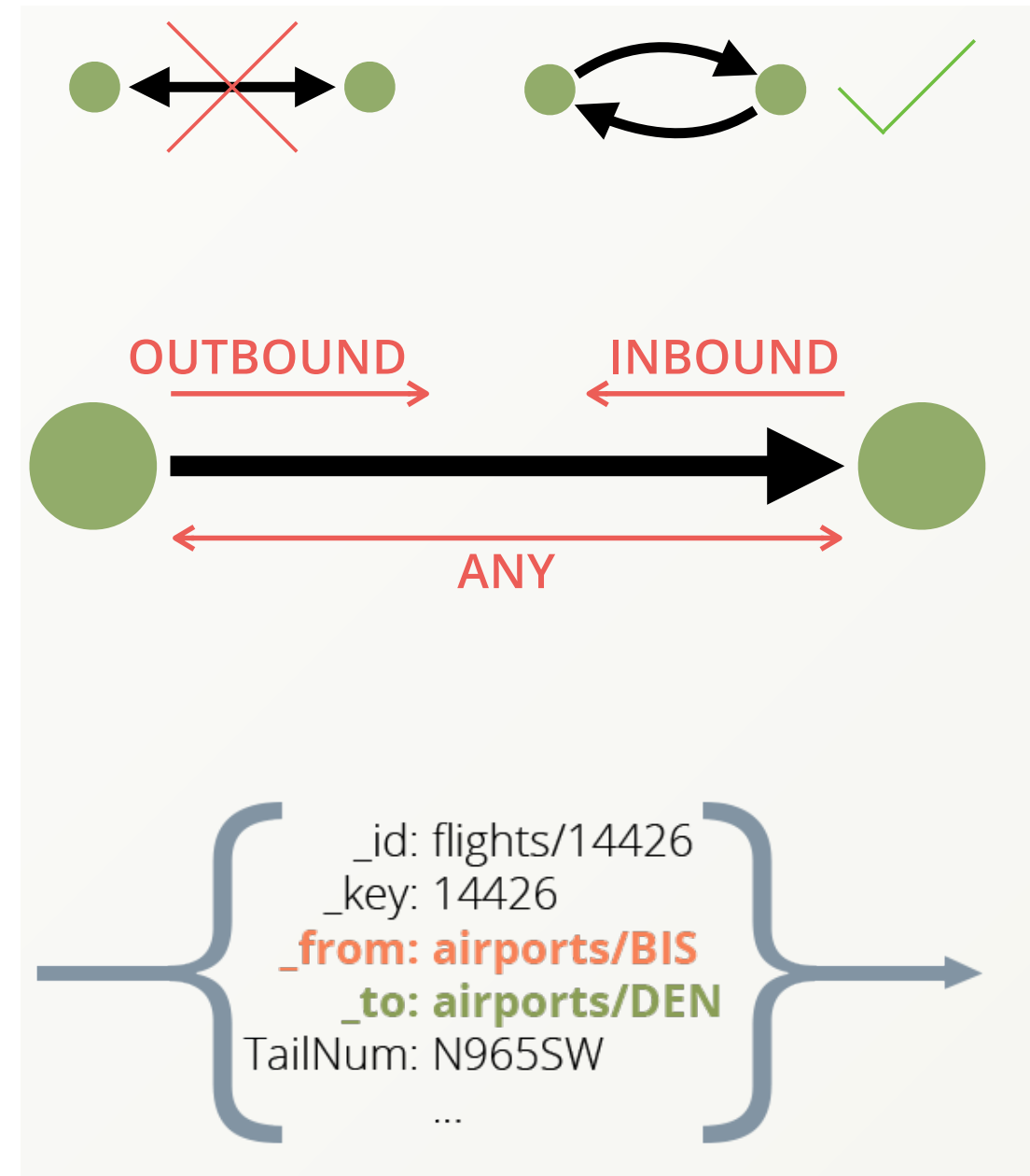
- Características únicas de AQL:
  - Posibilidad de combinar los **3 modelos de datos** en una sola consulta.
  - Podemos combinar uniones (joins), recorridos (traversal), filtros, operaciones geoespaciales y agregaciones en nuestras consultas



# DB en Grafos

## ArangoDB

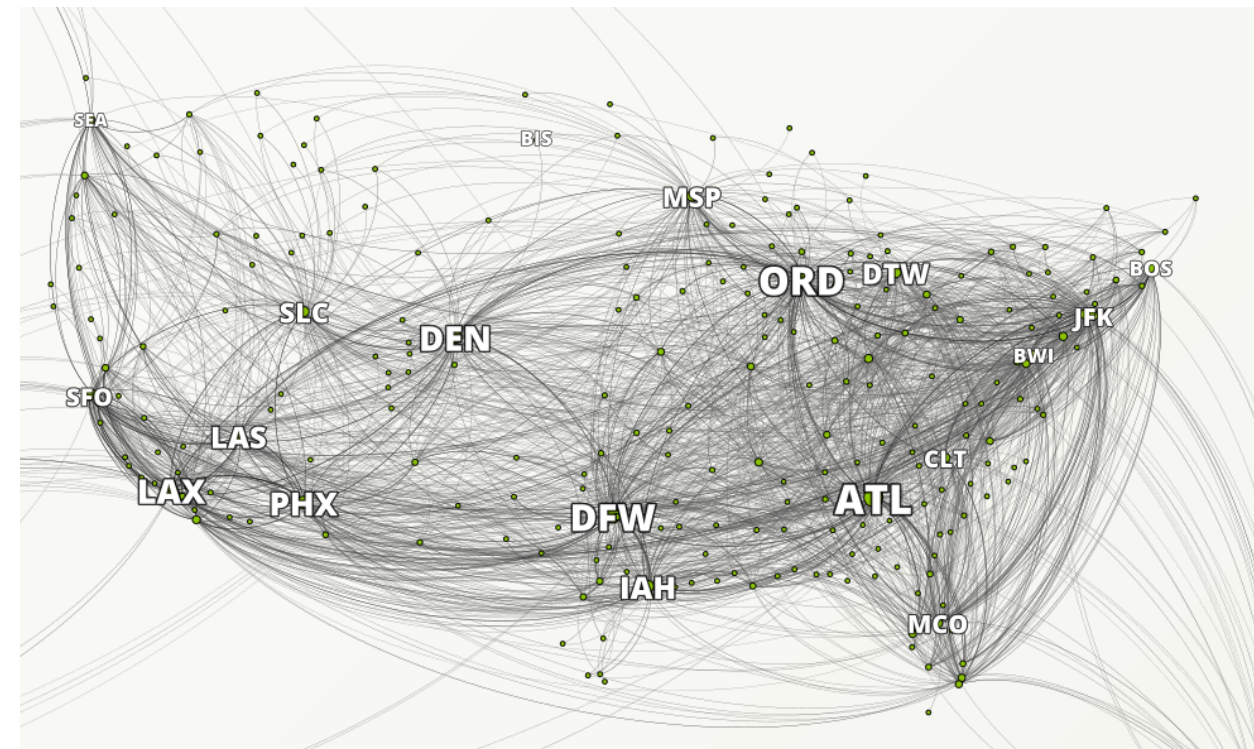
- En ArangoDB, cada arista tiene una sola dirección, no puede apuntar en ambos sentidos a la vez. Este modelo también se conoce como **grafo dirigido**.
- Pero la dirección se puede ignorar (seguir en CUALQUIER dirección - **ANY**) cuando nos movemos en el grafo, o seguir las aristas en dirección inversa (**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.



# 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

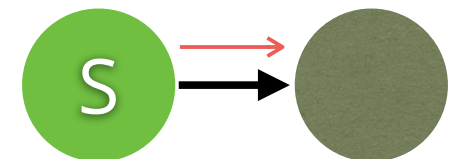
## AQL

```
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.

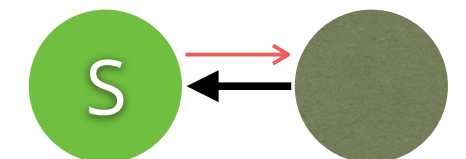
**OUTBOUND**

startVertex



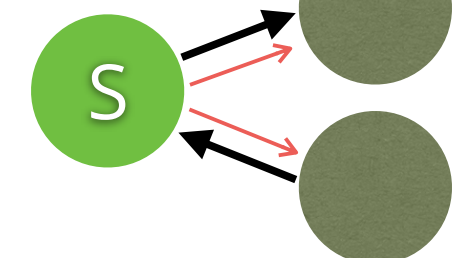
**INBOUND**

startVertex



**ANY**

startVertex

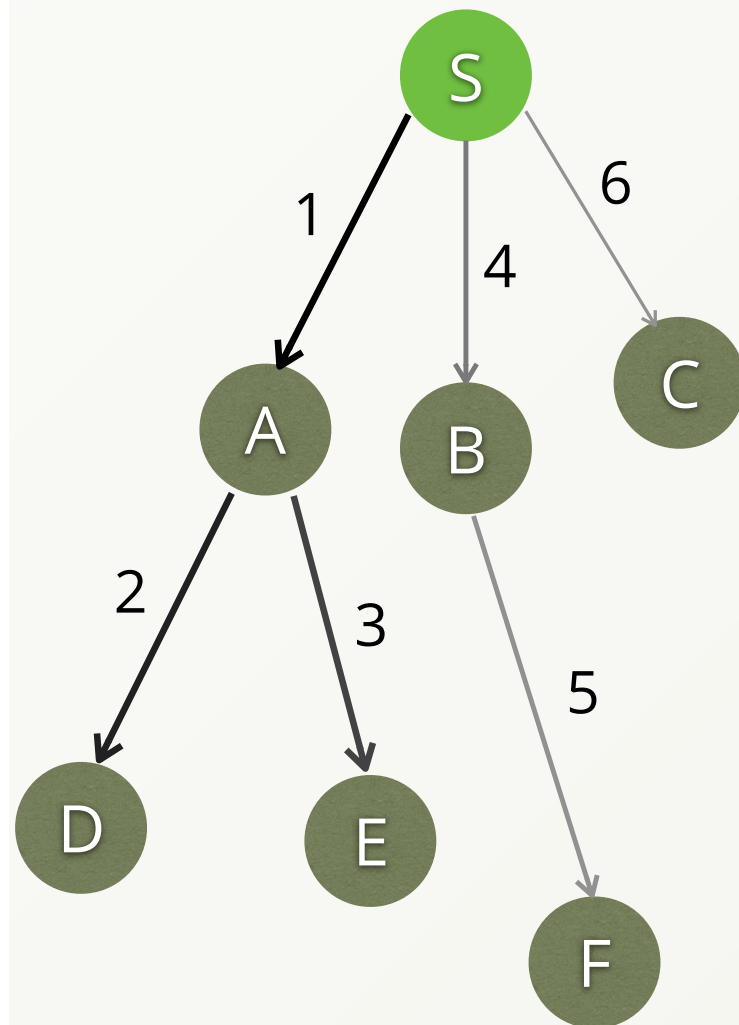


# Recorridos en ArangoDB

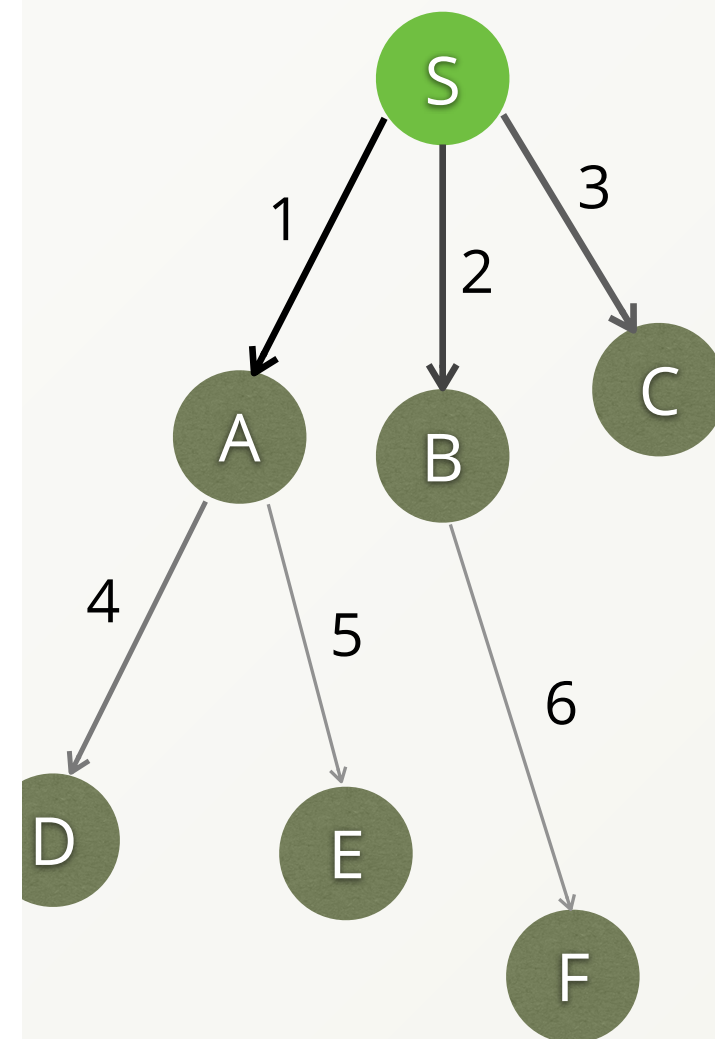
## Depth vs. Breadth-First Search

- 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



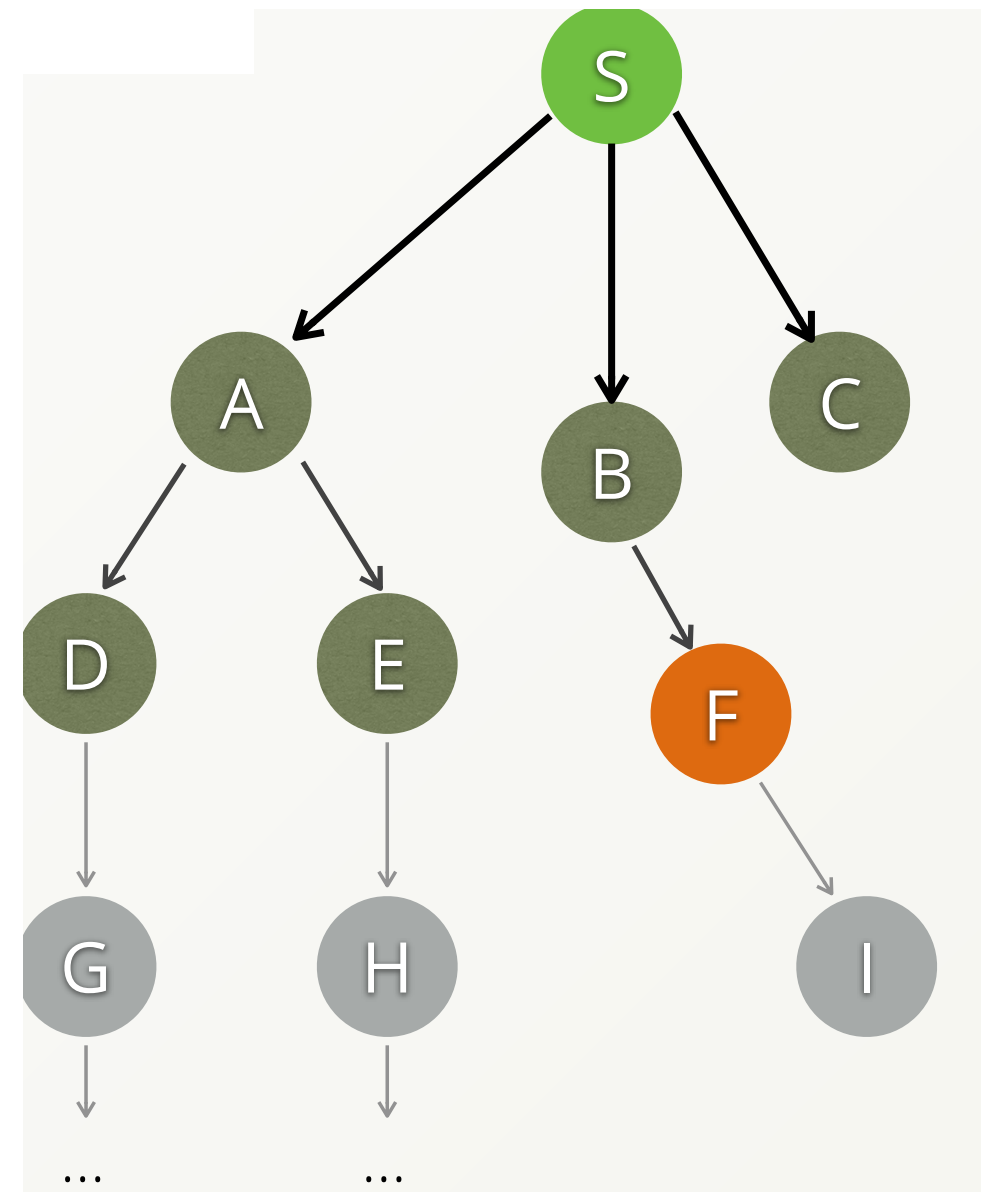


# Depth/Breath First search

## 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.

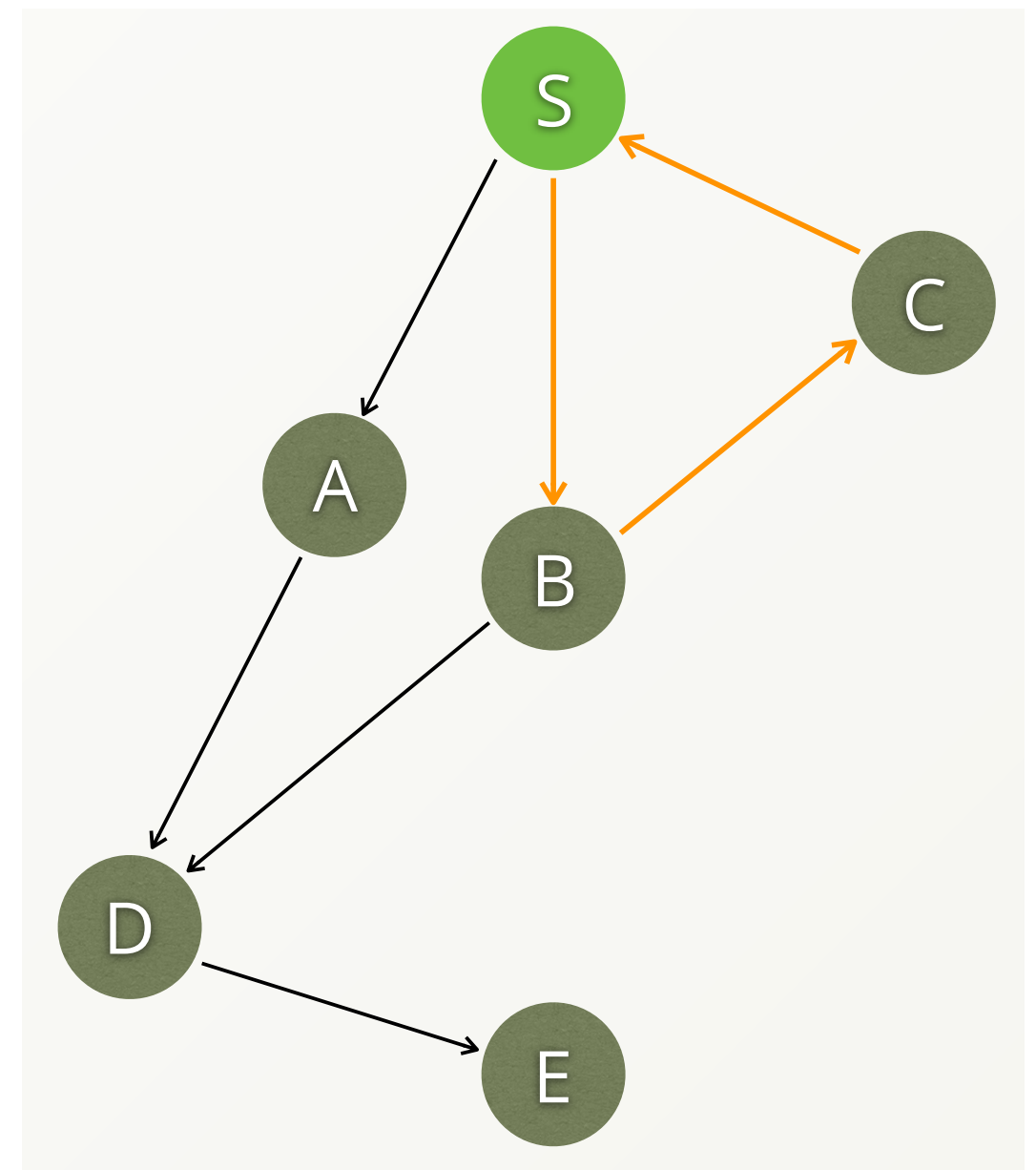
```
FOR v IN 1..10 OUTBOUND  
'verts/S' edges OPTIONS  
{bfs: true}  
FILTER v._key == 'F'  
LIMIT 1  
RETURN v
```



# 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

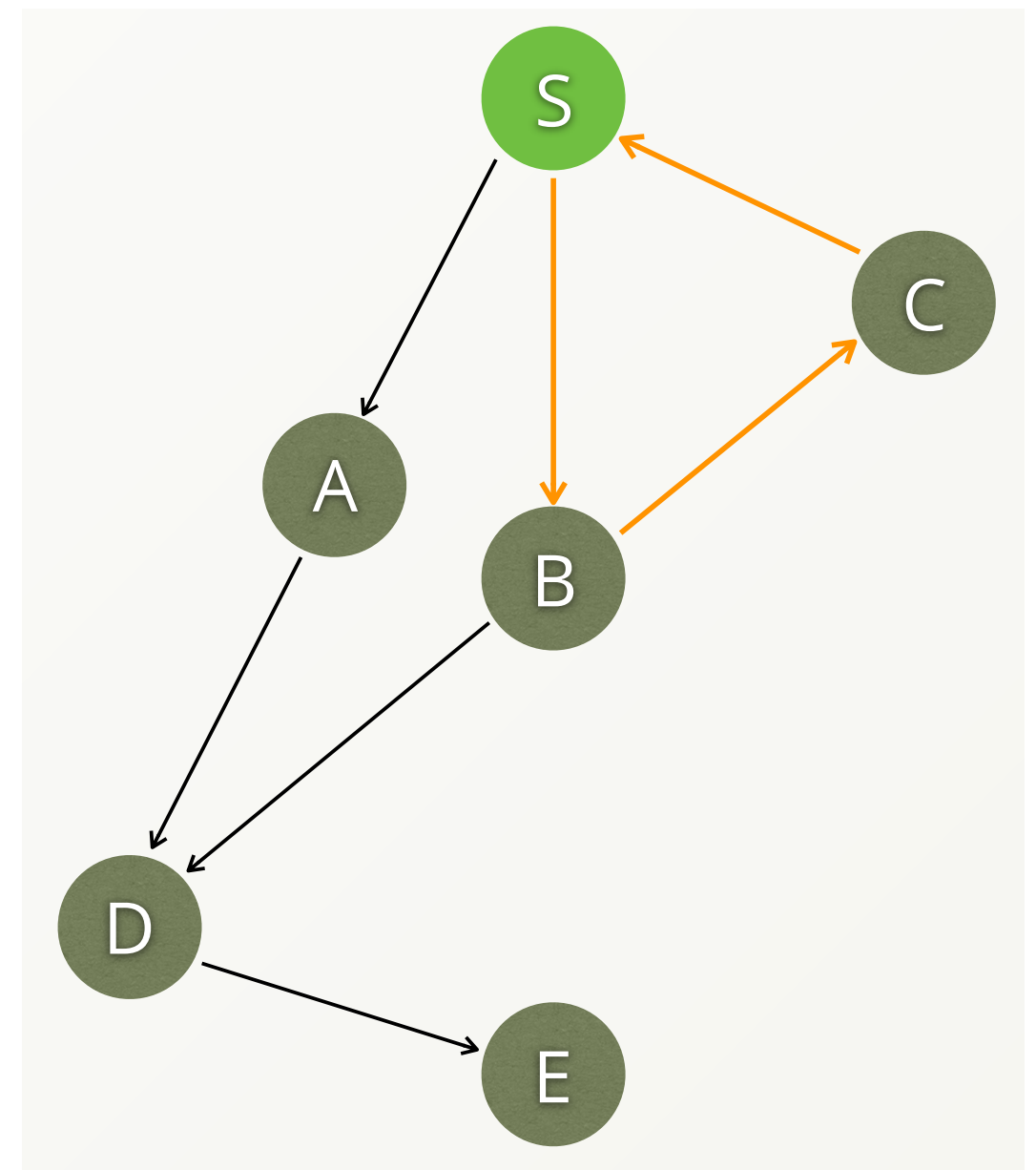
# Depth/Breath First search

## 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

## Syntax

- Una consulta AQL debe devolver un resultado (RETURN) o ejecutar una operación de modificación de datos (INSERT, UPDATE, REPLACE, REMOVE).
- **FOR**: Iterar sobre una colección o vista, todos los elementos de un arreglo, o recorrer un grafo.
- **RETURN**: retornar el resultado de una consulta.
- **FILTER**: Restringir los resultados a elementos que cumplan condiciones lógicas arbitrarias.
- **SEARCH**: Consultar una vista de arangosearch o search-alias.
- **SORT**: Ordenar un arreglo de resultados intermedios ya producidos.
- **LIMIT**: Reducir el número de elementos en el resultado a un máximo especificado.
- **LET**: Asignar un valor a una variable.
- **COLLECT**: Agrupar un arreglo por uno o varios criterios. También se puede contar y agregar.
- **WINDOW**: Realizar agregaciones sobre filas relacionadas.
- **REMOVE**: Eliminar documentos de una colección.
- **UPDATE**: Actualizar parcialmente documentos en una colección.
- **REPLACE**: Reemplazar completamente documentos en una colección.
- **INSERT**: Insertar nuevos documentos en una colección.
- **UPSERT**: Actualizar/reemplazar un documento existente o crearlo en caso de que no exista.
- **WITH**: Especificar las colecciones utilizadas en una consulta (solo al comienzo de la consulta).



# AQL

## Ejemplos y tipos de datos

```
FOR u IN users
  FILTER u.type == "newbie" && u.active == true
  RETURN u.name
```

```
FOR u IN users
  FOR f IN friends
    FILTER u.active == true && f.active == true && u.id ==
f.userId
  RETURN u.name
```

```
FOR u IN users
  LET friends = u.friends
  RETURN { "name" : u.name, "friends" : friends }
```

```
LET users = []
FOR u IN users
  RETURN u
```

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

```
//subconsultas
FOR p IN persons
  LET recommendations = ( // subconsulta comienza
    FOR r IN recommendations
      FILTER p.id == r.personId
      SORT p.rank DESC
      LIMIT 10
      RETURN r
    ) // termina
  RETURN { person : p, recommendations : recommendations }
```

Data type	Description
<b>null</b>	Valor vacío
<b>boolean</b>	Falso o verdadero
<b>number</b>	Numero con signo (real)
<b>string</b>	Texto codificado
<b>array</b> / list	Secuencia de valores referidos por sus posiciones.
<b>object</b> / document	Secuencia de valores referidos por sus nombres.

# AQL

## Operadores

Operadores	Descripción
<code>==</code>	Igualdad
<code>!=</code>	Distinto
<code>&lt;</code>	Menor
<code>&lt;=</code>	Menor igual
<code>&gt;</code>	Mayor
<code>&gt;=</code>	Mayor igual
<code>IN</code>	Prueba si un valor esta contenido en un arreglo/lista
<code>NOT IN</code>	Prueba si un valor no esta contenido en un arreglo/lista
<code>LIKE</code>	prueba si un valor de texto coincide con un patrón
<code>NOT LIKE</code>	prueba si un valor de text no coincide con un patrón
<code>=~</code>	prueba si un valor de text coincide con una expresión regular
<code>!~</code>	prueba si un valor de texto no coincide con una expresión regular

```
0 == null // falso
1 > 0 // verdadero
true != null // verdadero
45 <= "yikes!" // verdadero
65 != "65" // verdadero
65 == 65 // verdadero
1.23 > 1.32 // falso
1.5 IN [ 2, 3, 1.5 ] // verdadero
"foo" IN null // falso
42 NOT IN [ 17, 40, 50 ] // verdadero
"abc" == "abc" // verdadero
"abc" == "ABC" // falso
"foo" LIKE "f%" // verdadero
"foo" NOT LIKE "f%" // falso
```

```
25 > 1 && 42 != 7 // verdadero
22 IN [ 23, 42 ] || 23 NOT IN [ 22, 7 ] //v
25 != 25 //falso
```

- `+` suma
- `-` resta
- `*` multiplicacion
- `/` division
- `%` modulo

# AQL

## Consultas (DAQ y DMQ)

- DAQ (Data access Query)

```
FOR doc IN users
  FILTER doc.status == "active"
  SORT doc.name
  LIMIT 10
```

- DMQ (Data Modification Query)

```
INSERT {
  firstName: "Alex",
  name: "Perez",
  profession: "artista"
} INTO users
```

```
UPDATE "Alex" WITH {
  status: "activo",
  location: "Rancagua"
} IN users
```

```
REPLACE {
  _key: "user1", // no se puede modificar
  firstName: "Alex",
  name: "Peréz",
  status: "activo",
  level: "vip"
} IN users
```

```
REMOVE "Alex" IN users
```

- DMQ (Data Modification Query)

```
FOR i IN 1..10000
  INSERT {
    id: 100000 + i,
    edad: 18 + FLOOR(RAND() * 25),
    nombre: CONCAT('test', TO_STRING(i)),
    estado: i % 2 == 0 ? "active" : "not active",
    sexo: i % 3 == 0 ? "hombre" : i % 3 == 1 ?
    "mujer" : "no-informado"
  } IN users
```

```
FOR u IN users
  INSERT u IN backup
```

```
LET r1 = (FOR u IN users REMOVE u IN users)
LET r2 = (FOR u IN backup REMOVE u IN backup)
RETURN true
```

# AQL

## Operaciones de alto Nivel

```
FOR u IN users
  LET subquery = (FOR l IN locations RETURN
l.location)
  RETURN { "user": u, "locations": subquery }
```

```
FOR value IN ["foo", "bar", "bar", "baz", "foo"]
  RETURN DISTINCT value
```

```
FOR u IN users
  FILTER u.active == true
  SORT u.age ASC
  LIMIT 5
  FILTER u.sexo == "mujer"
  RETURN u
```

```
FOR u IN users
  SORT u.lastName, u.firstName, u.id DESC
  RETURN u
```

```
FOR u IN users
  LIMIT 5
  RETURN u
```

```
FOR u IN users
  LET numRecom = LENGTH(u.recomendaciones)
  RETURN {
    "usuario" : u,
    "numRecomendaciones" : numRecom,
    "esUsuarioPopular" : numRecom >= 10
  }
```

- La operación **COLLECT** se puede utilizar para agrupar datos por uno o varios criterios de grupo.

```
FOR u IN users
  COLLECT AGGREGATE minAge = MIN(u.age), maxAge =
MAX(u.age)
  RETURN {
    minAge,
    maxAge
  }
```

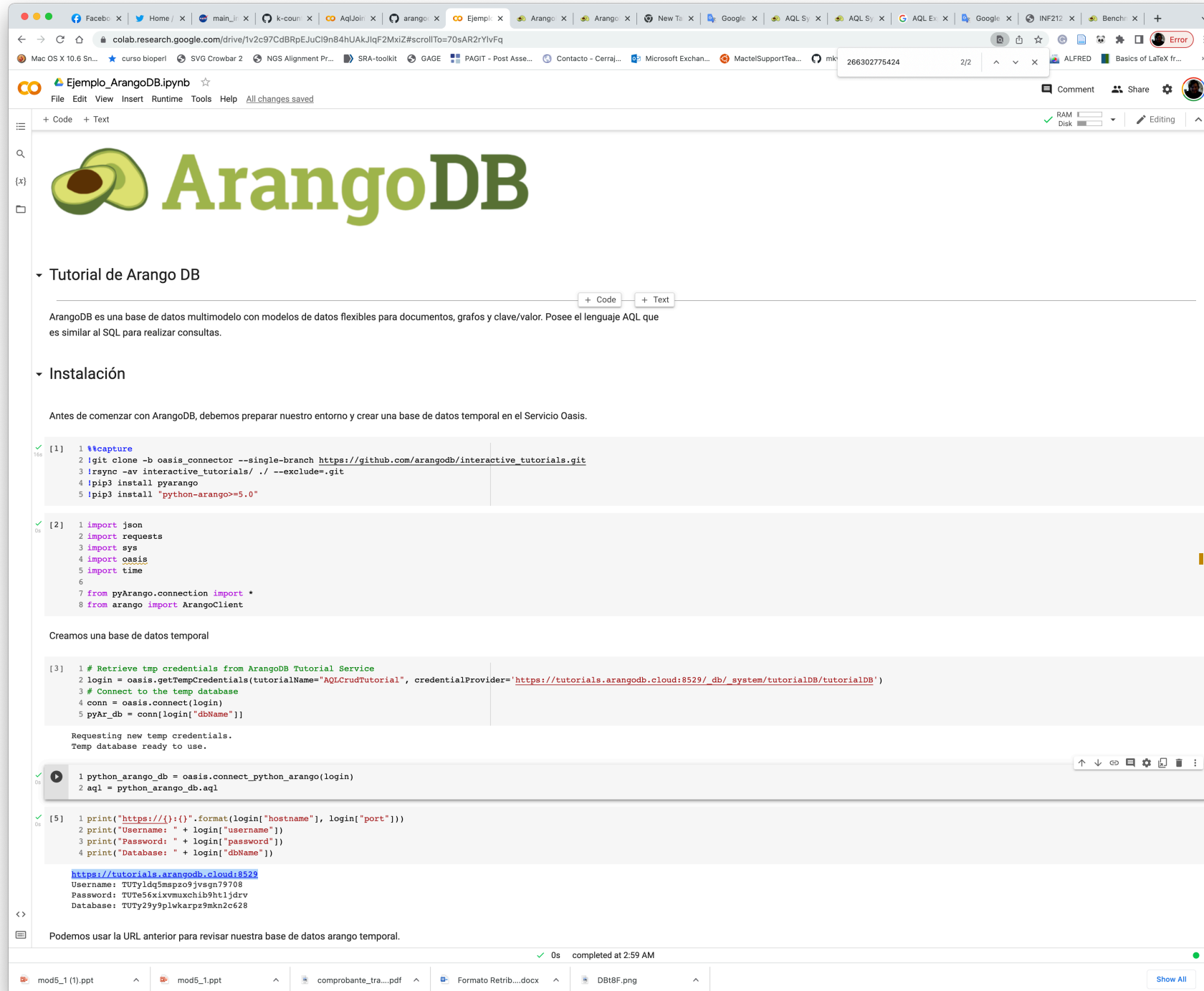
- LENGTH() / COUNT()
- MIN()
- MAX()
- SUM()
- AVERAGE() / AVG()
- STDDEV\_POPULATION() / STDDEV()
- STDDEV\_SAMPLE()
- VARIANCE\_POPULATION() / VARIANCE()
- VARIANCE\_SAMPLE()
- UNIQUE()
- SORTED\_UNIQUE()
- COUNT\_DISTINCT() / COUNT\_UNIQUE()
- BIT\_AND()
- BIT\_OR()
- BIT\_XOR()

```
WITH collection1 [, collection2 [, ... collectionN ] ]
```

- FOR ... IN collection
- INSERT ... INTO collection
- UPDATE ... IN collection
- GRAPH "graph-name" (via the graph definition)

# ArangoDB

## ArangoDB Google Colab



The screenshot shows a Google Colab notebook interface. At the top, there's a browser tab bar with various tabs open. The notebook title is "Ejemplo\_ArangoDB.ipynb". Below the title bar, there's a menu with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". The main content area features the ArangoDB logo and a section titled "Tutorial de Arango DB". The text describes ArangoDB as a multimodel database and mentions the AQL language. Below this, there's a section titled "Instalación" which includes instructions to prepare the environment and create a temporary database. The code cells show the following:

```
[1] 1 %capture
    2 !git clone -b oasis_connector --single-branch https://github.com/arangodb/interactive_tutorials.git
    3 !rsync -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://{0}:{1}".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: TUTyldq5mepzo9jvsgn79708
Password: TUTe56xixvmuxchib9htljdvr
Database: TUTy29y9plwkarpz9mkn2c628
```

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

At the bottom, there's a status bar showing "completed at 2:59 AM" and a file manager with several files listed.

[https://github.com/adigenova/uohdb/blob/main/code/NoSQL ArangoDB ejemplo.ipynb](https://github.com/adigenova/uohdb/blob/main/code/NoSQL%20ArangoDB%20ejemplo.ipynb)  
<https://www.arangodb.com/docs/stable/aql/tutorial.html>

# Consultas?

Consultas o comentarios?

Muchas gracias