THE BEST DB

Choose the best one.

Arely Viana

Software Developer

What is a DB?

ACID?

01

SQL

PostgreSQL

02

NoSQL

MongoDB

03

Key-Value Store

Redis

04

Graph

Neo4j

01. SQL

Structured Query Language

Docs

MySQL

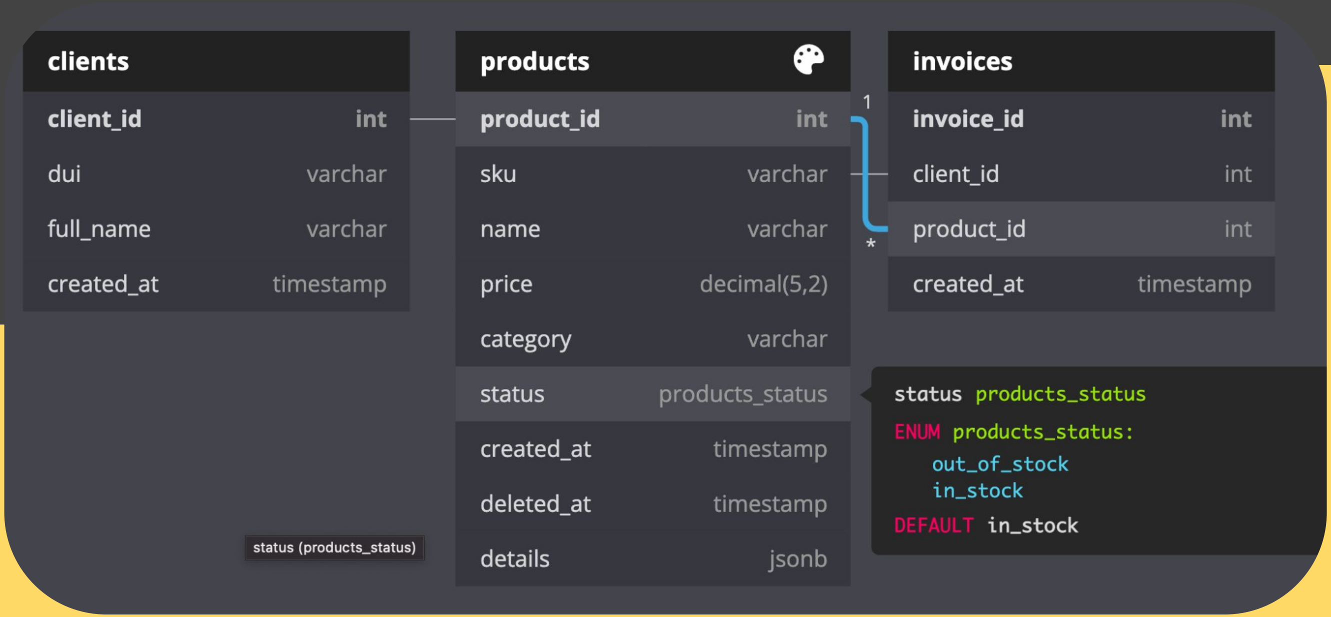
Oracle

SQLite vs PostgreSQL

Microsoft SQL Server

MariaDB

ERD



clients	
client_id	int
dui	varchar
full_name	varchar
created_at	timestamp

products	
product_id	int
sku	varchar
name	varchar
price	decimal(5,2)
category	varchar
status	products_status
created_at	timestamp
deleted_at	timestamp
details	jsonb

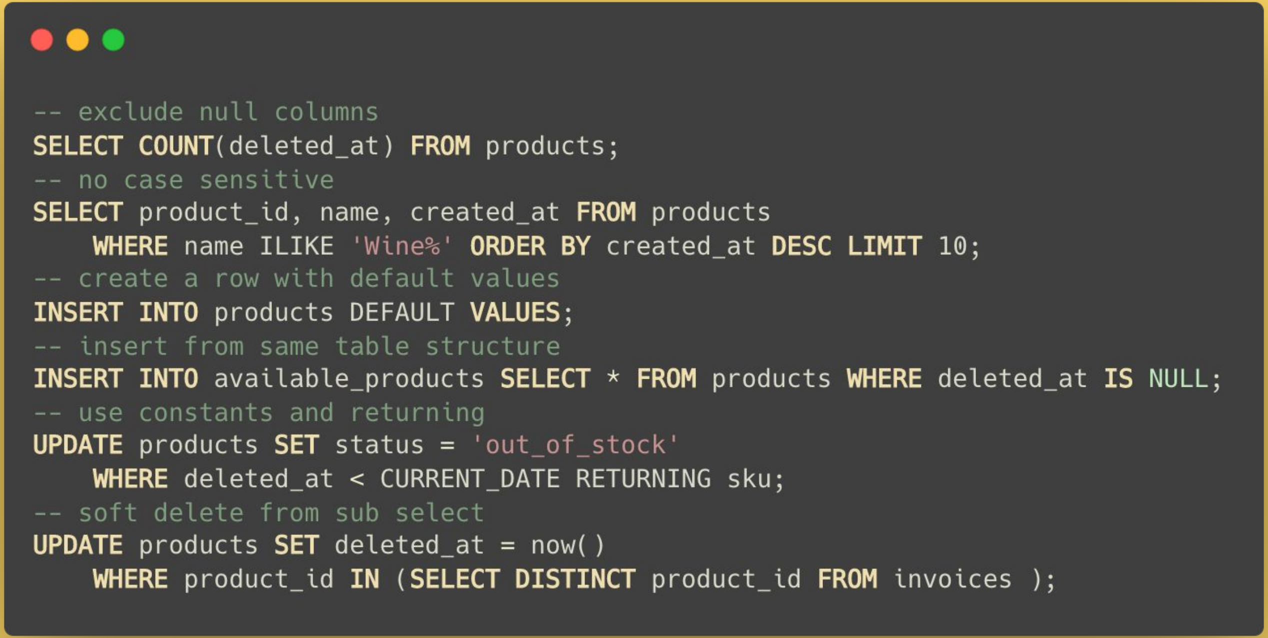
invoices	
invoice_id	int
client_id	int
product_id	int
created_at	timestamp

```
status products_status
ENUM products_status:
    out_of_stock
    in_stock
DEFAULT in_stock
```

- https://www.mockaroo.com/

- https://dbdiagram.io/home
- https://schemaspy.org/

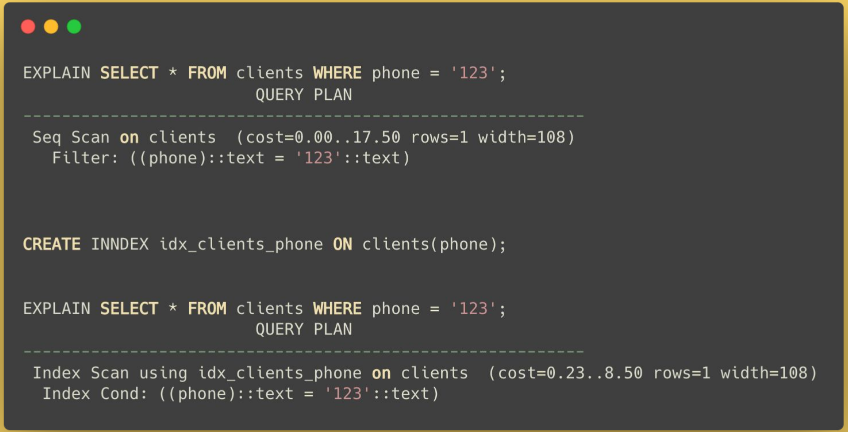
BASIC STATEMENTS



```sql
-- exclude null columns
SELECT COUNT(deleted_at) FROM products;
-- no case sensitive
SELECT product_id, name, created_at FROM products
    WHERE name ILIKE 'Wine%' ORDER BY created_at DESC LIMIT 10;
-- create a row with default values
INSERT INTO products DEFAULT VALUES;
-- insert from same table structure
INSERT INTO available_products SELECT * FROM products WHERE deleted_at IS NULL;
-- use constants and returning
UPDATE products SET status = 'out_of_stock'
    WHERE deleted_at < CURRENT_DATE RETURNING sku;
-- soft delete from sub select
UPDATE products SET deleted_at = now()
    WHERE product_id IN (SELECT DISTINCT product_id FROM invoices );
```

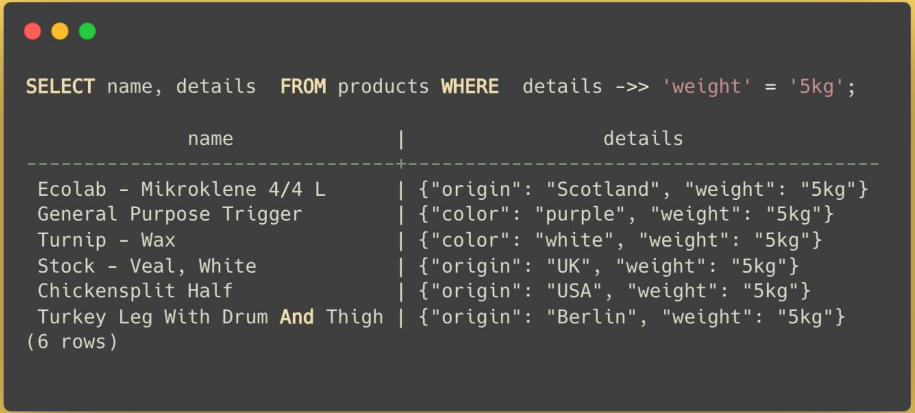
OTHER TOPICS

Index



```
EXPLAIN SELECT * FROM clients WHERE phone = '123';
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on clients  (cost=0.00..17.50 rows=1 width=108)
   Filter: ((phone)::text = '123'::text)



CREATE INNDEX idx_clients_phone ON clients(phone);


EXPLAIN SELECT * FROM clients WHERE phone = '123';
                        QUERY PLAN
-----------------------------------------------------------
 Index Scan using idx_clients_phone on clients  (cost=0.23..8.50 rows=1 width=108)
  Index Cond: ((phone)::text = '123'::text)
```

JSON data type



```
SELECT name, details  FROM products WHERE  details ->> 'weight' = '5kg';

             name               |               details
--------------------------------+--------------------------------------
 Ecolab - Mikroklene 4/4 L      | {"origin": "Scotland", "weight": "5kg"}
 General Purpose Trigger        | {"color": "purple", "weight": "5kg"}
 Turnip - Wax                   | {"color": "white", "weight": "5kg"}
 Stock - Veal, White            | {"origin": "UK", "weight": "5kg"}
 Chickensplit Half              | {"origin": "USA", "weight": "5kg"}
 Turkey Leg With Drum And Thigh | {"origin": "Berlin", "weight": "5kg"}
(6 rows)
```

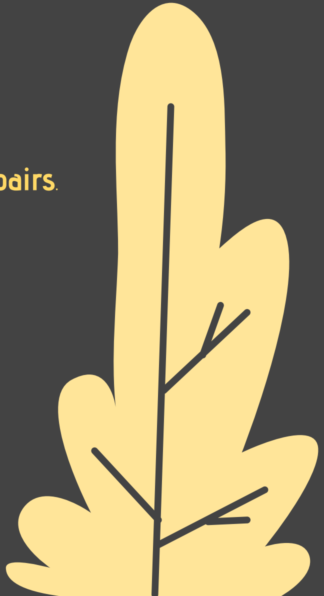
SQL Injection

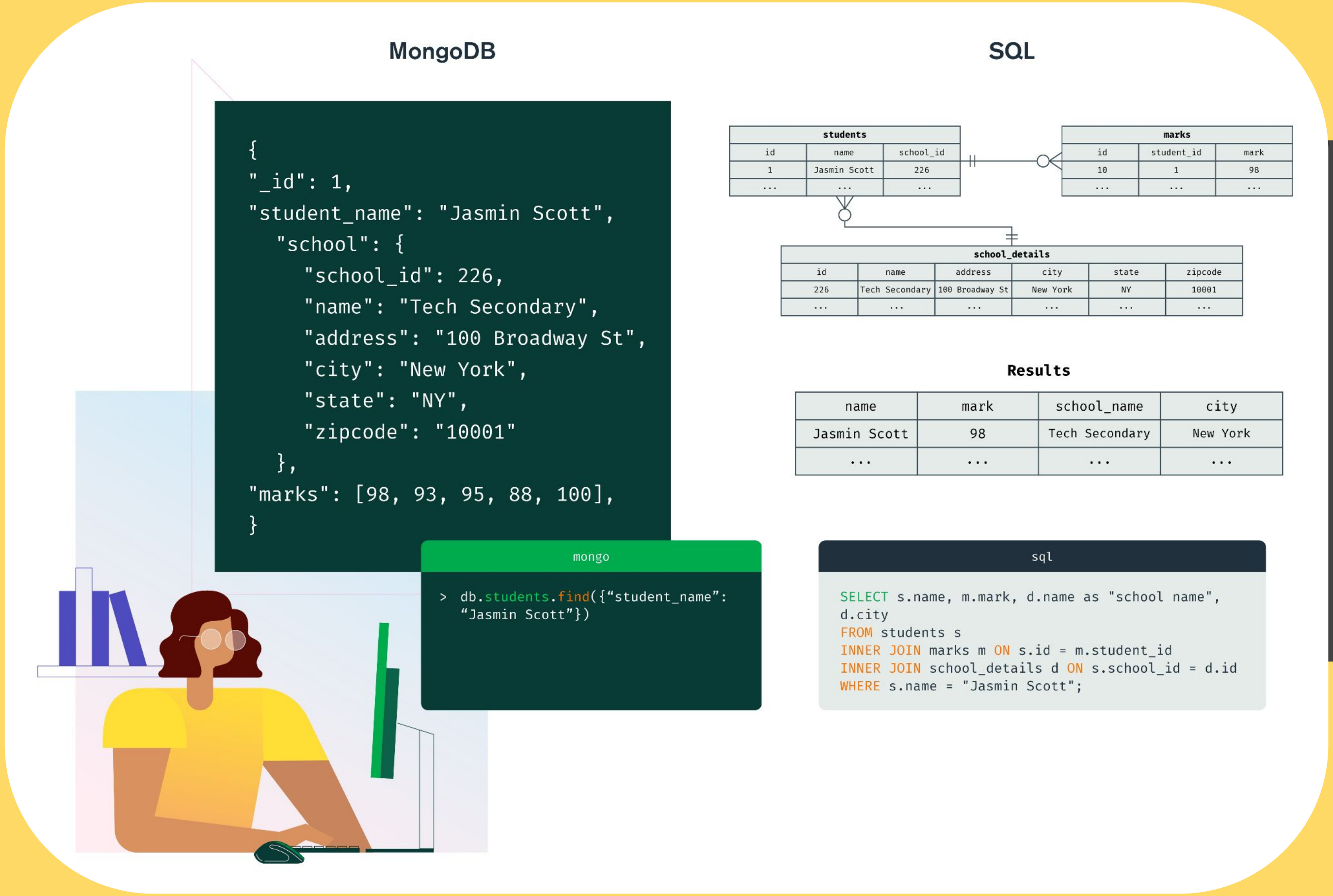
Framewroks - Drivers - Libraries

02. NoSQL

A record is a document, which is a data structure composed of **field and value pairs**.

Docs





MongoDB

```
{
"_id": 1,
"student_name": "Jasmin Scott",
  "school": {
    "school_id": 226,
    "name": "Tech Secondary",
    "address": "100 Broadway St",
    "city": "New York",
    "state": "NY",
    "zipcode": "10001"
  },
"marks": [98, 93, 95, 88, 100],
}
```

mongo

```
> db.students.find({“student_name”:
  “Jasmin Scott”})
```

SQL

students

id	name	school_id
1	Jasmin Scott	226
...	...	...

marks

id	student_id	mark
10	1	98
...	...	...

school_details

id	name	address	city	state	zipcode
226	Tech Secondary	100 Broadway St	New York	NY	10001
...	...	...	...	...	...

Results

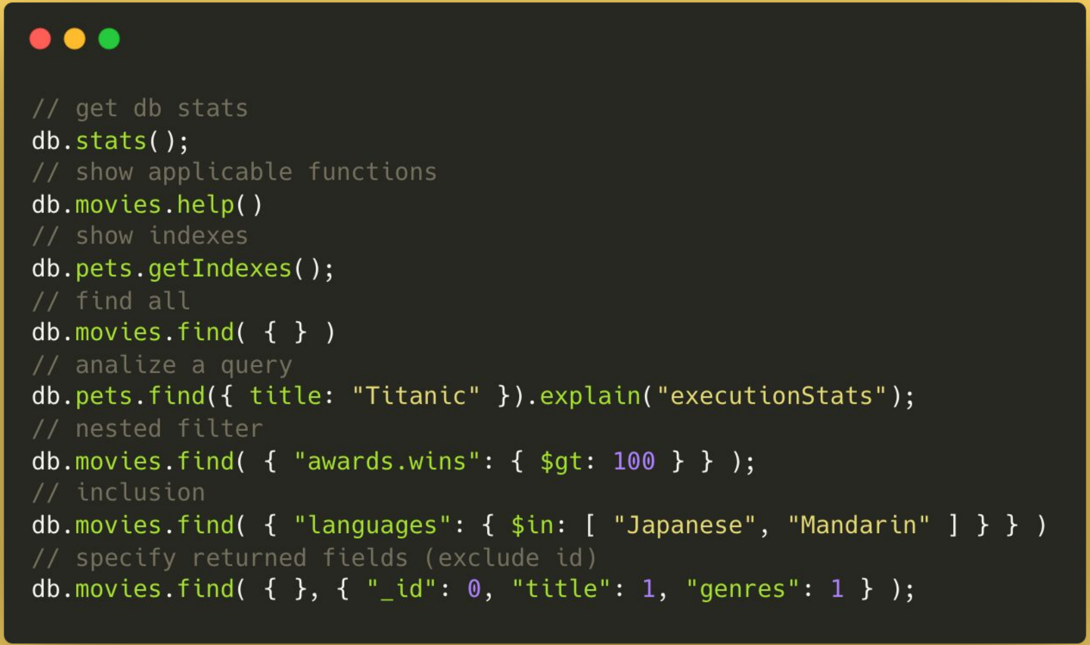
name	mark	school_name	city
Jasmin Scott	98	Tech Secondary	New York
...	...	...	...

sql

```
SELECT s.name, m.mark, d.name as "school name",
d.city
FROM students s
INNER JOIN marks m ON s.id = m.student_id
INNER JOIN school_details d ON s.school_id = d.id
WHERE s.name = "Jasmin Scott";
```

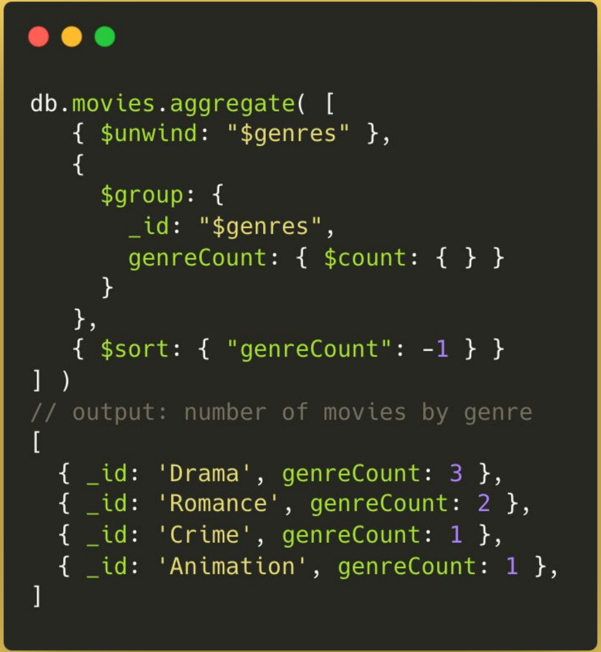
Collections = Tables
Documents = Records

BASIC STATEMENTS



```
// get db stats
db.stats();
// show applicable functions
db.movies.help()
// show indexes
db.pets.getIndexes();
// find all
db.movies.find( { } )
// analize a query
db.pets.find({ title: "Titanic" }).explain("executionStats");
// nested filter
db.movies.find( { "awards.wins": { $gt: 100 } } );
// inclusion
db.movies.find( { "languages": { $in: [ "Japanese", "Mandarin" ] } } )
// specify returned fields (exclude id)
db.movies.find( { }, { "_id": 0, "title": 1, "genres": 1 } );
```

AGGREGATE



```
db.movies.aggregate( [
   { $unwind: "$genres" },
   {
     $group: {
       _id: "$genres",
       genreCount: { $count: { } }
     }
   },
   { $sort: { "genreCount": -1 } }
] )
// output: number of movies by genre
[
  { _id: 'Drama', genreCount: 3 },
  { _id: 'Romance', genreCount: 2 },
  { _id: 'Crime', genreCount: 1 },
  { _id: 'Animation', genreCount: 1 },
]
```

Documentation

03. Key-Value Store

in-memory data structure store

store['my-key'] = 'my value'

Docs

redis

DATA TYPES

Strings
Lists
Hashes
Sets
Sorted Sets
Bitmaps
Hyperloglogs
Geospatial indexes
Streams

OPERATIONS

SET - MSET
GET -MGET
SETEX (key expiration)
APPEND (string)
INCR (numeric string)
LPUSH (list)
HGETALL (hash)

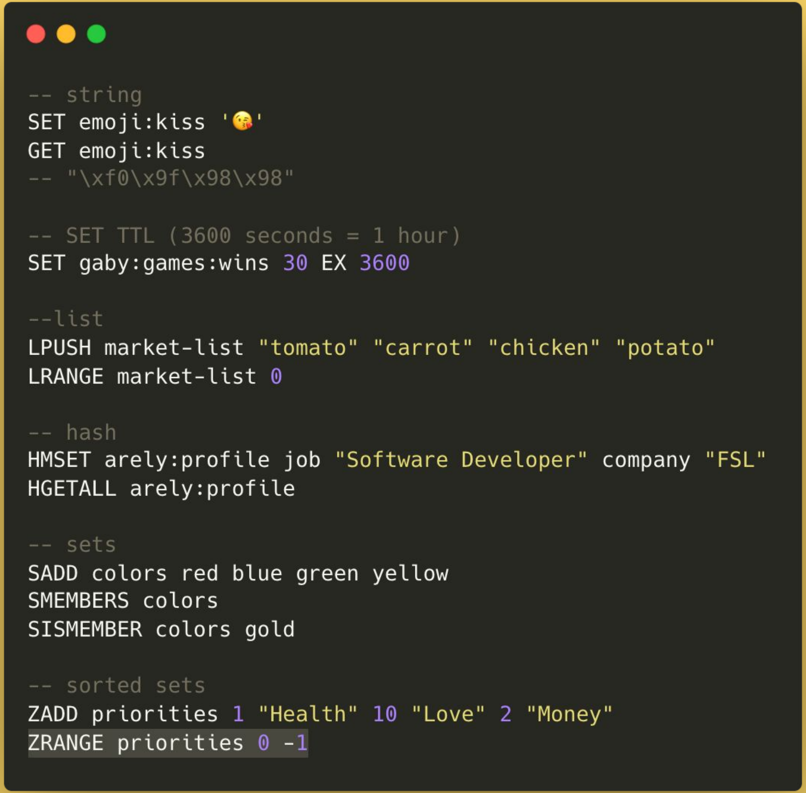
Commands...

NAMESPACES

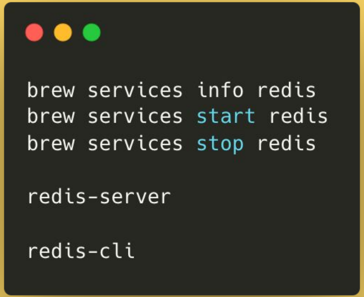
- 'user:supplier:norman'
- 'user/seller/jaime'

PUB/SUB
Transactions

BASIC COMMANDS



```
-- string
SET emoji:kiss '😘'
GET emoji:kiss
-- "\xf0\x9f\x98\x98"

-- SET TTL (3600 seconds = 1 hour)
SET gaby:games:wins 30 EX 3600

--list
LPUSH market-list "tomato" "carrot" "chicken" "potato"
LRANGE market-list 0

-- hash
HMSET arely:profile job "Software Developer" company "FSL"
HGETALL arely:profile

-- sets
SADD colors red blue green yellow
SMEMBERS colors
SISMEMBER colors gold

-- sorted sets
ZADD priorities 1 "Health" 10 "Love" 2 "Money"
ZRANGE priorities 0 -1
```



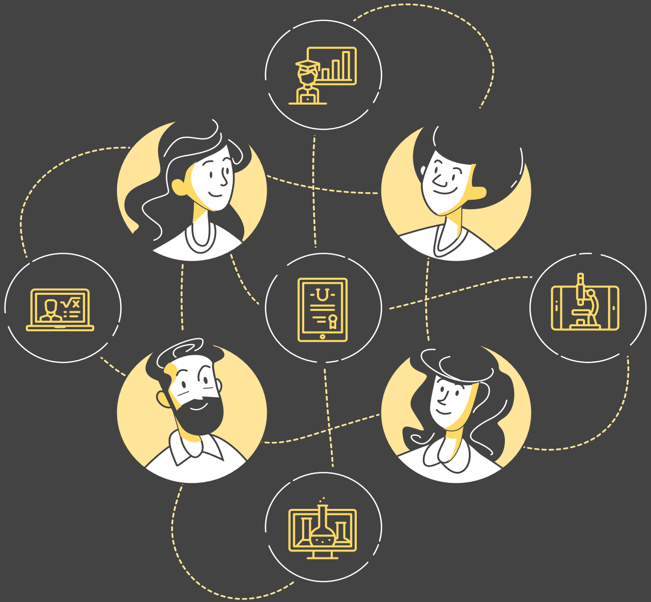
```
brew services info redis
brew services start redis
brew services stop redis

redis-server

redis-cli
```

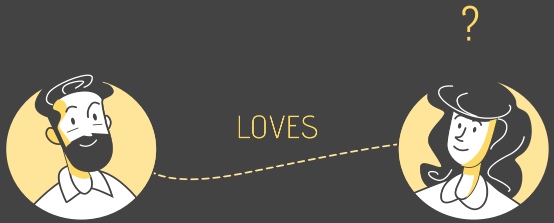
04. Graph

Graphs are the solution when **relationships** between data items are as important as the data **items themselves**.

Docs
Use Cases

CONCEPTS





|----------Node—---------|-Relationship-|--Node–|

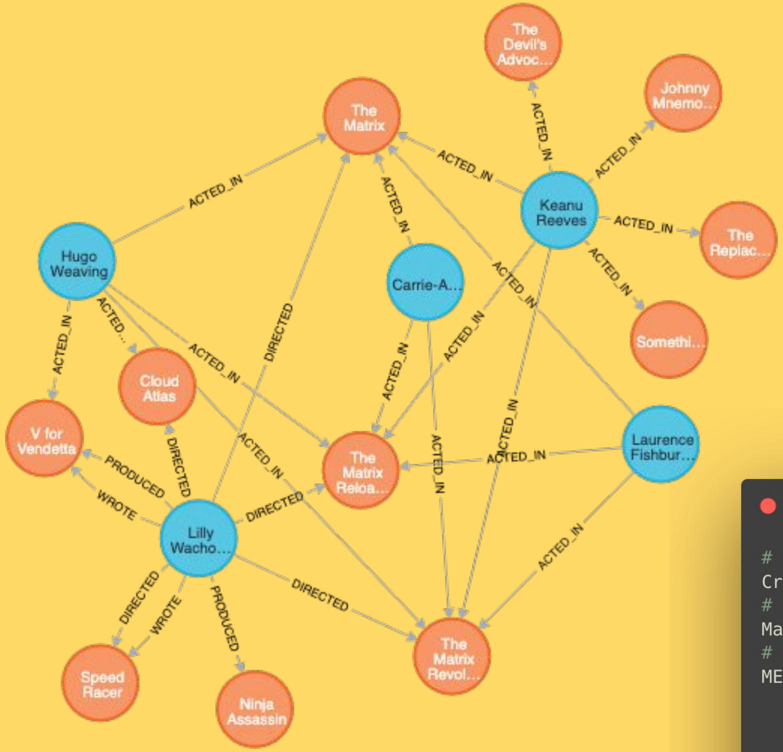
MATCH(:**Person** {name: "Rodrigo"}) -[:**LOVES**]->(**whom**) RETURN whom

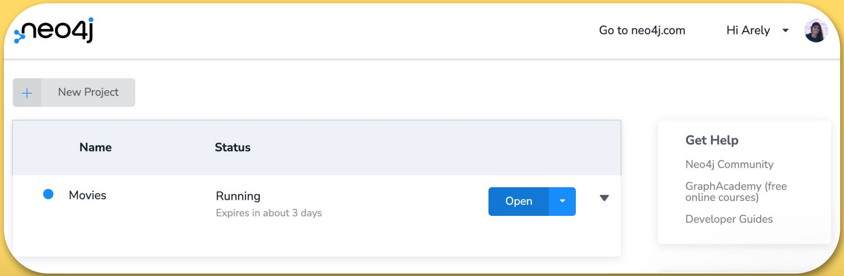
Property

Label

Variable

Neo4J





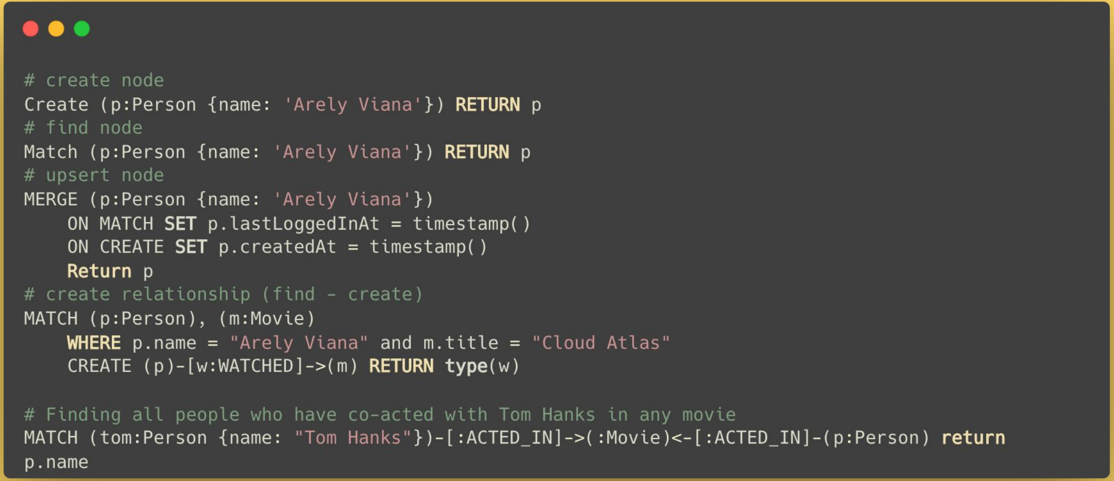
Name	Status
Movies	Running Expires in about 3 days

Get Help

Neo4j Community

GraphAcademy (free online courses)

Developer Guides



```
# create node
Create (p:Person {name: 'Arely Viana'}) RETURN p
# find node
Match (p:Person {name: 'Arely Viana'}) RETURN p
# upsert node
MERGE (p:Person {name: 'Arely Viana'})
    ON MATCH SET p.lastLoggedInAt = timestamp()
    ON CREATE SET p.createdAt = timestamp()
    Return p
# create relationship (find - create)
MATCH (p:Person), (m:Movie)
    WHERE p.name = "Arely Viana" and m.title = "Cloud Atlas"
    CREATE (p)-[w:WATCHED]->(m) RETURN type(w)

# Finding all people who have co-acted with Tom Hanks in any movie
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(p:Person) return
p.name
```

CONTACT

Github: https://github.com/areviana
LinkedIn: https://www.linkedin.com/in/areviana/

THANKS