PostgreSQL & Graph & Vector

윤명식 / MyoungSig.Youn

@MegazoneCloud

@PG.Conf 2023

Name : 윤명식
Company : 메가존클라우드
Email : myoungsig.youn@mz.co.kr
jazzlian@gmail.com
Phone. : 010-8948-9592

운이 좋아서 데이터베이스 쪽으로 일을 시작해서 현재 까지 다양한 데이터베이스를 사용한 일을 함.

지금은 그래프 데이터베이스 라는 이상한걸 하고 있고 벡터 데이터베이스에도 관심이 있음

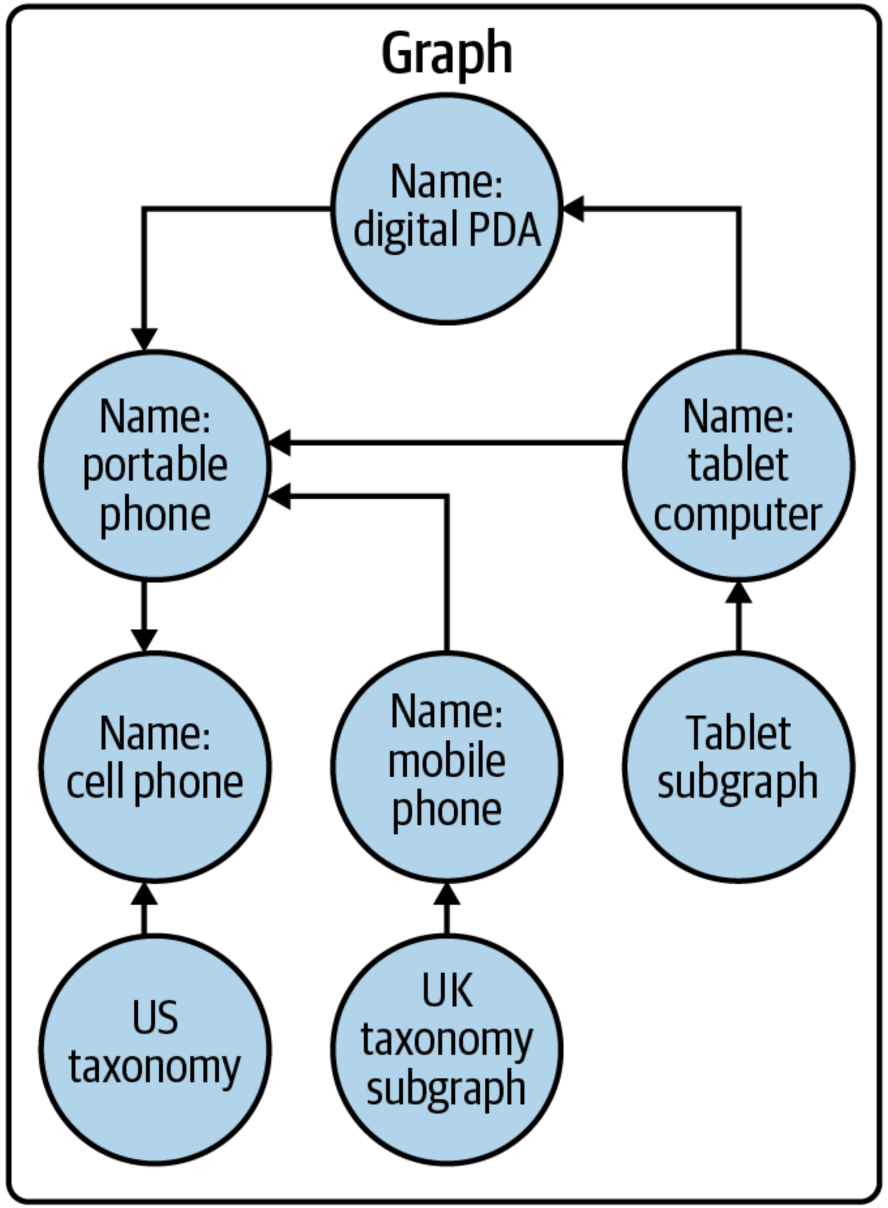
PostgreSQL + PostGIS
PostgreSQL + TimeScale
PostgreSQL + Oraface
PostgreSQL + Citus
PostgreSQL + hstore

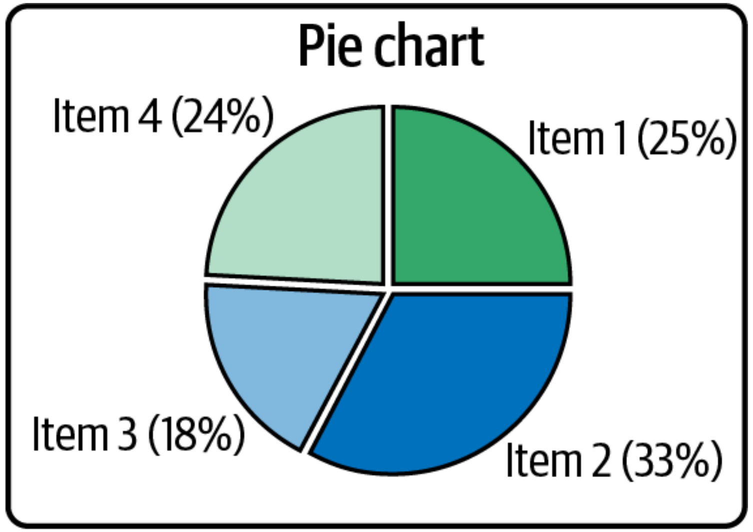
PostgreSQL + AWS
PostgreSQL + Azure
PostgreSQL + Google

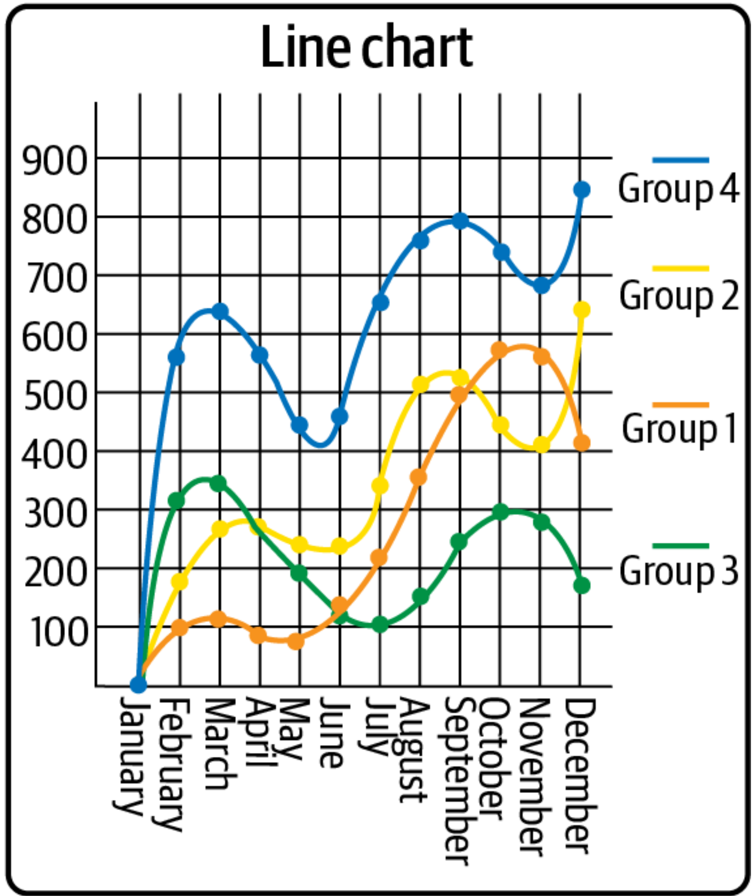
PostgreSQL + GRAPH = AGE

GRAPH

Which of the following is not the graph we are talking about?

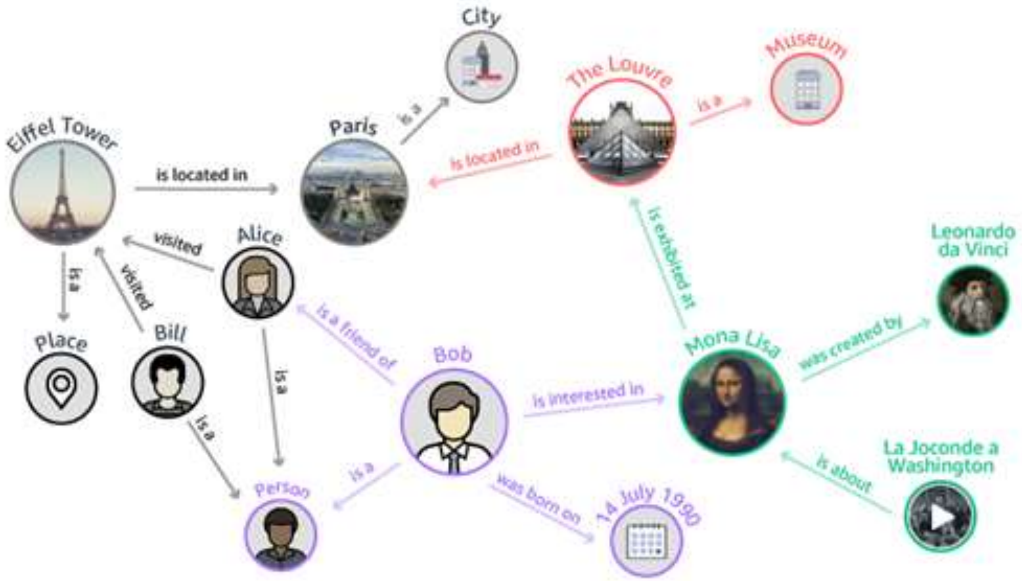


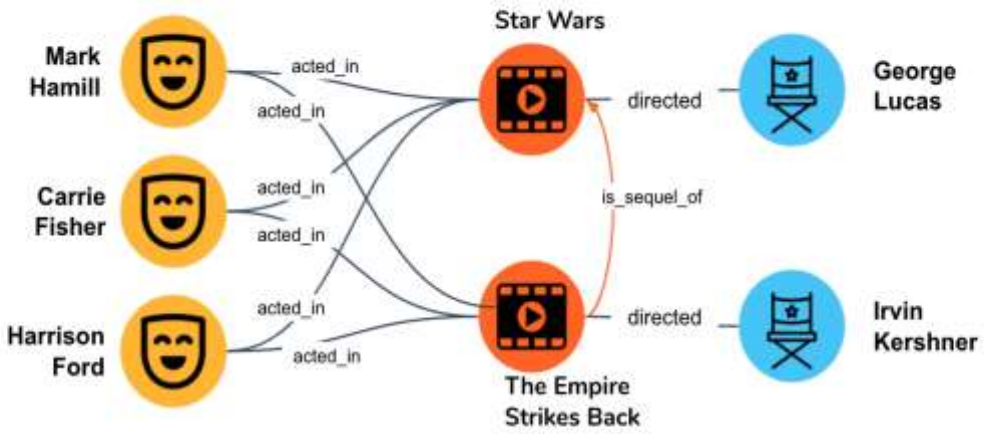




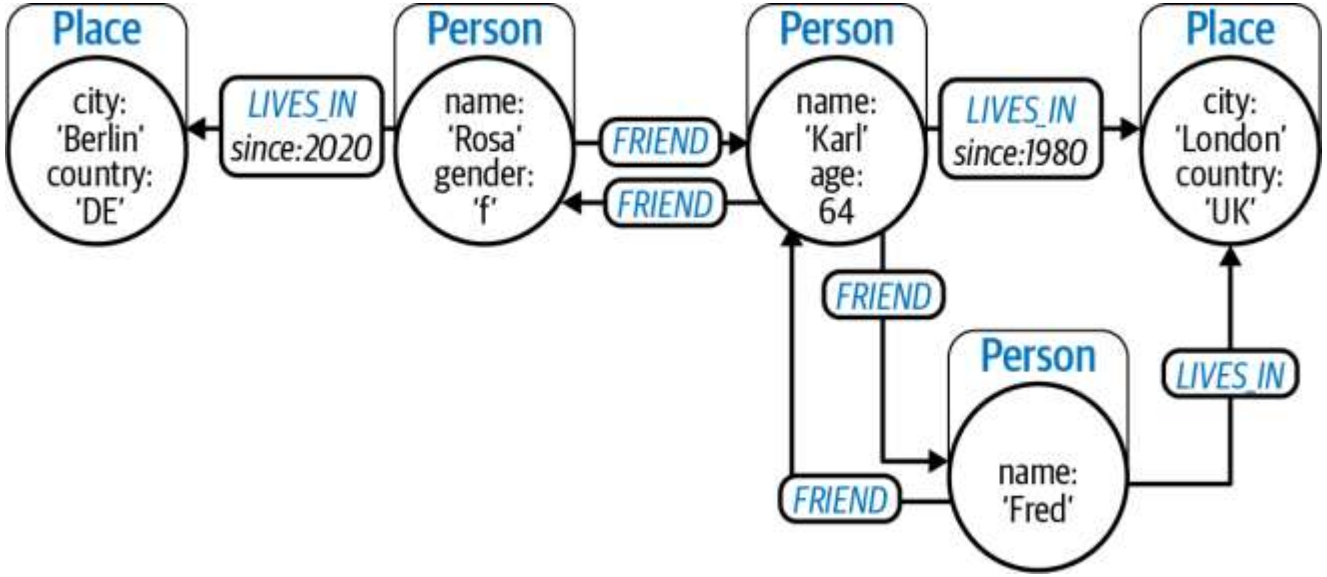
출처: Building Knowledge Graphs A Practitioners Guide

GRAPH

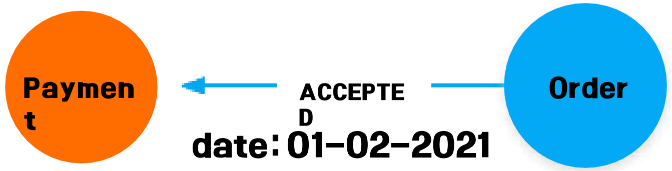




Real World Graphs



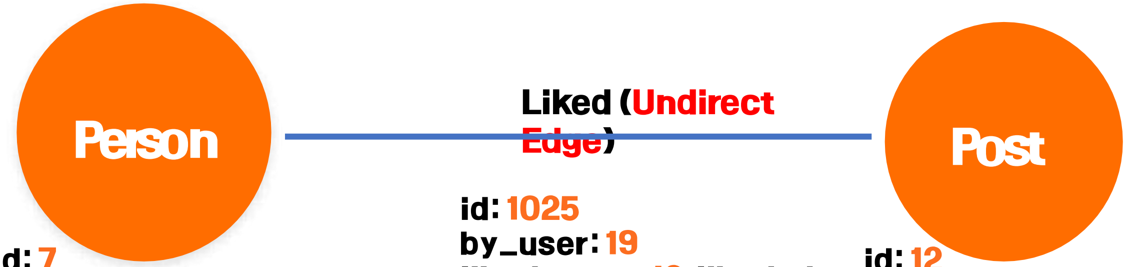
GRAPH



date: 01-02-2021

id: 0001
amount: $5

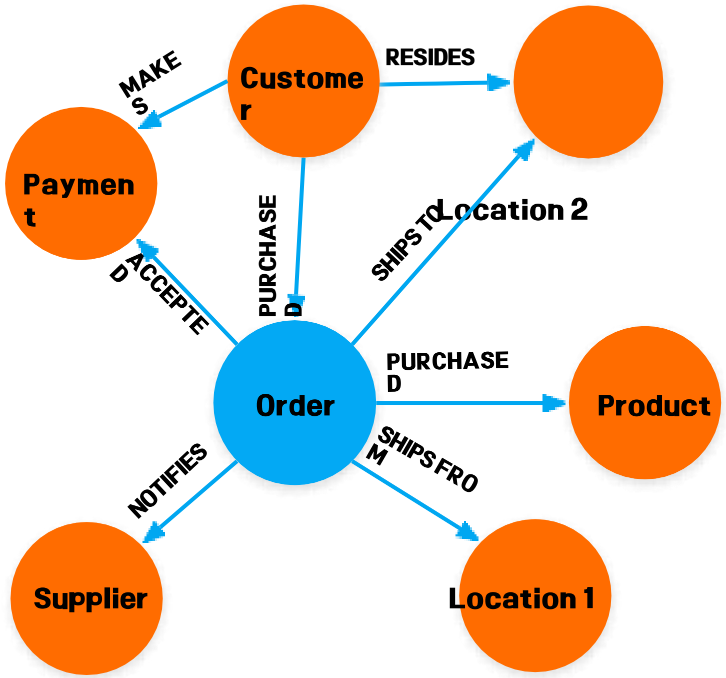
id: asdfg quanti
ty: 1



id: 7
email: vbarracks6@utexa
s.edu username: vbarrac
ks6
name: Virgie Barra
cks join_date: 20
21/09/02

id: 1025
by_user: 19
liked_post: 12 liked_da
te: 2021/07/11

id: 12
content: Suspendisse ornare ….
posted_date: 2020/07/31
posted_by: 7
deleted: False



GRAPH

Tables & Graphs

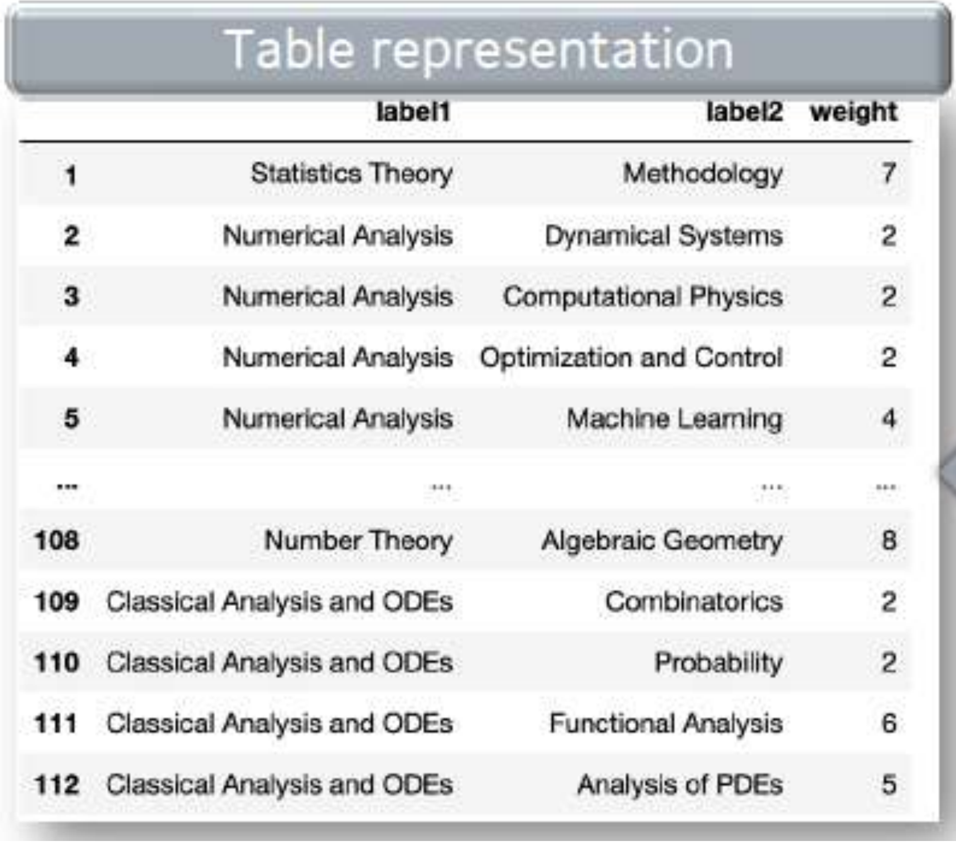
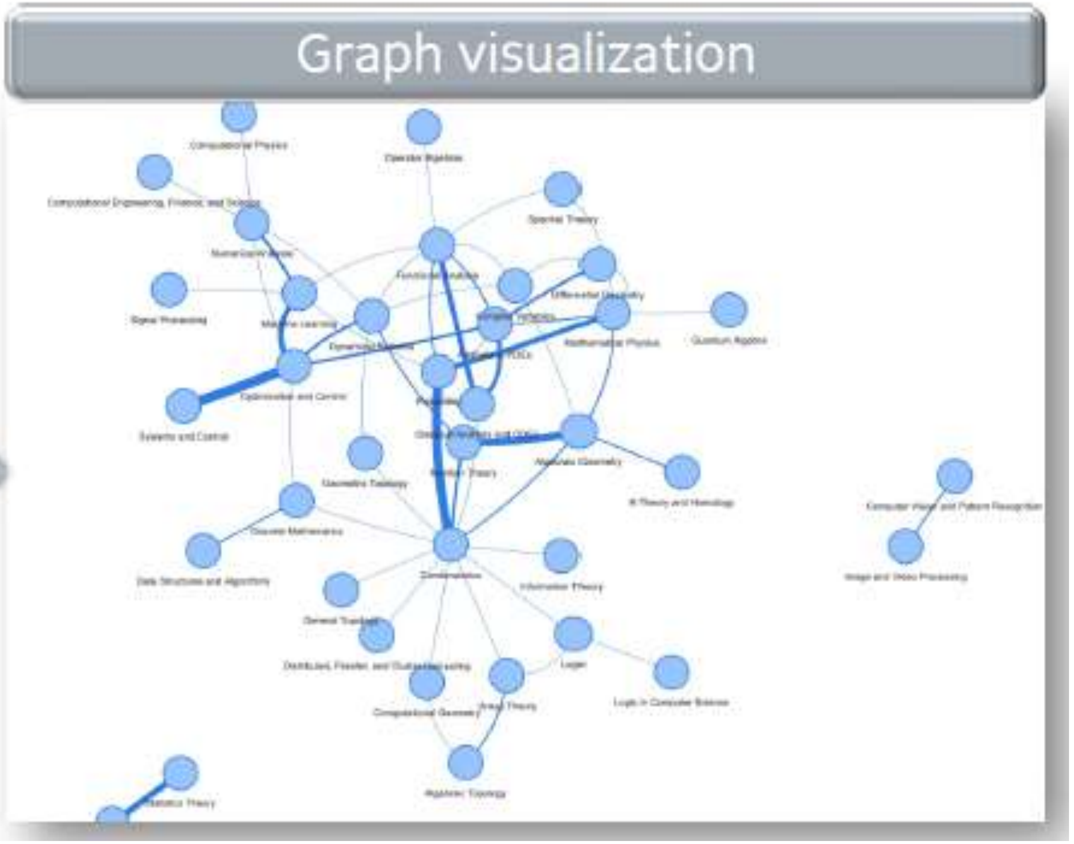


Table representation

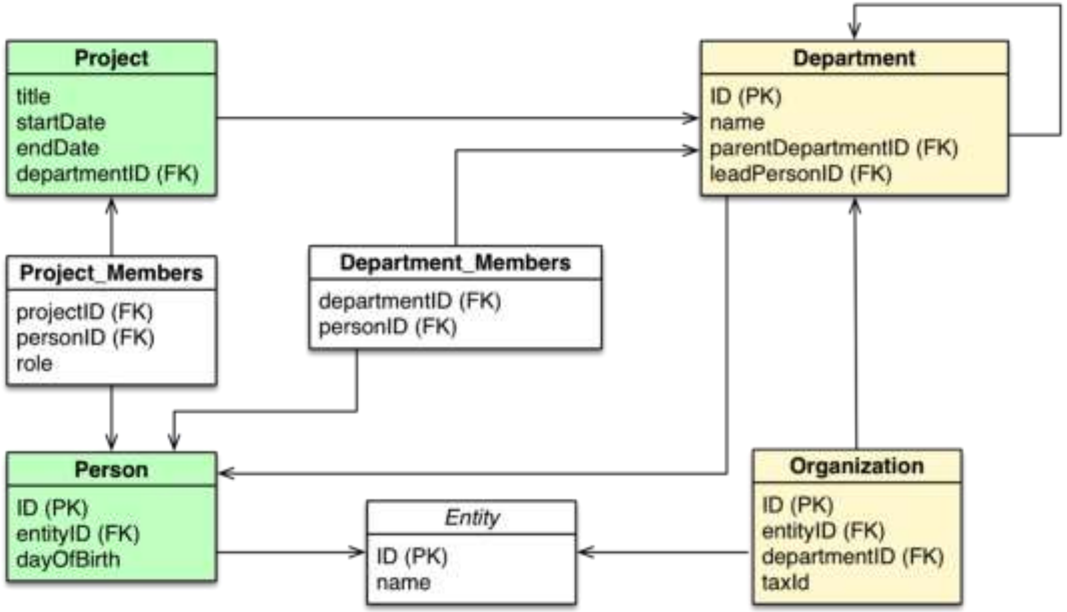
	label1	label2	weight
1	Statistics Theory	Methodology	7
2	Numerical Analysis	Dynamical Systems	2
3	Numerical Analysis	Computational Physics	2
4	Numerical Analysis	Optimization and Control	2
5	Numerical Analysis	Machine Learning	4
...	...	...	...
108	Number Theory	Algebraic Geometry	8
109	Classical Analysis and ODEs	Combinatorics	2
110	Classical Analysis and ODEs	Probability	2
111	Classical Analysis and ODEs	Functional Analysis	6
112	Classical Analysis and ODEs	Analysis of PDEs	5



Graph visualization

GRAPH

ERD



Project

title
startDate
endDate
departmentID (FK)

Department

ID (PK)
name
parentDepartmentID (FK)
leadPersonID (FK)

Project_Members

projectID (FK)
personID (FK)
role

Department_Members

departmentID (FK)
personID (FK)

Person

ID (PK)
entityID (FK)
dayOfBirth

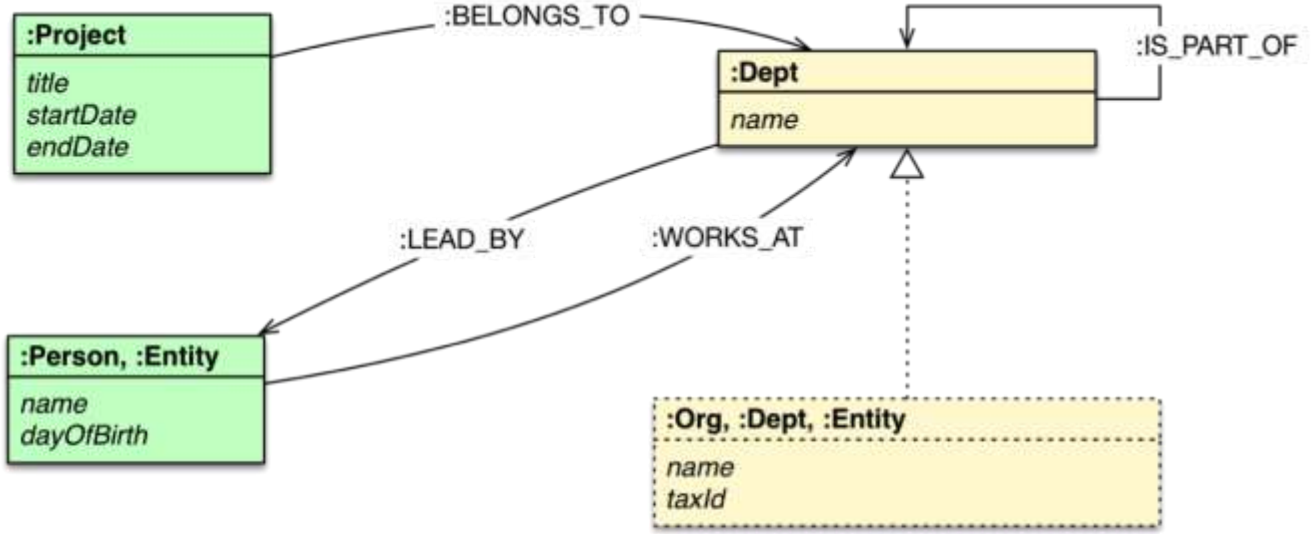
Entity

ID (PK)
name

Organization

ID (PK)
entityID (FK)
departmentID (FK)
taxId

META GRAPH



:Project

title
startDate
endDate

:BELONGS_TO

:Dept

name

:IS_PART_OF

:LEAD_BY

:WORKS_AT

:Person, :Entity

name
dayOfBirth

:Org, :Dept, :Entity

name
taxId

출처: Relational Database Modeling vs. Graph Database Modeling (neo4j.com)

IFA(Index-Free-Adjacency)

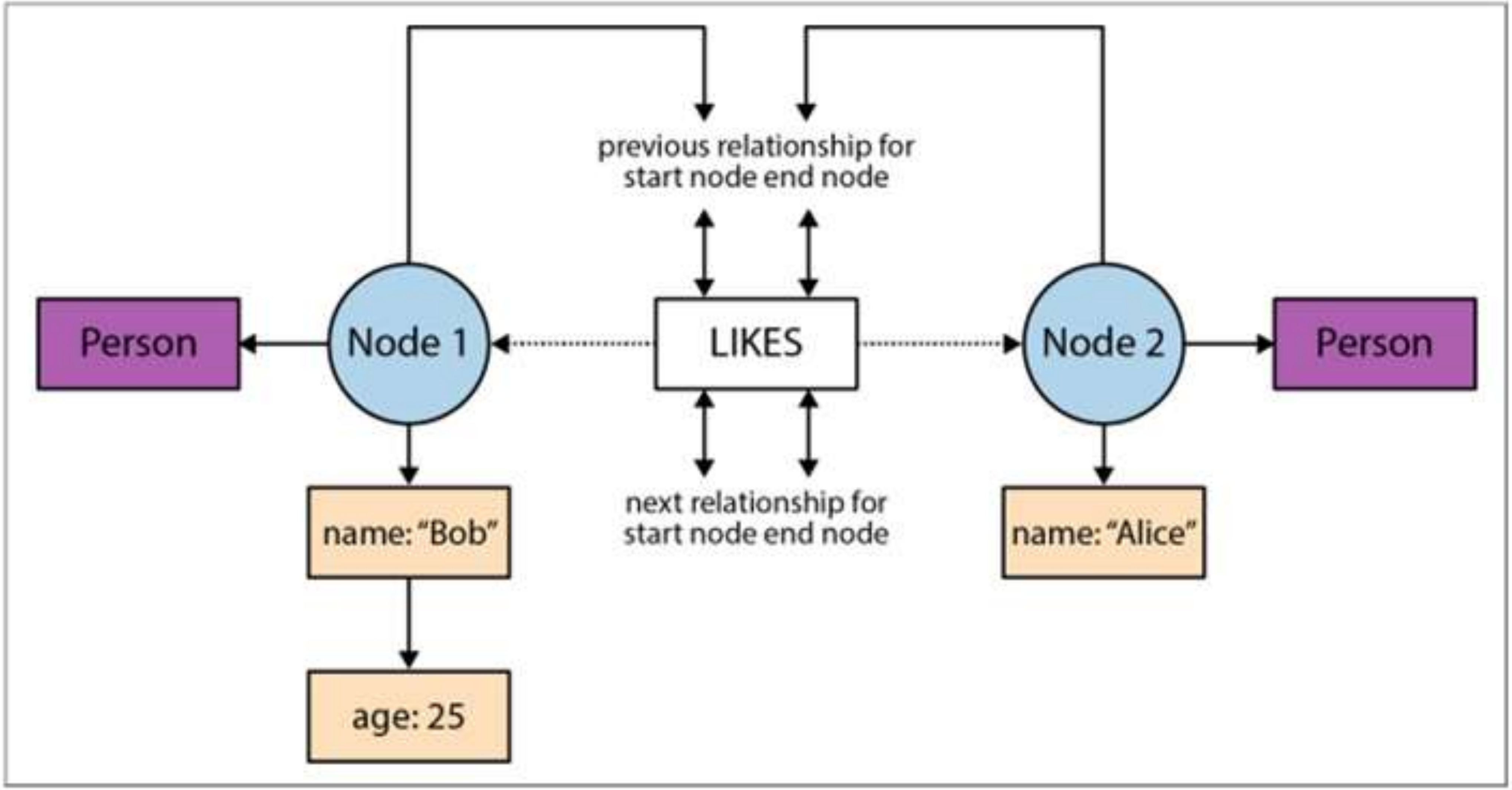
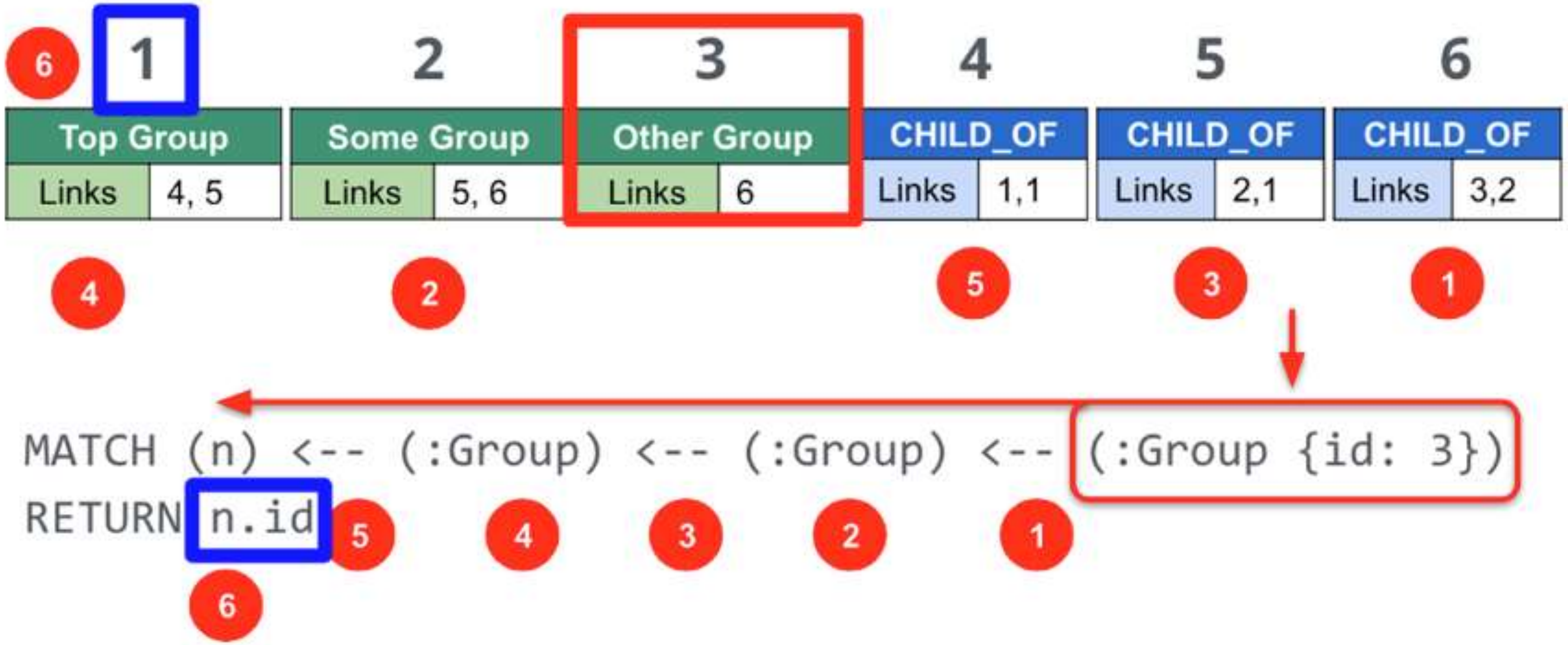


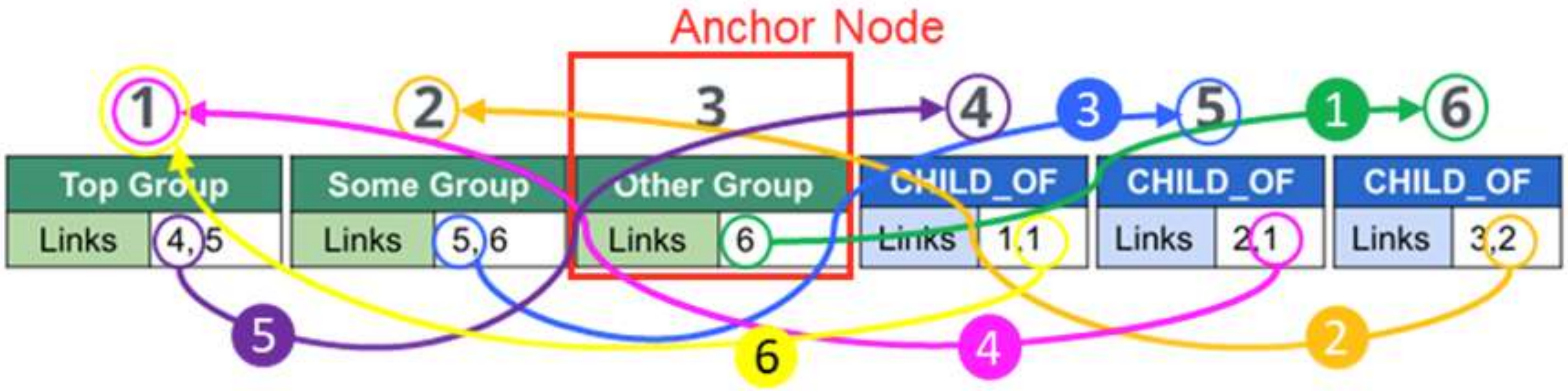
Figure 6-5. How a graph is physically stored in Neo4j

IFA(Index-Free-Adjacency)



1	2	3	4	5	6
Top Group	Some Group	Other Group	CHILD_OF	CHILD_OF	CHILD_OF
Links 4, 5	Links 5, 6	Links 6	Links 1,1	Links 2,1	Links 3,2

```
MATCH (n) <-- (:Group) <-- (:Group) <-- (:Group {id: 3})
RETURN n.id
```



ORACLE GRAPH

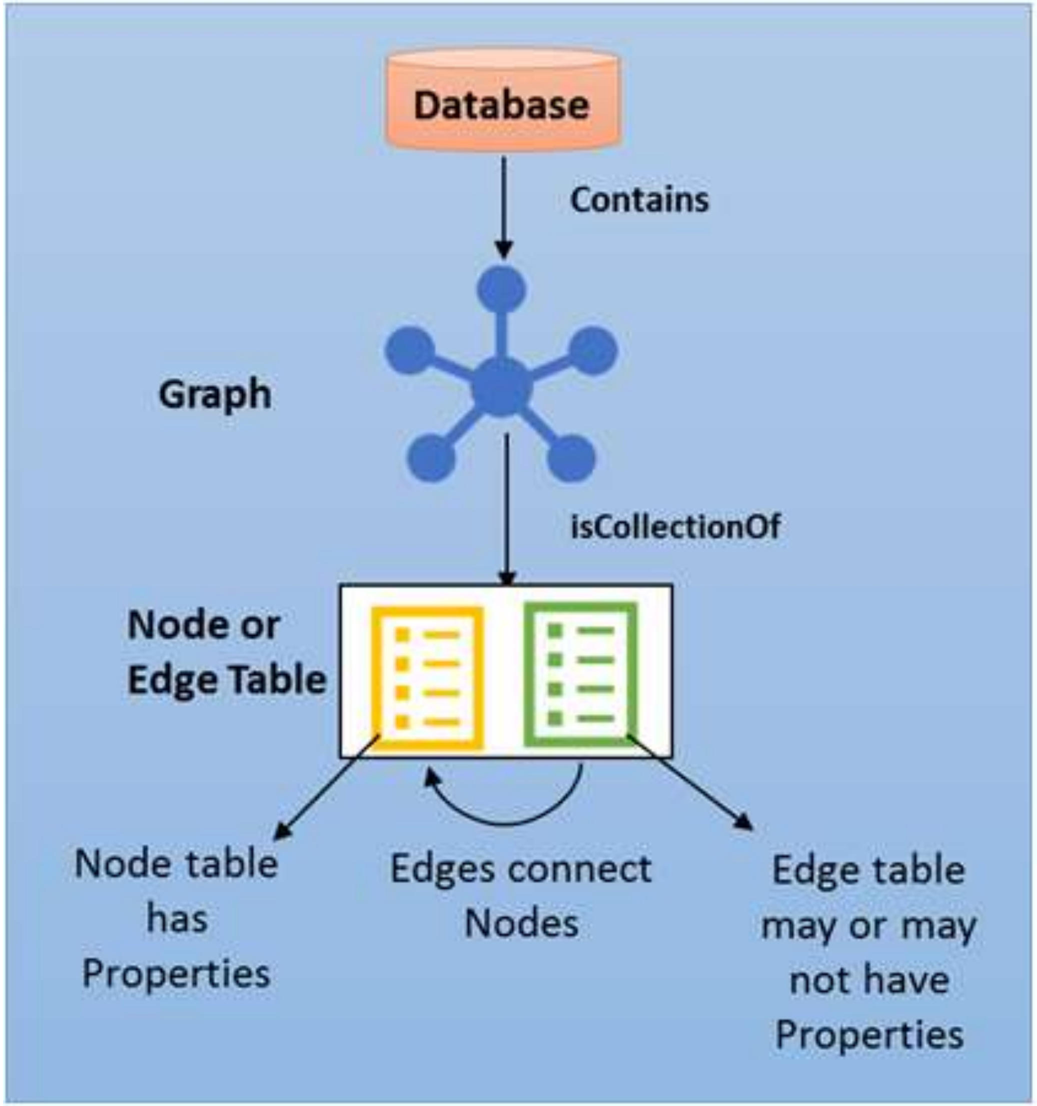
```
CREATE PROPERTY GRAPH BANK_GRAPH
   VERTEX TABLES (
     BANK_ACCOUNTS
     KEY (ID)
     PROPERTIES (ID, Name, Balance)
   )
   EDGE TABLES (
     BANK_TRANSFERS
     KEY (TXN_ID)
     SOURCE KEY (src_acct_id) REFERENCES BANK_ACCOUNTS(ID)
     DESTINATION KEY (dst_acct_id) REFERENCES BANK_ACCOUNT
S(ID)
     PROPERTIES (src_acct_id, dst_acct_id, amount)
   );
```

```
REM Check if there are any 3-hop (triangles) transfers th
at start and end at the same account
SELECT acct_id, COUNT(1) AS Num_Triangles
   FROM graph_table (BANK_GRAPH
      MATCH (src) - []->{3} (src)
      COLUMNS (src.id AS acct_id)
   ) GROUP BY acct_id ORDER BY Num_Triangles DESC;

  ACCT_ID   NUM_TRIANGLES
---------- ---------------
     918          3
     751          3
     534          3
     359          3
     119          2
     677          2
     218          2
...
118 rows selected.
```

Get started with property graphs in Oracle Database 23c Free – Developer Release

Oracle spatial and Graph

SQL Server GRAPH



```
-- Create a GraphDemo database
IF NOT EXISTS (SELECT * FROM sys.databases WHERE NAME = ‘graphdemo‘
)
  CREATE DATABASE GraphDemo;
GO

USE GraphDemo;
GO

-- Create NODE tables
CREATE TABLE Person (
 ID INTEGER PRIMARY KEY,
 name VARCHAR(100)
) AS NODE;

CREATE TABLE Restaurant (
 ID INTEGER NOT NULL,
 name VARCHAR(100),
 city VARCHAR(100)
) AS NODE;

CREATE TABLE City (
 ID INTEGER PRIMARY KEY,
 name VARCHAR(100),
 stateName VARCHAR(100)
) AS NODE;

-- Create EDGE tables.
CREATE TABLE likes (rating INTEGER) AS EDGE;
CREATE TABLE friendOf AS EDGE;
CREATE TABLE livesIn AS EDGE;
CREATE TABLE locatedIn AS EDGE;
```

출처: https://learn.microsoft.com/ko-kr/sql/relational-databases/graphs/sql-graph-architecture?view=sql-server-ver16

SQL Server GRAPH

```
-- Find Restaurants that John likes
SELECT Restaurant.name
FROM Person, likes, Restaurant
WHERE MATCH (Person-(likes)->Restaurant)
AND Person.name = ‘John‘;

-- Find Restaurants that John‘s friends like
SELECT Restaurant.name
FROM Person person1, Person person2, likes, friendOf, Restaurant
WHERE MATCH(person1-(friendOf)->person2-(likes)->Restaurant)
AND person1.name=‘John‘;

-- Find people who like a restaurant in the same city they live in
SELECT Person.name
FROM Person, likes, Restaurant, livesIn, City, locatedIn
WHERE MATCH (Person-(likes)->Restaurant-(locatedIn)->City AND Person-(livesIn)->City);
```

```
-- Find friends-of-friends-of-friends, excluding those cases where the relationship “loops back“.
-- For example, Alice is a friend of John; John is a friend of Mary; and Mary in turn is a friend of Alice.
-- This causes a “loop“ back to Alice. In many cases, it is necessary to explicitly check for such loops and exclude the results.
SELECT CONCAT(Person.name, ‘->‘, Person2.name, ‘->‘, Person3.name, ‘->‘, Person4.name)
FROM Person, friendOf, Person as Person2, friendOf as friendOffriend, Person as Person3, friendOf as friendOffriendOfFriend, Person as Person4
WHERE MATCH (Person-(friendOf)->Person2-(friendOffriend)->Person3-(friendOffriendOfFriend)->Person4)
AND Person2.name != Person.name
AND Person3.name != Person2.name
AND Person4.name != Person3.name
AND Person.name != Person4.name;
```

PostgreSQL + Apache AGE(BITNINE)

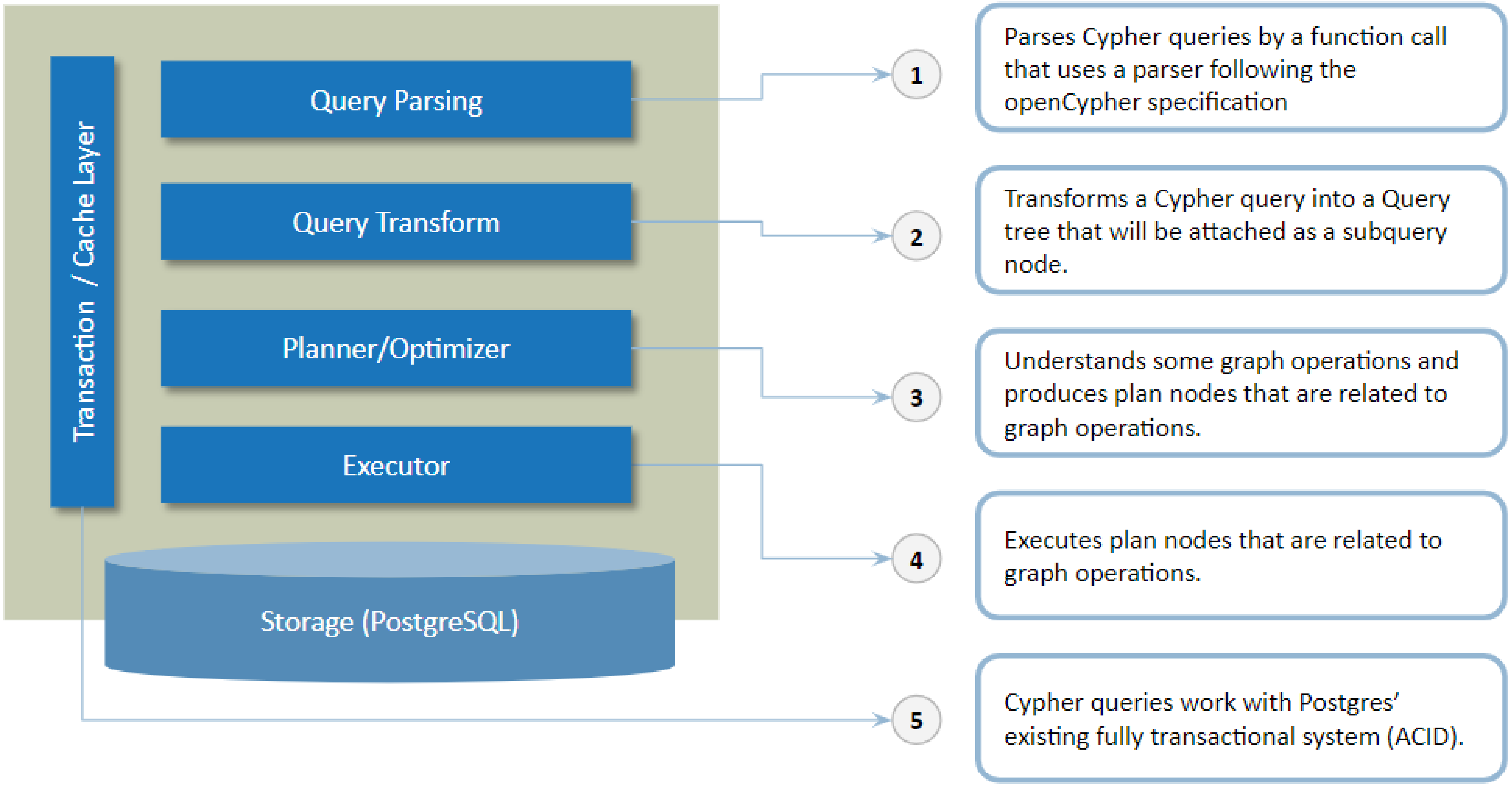
- Graph Database Plugin for PostgreSQL
- Hybrid Queries (OpenCypher And SQL)
- Fast Graph Query Processing
- Graph Visualization and Analytics
- Current PG15 support

https://age.apache.org/

```
CREATE EXTENSION age;

LOAD ‘age’;

SET search_path = ag_catalog, “$user“, public;

SELECT create_graph(‘graph_name’);

SELECT *
FROM cypher(‘graph_name‘, $$
   CREATE (:label {property:value})
$$) as (v agtype);

SELECT *
FROM cypher(‘graph_name‘, $$
   MATCH (v)
   RETURN v
$$) as (v agtype);

SELECT *
FROM cypher(‘graph_name‘, $$
   MATCH (a:Person), (b:Person)
   WHERE a.name = ‘Node A‘ AND b.name = ‘Node B‘
   CREATE (a)-[e:RELTYPE {name:a.name + ‘(-)‘ + b.name}]->(b)
   RETURN e
$$) as (e agtype);
```

Architecture

AGE Architecture



Transaction / Cache Layer

Query Parsing

Query Transform

Planner/Optimizer

Executor

Storage (PostgreSQL)

1 Parses Cypher queries by a function call that uses a parser following the openCypher specification

2 Transforms a Cypher query into a Query tree that will be attached as a subquery node.

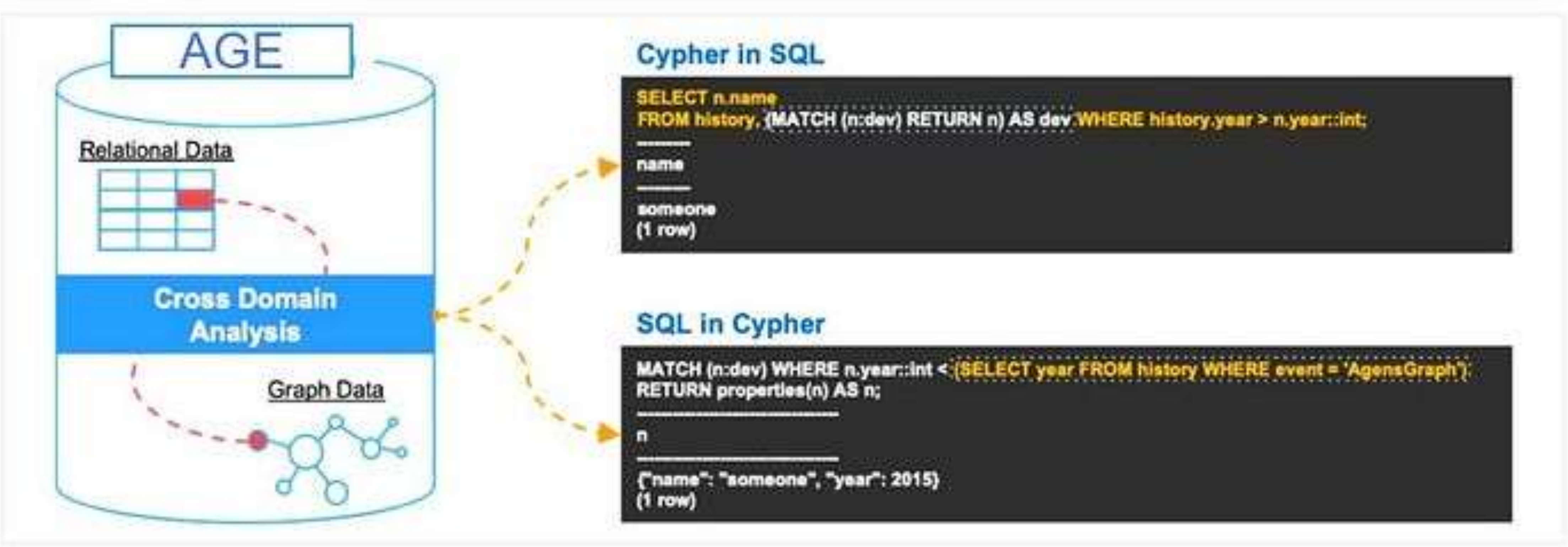
3 Understands some graph operations and produces plan nodes that are related to graph operations.

4 Executes plan nodes that are related to graph operations.

5 Cypher queries work with Postgres' existing fully transactional system (ACID).

이미지 출처: Apache AGE 공식 문서 홈페이지

Architecture



Cypher in SQL

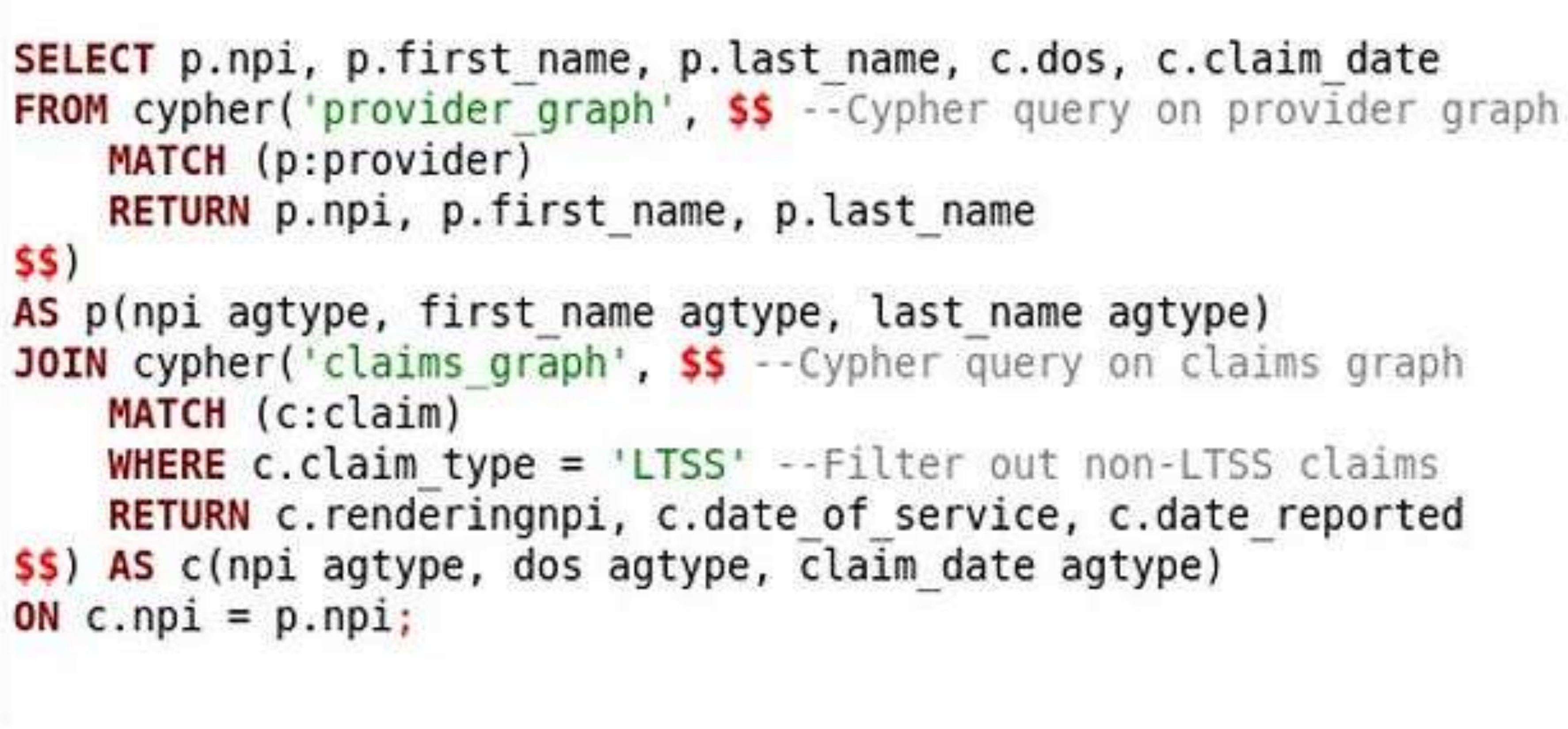
```
SELECT n.name
FROM history, (MATCH (n:dev) RETURN n) AS dev WHERE history.year > n.year::int;
--------
name
--------
someone
(1 row)
```

SQL in Cypher

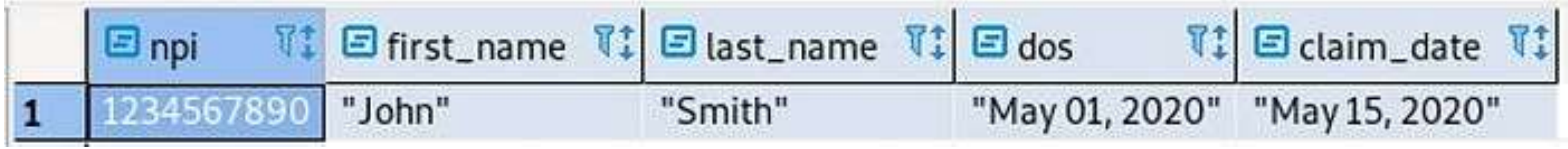
```
MATCH (n:dev) WHERE n.year::int < (SELECT year FROM history WHERE event = 'AgensGraph')
RETURN properties(n) AS n;
------------------------------
n
------------------------------
{"name": "someone", "year": 2015}
(1 row)
```

이미지 출처: [Postgres Build 2020] Apache AGE, the Extension of PostgreSQL | by Apache AGE | Medium

Architecture



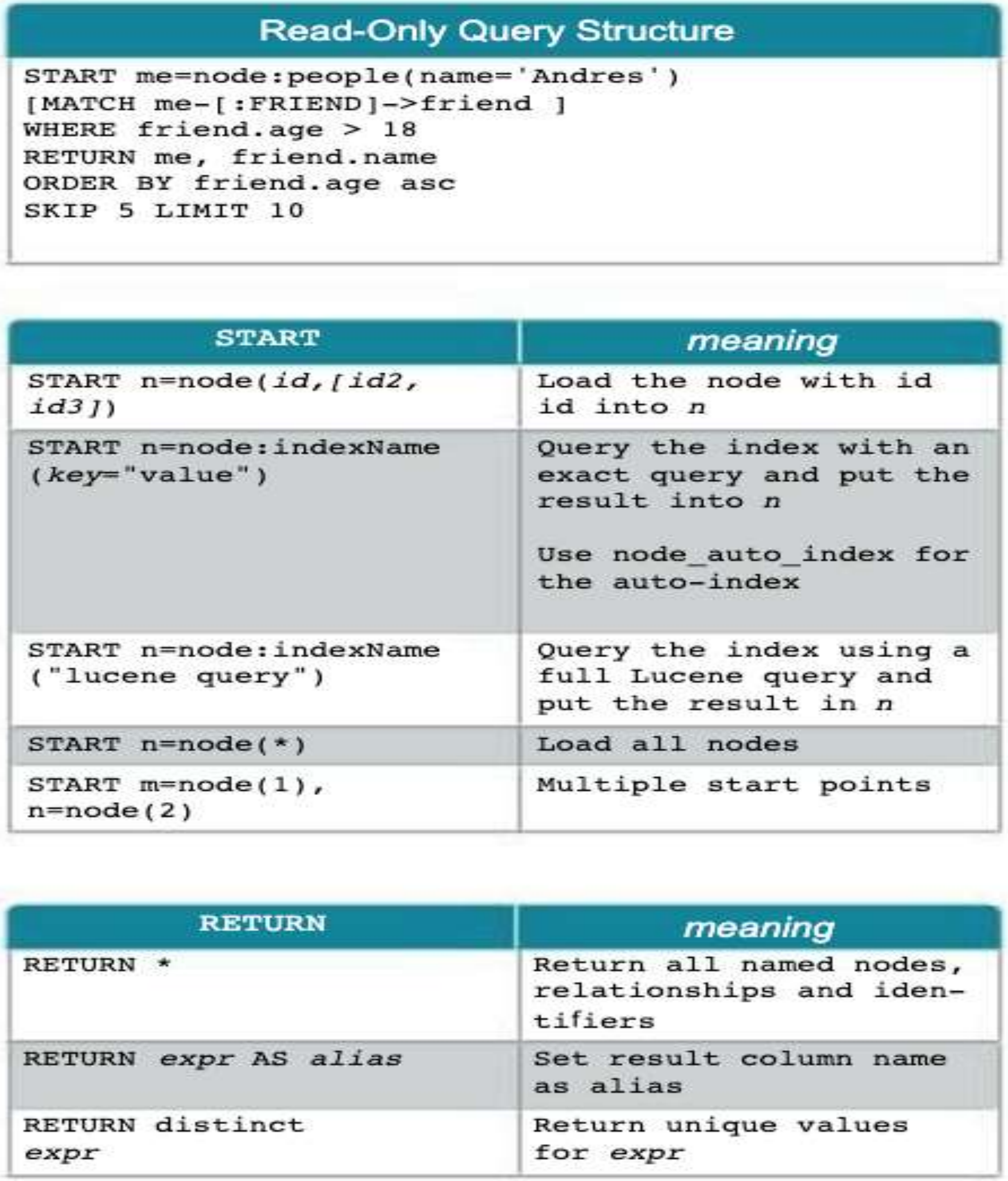
```
SELECT p.npi, p.first_name, p.last_name, c.dos, c.claim_date
FROM cypher('provider_graph', $$ --Cypher query on provider graph
    MATCH (p:provider)
    RETURN p.npi, p.first_name, p.last_name
$$)
AS p(npi agtype, first_name agtype, last_name agtype)
JOIN cypher('claims_graph', $$ --Cypher query on claims graph
    MATCH (c:claim)
    WHERE c.claim_type = 'LTSS' --Filter out non-LTSS claims
    RETURN c.renderingnpi, c.date_of_service, c.date_reported
$$) AS c(npi agtype, dos agtype, claim_date agtype)
ON c.npi = p.npi;
```



	npi	first_name	last_name	dos	claim_date
1	1234567890	"John"	"Smith"	"May 01, 2020"	"May 15, 2020"

이미지 출처: [Postgres Build 2020] Apache AGE, the Extension of PostgreSQL | by Apache AGE | Medium

Cypher Cheat Sheet

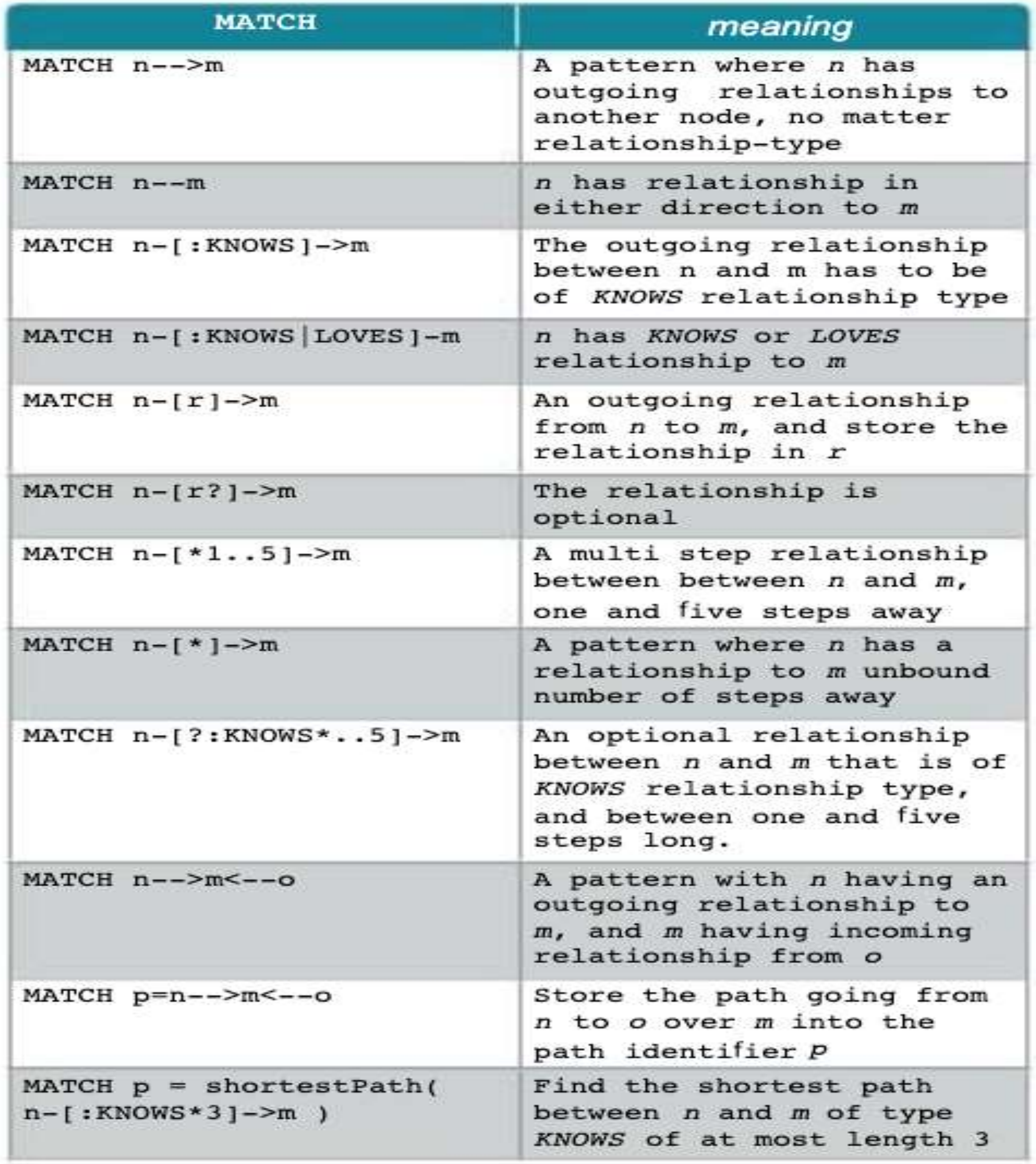


Read-Only Query Structure

```
START me=node:people(name='Andres')
[MATCH me-[:FRIEND]->friend ]
WHERE friend.age > 18
RETURN me, friend.name
ORDER BY friend.age asc
SKIP 5 LIMIT 10
```

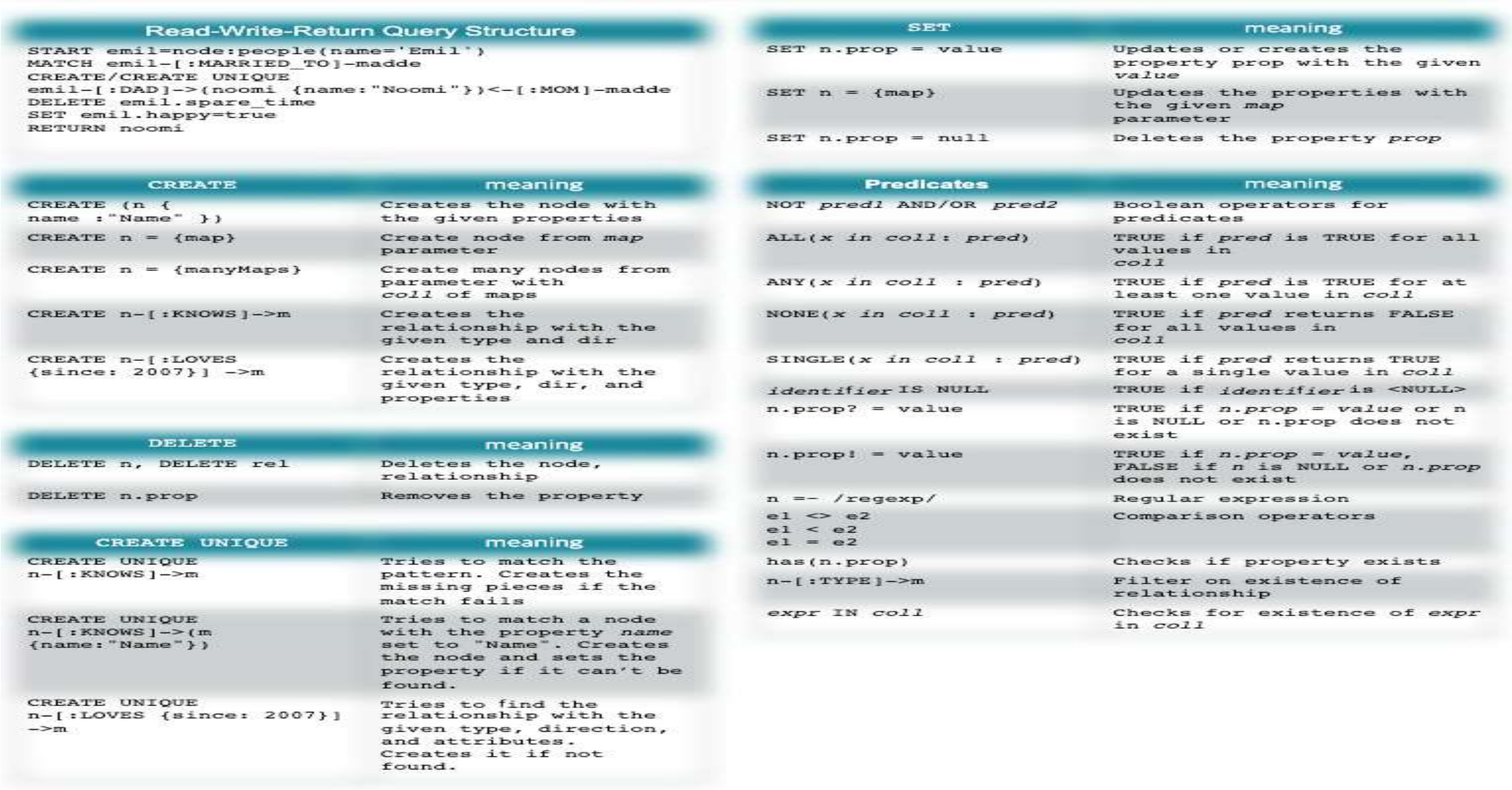
START	*meaning*
START n=node(*id*,*[id2, id3]*)	Load the node with id *id* into *n*
START n=node:indexName (*key*="value")	Query the index with an exact query and put the result into *n* Use node_auto_index for the auto-index
START n=node:indexName ("lucene query")	Query the index using a full Lucene query and put the result in *n*
START n=node(*)	Load all nodes
START m=node(1), n=node(2)	Multiple start points

RETURN	*meaning*
RETURN *	Return all named nodes, relationships and identifiers
RETURN *expr* AS *alias*	Set result column name as alias
RETURN distinct *expr*	Return unique values for *expr*



MATCH	*meaning*
MATCH n-->m	A pattern where *n* has outgoing relationships to another node, no matter relationship-type
MATCH n--m	*n* has relationship in either direction to *m*
MATCH n-[:KNOWS]->m	The outgoing relationship between n and m has to be of *KNOWS* relationship type
MATCH n-[:KNOWS|LOVES]-m	*n* has *KNOWS* or *LOVES* relationship to *m*
MATCH n-[r]->m	An outgoing relationship from *n* to *m*, and store the relationship in *r*
MATCH n-[r?]->m	The relationship is optional
MATCH n-[*1..5]->m	A multi step relationship between between *n* and *m*, one and five steps away
MATCH n-[*]->m	A pattern where *n* has a relationship to *m* unbound number of steps away
MATCH n-[?:KNOWS*..5]->m	An optional relationship between *n* and *m* that is of *KNOWS* relationship type, and between one and five steps long.
MATCH n-->m<--o	A pattern with *n* having an outgoing relationship to *m*, and *m* having incoming relationship from *o*
MATCH p=n-->m<--o	Store the path going from *n* to *o* over *m* into the path identifier *p*
MATCH p = shortestPath(n-[:KNOWS*3]->m)	Find the shortest path between *n* and *m* of type *KNOWS* of at most length 3

Cypher Cheat Sheet



Read-Write-Return Query Structure

```
START emil=node:people(name='Emil')
MATCH emil-[:MARRIED_TO]-madde
CREATE/CREATE UNIQUE
emil-[:DAD]->(noomi {name:"Noomi"})<-[:MOM]-madde
DELETE emil.spare_time
SET emil.happy=true
RETURN noomi
```

CREATE	meaning
CREATE (n { name :"Name" })	Creates the node with the given properties
CREATE n = {map}	Create node from *map* parameter
CREATE n = {manyMaps}	Create many nodes from parameter with *coll* of maps
CREATE n-[:KNOWS]->m	Creates the relationship with the given type and dir
CREATE n-[:LOVES {since: 2007}] ->m	Creates the relationship with the given type, dir, and properties

DELETE	meaning
DELETE n, DELETE rel	Deletes the node, relationship
DELETE n.prop	Removes the property

CREATE UNIQUE	meaning
CREATE UNIQUE n-[:KNOWS]->m	Tries to match the pattern. Creates the missing pieces if the match fails
CREATE UNIQUE n-[:KNOWS]->(m {name:"Name"})	Tries to match a node with the property *name* set to "Name". Creates the node and sets the property if it can't be found.
CREATE UNIQUE n-[:LOVES {since: 2007}] ->m	Tries to find the relationship with the given type, direction, and attributes. Creates it if not found.

SET	meaning
SET n.prop = value	Updates or creates the property prop with the given *value*
SET n = {map}	Updates the properties with the given *map* parameter
SET n.prop = null	Deletes the property *prop*

Predicates	meaning
NOT *pred1* AND/OR *pred2*	Boolean operators for predicates
ALL(*x in coll: pred*)	TRUE if *pred* is TRUE for all values in *coll*
ANY(*x in coll : pred*)	TRUE if *pred* is TRUE for at least one value in *coll*
NONE(*x in coll : pred*)	TRUE if *pred* returns FALSE for all values in *coll*
SINGLE(*x in coll : pred*)	TRUE if *pred* returns TRUE for a single value in *coll*
identifier IS NULL	TRUE if *identifier* is <NULL>
n.prop? = value	TRUE if *n.prop = value* or n is NULL or n.prop does not exist
n.prop! = value	TRUE if *n.prop = value*, FALSE if *n* is NULL or *n.prop* does not exist
n =~ /regexp/	Regular expression
e1 <> e2 e1 < e2 e1 = e2	Comparison operators
has(n.prop)	Checks if property exists
n-[:TYPE]->m	Filter on existence of relationship
expr IN *coll*	Checks for existence of *expr* in *coll*

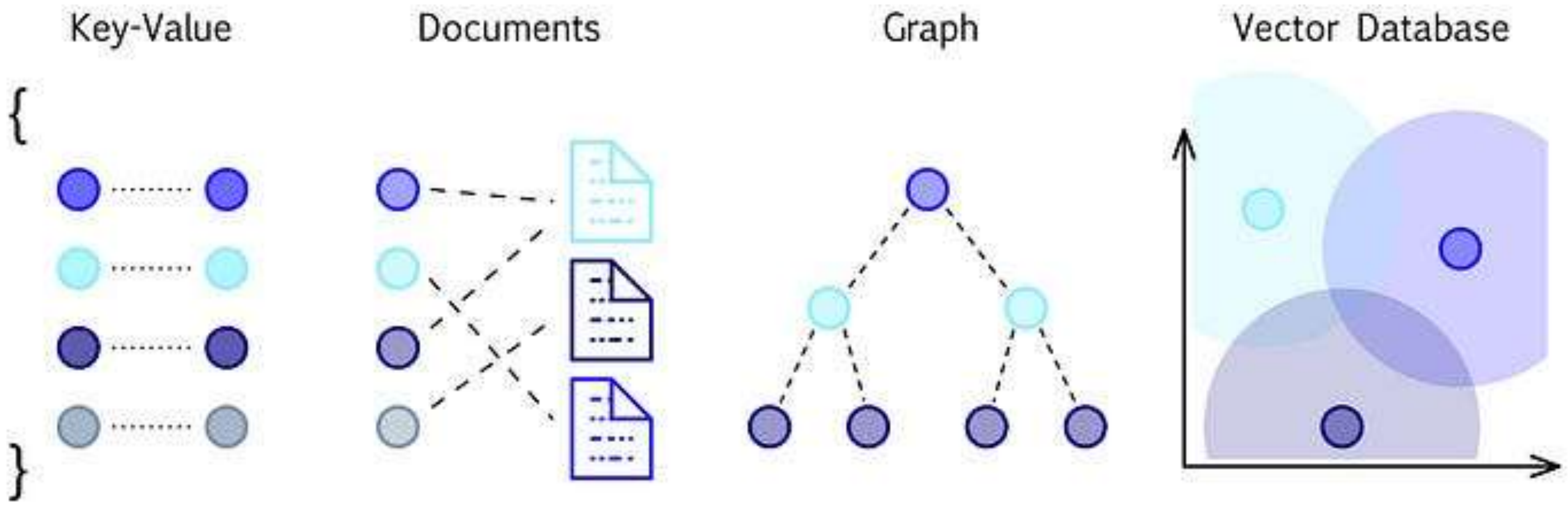
Demo

1. Setup — Apache AGE master documentation

2. Importing graph from files — Apache AGE master documentation

PostgreSQL + Vector = pgVector

VECTOR



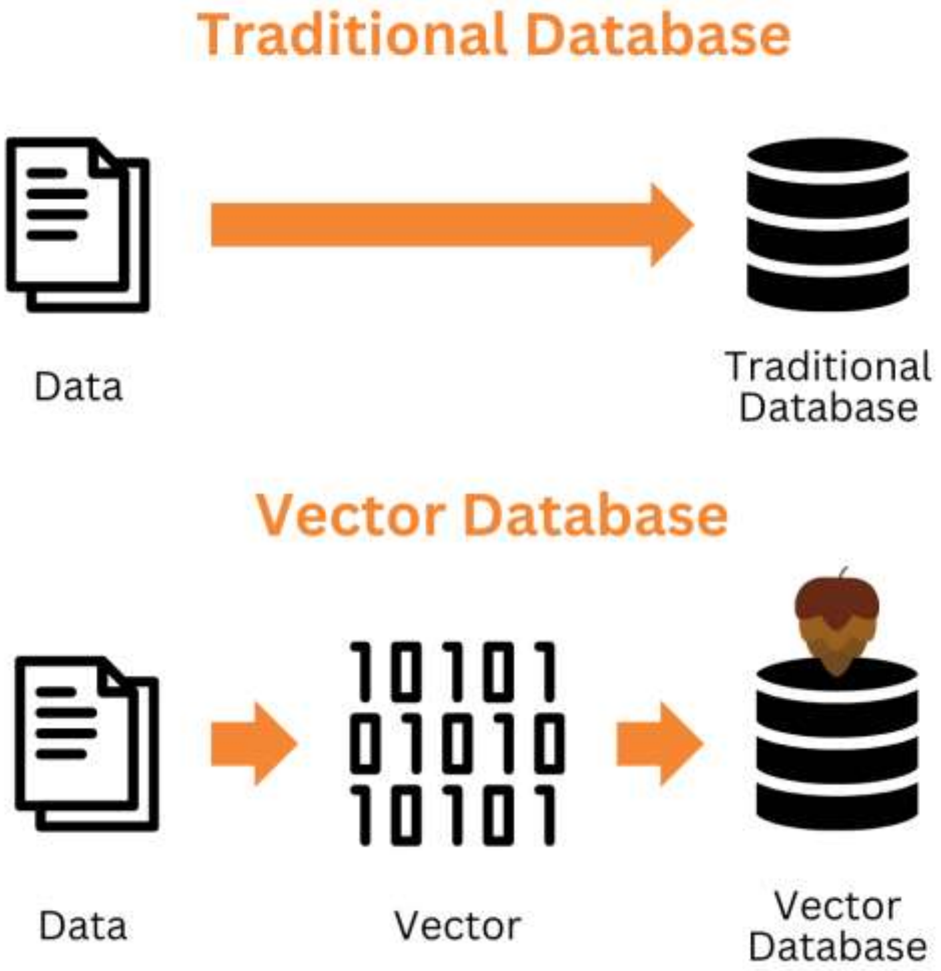
Vector

A vector is a mathematical representation of data that descri bes objects based on different characteristics or qualities.

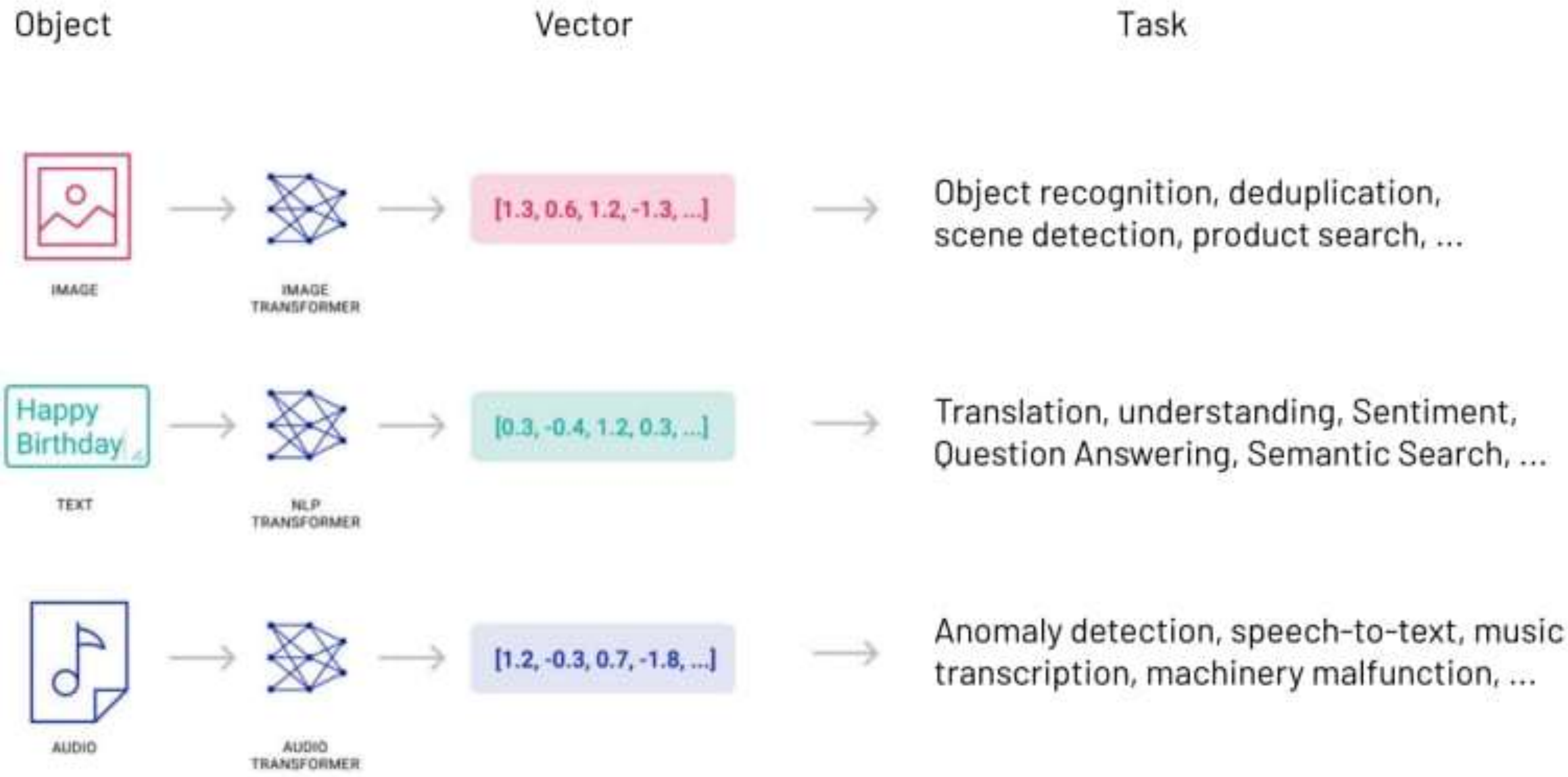
VECTOR

Vector Database

Vector databases are specialized databases for storing and querying large amounts of high–dimensional data optimized in vector form.

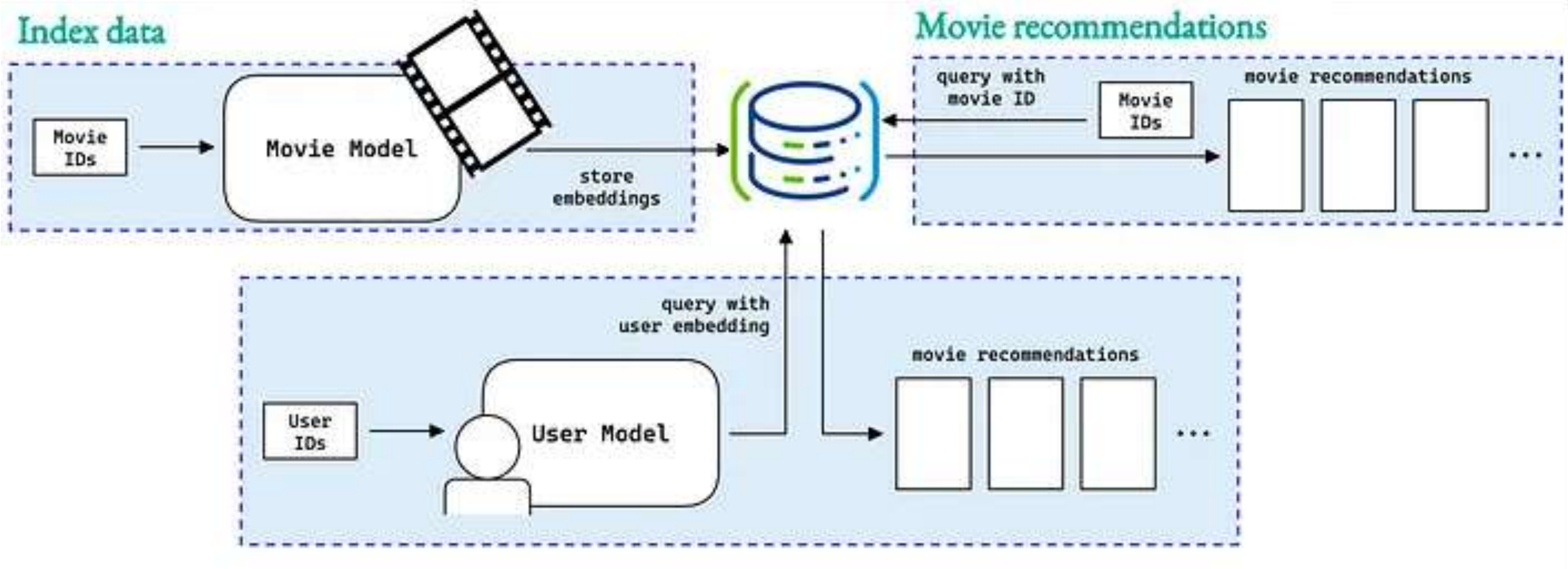


VECTOR


Object
Vector
Task
IMAGE
IMAGE TRANSFORMER
[1.3, 0.6, 1.2, -1.3, ...]
Object recognition, deduplication, scene detection, product search, ...
Happy Birthday
TEXT
NLP TRANSFORMER
[0.3, -0.4, 1.2, 0.3, ...]
Translation, understanding, Sentiment, Question Answering, Semantic Search, ...
AUDIO
AUDIO TRANSFORMER
[1.2, -0.3, 0.7, -1.8, ...]
Anomaly detection, speech-to-text, music transcription, machinery malfunction, ...

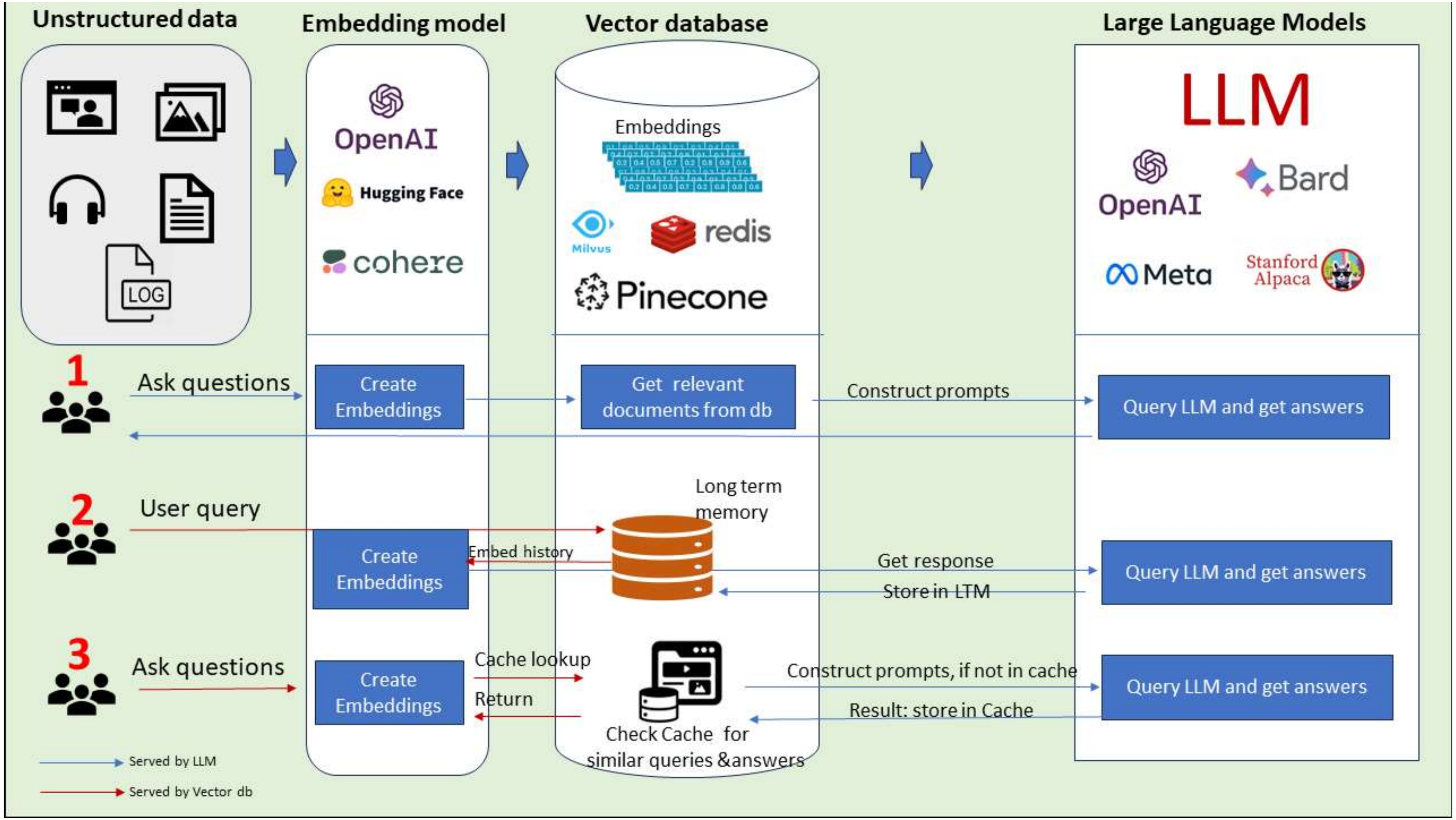
이미지 출처: https://dev.to/josethz00/vector-databases-5df1

VECTOR

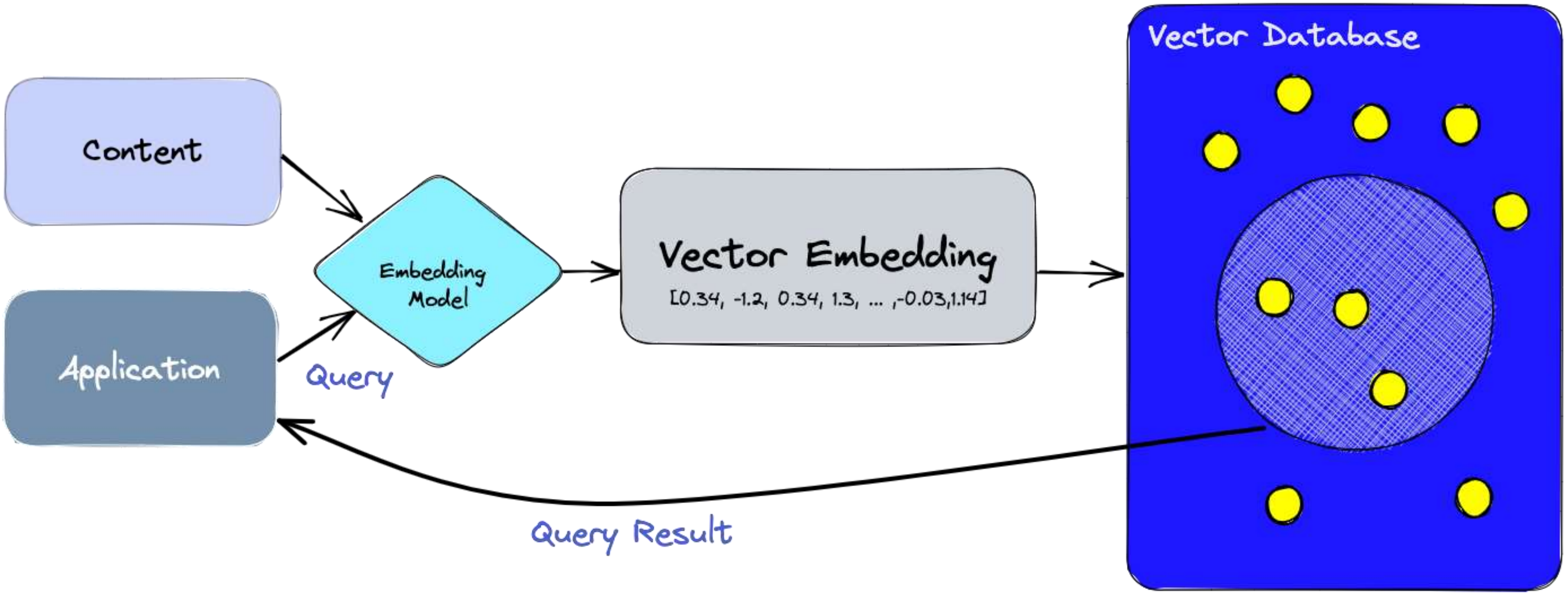


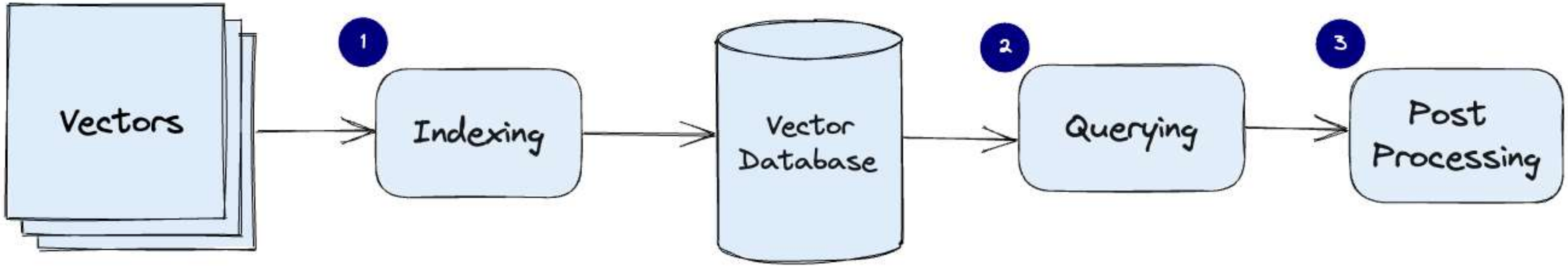
이미지 출처: https://rfriend.tistory.com/804

VECTOR


Unstructured data
Embedding model
Vector database
Large Language Models
LOG
OpenAI
Hugging Face
cohere
Embeddings
Milvus
redis
Pinecone
LLM
OpenAI
Bard
Meta
Stanford Alpaca
1
Ask questions
Create Embeddings
Get relevant documents from db
Construct prompts
Query LLM and get answers
2
User query
Long term memory
Create Embeddings
Embed history
Get response
Store in LTM
Query LLM and get answers
3
Ask questions
Create Embeddings
Cache lookup
Return
Check Cache for similar queries &answers
Construct prompts, if not in cache
Result: store in Cache
Query LLM and get answers
Served by LLM
Served by Vector db

VECTOR





이미지 출처: ChatGPT의 전두엽(장기기억 저장소)으로 각광받고 있는 Vector DB에 대해 알아보자 (sk.com)

VECTOR

Pinecone

milvus

Weaviate

PostgreSQL

Meta

Demo

GitHub - pgvector/pgvector: Open-source vector similarity search for Postgres

R, Python 분석과 프로그래밍의 친구 (by R Friend) :: [PostgreSQL, Greenplum] Greenplum의 pgvector와 OpenAI를 이용하여 대규모 AI 기반 검색 구축하기 (Building large-scale AI-powered search in Greenplum using pgvector and OpenAI) (tistory.com)

감사합니다.

윤명식 / MyoungSig.Youn

@MegazoneCloud