1. **Baza Grafowa – Neo4j**

Neo4j to baza grafowa, w której dane są przechowywane jako węzły (nodes) oraz krawędzie (relationships). Każdy węzeł i krawędź mogą mieć własne właściwości (properties). Zapytania do bazy odbywają się w języku **Cypher**.

W Neo4j dane są reprezentowane jako graf skierowany, np.:

(Alice) --[FRIEND]--> (Bob)

(Bob) --[WORKS\_AT]--> (CompanyX)

Neo4j obsługuje następujące typy wartości:

* **Podstawowe typy**
  + String – np. "Alice"
  + Integer – np. 42
  + Float – np. 3.14
  + Boolean – true, false
  + Null – NULL
* **Strukturalne typy**
  + List<T> – np. ["red", "green", "blue"]
  + Map<K, V> – np. {name: "Alice", age: 30}
* **Typy czasowe**
  + Date – np. 2025-03-24
  + Time – np. 12:00:00
  + LocalTime – np. 14:30:45
  + DateTime – np. 2025-03-24T12:00:00Z
  + LocalDateTime – np. 2025-03-24T14:30:45
  + Duration – np. P2DT3H4M (2 dni, 3 godziny, 4 minuty)

1. **Kolumnowy – Cassandra**

Apache Cassandra to baza kolumnowa, przechowująca dane w strukturze podobnej do tabel, ale w sposób bardziej elastyczny niż w relacyjnych bazach danych.  
W Cassandra dane są podzielone na **klastry**, **kolumnowe rodziny** i **wiersze**. Każdy wiersz może mieć różne kolumny.

**Brak natywnych relacji** – każda tabela działa niezależnie. Można odwzorować relacje, ale trzeba używać denormalizacji. Sprawia to, że gdy dana używana w wielu miejscach się zmieni należy zmienić ją w każdym miejscu

Przykładowa tabela w Cassandra

CREATE TABLE users (

id UUID PRIMARY KEY,

name TEXT,

age INT,

emails LIST<TEXT>

);

Cassandra obsługuje następujące typy:

* **Podstawowe typy**
  + ascii – znak ASCII, np. 'A'
  + bigint – liczba całkowita 64-bitowa, np. 9223372036854775807
  + blob – dane binarne, np. 0x89ABCDEF
  + boolean – true, false
  + counter – licznik, używany do inkrementacji (+1, -1)
  + decimal – liczby zmiennoprzecinkowe o zmiennej precyzji
  + double – liczby zmiennoprzecinkowe podwójnej precyzji (64-bit)
  + float – liczby zmiennoprzecinkowe pojedynczej precyzji (32-bit)
  + int – liczby całkowite 32-bitowe
  + smallint – liczby całkowite 16-bitowe
  + tinyint – liczby całkowite 8-bitowe
  + text – ciąg znaków (jak string)
  + uuid – identyfikator UUID, np. 123e4567-e89b-12d3-a456-426614174000
  + timeuuid – UUID z informacją o czasie
  + inet – adresy IP (IPv4 i IPv6)
  + varint – liczby całkowite o dowolnej wielkości
* **Strukturalne typy**
  + list<T> – np. ["email1", "email2"]
  + set<T> – np. { "football", "gaming" }
  + map<K, V> – np. { "home": "123456", "work": "654321" }
  + tuple<T1, T2, ...> – np. (1, "A", true)
  + frozen<T> – pozwala przechowywać struktury w niezmienialnej postaci
* **Typy czasowe**
  + timestamp – np. 2025-03-24 14:30:00
  + date – np. 2025-03-24
  + time – np. 14:30:00
  + duration – np. P2DT3H4M

Cassandra **nie obsługuje** typów węzłów czy relacji jak Neo4j.

1. **JSON - MongoDB**

JSON (JavaScript Object Notation) to format przechowywania danych używany przez bazy dokumentowe, np. MongoDB. Dane są zapisane w postaci **obiektów**.

Przykład dokumentu użytkownika:

{

"id": "123e4567-e89b-12d3-a456-426614174000",

"name": "Alice",

"age": 30,

"emails": ["alice@email.com"],

"address": {

"street": "Main St",

"city": "New York"

}

}

* **Podstawowe typy**
  + String – np. "Alice"
  + Number – np. 42, 3.14
  + Boolean – true, false
  + Null – null

1. **Strukturalne typy**
   * Array – np. ["email1", "email2"]
   * Object – np. { "name": "Alice", "age": 30 }

JSON **nie obsługuje** typów binarnych, dat, czasów czy UUID w sposób natywny. W bazach NoSQL takich jak **MongoDB** są one przechowywane w specjalnym formacie BSON, który dodaje obsługę:

* Date – np. ISODate("2025-03-24T14:30:00Z")
* ObjectId – np. 507f1f77bcf86cd799439011 (unikalny identyfikator)
* Binary – np. BinData(0, "QkN0ZXN0")
* Timestamp – np. {t: 1624000000, i: 1}

# Tabela mapowań:

|  |  |  |  |
| --- | --- | --- | --- |
| Python | MongoDB | Cassandra | Neo4j |
| str | String | Text | String |
| list | Array | List | List<T> |
| tuple | Array | Tuple | List<T> |
| dictionary | Object | map<K,V> | Map<K,V> |
| int | Int64 | Int | Integer |
| float | Double | Float | Float |
| complex | Object | Map<K,V> | Map<K,V> lub List lub String |
| bool | Boolean | Boolean | Boolean |
| set | Array | set | List with apoc.coll.toSet()  Ale nie ma jako takiej unikalnosci |
| frozenset | Array | frozen<T> | List |
| NoneType | Null | Null | Null |
| Date | Date | date | Date |

Neo4j:

Klasy - Klasa jest reprezentowana jako etykieta węzła.

Obiekty – Węzły z określonymi właściwościami.

Dziedziczenie – Etykieta węzła.

Relacje między obiektami – Krawędzie między węzłami.

MongoDB:

Klasy – Nie istnieją.

Obiekty – Każdy obiekt w Pythonie można zamienić na słownik i przechowywać jako dokument

Dziedziczenie – Nie ma wbudowanego mechanizmu obsługi dziedziczenia, ale można to rozwiązać za pomocą pola \_cls, które przechowuje typ obiektu. Podczas zapisu obiekt jest konwertowany na słownik (to\_dict()), a następnie zapisywany w MongoDB.

Relacje między obiektami – Referencje, czyli przechowywanie identyfikatora innego dokumentu

Cassandra:

Klasy - Klasa jest reprezentowana jako tabela.

Obiekty – Wiersz w tabeli.

Dziedziczenie – Nie wspierane. Można symulować jako pole type w tabeli.

Relacje między obiektami – W Cassamdra nie ma joinów, więc używa się denormalizacji lub kluczy obcych.