Big Data Management

Project Presentation

Valentina Sisti
1952657

Domain of Interest

♫ Music & Spotify Data

- Songs
- Artists
- Albums

- Genres
- Record Labels
- Music Charts

Data collection phase

 6 Datasets

 14 CSV Files

Data Collection Phase

Spotify®

8+M. Spotify Tracks Dataset

- *tracks.csv*
- *artists.csv*
- *albums.csv*
- *audio_features.csv*
- *r_albums_artists.csv*
- *r_albums_tracks.csv*
- *r_artist_genre.csv*
- *r_tracks_artist.csv*

Data Collection Phase

Billboard Hot 100 -
Full History Dataset

billboard-hot-100-tracks.csv

billboard HOT 100

Data Collection Phase

The Influence of Music, Spotify and AllMusic Data

influence_data.csv

Spotify®

ALLMUSIC

Data Collection Phase

record_labels.csv

Record Labels - All Universal Music Group Artists

UNIVERSAL MUSIC GROUP

Data Collection Phase

Exportify GENIUS Spotify

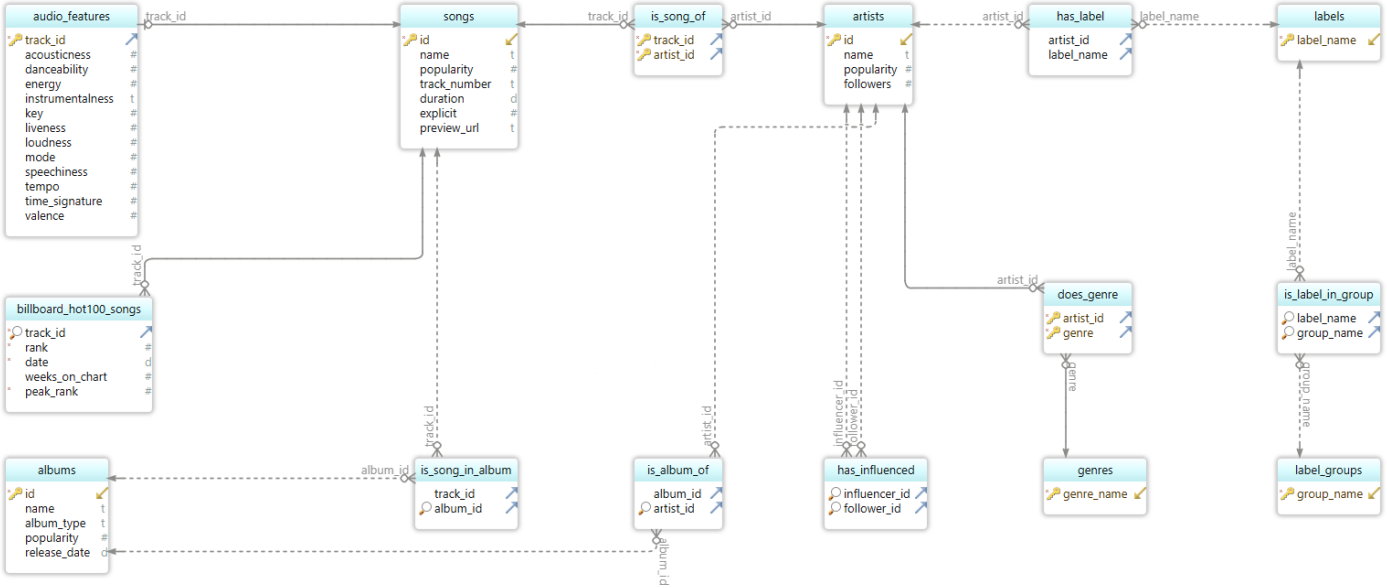
Problems of the Data

- Conceptual heterogenity
  - Concept of «id» may refer to Spotify ID, Billboard ID or custom ID internally defined
- *track_id* and *audio_feature_id* are the same in *tracks.csv*
- Songs' *duration* stored in milliseconds
- Albums' *release_date* stored in epoch format

Problems of the Data (2)

- Some data sources contain a ***huge volume*** of data
- A great part of it refers to quite ***unknown artists*** and songs
- In order to gain interesting insights, we want to restrict our discussion to just ***most relevant*** artists and songs

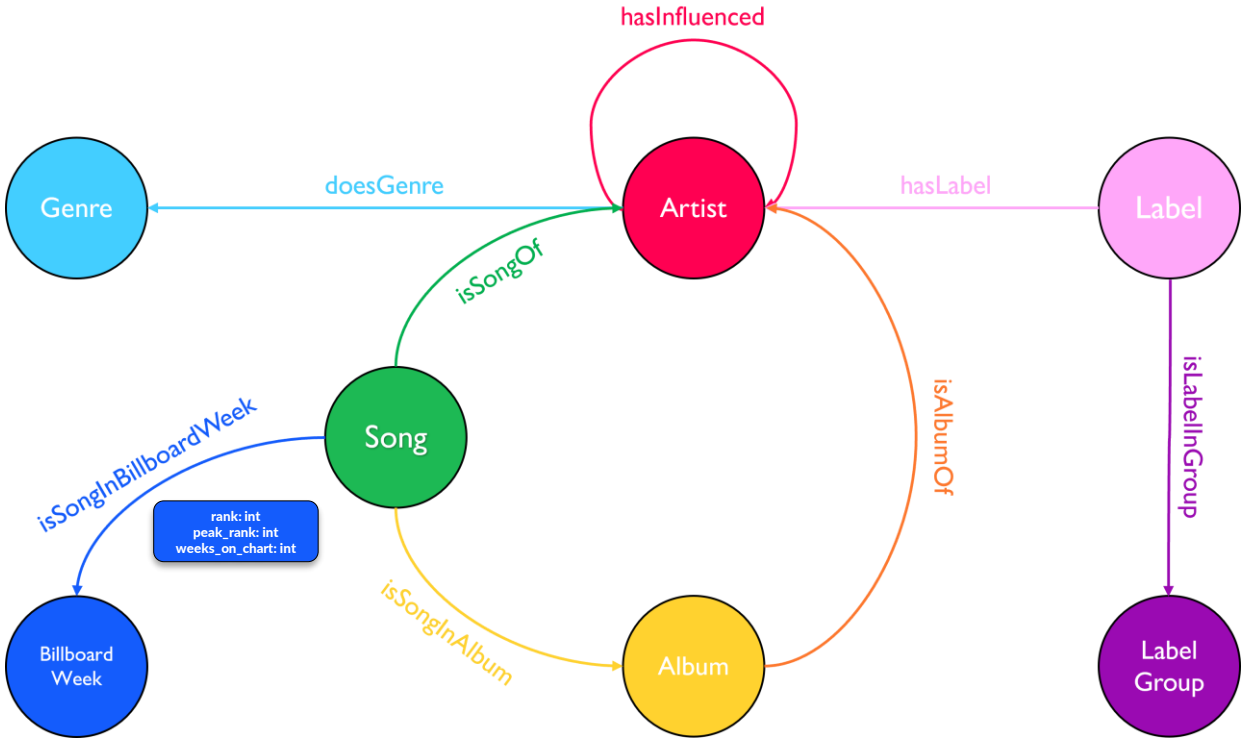
Resulting RDBMS



The need for a Graph Database

- *Relationships* are very predominant in this domain
- in a RBDMS we may need *many joins* when querying the data
- RDBMS suffer of the so-called ***join pain phenomenon***

Property Graph Model



Choosing a proper tool:

- **Multi-Model** NoSQL DBMS
- Combines *power* of **graphs** with *flexibility* of **documents**
- Entirely written in **Java**
- No *configuration* and *installation*
- Community edition *free* for commercial use
- Fully supports **ACID** transactions

OrientDB: Multi-Model

OrientDB engine supports the following models:

- *Graph*
- *Document*
- *Key/Value*
- *Object*

and combines the features of the four models into the core

OrientDB as a Graph Database

Property Graph which defines:

- **Vertices**
  - unique identifier
  - set of incoming edges
  - set of outgoing edges
- **Edges**
  - unique identifier
  - link to an incoming Vertex
  - link to and outgoing Vertex
  - label

Relational Model	Graph Model	OrientDB Graph Model
Table	Vertex and Edge Class	Class that extends "V" (for Vertex) and "E" (for Edges)
Row	Vertex	Vertex
Column	Vertex and Edge property	Vertex and Edge property
Relationship	Edge	Edge

OrientDB Classes

- Concept taken from **OOP** paradigm
- Define records
- Can be ***schema-less***, ***schema-full*** or ***hybrid***
- Can *inherit* from other classes
- Have one **cluster** defined as default cluster but can support multiple clusters

OrientDB Clusters

Classes: *logical* framework to organize data

Clusters: *physical* (or in-memory) space where data is stored

Examples:

- Cluster «cache»
  containing most accessed records
- Clusters «artists_it» , «artists_us»
  containing data on artists from Italy and USA

OrientDB Storages

OrientDB supports 3 storage types:

- **plocal** → persistent, disk-based
- **remote** → use of the network to access it
- **memory** → all data remains in memory

Each storage is composed of multiple *clusters*

OrientDB Query Language

- Since SQL is the most widely recognized standard, OrientDB uses **SQL** as its basic query language

- OrientDB then *enhances* SQL with some extensions to enable **graph functionalities**

OrientDB's SQL Dialect

- The classic JOIN syntax of SQL is not supported in OrientDB
  - Relationships are represented as LINKs instead of JOINs
- The * in projections is optional
  - Writing `SELECT * FROM Song` and `SELECT FROM Song` is equivalent
- Does not support the `HAVING` keyword
- OrientDB allows to SELECT only from *one* class
  - `SELECT FROM Song,Album` cannot be done in OrientDB

MATCH clause

- Queries the database in a declarative manner, using pattern matching

```
MATCH
  {
    [class: <class>],
    [as: <alias>],
    [where: (<whereCondition>)]
  }
  .<functionName>(){
    [class: <className>],
    [as: <alias>],
    [where: (<whereCondition>)],
    [while: (<whileCondition>)],
    [maxDepth: <number>],
    [depthAlias: <identifier> ],
    [pathAlias: <identifier> ],
    [optional: (true | false)]
  }*
  [,
    [NOT]
    {
      [as: <alias>],
      [class: <class>],
      [where: (<whereCondition>)]
    }
    .<functionName>(){
      [class: <className>],
      [as: <alias>],
      [where: (<whereCondition>)],
      [while: (<whileCondition>)],
      [maxDepth: <number>],
      [depthAlias: <identifier> ],
      [pathAlias: <identifier> ],
      [optional: (true | false)]
    }*
  ]*
RETURN [DISTINCT] <expression> [ AS <alias> ] [, <expression> [ AS <alias> ]]
GROUP BY <expression> [, <expression>]*
ORDER BY <expression> [, <expression>]*
SKIP <number>
LIMIT <number>
```

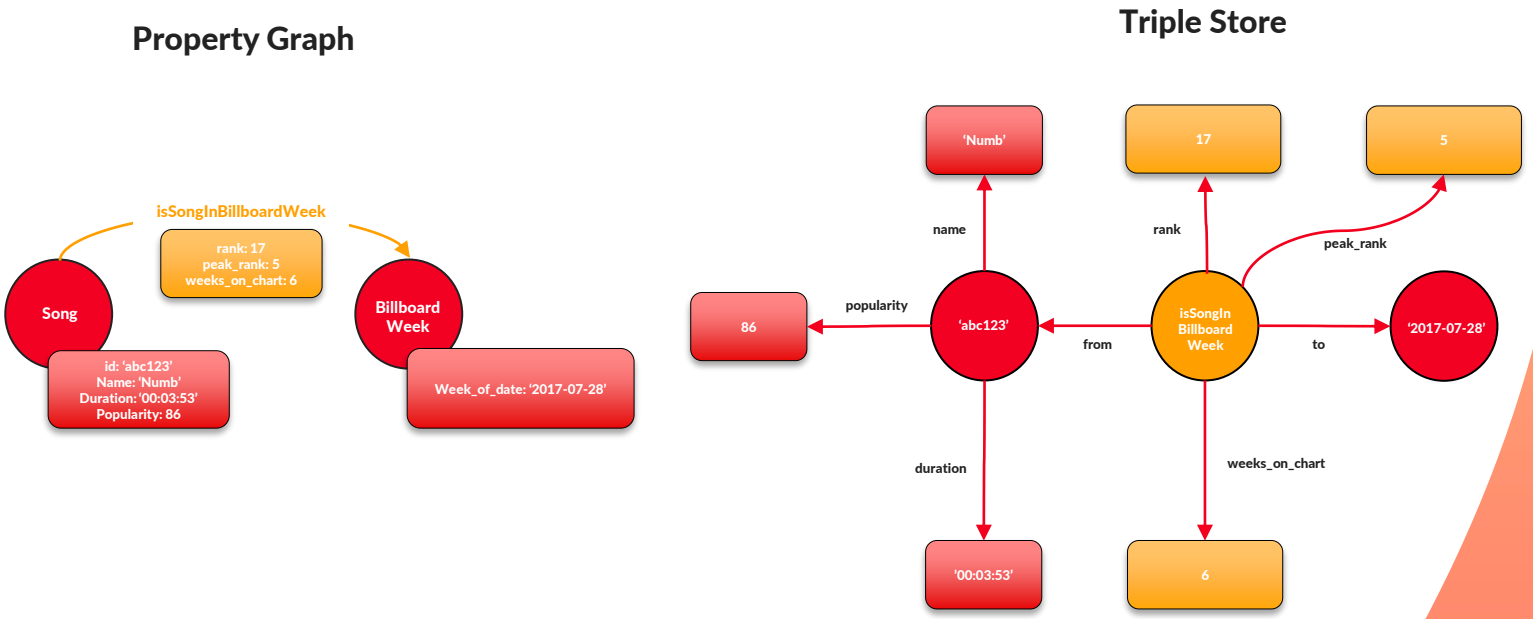
DEMO

Property Graph VS Triple Store

- Take a graph with *n* nodes
- 5 properties per node
- 5 relationships per node
- 1 label per node

- in RDF it will contain 11 x n triples
- When we have a 100 millions triple graph, that same data will probably be equivalent to labeled property graph with 10 millions nodes

Property Graph VS Triple Store: Attributes of Relationships



Which type of Graph Database?

Triple Stores

- best suited when we have a slow-changing dataset
- from what is known, they do not scale very well
- most used in when we want to derive ontologies

OrientDB: Document Model

- A document is a set of key/value pairs
- Values can hold primitive data types, embedded documents, or arrays of other values
- Documents are stored in *collections* enabling developers to group data as they decide. OrientDB uses the concepts of "classes" and "clusters" instead of "collections" for grouping documents
- OrientDB's Document model adds the concept of a "LINK" as a relationship between documents.

Relational Model	Document Model	OrientDB Document Model
Table	Collection	Class or Cluster
Row	Document	Document
Column	Key/value pair	Document field
Relationship	not available	Link

OrientDB: Key/Value Model

Relational Model	Key/Value Model	OrientDB Key/Value Model
Table	Bucket	Class or Cluster
Row	Key/Value pair	Document
Column	not available	Document field or Vertex/Edge property
Relationship	not available	Link

OrientDB: Object Model

Relational Model	Object Model	OrientDB Object Model
Table	Class	Class or Cluster
Row	Object	Document or Vertex
Column	Object property	Document field or Vertex/Edge property
Relationship	Pointer	Link