Mini-projet - de la modélisation à la visualisation

vos noms et prénoms

Qian chengzhi

la source (url) de vos données.

I selected the Steam game data from Kaggle as my dataset. The dataset is extremely large, containing over 50,000 entries, meeting the requirements for large-scale data analysis and making it suitable for statistical analysis. Furthermore, the data is already structured, with **appid** serving as the unique primary key. The Steam game dataset includes explicit category information such as “genres” and “categories,” which can be mapped to entity-property-value triples in an RDF graph. Furthermore, fields like ‘developer’ and “publisher” in the Steam data correspond to entities in Wikidata, providing an interface for subsequent tasks of mapping related content to Wikidata and enriching the information.

les différentes étapes suivies et ce que vous avez réalisé à chaque étape

1. Modélisation en RDF(S)

According to the experiment requirements, this phase involves two tasks:

(1) Design an RDFS ontology to define the vocabulary for modeling;

(2) Convert structured data into RDF and validate its correctness in the triplet repository.

1.1 Data cleansing

a) Due to the massive size of the dataset, importing and exporting it takes an extremely long time. Therefore, to facilitate data processing, we extracted the first 10,000 records.

b) I first removed columns containing large amounts of text or those less relevant to this project's query objectives, such as `detailed_description` and `about_the_game`. Directly converting these attributes to RDF would significantly inflate the size of triples and increase cleaning costs.

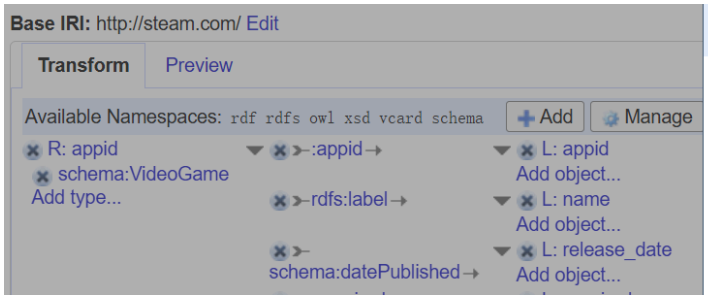
c) For numeric data types such as `price`, I transform them to number in OpenRefine. This ensures that the values in the resulting triplets are of type int or double and that these values carry the correct data types after RDF export.

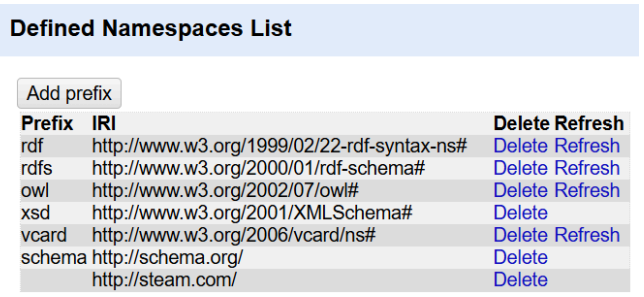
d) Steam data contains numerous multi-valued fields, such as `genres` and `categories`. I first used GREL to convert JSON arrays into text strings separated by the pipe character (|). Then, I executed Split multi-valued cells to break them down into true multi-valued units. This ensures that the subsequent RDF generation can accurately express the multi-valued property structure in RDF.

e) For the date data `release_date`, I performed date conversion and formatting (yyyy-MM-dd), enabling the date data to be expressed as `xsd:date` in RDF. This facilitates subsequent implementation of time-based queries and statistical requirements in SPARQL.

1.2 Ontology Design: Core Classes and Attributes

1.2.1 Namespace





First, I set the Base IRI to http://steam.com/ as the default namespace prefix for generating RDF in this project, providing a unified namespace for entities in this dataset.

Then, I added "http://steam.com/" to the prefix mapping table `Available Namespaces` to enable abbreviations and reuse of class and property URIs from standard vocabularies, facilitating subsequent expression of types and relationships in RDF templates.

1.2.2 Properties

- **Core property: schema: VideoGame**

I select `R: appid` as the root node and assign it the `schema:VideoGame` type. Steam data is essentially game data, and Schema.org already provides `schema:VideoGame` for handling game data. Therefore, we directly use `schema:VideoGame` as the central node in RDF. Since appid is unique for each game, it serves as the primary key.

- **Dates & Numeric property**

For numeric or date-type data like `required_age`, I simply modified the data type. In some cases, I adopted types from the schema to reduce customization and clarify semantics.

- **Text & External property**

For text-type data, some entries are plain text, which I simply set as text type. For data containing URLs, which are essentially web addresses, I set their data type to IRI. This allows subsequent access to the links and redirection to the official websites.

- **Multi-valued IRI Lists property**

I modeled the six fields—`categories`, `genres`, `developers`, `publishers`, `supported_languages`, and `full_audio_languages`—as IRIs rather than plain strings, making them reusable, referenceable, and alignable properties. This facilitates subsequent reverse lookups, enabling the formation of graph structures for easier statistical analysis and aggregation. Additionally, this approach enhances alignment capabilities with external Linked Open Data, making cross-dataset integration easier. This is the core of knowledge graph modeling!

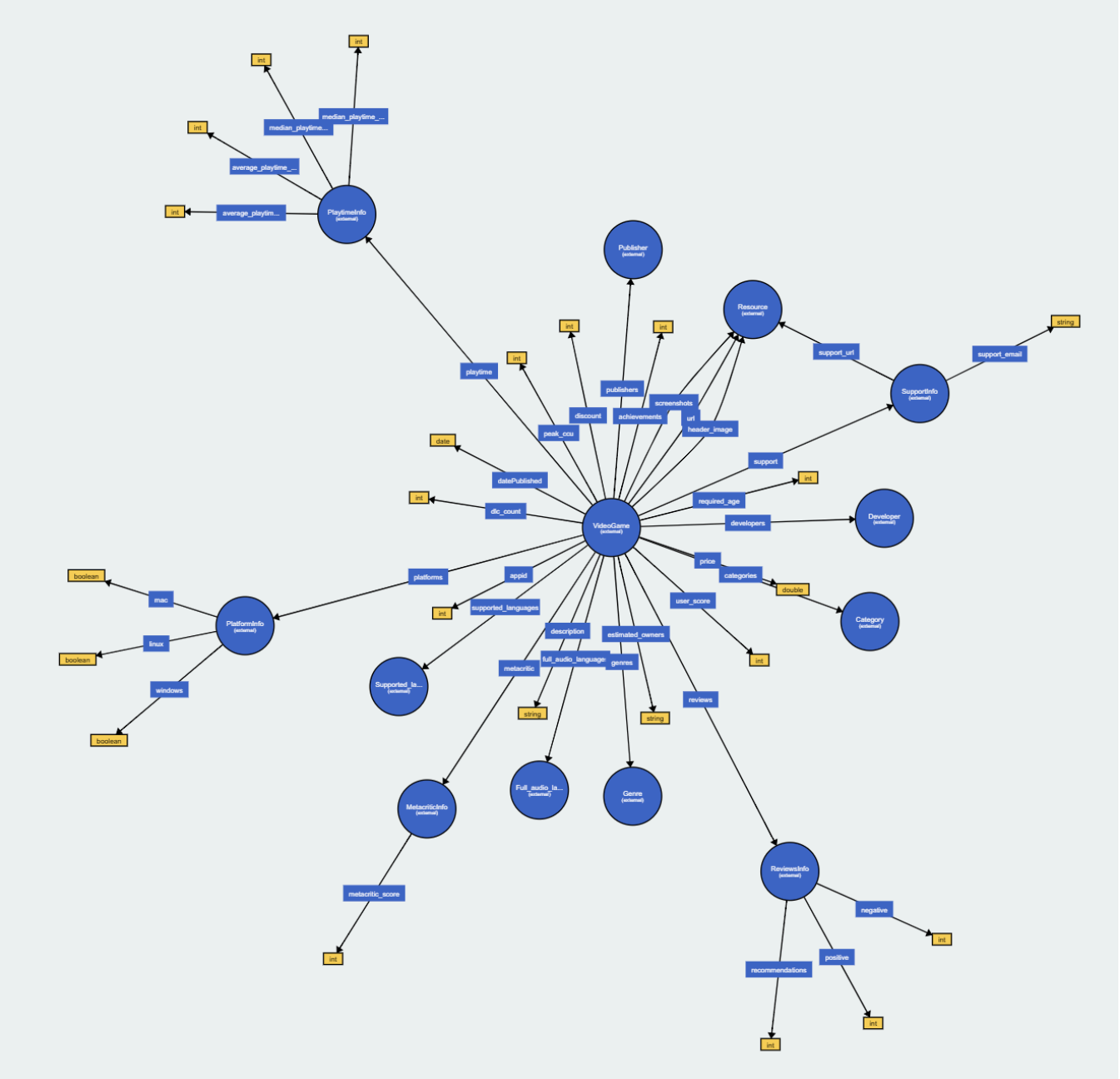
- **Structured Attribute Blocks**

I organize a group of strongly related, thematically consistent fields into composite property blocks, such as :support, :platforms, :metacritic, :reviews, :playtime, etc. This information isn't isolated single properties but rather a semantically complete sub-object—for example, customer support includes support_url/support_email. Using Blank Nodes consolidates properties under the same theme into a single structural unit. This approach yields a clearer graph that better aligns with real-world semantics while offering superior scalability. This step also facilitates more efficient querying: locate the relevant thematic block first, then extract specific metrics from it, enabling a more modular SPARQL access method.

1.2.3 RDFS relations

I first defined related classes centered around schema:VideoGame based on OpenRefine's RDF specifications (PlatformInfo, SupportInfo, ReviewsInfo, MetacriticInfo, PlaytimeInfo, etc.), distinguishing fields into object properties (pointing to literals) and data properties (pointing to resources). Then, I supplemented each property with rdfs:domain and rdfs:range to form an interpretable RDFS vocabulary.

Then, I passed the RDFS relationship into `webVOWL` and obtained the following relationship graph.



2. Point d’accès SPARQL.

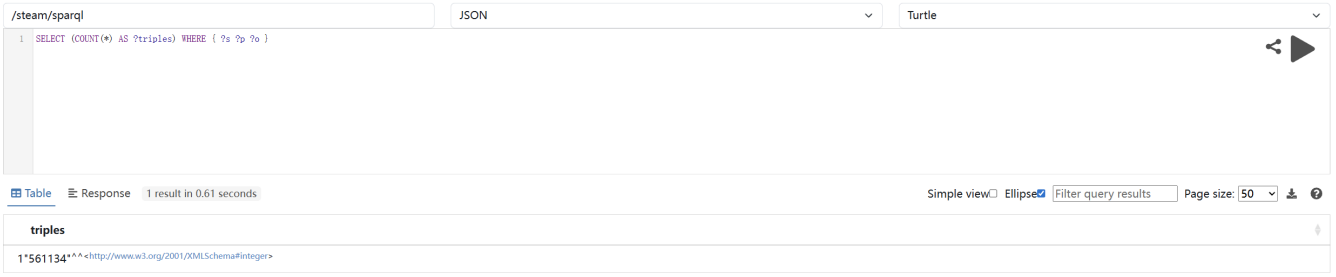
Create a dataset: In the Fuseki management interface, create a new dataset named “steam” and select “persistent” as the storage type for easier future management.

Load the RDF file exported from OpenRefine into this dataset. After importing, you can see in Fuseki that the data graph has been stored in the triplet repository.

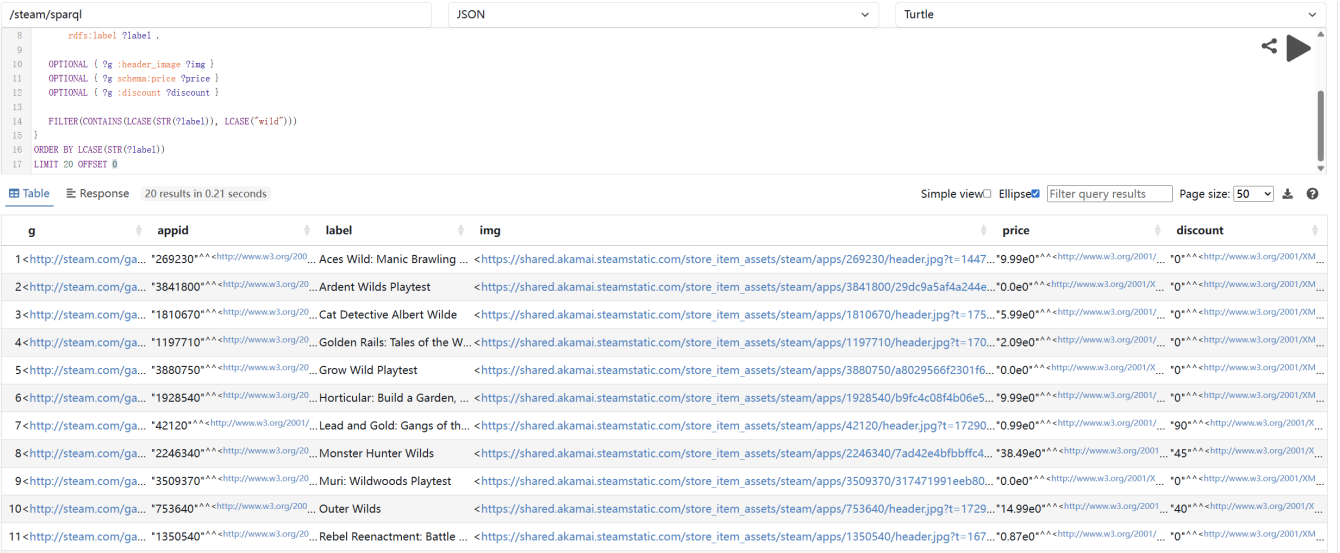
3. Elaboration de requêtes SPARQL.

Here I've referenced everything in the SPARQL documentation and tried numerous queries. This time I'll select some simple queries as examples. The specific queries I used will be detailed in the query section below.

Query the total number of triplets

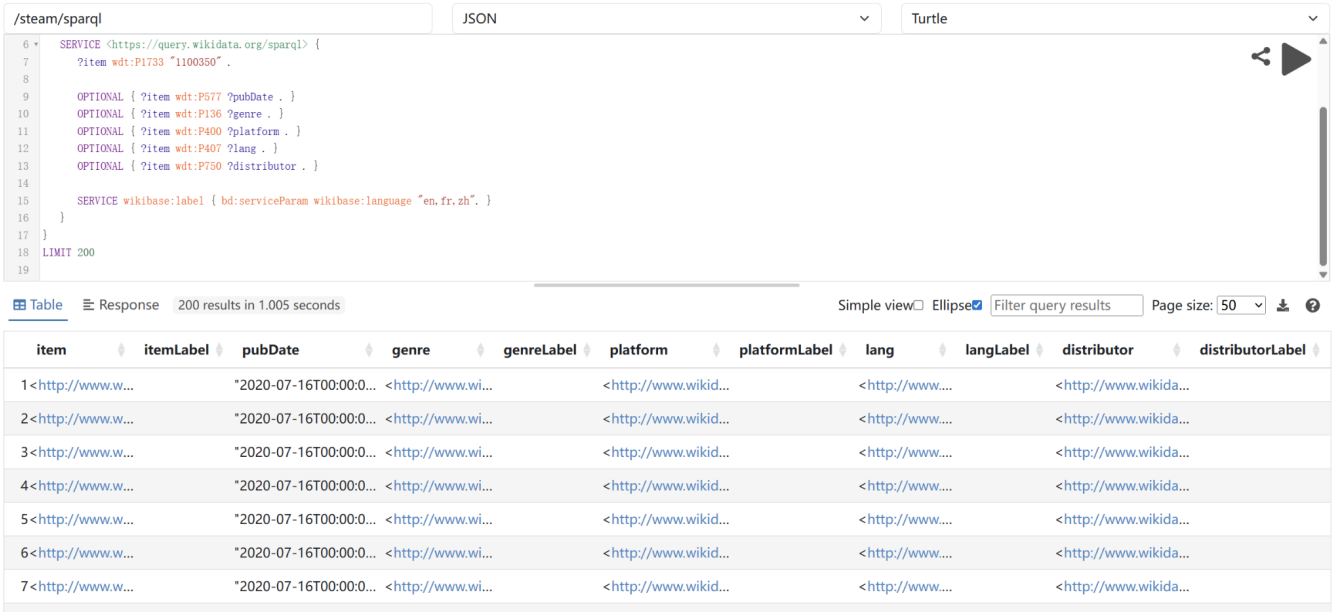


Query Type



4. Accès à des données liées.

Here, I connect the data to an external database. I link my game's ID to its Steam ID in Wikidata to retrieve Wikidata information about this game, which I use to supplement my own dataset. However, my investigation revealed that Wikidata's data itself is not as comprehensive as my dataset. My dataset covers more extensive content than Wikidata's data. Therefore, in my subsequent web pages, I did not incorporate content connected to Wikidata data.



5. Application Web

I first built a webpage myself to test the functionality, incorporating relevant search and chart features. Then I used GPT to generate a web page within the Angular framework, making it more visually appealing and practical. The specific features will be demonstrated later.

expliquer les parties du code (rdf, owl, sparql, js) qui a été produite par LLM et si vous l'avez modifié)

In this project, I used GPT solely to assist in generating the foundational framework and page structure for the web application, including the initial skeleton of frontend files along with minimal generic styles and placeholder code. The core semantic web-related implementations—such as designing and debugging SPARQL queries, modeling RDF/RDFS/OWL and authoring/refining Turtle files, implementing actual interaction logic with Fuseki endpoints, and final visualization and functional details—were all completed independently by me, with multiple iterations based on data characteristics. Therefore, GPT primarily served to enhance frontend development efficiency. I was responsible for verifying and ensuring the final delivered functionality aligned with the semantic data.

le squelette openrefine (rdf schema alignment),

I have already written the specific details earlier.

le schéma rdfs (turtle) et un extrait du fichier rdf (turtle) obtenu,

I have already written the specific details earlier.

les requetes sparql utilisées

Total number of games whose statistical names contain a specific keyword

```
SELECT (COUNT(DISTINCT ?g) AS ?total)
WHERE {
  ?g a schema:VideoGame ;
    rdfs:label ?label .
  FILTER(CONTAINS(LCASE(STR(?label)), LCASE("${safe}")))
}
`.trim();
```

Game Search and List Display Functionality

```
SELECT ?g ?appid ?label ?img ?price ?discount
WHERE {
  ?g :appid ?appid ;
    rdfs:label ?label .

  OPTIONAL { ?g :header_image ?img }
  OPTIONAL { ?g schema:price ?price }
  OPTIONAL { ?g :discount ?discount }

  FILTER(CONTAINS(LCASE(STR(?label)), LCASE("${safe}")))
}
ORDER BY LCASE(STR(?label))
LIMIT ${limit} OFFSET ${offset}
`.trim();
```

Retrieve complete details for a specific game based on the provided Steam app ID.

```
SELECT ?label ?appid ?price ?discount ?dlc ?owners ?peak ?age ?userScore ?img ?discription
       (GROUP_CONCAT(DISTINCT ?genreName; separator=" | ") AS ?genres)
       (GROUP_CONCAT(DISTINCT ?devName; separator=" | ") AS ?developers)
       (GROUP_CONCAT(DISTINCT ?pubName; separator=" | ") AS ?publishers)
       (GROUP_CONCAT(DISTINCT ?langName; separator=" | ") AS ?langs)
WHERE {
  ?g a schema:VideoGame ;
    :appid ?appid ;
    rdfs:label ?label .

  FILTER(STR(?appid) = "${appid}")

  OPTIONAL { ?g schema:price ?price }
  OPTIONAL { ?g :discount ?discount }
  OPTIONAL { ?g :dlc_count ?dlc }
  OPTIONAL { ?g :estimated_owners ?owners }
  OPTIONAL { ?g :peak_ccu ?peak }
  OPTIONAL { ?g :required_age ?age }
  OPTIONAL { ?g :user_score ?userScore }
  OPTIONAL { ?g :header_image ?img }
  OPTIONAL { ?g schema:description ?discription }

  # genres as resources
  OPTIONAL {
    ?g :genres ?genre .
    OPTIONAL { ?genre schema:name ?gn1 }
    OPTIONAL { ?genre rdfs:label ?gn2 }
    BIND(COALESCE(?gn1, ?gn2, STRAFTER(STR(?genre), STR(:))) AS ?genreName)
  }

  # developers as resources
  OPTIONAL {
    ?g :developers ?dev .
    OPTIONAL { ?dev schema:name ?dn1 }
    OPTIONAL { ?dev rdfs:label ?dn2 }
    BIND(COALESCE(?dn1, ?dn2, STRAFTER(STR(?dev), STR(:))) AS ?devName)
  }
```

```
  # publishers as resources
  OPTIONAL {
    ?g :publishers ?pub .
    OPTIONAL { ?pub schema:name ?pn1 }
    OPTIONAL { ?pub rdfs:label ?pn2 }
    BIND(COALESCE(?pn1, ?pn2, STRAFTER(STR(?pub), STR(:))) AS ?pubName)
  }

  # supported languages as resources
  OPTIONAL {
    ?g :supported_languages ?lang .
    OPTIONAL { ?lang schema:name ?ln1 }
    OPTIONAL { ?lang rdfs:label ?ln2 }
    BIND(COALESCE(?ln1, ?ln2, STRAFTER(STR(?lang), STR(:))) AS ?langName)
  }
}
GROUP BY ?label ?appid ?price ?discount ?dlc ?owners ?peak ?age ?userScore ?img ?discription
`.trim();
```

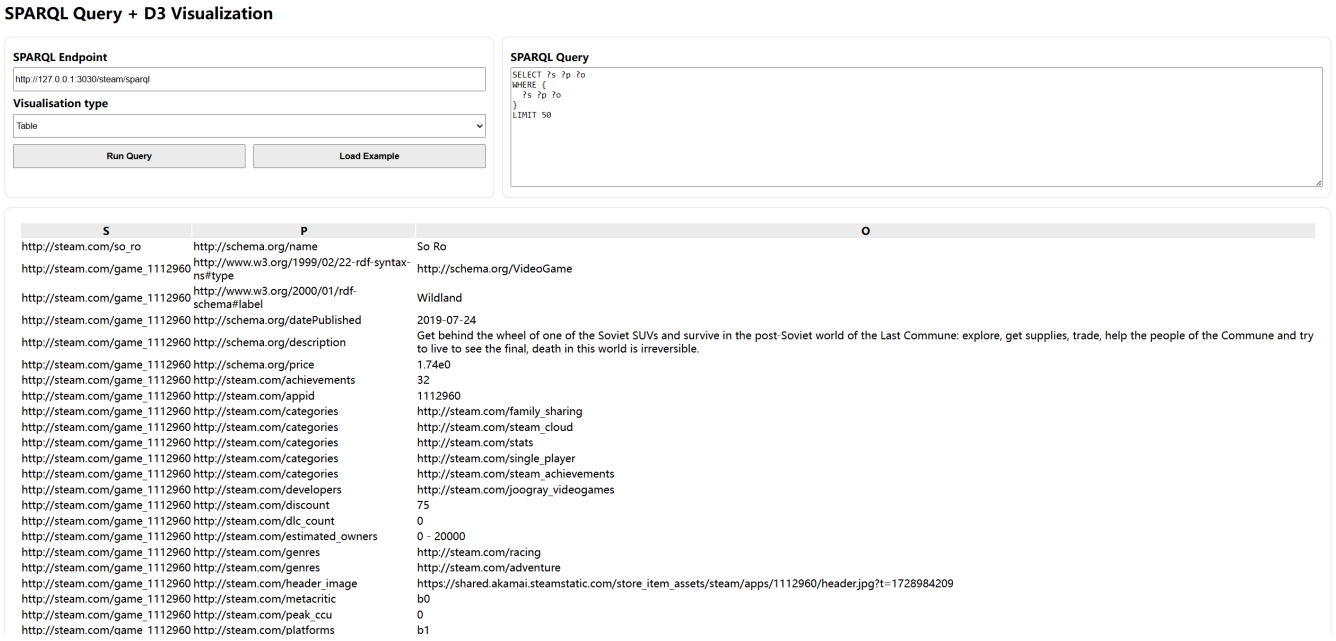
Number of games appearing in each genre within the statistical spectrum

```
SELECT (SAMPLE(?rawName) AS ?genreName) (COUNT(DISTINCT ?g) AS ?n)
WHERE {
  ?g a schema:VideoGame ;
    :genres ?genre .

  OPTIONAL { ?genre schema:name ?gn1 }
  OPTIONAL { ?genre rdfs:label ?gn2 }

  BIND(COALESCE(?gn1, ?gn2, STRAFTER(STR(?genre), STR(:))) AS ?rawName)

  BIND(LCASE(STR(?rawName)) AS ?key)
}
GROUP BY ?key
ORDER BY DESC(?n)
LIMIT ${limit}
`.trim();
```

This query is also crucial for generating the chart.

des photo d'écran de l'interface de votre application.

The webpage I made myself



SPARQL Query + D3 Visualization

SPARQL Endpoint

http://127.0.0.1:3030/steam/sparql

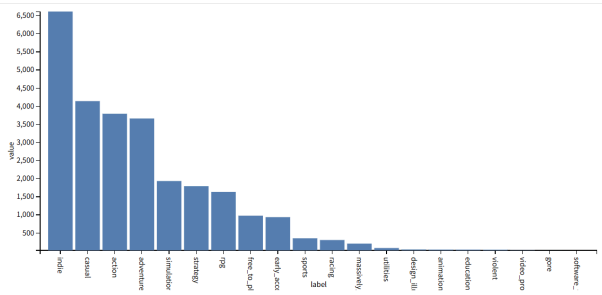
Visualisation type

Bar chart

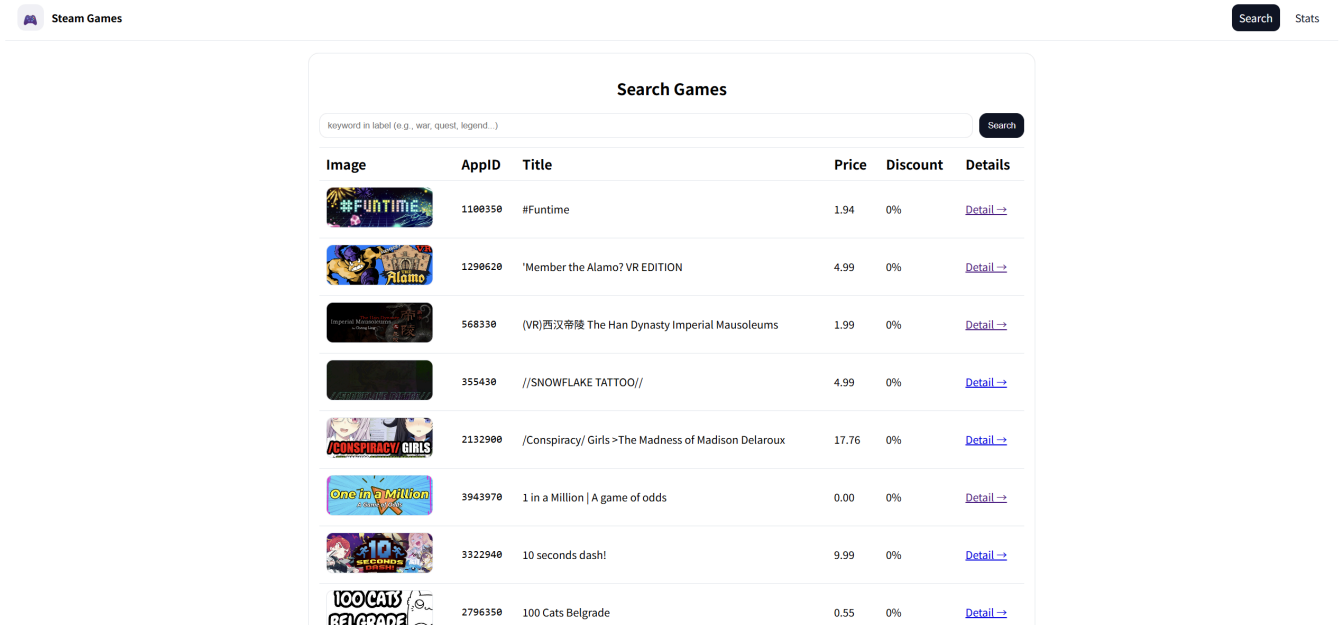
Run Query | Load Example

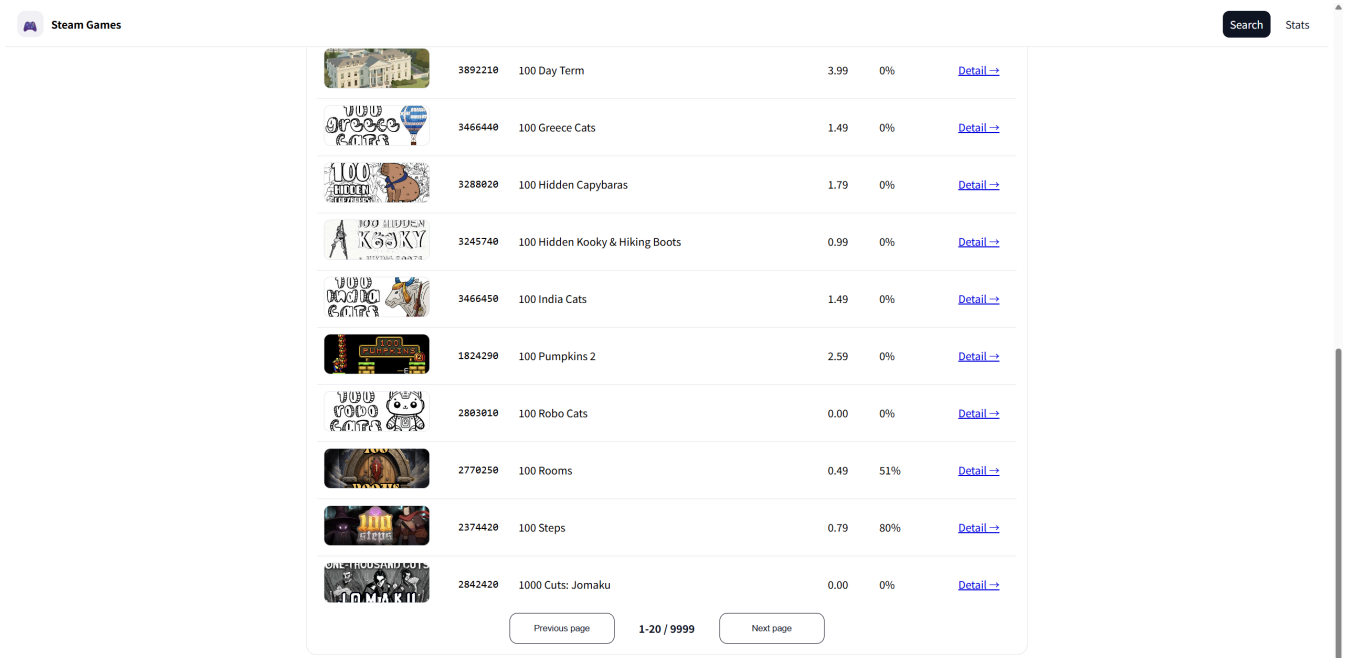
SPARQL Query

```
PREFIX schema: <http://schema.org/>

SELECT ?label (COUNT(DISTINCT ?g) AS ?value)
WHERE {
  ?g a schema:VideoGame ;
     :genres ?genre .

  BIND(LCASE(STRAFTER(STR(?genre), STR(:))) AS ?label)
}
GROUP BY ?label
ORDER BY DESC(?value)
LIMIT 20
```

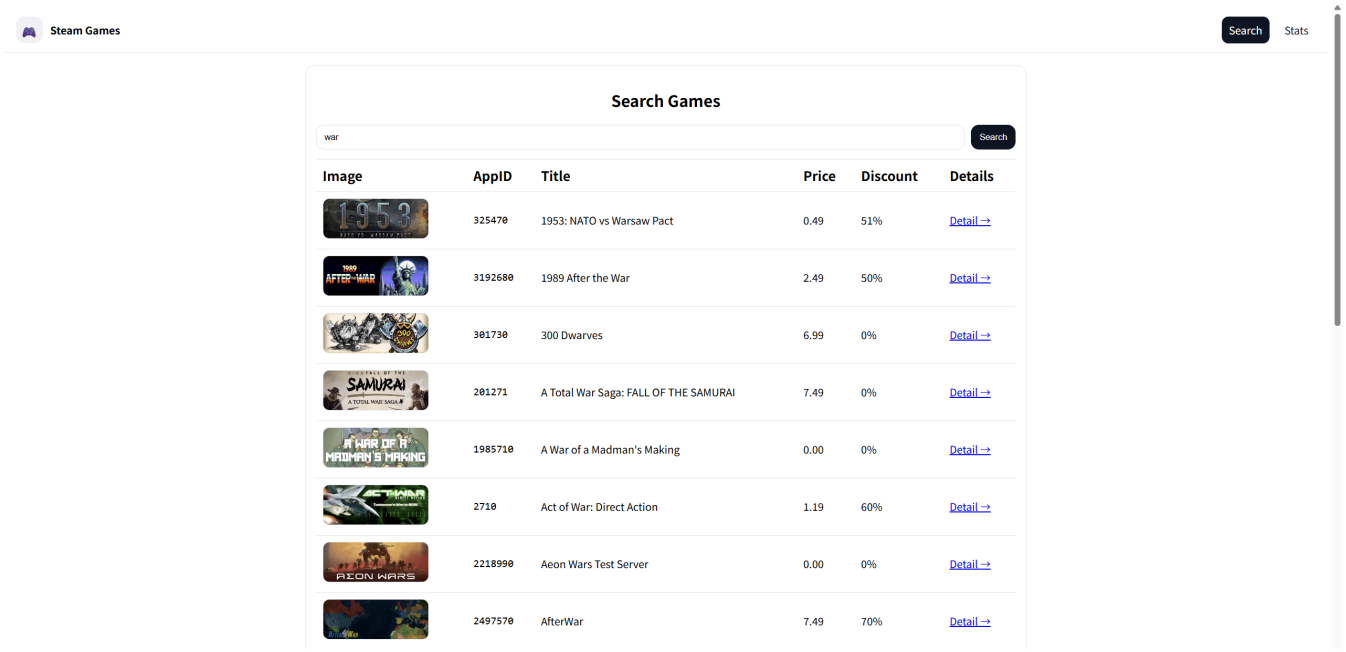


The webpage made by me and GPT





when we search



if we click on the detail

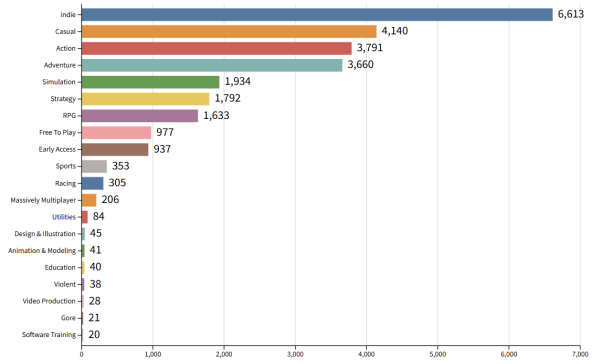
if we choose Stats, we can see the chart

Steam Games

Search Stats

Stats

Bar Pie



Top 20 genres by number of games.

Genre	Count
indie	6613
Casual	4140
Action	3791

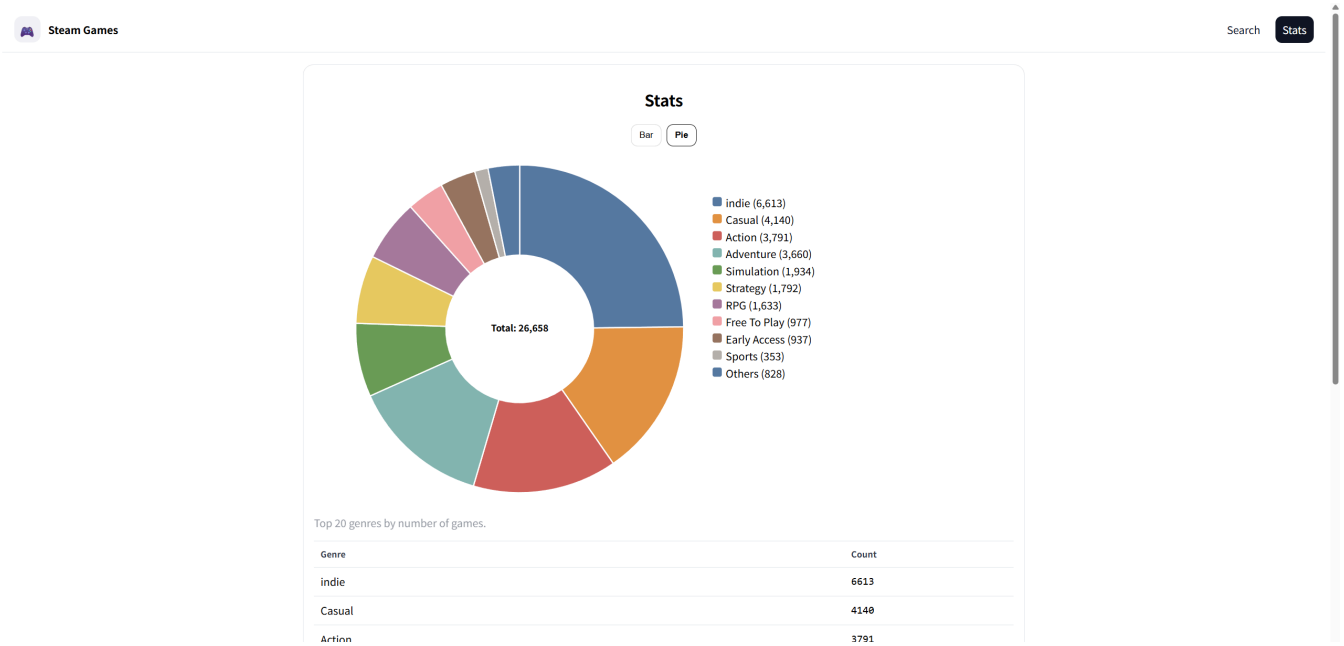
Steam Games

Search Stats

0 1,000 2,000 3,000 4,000 5,000 6,000 7,000

Top 20 genres by number of games.

Genre	Count
indie	6613
Casual	4140
Action	3791
Adventure	3660
Simulation	1934
Strategy	1792
RPG	1633
Free To Play	977
Early Access	937
Sports	353
Racing	305
Massively Multiplayer	206
Utilities	84
Design & Illustration	45
Animation & Modeling	41
Education	40
Violent	38
Video Production	28
Gore	21



l'adresse du git (sources + doc + exemples - voir ci-dessous)

https://github.com/ChengzhiQian/Web-application-QIAN-Chengzhi.git