# D3 - Knowledge Graph, Knowledge Extraction & Queries

Cono Cirone & Giulio Billi

February 2026

Link to TriplyDB: `https://triplydb.com/ConoCirone/TRAVELCOMPANIONAKR/`
Link to GitHub Repository:
`https://github.com/conocirone/german_travel_companion`

## 1 Changes from the Previous Submission

Following the comments received in the previous submission, we revised the ontology to improve its semantic expressiveness and to simplify future extensions.

- **Generalization of venue types.** We introduced a new class `VenueType` that acts as a parent class for the previously separate type classes `MuseumType`, `ParkType`, `ClubType`, and `SightType`.
- **Generalization of type relations.** We introduced `hasVenueType` as a general super-property, and aligned the existing `hasXType` properties as subproperties of it. The relation is defined with `PhysicalVenue` as its domain and `VenueType` as its range.
- **Revision of disjointness axioms.** To avoid excessive use of disjointness, we simplified the disjoint axioms in the ontology. The only remaining disjointness is between the classes `Tours` and `PhysicalVenue`.
- **Datatype compatibility for operating hours.** We modified the ranges of the data properties `opensAt` and `closesAt` to `xsd:string` to ensure compatibility with the HermiT reasoner, since the previous version caused reasoning issues due to unsupported datatype handling in our setup.
- **Addition of new functional data.** We introduced four new functional data properties to better support external references and meeting point metadata:
  - `hasURL` (`Tour` → `xsd:string`): URL link to the tour information page.
  - `hasImageURL` (`PhysicalVenue` → `xsd:string`): URL link to an image representing a physical venue.

- `hasMeetingPointDescription` (`MeetingPoint` → `xsd:string`): textual description of the meeting point location.
- `hasMapLink` (`MeetingPoint` → `xsd:string`): URL linking to the meeting point location on a map.

## 2 Data Collection and Enrichment Pipeline

### 2.1 TripAdvisor

#### 2.1.1 Scraping TripAdvisor

We experimented with multiple approaches to scrape TripAdvisor while minimizing bot-detection issues:

- **Camoufox + Playwright.** We initially used Camoufox together with Playwright to reduce automated browsing detection. However, this was not sufficient: our IP address was periodically banned. We attempted to mitigate this by implementing proxy rotation, but the free proxies available online were unreliable and frequently failed. Even when a proxy worked, TripAdvisor tended to ban it after scraping only a limited number of items.
- **SeleniumBase.** We then evaluated SeleniumBase, a Selenium-based framework that provides anti-detection features. In particular, its stealth/anti-detection mode allowed us to scrape the required data reliably without triggering repeated IP bans. This became our final scraping solution.

Scraping TripAdvisor alone was not sufficient for our ontology. In particular, we model two additional properties that were not directly available on the website:

- `LocationSetting` (indoor/outdoor)
- `BudgetTier` (free/low/medium/high)

Therefore, we needed an automated enrichment step. Classifying `LocationSetting` requires contextual understanding (e.g., determining whether an attraction is typically visited indoors or outdoors). Assigning `BudgetTier` requires knowledge of the base entry fee, which often requires querying external sources (e.g., official websites or aggregated pricing information).

#### 2.1.2 Local LLM-Based Attempts

Our initial idea was to run a fully local pipeline combining web search and an LLM:

- **DuckDuckGo Search + Llama 3.1.** For each attraction, we performed a DuckDuckGo web search and then prompted Llama 3.1 using the retrieved results as context, asking it to classify `LocationSetting` and `BudgetTier`. In practice, this approach was not sufficiently reliable: DuckDuckGo results were sometimes inaccurate or misleading, which in turn degraded the classification quality.

- **Perplexica + Llama 3.1.** We then switched to Perplexica, an AI-based search engine, which provided significantly higher-quality context. Using Perplexica (especially in reasoning mode) combined with Llama 3.1 produced correct classifications more consistently. However, this approach was not efficient: the processing time was approximately one minute per attraction, making it impractical at scale.

Despite these limitations, the key advantage of both local approaches was that they ran entirely offline (except for search) and could be integrated into a simple end-to-end pipeline: scrape TripAdvisor data, then automatically enrich the JSON dataset with `location_setting` and `budget_tier`.

#### 2.1.3 Final Enrichment Using Gemini

To keep the usage of LLM we decided to use Gemini 3 Pro. In fact it is a much more powerful LLM than Llama 3.1 and it is also much faster. Also the results are much more accurate. The prompt we used is the following:

Prompt used for enrichment

```
Task: Classify the attractions in the json file based on the name
and context. In case you are unsure about the price and the
location setting look up online for the correct informations.

   1. location_setting: 'indoor' or 'outdoor'
      - Museums, theaters, churches (interior visits) = indoor
      - Parks, squares, streets, monuments, outdoor markets =
      outdoor

   2. budget_tier: Based on the BASE ENTRY FEE to access the
   attraction ITSELF.
      - 'free': Public spaces (squares, streets, parks,
      neighborhoods, markets), monuments you can view from
      outside, churches with free entry, free museums
      - 'low': 1€-10€ entry fee
      - 'medium': 11€-20€ entry fee
      - 'high': >20€ entry fee

   IMPORTANT:
   - Ignore prices for guided tours, experiences, or activities -
   only consider the base entry fee.
   - Public squares (like Alexanderplatz, Potsdamer Platz),
   streets, parks, and neighborhoods are ALWAYS 'free'.
```

```
- If unsure and it's a public outdoor space, default to
'free'.

Return ONLY a valid JSON file with JSON code. No intro text.
You must add the two new fields to the ones which were already
in the file
Example: {{"location_setting": "indoor", "budget_tier":
"medium"}}
```

So we gave the JSON to Gemini and let him analyze it and enrich the file.

## 2.2 GetYourGuide

### 2.2.1 Scraping GetYourGuide

For the GetYourGuide platform, we developed a custom scraping solution using **Python** and **Playwright**. We selected Playwright over static parsers (like BeautifulSoup) because the website is highly dynamic, relying heavily on JavaScript for rendering content and pagination. Although GetYourGuide presented fewer anti-bot challenges than TripAdvisor, we prioritized reliability and stealth. To that end, we integrated `fake-useragent` to generate random browser signatures and implemented measures such as disabling automation flags (e.g., `navigator.webdriver`) and introducing random sleep intervals to mimic human behavior.

The development of the scraper evolved through two distinct phases to ensure data quality:

- **Version 1 (Proof of Concept).** The initial version focused on a single city (Berlin) to validate the core logic. During this phase, we identified critical missing data points. Specifically, self-guided activities (like museums or hop-on hop-off buses) often returned "N/A" for languages because the scraper only looked for "Live Tour Guides." Additionally, meeting point descriptions were often vague (e.g., "Open in Google Maps") without a usable address.
- **Version 2 (Multi-City & Enhanced Extraction).** The final version addressed these limitations and introduced scalability.
  - **Data Fixes:** We implemented fallback logic to explicitly check for "Audio Guides" if a live guide was missing, ensuring language data was captured for all activity types. We also modified the extraction logic to capture the specific Google Maps URL (`meeting_point_maps_link`), ensuring precise location data even when the text description was generic.
  - **Scalability:** The architecture was updated to iterate through a dictionary of target cities, generating individual JSON files for each city (e.g., Berlin, Munich, Hamburg) as well as a merged master dataset.

### 2.2.2 Data Processing with Gemini

The raw data scraped from GetYourGuide contained incomplete entries ("N/A" fields) and lacked specific properties required for our ontology (`LocationSetting` and `BudgetTier`). To address this, we employed Gemini 3 Pro in a two-step pipeline: Filtering and Enrichment.

**Step 1: Filtering and Cleaning** First, we processed the raw JSON files to remove noise. The scraper occasionally captured objects with insufficient data (e.g., items lacking descriptions or detail pages). We used the following prompt to clean the dataset:

Prompt used for Filtering

```
Task: Clean and filter the provided JSON file containing
GetYourGuide data.

Logic:
1. Analyze objects with fields marked as "N/A".
2. Attempt to infer the missing information by following the
provided link. If inference is impossible, delete the object.
3. Strict Deletion Rule: If an object contains only a "link" and
"title" but all other fields are "N/A", delete the object
immediately without searching since it's an object that don't
provide a detail page.

Output: Provide the filtered JSON.
```

**Step 2: Data Enrichment** Once the dataset was cleaned, we needed to populate the semantic properties for the ontology. We used the LLM to classify the location context and budget category based on the activity description and price.

Prompt used for Enrichment

```
Task: Classify the attractions in the JSON file based on the name
and context. If unsure about the price or location setting,
search online for the correct information.

Add the following fields to the objects:

1. location_setting: 'indoor' or 'outdoor'
   - Indoor: Museums, theaters, churches (interior visits), or
   activities inside a building.
   - Outdoor: Parks, squares, streets, monuments, outdoor
   markets.
```

```
    - Rule: If the tour is around the city, in a park, between
    different places, or uses a bus, set location_setting =
    'outdoor'.

 2. budget_tier: Based on the price.
    - 'free': Public spaces (squares, streets, parks,
    neighborhoods), monuments viewable from outside, free museums.
    - 'low': 1€ - 10€ entry fee.
    - 'medium': 11€ - 20€ entry fee.
    - 'high': > 20€ entry fee.


 IMPORTANT: If you are unsure and the object is a public outdoor
 space, default to 'free'.

 Output: Return ONLY a valid JSON file with the new fields added.
 No introductory text.
```

### 2.3 Note on Data Processing Pipeline

The data processing pipeline begins with a scraping phase that generates two distinct outputs: individual JSON files for each specific city and a single master file containing all cities organized by name. These individual city files are then processed through filtering and enrichment prompts to inject additional metadata, specifically identifying the `location_setting`, `budget_tier`, and an explicit `city` field for each entry. The process concludes with the combine_tours script that merges these enriched files into a final, flattened array of objects saved as `all_city_tours.json`.

- **Initial Extraction:** The scraper produces `city-name_tours.json` files and a combined key-value JSON.
- **AI Refinement:** Prompts are applied to single-city files to filter content and add descriptive tags.
- **Final Aggregation:** All refined data is merged into a single, high-utility array of objects.

### Data Structure Evolution

```
"Berlin": [
    {
        "title": "Berlin: City Sightseeing Hop-On Hop-Off Bus
        Tour",
        "price": "€21",
        "link": "...",
        "duration": "2 hours",
        "languages": "Spanish, English...",
        "meeting_point": "...",
        "meeting_point_maps_link": "..."
    }
]
```

Enriched Master File (`all_city_tours.json`)

```
[
    {
        "title": "Berlin: City Sightseeing Hop-On Hop-Off Bus
        Tour",
        "price": "€21",
        "link": "...",
        "duration": "2 hours",
        "languages": "Spanish, English...",
        "meeting_point": "...",
        "meeting_point_maps_link": "...",
        "location_setting": "outdoor",
        "budget_tier": "high",
        "city": "Berlin"
    }
]
```

## 3 Statistics

The datasets' statistics are provided in the `stats` folder included with the submission.

**Note on TripAdvisor statistics.** For TripAdvisor, many attractions have `operating_hours` set to `null`. This is not a scraper issue: in many cases, the information is simply not available on the TripAdvisor page. In addition, some attraction in the Parks and Sights & Landmarks categories may not have opening and closing hours by default. For these reasons, the TripAdvisor statistics show a relatively high number of missing `operating_hours` values.

## 4 Scraping Reproduction

To reproduce the data extraction phase, please refer to our GitHub repository linked at the beginning of this document. It is important to note that both `TripAdvisor` and `GetYourGuide` implement strict anti-scraping measures and frequently update their DOM structures (specifically `div` identifiers).

- **Selector Volatility:** Frequent changes to website layouts may cause the current scraper selectors to become obsolete.

- **Temporal Decay:** As these scripts were developed three weeks ago, subsequent platform updates may have impacted the reliability of the scraping logic.