Web Scraping with Web Scraper

DATA 351: Data Management with SQL

Lucas P. Cordova, Ph.D.

2026-04-08

This session introduces the Web Scraper browser extension for Chromium-based browsers. We follow the official "Scraping a site" workflow (sitemap, selectors, scrape, export), then walk through a live-style demo: paginated Discogs search results, a Link selector into each release detail page, and Text selectors for community statistics on that page (for example Have, Want, and rating summaries). Primary reference: Web Scraper documentation.

Table of contents

1 Web Scraping in DATA 351

1.1 Why this tool?

You already know SQL is for data you control. Much of the world's data still lives in HTML pages. For **structured extraction at human scale** (assignments, prototypes, one-off exports),

a point-and-click scraper inside the browser keeps you close to the page and avoids writing a full crawler on day one.

This lecture uses **Web Scraper** (Chromium extension from the Chrome Web Store). It is not the only option, but it matches our goals: **sitemap**, **selector tree**, **preview**, **CSV export**.

1.2 What you should be able to do after today

By the end of class, you will be able to:

- Install the extension and open it from Developer Tools
- Create a **sitemap** with one or more **start URLs**, including **numeric ranges** for numbered pages
- Build a **selector tree** (for example, Link selectors with child Text selectors)
- Run a scrape, tune **request interval** and **page load delay**, then **browse** and **export CSV**
- Sketch a **multi-page Discogs search** workflow that **follows links to release pages** and pulls **detail-page statistics**, and name the ethical constraints (terms of service, robots, politeness)

1.3 Official documentation (read alongside the slides)

The steps and screenshots in this deck follow the official guide:

- **Scraping a site** (start URL, ranges, selectors, scrape, export) 1

Keep that page open in a tab while you work. Figures such as the **news site** example (selector tree with a Link selector and a child Text selector) are shown there in full.

2 Part 1: Install and open

2.1 Install the extension

Install **Web Scraper** from the Chrome Web Store (works in Chromium-based browsers that allow extensions):

- Web Scraper in the Chrome Web Store

After installation, **restart the browser** or only use the tool in tabs opened **after** installation so the extension loads cleanly. Requirements are listed under **Installation** 2.

2.2 Open Developer Tools and the Web Scraper tab

Web Scraper lives inside **Developer Tools**, not only as a toolbar icon.

Shortcuts:

Platform	Open Developer Tools
Windows / Linux	`Ctrl+Shift+I` or `F12`
macOS	`Cmd+Opt+I`

Then open the **Web Scraper** tab inside the tools panel. **Open Web Scraper** includes a figure for Chrome 3.

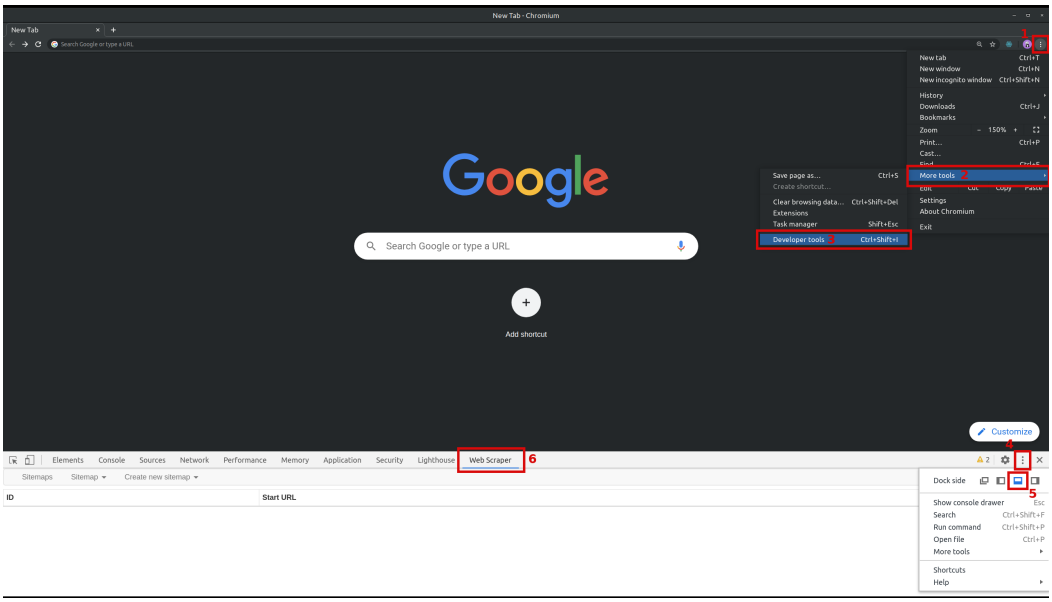


Figure 1: Web Scraper tab in Chrome Developer Tools (documentation figure).

3 Part 2: Create a sitemap

3.1 Start URL

A **sitemap** is your recipe for one scrape job. The first setting is the **start URL**: the page where crawling begins. You can add **multiple** start URLs (for example, several search queries) using the + control next to the URL field. After creation, start URLs also appear under **Edit metadata** in the sitemap menu 1.

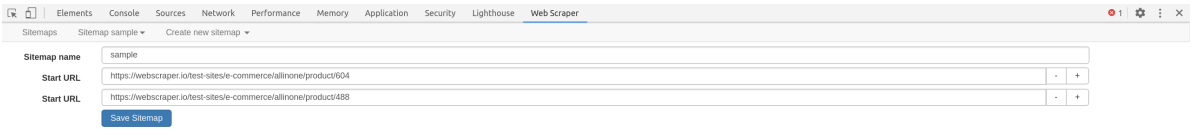


Figure 2: Sitemap editor: start URL field and + to add more start URLs (documentation figure).

3.2 Numeric ranges in the start URL

When page URLs contain a number, you can replace that segment with a **range** instead of listing every page by hand [1]:

Pattern	Example meaning
`[1-100]`	Pages 1 through 100
`[001-100]`	Zero-padded, e.g. `001`, `002`, …
`[0-100:10]`	Step by 10: `0`, `10`, `20`, …

Examples from the docs:

- `https://example.com/page/[1-3]` yields `/page/1`, `/page/2`, `/page/3`
- `https://example.com/page/[001-100]` matches three-digit paths
- `https://example.com/page/[0-100:10]` yields every tenth value

4 Part 3: Create selectors

4.1 Selector tree and order

Selectors are organized in a **tree**. The extension runs them in tree order: parent selectors run first, then children on the pages those parents open [1].

Classic pattern from the documentation:

- **Link selector** on a listing page: collect many links (for example, every article link)
- **Child Text selector** on each article page: pull the fields you need from that page

Use **Element preview** and **Data preview** when you build each selector so you know the CSS selection matches real nodes.

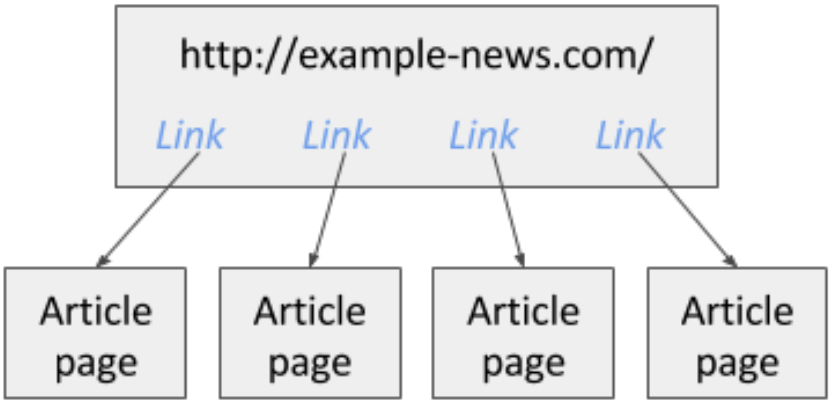


Figure 3: Example listing page with article links (documentation news-site example).

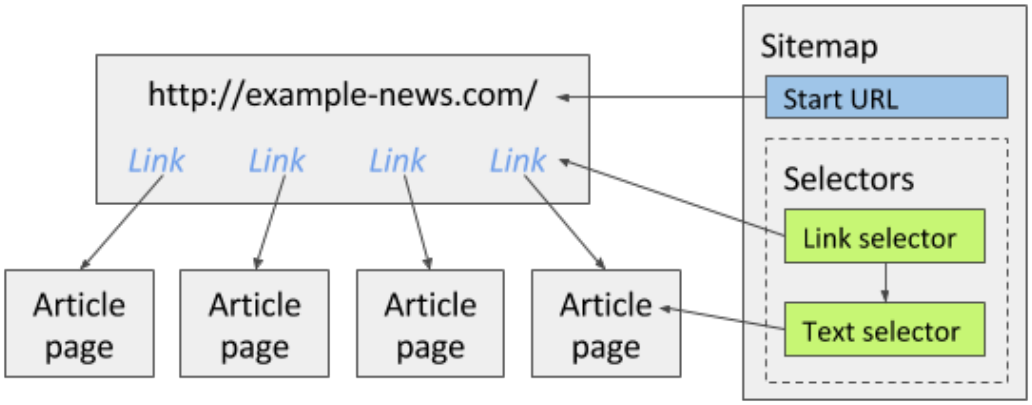


Figure 4: Selector tree for that site: Link selector, then child Text on each article page (documentation figure).

4.2 Core selector types to read next

The docs recommend being comfortable with at least **Link selector** and **Text selector** 4, 5. Link selectors follow `href` values and pass child selectors to the destination page. If clicking a result **does not** change the URL (heavy AJAX), read **Pagination selector** instead of forcing a Link selector 4.

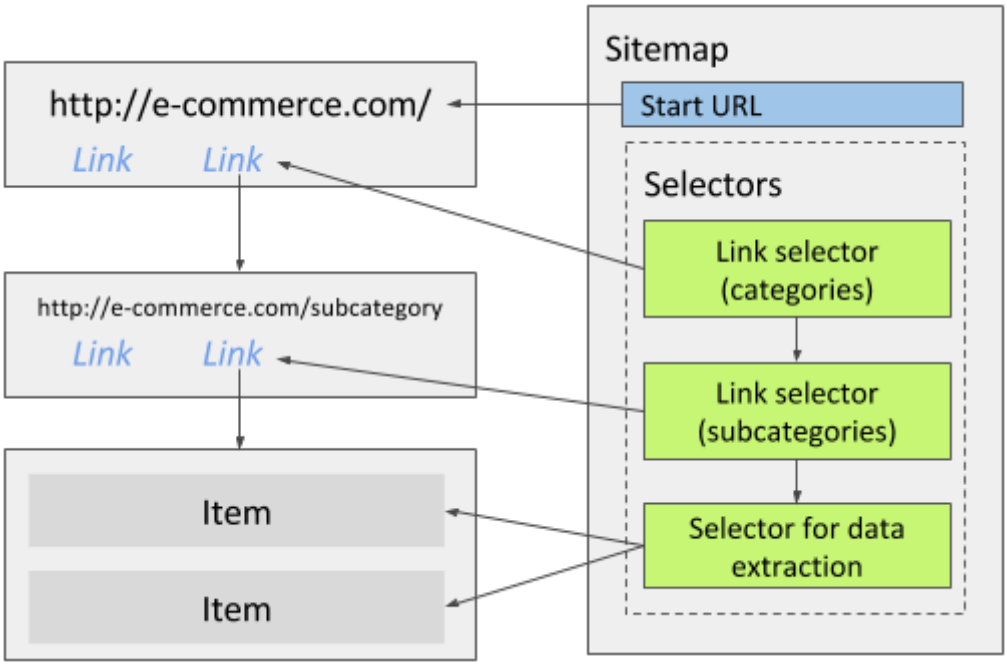


Figure 5: Nested Link selectors for multi-level navigation (documentation figure).

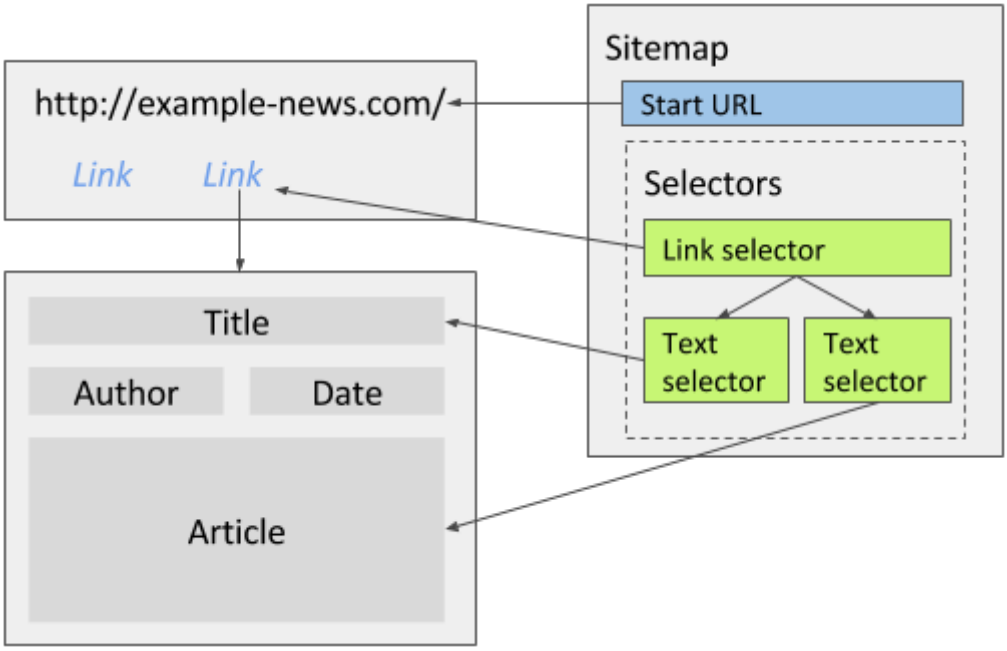


Figure 6: Several Text selectors defined for one page (documentation figure).

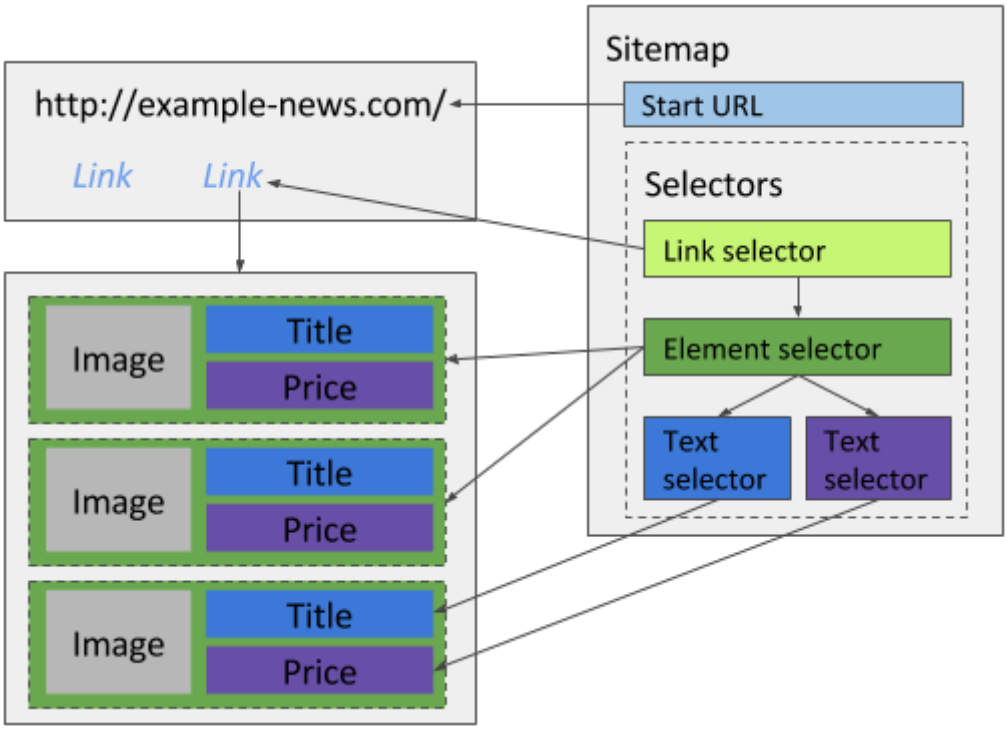


Figure 7: Multiple elements matched by Text selectors, and one selector returning many rows (documentation figures).

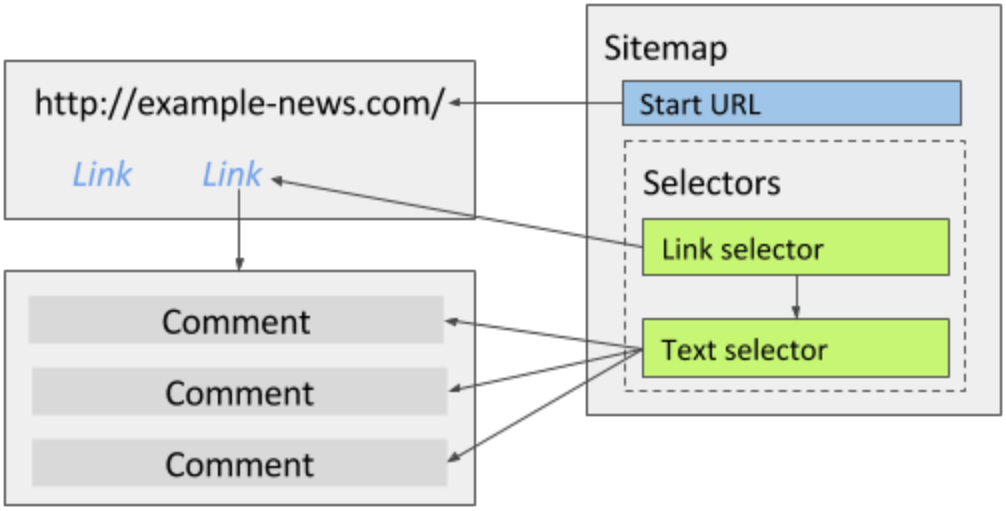
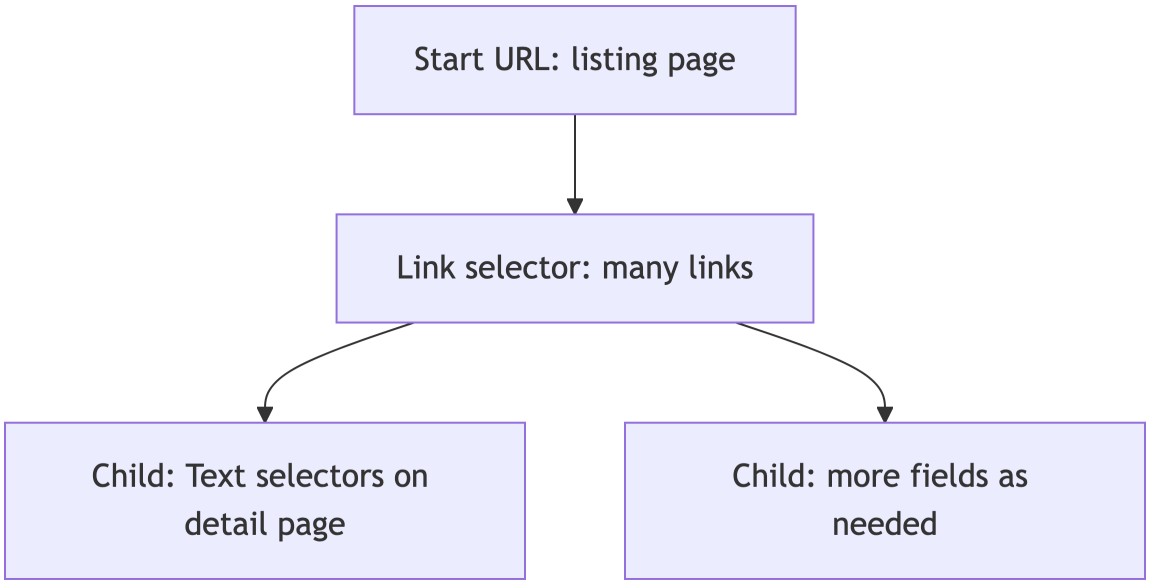


Figure 8: Text selector with multiple matches per page (documentation figure).



5 Part 4: Run the scrape and export

5.1 Scrape panel

When the sitemap is ready, open the **Scrape** panel and start the job. A **popup window** loads pages and extracts rows. When it finishes, the popup closes and you get a completion notice 1.

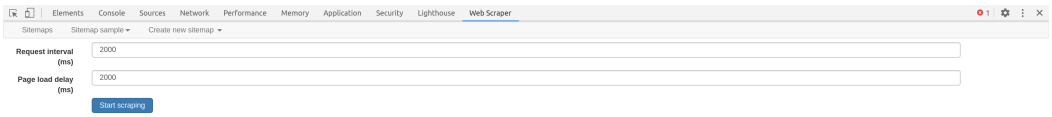


Figure 9: Scrape panel: start the job from the Scrape view (documentation figure).

Two knobs matter for fragile sites:

- **Request interval**: minimum time between HTTP requests (be polite; reduces load and bans)
- **Page load delay**: wait after load before running selectors (helps with late-rendered content)

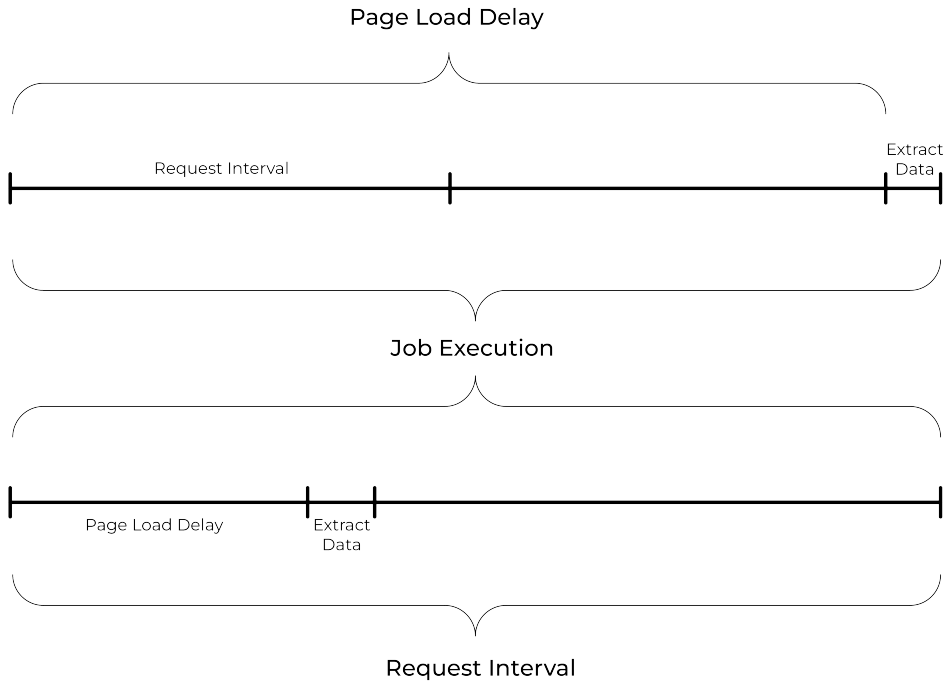


Figure 10: Request interval and page load delay fields in the Scrape panel (documentation figure).

5.2 Browse and CSV

After scraping:

- **Browse** panel: inspect extracted rows in the tool
- **Export data as CSV**: download for spreadsheets or later loading into SQL

6 Part 5: Demo

6.1 Goal and URL

Goal: build one table where each row is a **release** you care about. Rows should combine:

- **Listing fields** from the search results (for example title and artist line as shown on the card), and

- **Statistics from the release detail page** after you follow the link (for example community **Have** and **Want** counts, **average rating**, and **number of ratings**, or other stats visible in the header or sidebar)

Discogs lays these out in HTML that can change; use **Element preview** and **Data preview** on a real release page to lock selectors.

Start from **at least five** search result pages for **releases**, sorted by community **have** count (descending):

`https://www.discogs.com/search?sort=have%2Cdesc&type=release`

Pagination: Discogs search uses a `page` query parameter. For five pages, a range start URL matches the documentation pattern 1:

```
1 https://www.discogs.com/search?sort=have%2Cdesc&type=release&page=[1-5]
```

If your browser copies the URL with a leading slash path variant, keep the query string identical aside from `page=[1-5]`. You can instead add **five separate start URLs** (`page=1` … `page=5`) using the + URL field.

6.2 Ethics and housekeeping

Discogs is a real commercial community. For class:

- Stay within **Discogs Developer Terms** and general **Terms of Use**; do not republish bulk data or circumvent access controls 6, 7
- Prefer **moderate** request interval and delay; this is a teaching demo, not production mirroring
- Site HTML changes; if selectors break, update CSS with Element preview

6.3 Suggested sitemap (recommended: listing plus detail statistics)

Primary path for class (search, then each release page):

1. **Start URL** with `page=[1-5]` as above (or five explicit `page=` URLs).
2. **Link selector** (**multiple** enabled) whose CSS matches the **anchor** for each search result that points to a **`/release/`** URL. This opens the **detail** page for each album. Name it clearly (for example `release_link`).
3. As **children of that Link selector**, add **Text** selectors that run **on the release page**, not on the search grid. Aim for at least:
   - One statistic for **Have** (community copies owned)
   - One for **Want** (community want list)

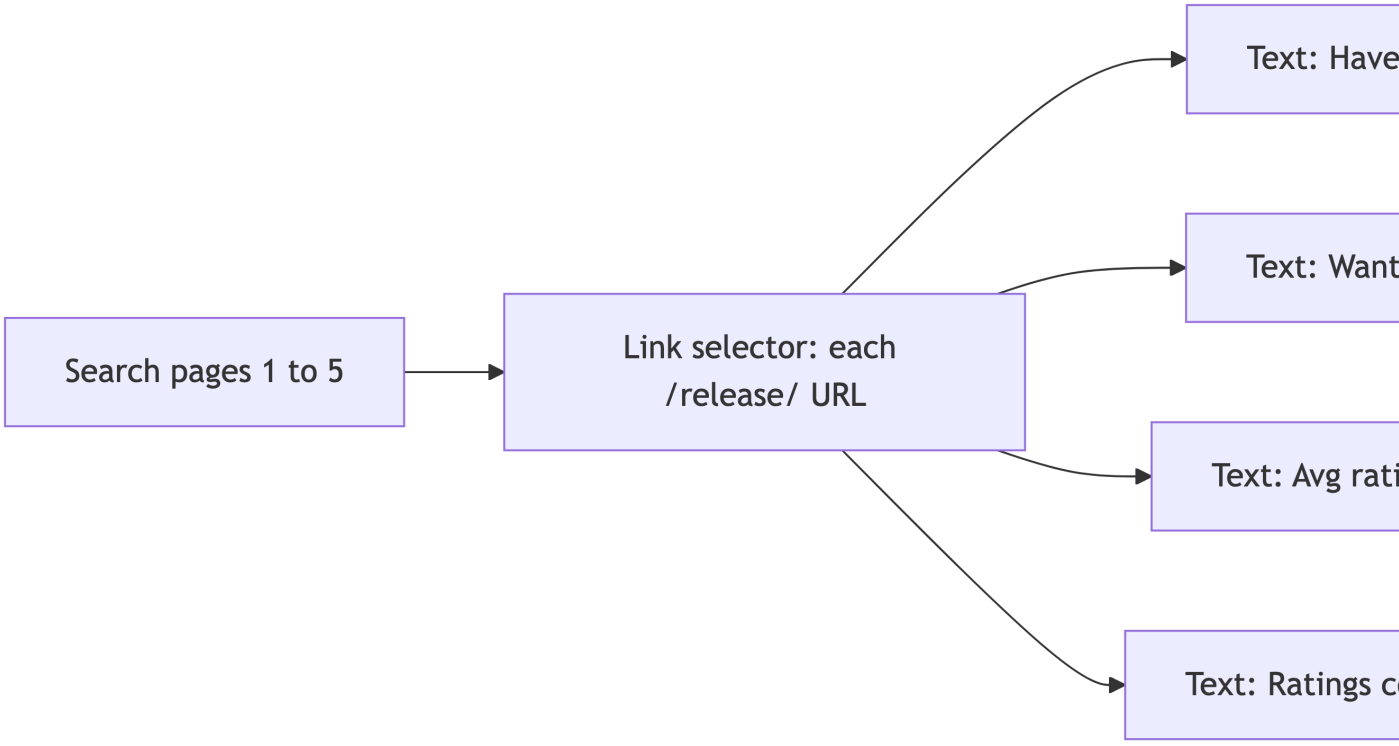
- **Average rating** and **rating count**, if shown, or substitute other numeric fields you see in the page chrome (for example **Last sold** or **Lowest** price) if the layout differs that day

4. **Optional sibling Text selectors** under the same Link (still on the detail page) for title or catalog number if you want columns that only appear on the detail view.

Optional warm-up (listing only): before you add the Link selector, you can add **Text** selectors on the **search** page scoped to each card to capture title and artist from the snippet. Those columns are redundant with the detail page for some fields, but they help verify selectors before you crawl deeper.

Tips:

- Detail URLs look like `https://www.discogs.com/release/...` . Avoid confusing them with **master** or **artist** links if multiple anchors appear in the same card; restrict the Link selector with a CSS filter or parent scope so you only enqueue releases.
- If a statistic is split across elements (label plus number), try a **parent Element** on the stat block, then a **Text** child, or one **Text** with a tighter selector. Re-check after **Data preview**.
- If result links behave like **in-page** loading without a stable URL, switch to **Pagination** or adjust link type per **Link selector** 4.



6.4 Live demo checklist

1. Open the search URL in a fresh tab; confirm you see results
2. Open a **single release** in another tab; note where **Have**, **Want**, and **rating** numbers live in the DOM for your Text selectors
3. Open Developer Tools, then the **Web Scraper** tab
4. **Create sitemap**, paste the start URL (range or five URLs)
5. Add the **Link** selector to `/release/` pages; then add **Text** children for **detail statistics** (and optional listing fields). Verify with **Element preview** / **Data preview** on both search and release views
6. **Scrape** with conservative timing; watch the popup until completion (detail pages mean **more** requests than listing-only)
7. **Browse** rows: confirm one row per release and non-empty stat columns
8. **Export CSV**

7 References

7.1 Sources

1. Web Scraper Documentation. **Scraping a site.** https://webscraper.io/documentation/scraping-a-site
2. Web Scraper Documentation. **Installation.** https://webscraper.io/documentation/installation
3. Web Scraper Documentation. **Open Web Scraper.** https://webscraper.io/documentation/open-web-scraper
4. Web Scraper Documentation. **Link selector.** https://webscraper.io/documentation/selectors/link-selector
5. Web Scraper Documentation. **Text selector.** https://webscraper.io/documentation/selectors/text-selector
6. Discogs. **Developer Terms of Service.** https://www.discogs.com/developers/terms
7. Discogs. **Terms of Use.** https://www.discogs.com/help/doc/terms-of-use

7.2 Figures from the Web Scraper documentation

Embedded screenshots are the same figures used on **Scraping a site** 1, **Open Web Scraper** 3, **Link selector** 4, and **Text selector** 5. Copies are stored under `data351/assets/images/webscraper-docs/` for this deck.

7.3 Video tutorials (from the docs)

Linked from the official pages 1, 3:

- How to create a sitemap
- How to add multiple start URLs
- How to open Web Scraper for the first time