Equijoin Operation (Experimental)

The Equijoin operation in DocETL is an experimental feature designed for joining two datasets based on flexible, LLM-powered criteria. It leverages many of the same techniques as the Resolve operation, but applies them to the task of joining datasets rather than deduplicating within a single dataset.

Motivation

While traditional database joins rely on exact matches, real-world data often requires more nuanced joining criteria. Equijoin allows for joins based on semantic similarity or complex conditions, making it ideal for scenarios where exact matches are impossible or undesirable.

🚀 Example: Matching Job Candidates to Job Postings

Let's explore a practical example of using the Equijoin operation to match job candidates with suitable job postings based on skills and experience.

```yaml
- name: match_candidates_to_jobs
  type: equijoin
  comparison_prompt: |
    Compare the following job candidate and job posting:

    Candidate Skills: {{ left.skills }}
    Candidate Experience: {{ left.years_experience }}

    Job Required Skills: {{ right.required_skills }}
    Job Desired Experience: {{ right.desired_experience }}

    Is this candidate a good match for the job? Consider both the overlap in
skills and the candidate's experience level. Respond with "True" if it's a
good match, or "False" if it's not a suitable match.
```

This Equijoin operation matches job candidates to job postings:

1. It uses the `comparison_prompt` to determine if a candidate is a good match for a job.
2. The operation can be optimized to use efficient blocking rules, reducing the number of comparisons.

Jinja2 Syntax with left and right

The prompt template uses Jinja2 syntax, allowing you to reference input fields directly (e.g., `left.skills`). You can reference the left and right documents using `left` and `right` respectively.

Performance Consideration

For large datasets, running comparisons with an LLM can be time-consuming. It's recommended to optimize your pipeline using `docetl build pipeline.yaml` to generate efficient blocking rules for the operation.

Blocking

Like the Resolve operation, Equijoin supports blocking techniques to improve efficiency. For details on how blocking works and how to implement it, please refer to the Blocking section in the Resolve operation documentation.

Adding Blocking Rules

Equijoin lets you specify **explicit blocking logic** to skip record pairs that are obviously unrelated *before* any LLM calls are made.

`blocking_keys`

Provide one or more **field names** for each side of the join. The selected values are concatenated and **embedded**; the cosine similarity of the left vs. right embeddings is then compared against `blocking_threshold` (defaults to `1.0`). If the similarity meets or exceeds that threshold, the pair moves on to the `comparison_prompt`; otherwise it is skipped.
If you omit `blocking_keys`, **all key–value pairs of each record are embedded by default**.

```yaml
blocking_keys:
  left:
    - medicine
  right:
    - extracted_medications
```

`blocking_threshold`

Optionally set a numeric `blocking_threshold` (0 – 1) representing the minimum cosine similarity (computed with the selected `embedding_model`) that the concatenated blocking keys must achieve to be considered a candidate pair. Anything below the threshold is filtered out without invoking the LLM.

```
blocking_threshold: 0.35
embedding_model: text-embedding-3-small
```

A full Equijoin step combining both ideas might look like:

```
- name: join_meds_transcripts
  type: equijoin
  blocking_keys:
    left:
      - medicine
    right:
      - extracted_medications
  blocking_threshold: 0.3535
  embedding_model: text-embedding-3-small
  comparison_prompt: |
    Compare the following medication names:

    {{ left.medicine }}

    {{ right.extracted_medications }}

    Determine if these entries refer to the same medication.
```

Auto-generating Rules (Experimental)

`docetl build pipeline.yaml` can call the **Optimizer** to propose `blocking_keys` and an appropriate `blocking_threshold` based on a sample of your data. This feature is experimental; always review the suggested rules to ensure they do not exclude valid matches.

Parameters

Equijoin shares many parameters with the Resolve operation. For a detailed list of required and optional parameters, please see the Parameters section in the Resolve operation documentation.

Key differences for Equijoin include:

- `resolution_prompt` is not used in Equijoin.
- `limits` parameter is specific to Equijoin, allowing you to set maximum matches for each left and right item.

Incorporating Into a Pipeline

Here's an example of how to incorporate the Equijoin operation into a pipeline using the job candidate matching scenario:

```yaml
model: gpt-4o-mini

datasets:
  candidates:
    type: file
    path: /path/to/candidates.json
  job_postings:
    type: file
    path: /path/to/job_postings.json

operations:
  - name: match_candidates_to_jobs:
    type: equijoin
    comparison_prompt: |
      Compare the following job candidate and job posting:

      Candidate Skills: {{ left.skills }}
      Candidate Experience: {{ left.years_experience }}

      Job Required Skills: {{ right.required_skills }}
      Job Desired Experience: {{ right.desired_experience }}

      Is this candidate a good match for the job? Consider both the overlap in skills and the candidate's experience level. Respond with "True" if it's a good match, or "False" if it's not a suitable match.

pipeline:
  steps:
    - name: match_candidates_to_jobs
      operations:
        - match_candidates_to_jobs:
            left: candidates
            right: job_postings

  output:
    type: file
    path: "/path/to/matched_candidates_jobs.json"
```

This pipeline configuration demonstrates how to use the Equijoin operation to match job candidates with job postings. The pipeline reads candidate and job posting data from JSON files, performs the matching using the defined comparison prompt, and outputs the results to a new JSON file.

Best Practices

1. **Leverage the Optimizer**: Use `docetl build pipeline.yaml` to automatically generate efficient blocking rules for your Equijoin operation.

2. **Craft Thoughtful Comparison Prompts**: Design prompts that effectively determine whether two records should be joined based on your specific use case.
3. **Balance Precision and Recall**: When optimizing, consider the trade-off between catching all potential matches and reducing unnecessary comparisons.
4. **Mind Resource Constraints**: Use `limit_comparisons` if you need to cap the total number of comparisons for large datasets.
5. **Iterate and Refine**: Start with a small sample of your data to test and refine your join criteria before running on the full dataset.

For additional best practices that apply to both Resolve and Equijoin operations, see the Best Practices section in the Resolve operation documentation.