Chroma DB Filtering

Estimated Reading Time: 15 minutes

Filtering in **Chroma DB** fundamentally differs from traditional SQL-based filtering due to its emphasis on **vector similarity** and flexible **metadata querying**. While SQL databases rely on structured schemas and declarative logic to retrieve exact matches, Chroma DB is designed for **unstructured data** and **semantic search**, making it well-suited for AI-driven applications.

Chroma DB supports two primary types of filtering:

Filter Type	Description	Comparison to SQL
Metadata Filtering	Filters based on document metadata, such as `"topic": "history"` or `"date": "2023-01-15"`.	Similar to SQL `WHERE` clauses, but more flexible and can be combined with vector search.
Document Filtering	Filters based on document content using keyword presence (e.g., `$contains`, `$not_contains`).	Comparable to SQL's `CONTAINS` or `LIKE` operators, but more powerful when integrated with vector search.

This dual-filtering approach allows Chroma DB to support complex, context-aware queries that go beyond the capabilities of traditional relational databases.

> Document filtering in Chroma DB is also referred to as **full text search**.

Metadata Filtering in Chroma DB

Metadata filtering in Chroma DB can be performed by using the `where` parameter inside the `.query()`, `.get()`, or `.delete()` methods.

For basic metadata matches, where you want to find only equivalent matches, use the following syntax:

```
where={"key": "value"}
```

For instance, the following code will `get` only those documents within `collection` where the metadata `key` is exactly equal to `value` :

```
collection.get(
    where={"key": "value"}
)
```

Similar syntax can be used within `.query()` or `.delete()` methods.

For defining even more complex filters, Chroma DB supports the following metadata filtering operators:

`$eq` - equal to (string, int, float)
`$ne` - not equal to (string, int, float)
`$gt` - greater than (int, float)
`$gte` - greater than or equal to (int, float)
`$lt` - less than (int, float)
`$lte` - less than or equal to (int, float)

These operators can be applied using the following syntax, using `$eq` as an example:

```
where={"key": {"$eq": "value"}}
```

Note that the above is identical to the following:

```
where={"key": "value"}
```

In other words, not providing an operator is equivalent to using the `$eq` operator.

Combining different filters can be achieved using the `$and` and `$or` logical operators as follows:

```
collection.get(
    where={
        "$and": [
            {"key": {"$eq": "value1"}},
            {"key": {"$ne": "value2"}}
        ]
    }
)
```

The above would `get` only the documents where `key` is equal to `value1` *and* `key` is not equal to `value2` from `collection`. Similar syntax can be used for `$or` as well as the `.query()` and `.delete()` methods.

Finally, to use lists in filters, use the `$in` and `$nin` operators, where we note that `$nin` stands for "not in":

```
        where={"key": {"$nin":["value1", "value2"]}}
```

The above code will find all documents where `key` is not equal to either `value1` or `value2`.

📝 Document Filtering in Chroma DB

Document filtering in Chroma DB can be performed by supplying `$contains` or `$not_contains` to the `where_document` parameter inside the `.query()`, `.get()`, or `.delete()` methods using the following syntax:

```
        where_document={"$contains":"value"}
```

The above would find all documents that contain `value` in the text of the document.

Moreover, note that you can combine multiple document filters using the `$and` and `$or` document operators in an analogous way to the metadata filters.

A full example of metadata and document filtering in Chroma DB

The following presents a full example of metadata and document filtering in Chroma DB.

🔧 Setup

The following commands import `chromadb` and its embedding utilities, then create an embedding function object used to generate vector embeddings:

```
import chromadb
from chromadb.utils import embedding_functions
ef = embedding_functions.SentenceTransformerEmbeddingFunction(
    model_name="all-MiniLM-L6-v2"
)
```

📂 Creating Collections

Collections help organize your data in Chroma DB.

Create a Collection

```
client = chromadb.Client()
collection = client.create_collection(
    name="filter_demo",
    metadata={"description": "Used to demo filtering in ChromaDB"},
    configuration={
        "embedding_function": ef
    }
)
print(f"Collection created: {collection.name}")
```

Output:

```
Collection created: filter_demo
```

Adding Documents to Collections

Use `add` to insert documents with optional metadata.

```
collection.add(
    documents=[
        "This is a document about LangChain",
        "This is a reading about LlamaIndex",
        "This is a book about Python",
        "This is a document about pandas",
        "This is another document about LangChain"
    ],
    metadatas=[
        {"source": "langchain.com", "version": 0.1},
        {"source": "llamaindex.ai", "version": 0.2},
        {"source": "python.org", "version": 0.3},
        {"source": "pandas.pydata.org", "version": 0.4},
        {"source": "langchain.com", "version": 0.5},
    ],
    ids=["id1", "id2", "id3", "id4", "id5"]
)
```

Filter using Metadata

The following finds all documents where the source is "langchain.com":

```
collection.get(
    where={"source": {"$eq": "langchain.com"}}
)
```

output:

```
{'ids': ['id1', 'id5'],
 'embeddings': None,
 'documents': ['This is a document about LangChain',
  'This is another document about LangChain'],
 'uris': None,
 'included': ['metadatas', 'documents'],
 'data': None,
 'metadatas': [{'source': 'langchain.com', 'version': 0.1},
  {'version': 0.5, 'source': 'langchain.com'}]}
```

The above produced correct output, but suppose we were only interested in LangChain documents with versions less than 0.3. The following finds all documents where the source is "langchain.com" with versions less than 0.3:

```
collection.get(
    where={
        "$and": [
            {"source": {"$eq": "langchain.com"}},
            {"version": {"$lt": 0.3}}
        ]
    }
)
```

Output:

```
{'ids': ['id1'],
 'embeddings': None,
 'documents': ['This is a document about LangChain'],
 'uris': None,
 'included': ['metadatas', 'documents'],
 'data': None,
 'metadatas': [{'source': 'langchain.com', 'version': 0.1}]}
```

Now, let's make an even more complicated filtering rule, one that combines logical operators with lists in filters. The following retrieves all documents about LangChain and LlamaIndex with a version less than 0.3:

```
collection.get(
    where={
        "$and": [
            {"source": {"$in": ["langchain.com", "llamaindex.ai"]}},
            {"version": {"$lt": 0.3}}
        ]
    }
)
```

Output:

```
{'ids': ['id1', 'id2'],
 'embeddings': None,
 'documents': ['This is a document about LangChain',
  'This is a reading about LlamaIndex'],
 'uris': None,
 'included': ['metadatas', 'documents'],
 'data': None,
 'metadatas': [{'source': 'langchain.com', 'version': 0.1},
  {'source': 'llamaindex.ai', 'version': 0.2}]}
```

Filter using Document Content

Suppose we wanted to find documents that include the word "pandas" in the text. The following performs a full text search for such documents:

```
collection.get(
    where_document={"$contains":"pandas"}
)
```

Output:

```
{'ids': ['id4'],
 'embeddings': None,
 'documents': ['This is a document about pandas'],
 'uris': None,
 'included': ['metadatas', 'documents'],
 'data': None,
 'metadatas': [{'source': 'pandas.pydata.org', 'version': 0.4}]}
```

💡 Document filtering is case-sensitive in Chroma DB. Therefore, searching for "Pandas" will not find any documents.

🏷️ + 📝 Combine Metadata and Document Content Filters

Of course, we can combine metadata and document filters. The following looks for all documents containing "LangChain" or "Python" with version numbers greater than 0.1:

```
collection.get(
    where={"version": {"$gt": 0.1}},
    where_document={
        "$or": [
            {"$contains": "LangChain"},
            {"$contains": "Python"}
        ]
    }
)
```

Output:

```
{'ids': ['id3', 'id5'],
 'embeddings': None,
 'documents': ['This is a book about Python',
  'This is another document about LangChain'],
 'uris': None,
 'included': ['metadatas', 'documents'],
 'data': None,
 'metadatas': [{'version': 0.3, 'source': 'python.org'},
  {'source': 'langchain.com', 'version': 0.5}]}
```

✅ Wrap-up

In this reading, you explored how to implement Chroma DB's robust metadata and document filtering features to selectively include or exclude documents based on a predefined set of criteria. Metadata filtering allows you to target documents using structured attributes like tags, topics, or timestamps, while document filtering enables you to search within the content itself—either through keyword matching or semantic relevance. Together, these filtering techniques empower you to build more precise, context-aware queries that enhance the quality and relevance of your search results in AI-driven applications.

Author(s)

[Wojciech "Victor" Fulmyk](#)