Collection Objects

```
class Collection(BaseModel)
```

count

```
def count() -> int
```

The total number of embeddings added to the database

Returns:

- `int` - The total number of embeddings added to the database

add

```
def add(ids: OneOrMany[ID],
        embeddings: Optional[OneOrMany[Embedding]] = None,
        metadatas: Optional[OneOrMany[Metadata]] = None,
        documents: Optional[OneOrMany[Document]] = None) -> None
```

Add embeddings to the data store.

Arguments:

- `ids` - The ids of the embeddings you wish to add
- `embeddings` - The embeddings to add. If None, embeddings will be computed based on the documents using the embedding_function set for the Collection. Optional.
- `metadatas` - The metadata to associate with the embeddings. When querying, you can filter on this metadata. Optional.
- `documents` - The documents to associate with the embeddings. Optional.

Returns:

None

Raises:

- `ValueError` - If you don't provide either embeddings or documents
- `ValueError` - If the length of ids, embeddings, metadatas, or documents don't match
- `ValueError` - If you don't provide an embedding function and don't provide embeddings
- `ValueError` - If you provide both embeddings and documents
- `ValueError` - If you provide an id that already exists

get

```
def get(ids: Optional[OneOrMany[ID]] = None,
        where: Optional[Where] = None,
        limit: Optional[int] = None,
        offset: Optional[int] = None,
        where_document: Optional[WhereDocument] = None,
        include: Include = ["metadatas", "documents"]) -> GetResult
```

Get embeddings and their associate data from the data store. If no ids or where filter is provided returns all embeddings up to limit starting at offset.

Arguments:

- `ids` - The ids of the embeddings to get. Optional.
- `where` - A Where type dict used to filter results by. E.g. `{"color" : "red", "price": 4.20}` . Optional.
- `limit` - The number of documents to return. Optional.
- `offset` - The offset to start returning results from. Useful for paging results with limit. Optional.
- `where_document` - A WhereDocument type dict used to filter by the documents. E.g. `{$contains: {"text": "hello"}}` . Optional.
- `include` - A list of what to include in the results. Can contain `"embeddings"` , `"metadatas"` , `"documents"` . Ids are always included. Defaults to `["metadatas", "documents"]` . Optional.

Returns:

- `GetResult` - A GetResult object containing the results.

peek

```
def peek(limit: int = 10) -> GetResult
```

Get the first few results in the database up to limit

Arguments:

- `limit` - The number of results to return.

Returns:

- `GetResult` - A GetResult object containing the results.

query

```
def query(
        query_embeddings: Optional[OneOrMany[Embedding]] = None,
        query_texts: Optional[OneOrMany[Document]] = None,
        n_results: int = 10,
        where: Optional[Where] = None,
        where_document: Optional[WhereDocument] = None,
        include: Include = ["metadatas", "documents",
                           "distances"]) -> QueryResult
```

Get the n_results nearest neighbor embeddings for provided query_embeddings or query_texts.

Arguments:

- `query_embeddings` - The embeddings to get the closes neighbors of. Optional.
- `query_texts` - The document texts to get the closes neighbors of. Optional.
- `n_results` - The number of neighbors to return for each query_embedding or query_texts. Optional.
- `where` - A Where type dict used to filter results by. E.g. `{"color" : "red", "price": 4.20}` . Optional.
- `where_document` - A WhereDocument type dict used to filter by the documents. E.g. `{$contains: {"text": "hello"}}` . Optional.

- `include` - A list of what to include in the results. Can contain `"embeddings"` , `"metadatas"` , `"documents"` , `"distances"` . Ids are always included. Defaults to `["metadatas", "documents", "distances"]` . Optional.

Returns:

- `QueryResult` - A QueryResult object containing the results.

Raises:

- `ValueError` - If you don't provide either query_embeddings or query_texts
- `ValueError` - If you provide both query_embeddings and query_texts

modify

```
def modify(name: Optional[str] = None,
           metadata: Optional[CollectionMetadata] = None) -> None
```

Modify the collection name or metadata

Arguments:

- `name` - The updated name for the collection. Optional.
- `metadata` - The updated metadata for the collection. Optional.

Returns:

None

update

```
def update(ids: OneOrMany[ID],
           embeddings: Optional[OneOrMany[Embedding]] = None,
           metadatas: Optional[OneOrMany[Metadata]] = None,
           documents: Optional[OneOrMany[Document]] = None) -> None
```

Update the embeddings, metadatas or documents for provided ids.

Arguments:

- `ids` - The ids of the embeddings to update
- `embeddings` - The embeddings to add. If None, embeddings will be computed based on the documents using the embedding_function set for the Collection. Optional.
- `metadatas` - The metadata to associate with the embeddings. When querying, you can filter on this metadata. Optional.
- `documents` - The documents to associate with the embeddings. Optional.

Returns:

None

upsert

```
def upsert(ids: OneOrMany[ID],
           embeddings: Optional[OneOrMany[Embedding]] = None,
           metadatas: Optional[OneOrMany[Metadata]] = None,
           documents: Optional[OneOrMany[Document]] = None) -> None
```

Update the embeddings, metadatas or documents for provided ids, or create them if they don't exist.

Arguments:

- `ids` - The ids of the embeddings to update
- `embeddings` - The embeddings to add. If None, embeddings will be computed based on the documents using the embedding_function set for the Collection. Optional.
- `metadatas` - The metadata to associate with the embeddings. When querying, you can filter on this metadata. Optional.
- `documents` - The documents to associate with the embeddings. Optional.

Returns:

None

delete

```
def delete(ids: Optional[IDs] = None,
           where: Optional[Where] = None,
             where_document: Optional[WhereDocument] = None) -> None
```

Delete the embeddings based on ids and/or a where filter

Arguments:

- `ids` - The ids of the embeddings to delete
- `where` - A Where type dict used to filter the delection by. E.g. `{"color" : "red", "price": 4.20}` . Optional.
- `where_document` - A WhereDocument type dict used to filter the deletion by the document content. E.g. `{$contains: {"text": "hello"}}` . Optional.

Returns:

None

Edit this page