日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2023年8月1日火曜日

Llama.cpp_serverのOpenAI互換APIとChromaを使ってIn-context Learningを行なう

以前の記事で、Llama_cpp.serverのOpenAI互換APIとベクトル・データベースとしてPineconeを使ったIn-context Learningを行なうAPEXアプリケーションを作成しました。そのPineconeの部分をChromaに置き換えてみました。

ベクトル・データベースを置き換えただけなので、APEXアプリケーション自体にほとんど変更はありません。

置き換えたAPEXアプリケーションのエクスポートは以下になります。

https://github.com/ujnak/apexapps/blob/master/exports/vector-documents-search-chroma.zip

ChromaにAPEXからアクセスするためのパッケージCHROMA_APIを作成しています。JavaScript APIを参考にしていますが、実装を省略した部分は多々あります。(例えば引数としてstringまたはstring[]を受け取る部分をstring[]に限定していたりします)。

```
create or replace package chroma_api
as
/**
 * PL/SQL package to call Chroma REST API.
 *
 * Ref:
 * https://github.com/chroma-core/chroma/tree/main/clients/js/src
 */

/**
 * Returns the version of the Chroma API.
 *
 * @param {p_server} Chroma API Server
 */
function version(
    p_server in varchar2
)
return varchar2;

/**
 * Returns a heartbeat from the Chroma API.
 *
```

```
 * @param {p_server} Chroma API Server
 * @returns {number} nanosecond heartbeat
 */
function heartbeat(
    p_server in varchar2
)
return number;

/**
 * Creates a new collection with the specified properties.
 *
 * @param {p_server} Chroma API Server
 * @param {p_name} The name of the collection.
 * @param {p_metadata} Optional metadata associated with the collection.
 * @param {p_embedding_function} currently ignored.
 * @returns {varchar2} id of created collection.
 *
 * hnsw:space and description usually in metadata.
 * Valid options for hnsw:space are "l2", "ip, "or "cosine". The default is "l2".
 * ----
 * metadata = { "metadata": {
 *   "hnsw:space": "cosine",
 *   "description"; "description"
 * };
 * ----
 */
function create_collection(
    p_server              in varchar2
    ,p_name               in varchar2
    ,p_metadata           in varchar2 default null
    ,p_embedding_function in varchar2 default null
)
return clob;

/* return id of created collection */
function create_collection_id(
    p_server              in varchar2
    ,p_name               in varchar2
    ,p_metadata           in varchar2 default null
    ,p_embedding_function in varchar2 default null
)
return varchar2;

/**
 * Gets or creates a collection with the specified properties.
 */
function get_or_create_collection(
```

```
    p_server              in varchar2
    ,p_name               in varchar2
    ,p_metadata           in varchar2 default null
    ,p_embedding_function in varchar2 default null
)
return clob;

/* return id of get or created collection */
function get_or_create_collection_id(
    p_server              in varchar2
    ,p_name               in varchar2
    ,p_metadata           in varchar2 default null
    ,p_embedding_function in varchar2 default null
)
return varchar2;

/**
 * Gets a collection with the specified name.
 */
function get_collection(
    p_server in varchar2
    ,p_name  in varchar2
)
return clob;

/* return id of get collection */
function get_collection_id(
    p_server in varchar2
    ,p_name  in varchar2
)
return varchar2;

/**
 * List all collections.
 */
function list_collections(
    p_server in varchar2
)
return clob;

/**
 * Deletes a collection with the specified name.
 */
function delete_collection(
    p_server in varchar2
    ,p_name  in varchar2
)
```

```
return boolean;

/**
 * Add items to the collection
 */
function add_items(
    p_server         in varchar2
    ,p_collection_id in varchar2
    ,p_ids           in json_array_t default null
    ,p_embeddings    in json_array_t default null
    ,p_metadatas     in json_array_t default null
    ,p_documents     in json_array_t default null
)
return clob;

/**
 * Upsert items to the collection
 */
function upsert_items(
    p_server         in varchar2
    ,p_collection_id in varchar2
    ,p_ids           in json_array_t default null
    ,p_embeddings    in json_array_t default null
    ,p_metadatas     in json_array_t default null
    ,p_documents     in json_array_t default null
)
return clob;

/**
 * Update the embeddings, documents, and/or metadatas of existing items
 */
function update_items(
    p_server         in varchar2
    ,p_collection_id in varchar2
    ,p_ids           in json_array_t default null
    ,p_embeddings    in json_array_t default null
    ,p_metadatas     in json_array_t default null
    ,p_documents     in json_array_t default null
)
return clob;

/**
 * Count the number of items in the collection
 */
function count_items(
    p_server         in varchar2
    ,p_collection_id in varchar2
```

```
)
return number;

/**
 * Modify the collection name or metadata
 */
function modify_collection(
    p_server         in varchar2
    ,p_collection_id in varchar2
    ,p_name          in varchar2 default null
    ,p_metadata      in varchar2 default null
)
return clob;

/**
 * Get items from the collection
 *
 * {
 *   ids: ["id1", "id2"],
 *   where: { "key": "value" },
 *   limit: 10,
 *   offset: 0,
 *   include: ["embeddings", "metadatas", "documents"],
 *   whereDocument: { $contains: "value" },
 * }
 */
function get_items(
    p_server          in varchar2
    ,p_collection_id  in varchar2
    ,p_ids            in json_array_t default null
    ,p_where          in varchar2     default null
    ,p_limit          in number       default null
    ,p_offset         in number       default null
    ,p_include        in json_array_t default null
    ,p_where_document in varchar2     default null
)
return clob;

/**
 * Performs a query on the collection using the specified parameters.
 */
function query_items_by_embeddings(
    p_server          in varchar2
    ,p_collection_id  in varchar2
    ,p_query_embeddings in json_array_t
    ,p_n_results        in number       default 1
    ,p_where            in varchar2     default null
```

```
    ,p_include          in json_array_t default null
)
return clob;

/**
 * Peek inside the collection
 */
function peek_items(
    p_server          in varchar2
    ,p_collection_id  in varchar2
    ,p_limit          in number
)
return clob;

/**
 * Deletes items from the collection.
 */
function delete_items(
    p_server           in varchar2
    ,p_collection_id   in varchar2
    ,p_ids             in json_array_t default null
    ,p_where           in varchar2     default null
    ,p_where_document  in varchar2     default null
)
return varchar2;

end chroma_api;
/

create or replace package body chroma_api
as
C_PATH constant varchar2(10) := '/api/v1/';

/* version */
function version(
    p_server in varchar2
)
return varchar2
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'version';
    l_response clob;
    e_version_failed exception;
begin
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'GET'
```

```
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_version_failed;
    end if;
    return l_response;
end version;

/* heartbeat */
function heartbeat(
    p_server in varchar2
)
return number
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'heartbeat';
   l_response clob;
   l_response_json json_object_t;
   l_heartbeat number;
   e_heartbeat_failed exception;
 begin
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'GET'
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_heartbeat_failed;
    end if;
    l_response_json := json_object_t(l_response);
    l_heartbeat := l_response_json.get_number('nanosecond heartbeat');
    return l_heartbeat;
end heartbeat;

/* create collection */
function create_collection(
    p_server              in varchar2
    ,p_name               in varchar2
    ,p_metadata           in varchar2 default null -- JSON
    ,p_embedding_function in varchar2 default null
)
return clob
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections';
    l_request  clob;
    l_response clob;
    e_create_collection_failed exception;
```

```
begin
    select json_object(
        key 'name' value p_name,
        key 'metadata' value p_metadata format json
    ) into l_request
    from dual;
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'POST'
        ,p_body => l_request
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_create_collection_failed;
    end if;
    return l_response;
end create_collection;

/* create collection id */
function create_collection_id(
    p_server              in varchar2
    ,p_name               in varchar2
    ,p_metadata           in varchar2 default null -- JSON
    ,p_embedding_function in varchar2 default null
)
return varchar2
as
    l_id varchar2(40);
    l_response clob;
    l_response_json json_object_t;
begin
    l_response := create_collection(
        p_server    => p_server
        ,p_name     => p_name
        ,p_metadata => p_metadata
        ,p_embedding_function => p_embedding_function
    );
    l_response_json := json_object_t(l_response);
    l_id := l_response_json.get_string('id');
    return l_id;
end create_collection_id;

/* get_or_create_collection */
function get_or_create_collection(
    p_server              in varchar2
    ,p_name               in varchar2
```

```
    ,p_metadata           in varchar2 default null -- JSON
    ,p_embedding_function in varchar2 default null
)
return clob
as
    l_response clob;
begin
    l_response := get_collection(
        p_server => p_server
        ,p_name  => p_name
    );
    return l_response;
exception
    when others then
        l_response := create_collection(
            p_server => p_server
            ,p_name  => p_name
            ,p_metadata => p_metadata
            ,p_embedding_function => p_embedding_function
        );
        return l_response;
end get_or_create_collection;

function get_or_create_collection_id(
    p_server in varchar2
    ,p_name  in varchar2
    ,p_metadata           in varchar2 default null -- JSON
    ,p_embedding_function in varchar2 default null
)
return varchar2
as
    l_response clob;
    l_response_json json_object_t;
    l_id varchar2(40);
begin
    l_response := get_or_create_collection(
        p_server => p_server
        ,p_name => p_name
        ,p_metadata => p_metadata
        ,p_embedding_function => p_embedding_function
    );
    l_response_json := json_object_t(l_response);
    l_id := l_response_json.get_string('id');
    return l_id;
end get_or_create_collection_id;

/* get collection */
```

```
function get_collection(
    p_server in varchar2
    ,p_name  in varchar2
)
return clob
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections/' || p_name;
    l_response clob;
    e_get_collection_failed exception;
begin
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'GET'
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
       raise e_get_collection_failed;
    end if;
    return l_response;
end get_collection;

function get_collection_id(
    p_server in varchar2
    ,p_name  in varchar2
)
return varchar2
as
    l_response clob;
    l_response_json json_object_t;
    l_id varchar2(40);
begin
    l_response := get_collection(
        p_server => p_server
        ,p_name => p_name
    );
    l_response_json := json_object_t(l_response);
    l_id := l_response_json.get_string('id');
    return l_id;
end get_collection_id;

/* list all collections */
function list_collections(
    p_server in varchar2
)
return clob
as
```

```
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections';
    l_response clob;
    e_list_collections_failed exception;
begin
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'GET'
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_list_collections_failed;
    end if;
    return l_response;
end list_collections;

/* delete collection */
function delete_collection(
    p_server in varchar2
    ,p_name  in varchar2
)
return boolean
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections/' || p_name;
    l_response clob;
    l_status boolean;
    e_delete_collection_failed exception;
begin
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'DELETE'
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_delete_collection_failed;
    end if;
    l_status := l_response = 'null';
    return l_status;
end delete_collection;

/* validate */
function validate(
    p_require_embeddings_or_documents in boolean
    ,p_ids        in json_array_t -- string | string[]
    ,p_embeddings in json_array_t -- number[] | number[][] | undefined
    ,p_metadatas  in json_array_t -- object | object[]
```

```
    ,p_documents  in json_array_t -- string | string[]
)
return boolean
as
    l_ids_c pls_integer;
    l_ids   varchar2(4000);
begin
    /*
     * Currently, no embedding function supported.
     * Therefore, p_embeddings must have, p_documents is ignored.
     */
    if p_embeddings is null then
        apex_debug.info('embeddings missed.');
        return false;
    end if;

    /* skip: validate all id are string */
    l_ids_c := p_ids.get_size();

    /* ids, embeddings, metadatas, and documents must all be the same length */
    if (p_embeddings is not null and l_ids_c <> p_embeddings.get_size())
    or (p_metadatas  is not null and l_ids_c <> p_metadatas.get_size())
    then
        apex_debug.info('ids, embeddings, metadatas, and documents must all be the same length.
        return false;
    end if;

    /* skip: validate all ids are unique */

    return true;
end validate;

/**
 * add/upsert/update items to the collection
 */
function op_items_common(
    p_operation      in varchar2
    ,p_server        in varchar2
    ,p_collection_id in varchar2
    ,p_ids           in json_array_t
    ,p_embeddings    in json_array_t
    ,p_metadatas     in json_array_t
    ,p_documents     in json_array_t
)
return clob
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections/' || p_collection_i
```

```
    l_request clob;
    l_request_json json_object_t;
    l_response clob;
    p_parameter_invalid  exception;
    p_add_vectors_failed exception;
begin
    if not validate(
        p_require_embeddings_or_documents => true
        ,p_ids        => p_ids
        ,p_embeddings => p_embeddings
        ,p_metadatas  => p_metadatas
        ,p_documents  => p_documents
    ) then
        raise p_parameter_invalid;
    end if;

    l_request_json := json_object_t;
    if p_ids is not null then
        l_request_json.put('ids', p_ids);
    end if;
    if p_embeddings is not null then
        l_request_json.put('embeddings', p_embeddings);
    end if;
    if p_metadatas is not null then
        l_request_json.put('metadatas', p_metadatas);
    end if;
    if p_documents is not null then
        l_request_json.put('documents', p_documents);
    end if;
    l_request := l_request_json.to_clob();
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'POST'
        ,p_body => l_request
    );
    if apex_web_service.g_status_code not in (200,201) then
        apex_debug.info(l_response);
        raise p_add_vectors_failed;
    end if;
    return l_response;
end op_items_common;

/* add */
function add_items(
    p_server        in varchar2
    ,p_collection_id in varchar2
```

```
    ,p_ids           in json_array_t
    ,p_embeddings    in json_array_t
    ,p_metadatas     in json_array_t
    ,p_documents     in json_array_t
)
return clob
as
begin
    return op_items_common(
        p_operation => 'add'
        ,p_server => p_server
        ,p_collection_id => p_collection_id
        ,p_ids           => p_ids
        ,p_embeddings    => p_embeddings
        ,p_metadatas     => p_metadatas
        ,p_documents     => p_documents
    );
end add_items;

/* upsert */
function upsert_items(
    p_server        in varchar2
    ,p_collection_id in varchar2
    ,p_ids           in json_array_t
    ,p_embeddings    in json_array_t
    ,p_metadatas     in json_array_t
    ,p_documents     in json_array_t
)
return clob
as
begin
    return op_items_common(
        p_operation => 'upsert'
        ,p_server => p_server
        ,p_collection_id => p_collection_id
        ,p_ids           => p_ids
        ,p_embeddings    => p_embeddings
        ,p_metadatas     => p_metadatas
        ,p_documents     => p_documents
    );
end upsert_items;

/* update */
function update_items(
    p_server        in varchar2
    ,p_collection_id in varchar2
    ,p_ids           in json_array_t
```

```
    ,p_embeddings    in json_array_t
    ,p_metadatas     in json_array_t
    ,p_documents     in json_array_t
)
return clob
as
begin
    return op_items_common(
        p_operation => 'update'
        ,p_server => p_server
        ,p_collection_id => p_collection_id
        ,p_ids           => p_ids
        ,p_embeddings    => p_embeddings
        ,p_metadatas     => p_metadatas
        ,p_documents     => p_documents
    );
end update_items;

/* count */
function count_items(
    p_server         in varchar2
    ,p_collection_id in varchar2
)
return number
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections/' || p_collection_i
    l_response clob;
    e_count_vectors_failed exception;
begin
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'GET'
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_count_vectors_failed;
    end if;
    return to_number(l_response);
end count_items;

/* modify */
function modify_collection(
    p_server         in varchar2
    ,p_collection_id in varchar2
    ,p_name          in varchar2
    ,p_metadata      in varchar2
```

```
)
return clob
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections/' || p_collection_i
    l_request_json json_object_t;
    l_request clob;
    l_response clob;
    e_modify_collection_failed exception;
begin
    l_request_json := json_object_t();
    if p_name is not null then
        l_request_json.put('new_name', p_name);
    end if;
    if p_metadata is not null then
        l_request_json.put('new_metadata', json_object_t(p_metadata));
    end if;
    l_request := l_request_json.to_clob();
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'PUT'
        ,p_body => l_request
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_modify_collection_failed;
    end if;
    return l_response;
end modify_collection;

/* get */
function get_items(
    p_server          in varchar2
    ,p_collection_id  in varchar2
    ,p_ids            in json_array_t
    ,p_where          in varchar2
    ,p_limit          in number
    ,p_offset         in number
    ,p_include        in json_array_t
    ,p_where_document in varchar2
)
return clob
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections/' || p_collection_i
    l_request_json json_object_t;
    l_request clob;
    l_response clob;
```

```
    e_get_collection_failed exception;
begin
    l_request_json := json_object_t();
    if p_ids is not null then
        l_request_json.put('ids', p_ids);
    end if;
    if p_where is not null then
        l_request_json.put('where', json_object_t(p_where));
    end if;
    if p_limit is not null then
        l_request_json.put('limit', p_limit);
    end if;
    if p_offset is not null then
        l_request_json.put('offset', p_offset);
    end if;
    if p_include is not null then
        l_request_json.put('include', p_include);
    end if;
    if p_where_document is not null then
        l_request_json.put('whereDocument', json_object_t(p_where_document));
    end if;
    l_request := l_request_json.to_clob();
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url          => C_ENDPOINT
        ,p_http_method => 'POST'
        ,p_body        => l_request
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_get_collection_failed;
    end if;
    return l_response;
end get_items;

/* query */
function query_items_by_embeddings(
    p_server          in varchar2
    ,p_collection_id  in varchar2
    ,p_query_embeddings in json_array_t
    ,p_n_results        in number
    ,p_where            in varchar2
    ,p_include          in json_array_t
)
return clob
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections/' || p_collection_i
```

```
    l_request_json json_object_t;
    l_request clob;
    l_response clob;
    e_query_collection_failed exception;
begin
    l_request_json := json_object_t();
    l_request_json.put('query_embeddings', p_query_embeddings);
    l_request_json.put('n_results', p_n_results);
    if p_where is not null then
        l_request_json.put('where', json_object_t(p_where));
    end if;
    if p_include is not null then
        l_request_json.put('include', p_include);
    end if;
    l_request := l_request_json.to_clob();
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'POST'
        ,p_body => l_request
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_query_collection_failed;
    end if;
    return l_response;
end query_items_by_embeddings;

/* peek */
function peek_items(
    p_server          in varchar2
    ,p_collection_id  in varchar2
    ,p_limit           in number
)
return clob
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections/' || p_collection_i
    l_request clob;
    l_response clob;
    e_peek_collection_failed exception;
begin
    select json_object(
        key 'limit' value p_limit
    ) into l_request from dual;
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
```

```
        ,p_http_method => 'POST'
        ,p_body => l_request
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_peek_collection_failed;
    end if;
    return l_response;
end peek_items;

/* delete */
function delete_items(
    p_server           in varchar2
    ,p_collection_id   in varchar2
    ,p_ids             in json_array_t
    ,p_where           in varchar2
    ,p_where_document  in varchar2
)
return varchar2
as
    C_ENDPOINT constant varchar2(200) := p_server || C_PATH || 'collections/' || p_collection_i
    l_request clob;
    l_request_json json_object_t;
    l_response clob;
    e_delete_collection_failed exception;
begin
    l_request_json := json_object_t();
    if p_ids is not null then
        l_request_json.put('ids', p_ids);
    end if;
    if p_where is not null then
        l_request_json.put('where', json_object_t(p_where));
    end if;
    if p_where_document is not null then
        l_request_json.put('whereDocument', p_where_document);
    end if;
    l_request := l_request_json.to_clob();
    apex_web_service.set_request_headers('Content-Type', 'application/json');
    l_response := apex_web_service.make_rest_request(
        p_url => C_ENDPOINT
        ,p_http_method => 'POST'
        ,p_body => l_request
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_delete_collection_failed;
    end if;
```

```
    return l_response;
end delete_items;

end chroma_api;
/
```

chorma_api.sql hosted with ❤️ by GitHub view raw

パッケージKB_LLM_UTILのPineconeの呼び出し部分を、パッケージCHROMA_APIのファンクションの呼び出しに置き換えています。

```
create or replace package kb_llm_util as
/**
 * OpenAIのembeddingとchat/completions APIを呼び出すように
 * 改変したパッケージ。
 *
 * Llama_cpp.serverのOpenAI互換APIを使用しているので、本家の
 * OpenAIのAPIで変更が必要なところもある可能性はあります。
 */

/**
 * 表KB_DOCUMENTSのBLOB列CONTENTをCLOB列のCONTENT_TEXTに
 * 単純にコピーする。
 *
 * PDFをソースにするといったことはせず、MIMEタイプはtext/plainを想定
 * している。
 */
procedure apply_auto_filter(
  p_id in number
);

/**
 * 表KB_DOCUMENTSの列CONTENT_TEXTに保存されている文章をチャンクに分割する。
 * 分割されたチャンクは表KB_CHUNKSに保存される。
 *
 * LlamaIndexでいうところのNode Parserに該当する処理を行う。
 * https://gpt-index.readthedocs.io/en/latest/core_modules/data_modules/node_parsers/root.html
 *
 * どのような形でチャンクに分割するかはとても重要。以下では、
 * CHR(10) || '---' || CHR(10) または CHR(10) || '===' || CHR(10) で
 * チャンクに分割する。
 */
procedure split_into_chunks(
    p_id in number
    ,p_primary_separator   in varchar2
    ,p_secondary_separator in varchar2
    ,p_limit in number default 4000
);
```

```
/**
 * 表KB_CHUNKSに保存した列CHUNKのベクトル埋め込みを作成する。
 * OpenAIの/v1/embeddingを呼び出す。
 * 実際はLlama_cpp.serverで、Llama2の7bは4096、13Bは5120の
 * 次元のベクトル埋め込みを生成する。
 *
 * 生成したベクトル埋め込みは列EMBEDDINGに保存する。
 */
procedure generate_embeddings(
    p_id in number
    ,p_collection_name in varchar2 default 'EMBEDDINGS'
    ,p_model_name      in varchar2 default 'text-embedding-ada-002'
    ,p_endpoint        in varchar2 default null
    ,p_cred_id         in varchar2 default null
);

/**
 * 生成したベクトル埋め込みをPineconeのインデックスにUpsertする。
 */
procedure upsert_vectors(
    p_id        in number
    ,p_endpoint in varchar2
    ,p_index    in varchar2
);

/**
 * 質問の送信と回答の表示。
 *
 * 質問の文字列のベクトル埋め込みを生成し、Pineconeのインデックスを検索する。
 * 回答数はp_top_kで指定する。検索結果のチャンクは、scoreの良い順番で連結する。
 *
 * LlamaIndexでいうところのResponse Synthesizerに該当する処理を行う。
 * https://gpt-index.readthedocs.io/en/latest/core_modules/query_modules/response_synthesizers/
 * OpenAIのAPIであれば、LlamaIndexでのpromptは以下。
 * https://gpt-index.readthedocs.io/en/latest/core_modules/model_modules/prompts.html
 *
 * プロンプトの生成方法については、暫定的なもので要調整。
 */
procedure ask(
    p_question in varchar2
    ,p_prompt_system in varchar2
    ,p_top_k   in number
    ,p_index   in varchar2
    ,p_answer  out varchar2
    ,p_question_id out number
    ,p_score_limit   in number    default 0
```

```
    ,p_model_name    in varchar2  default 'text-embedding-ada-002'
    ,p_endpoint      in varchar2  default null
    ,p_cred_id       in varchar2  default null
    ,p_generate_model_name in varchar2 default 'gpt-3.5-turbo'
    ,p_temperature         in number   default 0.9
    ,p_max_tokens          in number   default 256
    ,p_prompt_template in clob
);

/**
 * 削除された文書のベクトルをPineconeのインデックスから削除する。
 */
procedure delete_vectors(
    p_id        in number
    ,p_endpoint in varchar2
    ,p_index    in varchar2
);
end;
/

create or replace package body kb_llm_util as
C_OPENAI_API_TIMEOUT constant number := 360; -- 6 min.

/**
 * Extract text string from BLOB column.
 */
procedure apply_auto_filter(
    p_id in number
)
as
    l_content       kb_documents.content%type;
    l_content_text  kb_documents.content_text%type;
    l_is_failed     kb_documents.is_failed%type := 'N';
begin
    /* assume mime type is 'text/plain' so simply convert blob to clob. */
    update kb_documents set content_text = to_clob(content), is_failed = 'N'
    where id = p_id;
end apply_auto_filter;

/**
 * Split document into chunks.
 */
procedure split_into_chunks(
    p_id in number
    ,p_primary_separator   in varchar2
    ,p_secondary_separator in varchar2
    ,p_limit in number
```

```
)
as
    l_content_text kb_documents.content_text%type;
    l_chunk        kb_chunks.chunk%type;
    l_seq          kb_chunks.seq%type;
    l_split_chars  kb_chunks.split_chars%type;
    l_split        number;
begin
    /* Delete chunks currently exists for update. */
    delete from kb_chunks where document_id = p_id;
    select content_text into l_content_text from kb_documents where id = p_id;

    l_seq := 1;
    while true
    loop
        l_split_chars := p_primary_separator;
        l_split := instr(l_content_text, l_split_chars);
        if (l_split > p_limit) or (l_split = 0 and length(l_content_text) > p_limit) then
            l_split_chars := p_secondary_separator;
            l_split := instr(l_content_text, l_split_chars);
            if (l_split > p_limit) or (l_split = 0 and length(l_content_text) > p_limit) then
                l_split_chars := '';
                l_split := p_limit;
            end if;
        end if;
        if l_split = 0 then
            l_chunk := trim(l_content_text);
            if length(l_chunk) > 0 then
                insert into kb_chunks(document_id, seq, chunk, split_chars) values(p_id, l_seq,
            end if;
            exit;
        else
            l_chunk := trim(substr(l_content_text, 1, l_split));
            -- dbms_output.put_line('START CHUNK');
            -- dbms_output.put_line(l_chunk);
            insert into kb_chunks(document_id, seq, chunk, split_chars) values(p_id, l_seq, l_c
            l_content_text := substr(l_content_text, l_split+length(l_split_chars));
            l_seq := l_seq + 1;
        end if;
    end loop;
end split_into_chunks;

/**
 * Generate each embedding from chunks.
 */
procedure generate_embeddings(
    p_id in number
```

```
    ,p_collection_name in varchar2
    ,p_model_name      in varchar2
    ,p_endpoint        in varchar2
    ,p_cred_id         in varchar2
)
as
    l_count   number;
    l_request clob;
    l_request_json json_object_t;
    l_texts        json_array_t;
    l_response clob;
    l_response_json json_object_t;
    l_data          json_array_t;
    l_embedding_obj json_object_t;
    l_embedding     json_array_t;
    l_embedding_clob clob;
    l_chunk_id kb_chunks.id%type;
    e_llm_embed_failed exception;
begin
    while true
    loop
        /* exit if no candidate for generating embedding. */
        select count(*) into l_count from kb_chunks where embedding is null and document_id = p
        if l_count = 0 then
            exit;
        end if;
        /* select only 1 chunk to generate embeddings. */
        apex_collection.create_or_truncate_collection(p_collection_name);
        l_count := 1;
        for r in (
            select id, chunk from kb_chunks where embedding is null and document_id = p_id
        )
        loop
            apex_collection.add_member(
                p_collection_name => p_collection_name
                ,p_n001 => r.id
                ,p_clob001 => r.chunk
            );
            l_count := l_count + 1;
            if l_count > 1 then
                exit;
            end if;
        end loop;
        apex_collection.resequence_collection(p_collection_name);
        /* create a request body for OpenAI embedding. */
        l_texts := json_array_t();
        for r in (
```

```
            select clob001 from apex_collections where collection_name = p_collection_name orde
        )
        loop
            l_texts.append(r.clob001);
        end loop;
        l_request_json := json_object_t();
        l_request_json.put('model', p_model_name);
        l_request_json.put('input', l_texts);
        l_request := l_request_json.to_clob;
        apex_debug.info(l_request);
        /* call OpenAI Embedding */
        apex_web_service.clear_request_headers;
        apex_web_service.set_request_headers('Accept','application/json', p_reset => false);
        apex_web_service.set_request_headers('Content-Type','application/json', p_reset => fals
        l_response := apex_web_service.make_rest_request(
            p_url => p_endpoint || '/v1/embeddings'
            ,p_http_method => 'POST'
            ,p_body => l_request
            ,p_credential_static_id => p_cred_id
            ,p_transfer_timeout => C_OPENAI_API_TIMEOUT
        );
        if apex_web_service.g_status_code <> 200 then
            apex_debug.info(l_response);
            raise e_llm_embed_failed;
        end if;
        /* update embedding */
        l_response_json := json_object_t(l_response);
        l_data := l_response_json.get_array('data');
        for i in 1..l_data.get_size
        loop
            l_embedding_obj := json_object_t(l_data.get(i-1));
            l_embedding := l_embedding_obj.get_array('embedding');
            l_embedding_clob := l_embedding.to_clob;
            select n001 into l_chunk_id from apex_collections where collection_name = p_collect
            update kb_chunks set embedding = l_embedding_clob where id = l_chunk_id;
        end loop;
    end loop;
end generate_embeddings;

/**
 * store embeddings in vector database.
 */
procedure upsert_vectors(
    p_id in number
    ,p_endpoint in varchar2
    ,p_index in varchar2
)
```

```
as
    l_count number;
    l_ids json_array_t;
    l_embeddings json_array_t;
    l_response clob;
begin
    while true
    loop
        /* exit if all embeddings are stored. */
        select count(*) into l_count from kb_chunks
        where document_id = p_id and embedding is not null and (is_indexed is null or is_indexe
        if l_count = 0 then
            exit;
        end if;
        /* store 10 embeddings in single upsert request */
        l_count := 1;
        l_ids        := json_array_t();
        l_embeddings := json_array_t();
        for r in (
            select id, embedding from kb_chunks
            where embedding is not null and (is_indexed is null or is_indexed <> 'Y')
        )
        loop
            l_ids.append(to_char(r.id));
            l_embeddings.append(json_array_t(r.embedding));
            update kb_chunks set is_indexed = 'Y' where id = r.id;
            l_count := l_count + 1;
            if l_count > 10 then
                exit;
            end if;
        end loop;
        /* Store to Chroma */
        l_response := chroma_api.upsert_items(
            p_server => p_endpoint
            ,p_collection_id => p_index
            ,p_ids => l_ids
            ,p_embeddings => l_embeddings
        );
        apex_debug.info(l_response);
    end loop;
 end upsert_vectors;

/**
 * Qusetion and Answer
 */
procedure ask(
     p_question     in varchar2
```

```
    ,p_prompt_system in varchar2
    ,p_top_k       in number
    ,p_index       in varchar2
    ,p_answer      out varchar2
    ,p_question_id out number
    ,p_score_limit   in number
    ,p_model_name    in varchar2
    ,p_endpoint      in varchar2
    ,p_cred_id       in varchar2
    ,p_generate_model_name in varchar2
    ,p_temperature   in number
    ,p_max_tokens    in number
    ,p_prompt_template in clob
)
as
   l_request        clob;
   l_request_json   json_object_t;
   l_response       clob;
   l_response_json  json_object_t;
   l_data           json_array_t;
   l_embedding_obj  json_object_t;
   l_embedding      json_array_t;
   l_embeddings     json_array_t;
   l_embedding_clob clob;
   l_question_id    kb_questions.id%type;
   /* Chroma */
   l_ids       json_array_t;
   l_distances json_array_t;
   /* search result */
   l_chunk_id varchar2(400);
   l_score    number;

   l_messages json_array_t;
   l_content_system  json_object_t;
   l_content_user    json_object_t;

   l_prompt           kb_responses.prompt%type;
   l_context_str      clob;
   l_generations      json_array_t;
   l_generated_answer kb_responses.generated_answer%type;

   l_choices json_array_t;
   l_message json_object_t;

   e_llm_embed_failed     exception;
   e_llm_generate_failed  exception;
   e_bad_prompt_type      exception;
```

```
begin
   /* generate embedding from question */
   select json_object(
       key 'input'     value p_question
       ,key 'model'    value p_model_name
   returning clob) into l_request from dual;
   apex_web_service.clear_request_headers;
   apex_web_service.set_request_headers('Accept','application/json', p_reset => false);
   apex_web_service.set_request_headers('Content-Type','application/json', p_reset => false);
   l_response := apex_web_service.make_rest_request(
       p_url => p_endpoint || '/v1/embeddings'
       ,p_http_method => 'POST'
       ,p_body => l_request
       ,p_credential_static_id => p_cred_id
       ,p_transfer_timeout => C_OPENAI_API_TIMEOUT
   );
   if apex_web_service.g_status_code <> 200 then
       apex_debug.info(l_response);
       raise e_llm_embed_failed;
   end if;
   l_response_json  := json_object_t.parse(l_response);
   l_data           := l_response_json.get_array('data');
   l_embedding_obj  := json_object_t(l_data.get(0));
   l_embedding      := l_embedding_obj.get_array('embedding');
   l_embedding_clob := l_embedding.to_clob;
   /* store question in table KB_QUESTIONS. */
   insert into kb_questions(question, embedding) values(p_question, l_embedding_clob)
   returning id into l_question_id;
   p_question_id := l_question_id;
   /*
    * query Chroma by embedding generated from the question.
    */
   l_embeddings := json_array_t();
   l_embeddings.append(l_embedding);
   l_response := chroma_api.query_items_by_embeddings(
       p_server => p_endpoint
       ,p_collection_id => p_index
       ,p_query_embeddings => l_embeddings
       ,p_n_results => p_top_k
   );
   /* store response from Chroma in table KB_ANSWERS. */
   l_response_json := json_object_t.parse(l_response);
   apex_debug.info(l_response);
   l_ids       := l_response_json.get_array('ids');
   l_ids       := json_array_t(l_ids.get(0));
   l_distances := l_response_json.get_array('distances');
   l_distances := json_array_t(l_distances.get(0));
```

```
for i in 1..l_ids.get_size
loop
    l_chunk_id  := l_ids.get_string(i-1);
    l_score     := l_distances.get_number(i-1);
    insert into kb_answers(question_id, chunk_id, score) values(l_question_id, l_chunk_id,
end loop;
/*
 * Create Prompt for OpenAI chat completions.
 */
l_context_str := '';
for r in (
    select c.chunk from kb_answers a join kb_chunks c on a.chunk_id = c.id
    where a.question_id = l_question_id and a.score > p_score_limit
    order by a.score desc
)
loop
    l_context_str := l_context_str || r.chunk;
end loop;
l_prompt := p_prompt_template;
l_prompt := replace(l_prompt, '{context_str}', l_context_str);
l_prompt := replace(l_prompt, '{query_str}', p_question);
/*
 * call OpenAI chat completions with the prompt.
 */
l_request_json := json_object_t();
l_messages := json_array_t();
l_content_system := json_object_t();
l_content_system.put('role','system');
l_content_system.put('content', p_prompt_system);
l_content_user   := json_object_t();
l_content_user.put('role','user');
l_content_user.put('content', l_prompt);
l_messages.append(l_content_system);
l_messages.append(l_content_user);
l_request_json.put('messages', l_messages);
l_request_json.put('temperature', p_temperature);
l_request_json.put('max_tokens', p_max_tokens);
l_request_json.put('model', p_generate_model_name);
l_request := l_request_json.to_clob;
apex_debug.info(l_request);
apex_web_service.clear_request_headers;
apex_web_service.set_request_headers('Accept','application/json', p_reset => false);
apex_web_service.set_request_headers('Content-Type','application/json', p_reset => false);
l_response := apex_web_service.make_rest_request(
    p_url => p_endpoint || '/v1/chat/completions'
    ,p_http_method => 'POST'
    ,p_body => l_request
```

```
        ,p_credential_static_id => p_cred_id
        ,p_transfer_timeout => C_OPENAI_API_TIMEOUT
    );
    if apex_web_service.g_status_code <> 200 then
        apex_debug.info(l_response);
        raise e_llm_generate_failed;
    end if;
    l_response_json := json_object_t(l_response);
    l_choices := l_response_json.get_array('choices');
    l_message := json_object_t(l_choices.get(0)).get_object('message');
    l_generated_answer := l_message.get_string('content');
    /*
     * store reponse generated by OpenAI chat/completions for further review.
     */
    insert into kb_responses(question_id, iteration, prompt, generated_answer)
    values(l_question_id, 1, l_prompt, l_generated_answer);
    p_answer := l_generated_answer;
end ask;

/* delete vectors from Chroma */
procedure delete_vectors(
    p_id        in number
    ,p_endpoint in varchar2
    ,p_index    in varchar2
)
as
    l_request      clob;
    l_request_json json_object_t;
    l_vectors      json_array_t;
    l_response     clob;
begin
    l_vectors := json_array_t();
    for r in (select id from kb_chunks where document_id = p_id)
    loop
        l_vectors.append(to_char(r.id));
    end loop;
    l_response := chroma_api.delete_items(
        p_server => p_endpoint
        ,p_collection_id => p_index
        ,p_ids => l_vectors
    );
    /* delete chuks of the document from kb_chunks. */
    delete from kb_chunks where document_id = p_id;
end delete_vectors;
end kb_llm_util;
/
```

kb_llm_util.sql hosted with ❤️ by GitHub view raw

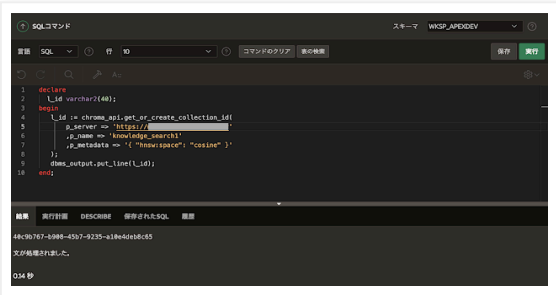
APEXアプリケーションの**置換文字列**の**G_INDEX**に、Chromaに作成したコレクションのIDを設定します。**G_ENDPOINT**として、LLama_cpp.serverとChromaが稼働しているホストを指すURLを設定します。

パッケージCHROMA_APIにはコレクションを作成するファンクションCREATE_COLLECTIONまたはGET_OR_CREATE_COLLECTIONが含まれています。そのファンクションを呼び出して、Chromaのコレクションを作成します。

metadataの**hnsw:space**に**cosine**を指定し、ベクトル検索時にコサイン類似度を使うようにします。

```
declare
  l_id varchar2(40);
begin
   l_id := chroma_api.get_or_create_collection_id(
       p_server => 'https://ホスト名'
       ,p_name => 'knowledge_search'
       ,p_metadata => '{ "hnsw:space": "cosine" }'
   );
   dbms_output.put_line(l_id);
end;
```

create_chroma_collection.sql hosted with ❤️ by GitHub view raw



作成されているコレクションを一覧するには、LIST_COLLECTIONSを呼び出します。

```
declare
    l_response clob;
begin
    /* list collections */
    l_response := chroma_api.list_collections(
        p_server => 'https://ホスト名'
```

```
    );
    dbms_output.put_line(l_response);
end;
/
```

list_chroma_collection.sql hosted with ❤️ by GitHub view raw

作成したコレクションを削除するには、DELETE_COLLECTIONを呼び出します。

```
declare
  l_success boolean;
begin
   l_success := chroma_api.delete_collection(
        p_server => 'https://ホスト名'
       ,p_name => 'knowledge_search1'
   );
end;
```

delete_collection.sql hosted with ❤️ by GitHub view raw

以上が、Chromaに切り替えた変更点です。

Oracle APEXのアプリケーション作成の参考になれば幸いです。

完

Yuji N. 時刻: 16:09

共有

‹ ホーム ›

ウェブ バージョンを表示

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示