Sorting and Pagination

The sorting module is available for advanced data sorting and pagination. Data can be sorted within a column, via association, or with a dynamic query.

- The feature was introduced here:https://github.com/blockscout/blockscout/pull/8611
- Features are implemented here:https://github.com/blockscout/blockscout/blob/master/apps/explorer/lib/explorer/sorting_helper.ex
- [An Example](#)
- is located at the bottom of this doc.

Ordering

To order any Ecto schema, use theExplorer.SortingHelper.apply_sorting/3 function. This function requires three arguments:

- The Ecto query to be sorted (query
-)
- User-provided sorting parameters (sorting
-)
- Default sorting parameters (default_sorting
-)

Sorting Parameters

Sorting parameters follow the typeExplorer.SortingHelper.sorting_params/0 , which can be one of the following tuples:

- {ordering, column}
- {ordering, column, binding}
- {:dynamic, column, ordering, Ecto.Query.dynamic_expr()}

Where:

- ordering
- can be:asc
- ,:asc_nulls_first
- ,:asc_nulls_last
- ,:desc
- ,:desc_nulls_first
- , or:desc_nulls_last
- .
- column
- is an atom representing the column name to sort by.
- binding
- is an atom denoting the association or binding to use for the column.
-

Pagination

Pagination is handled by theExplorer.SortingHelper.page_with_sorting/4 function in conjunction with ordering. It uses the providedpaging_options along with the sorting parameters to return a paginated query.

We use cursor based pagination because blockchain data may be quite large. After merging default and user provided sorting options we generate a dynamic query with a recursive condition for each column.

Blockscout employs cursor-based pagination for efficient handling of large blockchain datasets. This approach is facilitated by theExplorer.SortingHelper.page_with_sorting/4 function, which integrates sorting parameters with pagination options.

Cursor-Based Pagination

Cursor-based pagination is particularly suitable for blockchain data due to its potential size and frequent data updates. This method ensures that the same records are not retrieved more than once, even if new data is inserted between requests.

Pagination Implementation

Thepage_with_sorting function accepts the following arguments:

1. The Ecto query object.
2. Paging options that specify the cursor position and the page size.
3. User-provided sorting parameters.

4. Default sorting parameters.
5.

By combining the default and user-provided sorting options, a dynamic query is constructed. This includes 'where' conditions that are applied recursively for each column involved in the sorting. This ensures that the pagination cursor moves accurately through the sorted records.

Using this function, developers can create paginated endpoints that respond quickly and consistently, even with the ever-growing amount of blockchain data.

Example

```
Copy @default_sorting [ desc_nulls_last: :circulating_market_cap, desc_nulls_last: :fiat_value, desc_nulls_last:
:holder_count, asc: :name, asc: :contract_address_hash ]

...

@doc """ Lists the topt:__MODULE__.t/0's'. """ @speclist_top(String.t()|nil,[ Chain.paging_options() |
{:sorting,SortingHelper.sorting_params()} |{:token_type,[String.t()]} ])::[Token.t()] deflist_top(filter,options\[])do
paging_options=Keyword.get(options,:paging_options,Chain.default_paging_options())
token_type=Keyword.get(options,:token_type,nil) sorting=Keyword.get(options,:sorting,[])

query=from(tinToken,preload: [:contract_address])

sorted_paginated_query= query |>apply_filter(token_type) |>SortingHelper.apply_sorting(sorting,@default_sorting)
|>SortingHelper.page_with_sorting(paging_options,sorting,@default_sorting)

filtered_query= casefilter&&filter!==""&&Search.prepare_search_term(filter)do {:some,filter_term}-> sorted_paginated_query
|>where(fragment("to_tsvector('english', symbol || ' ' || name) @@ to_tsquery(?)",^filter_term))

_-> sorted_paginated_query end

filtered_query |>Chain.select_repo(options).all() end
```

User-provided sorting is obtained from the query parameters of a REST API call through thetokens_sorting/1 function:

```
Copy @spectokens_sorting(%{required(String.t())=>String.t()})::[{:sorting,SortingHelper.sorting_params()}]
deftokens_sorting(%{"sort"=>sort_field,"order"=>order})do [sorting:do_tokens_sorting(sort_field,order)] end

deftokens_sorting(_),do: []

defpdo_tokens_sorting("fiat_value","asc"),do: [asc_nulls_first: :fiat_value] defpdo_tokens_sorting("fiat_value","desc"),do:
[desc_nulls_last: :fiat_value] defpdo_tokens_sorting("holder_count","asc"),do: [asc_nulls_first: :holder_count]
defpdo_tokens_sorting("holder_count","desc"),do: [desc_nulls_last: :holder_count]
defpdo_tokens_sorting("circulating_market_cap","asc"),do: [asc_nulls_first: :circulating_market_cap]
defpdo_tokens_sorting("circulating_market_cap","desc"),do: [desc_nulls_last: :circulating_market_cap]
defpdo_tokens_sorting(,),do: []
```

This function converts query parameters into sorting parameters understood by the sorting module.

Last updated10 months ago