TYPO3.Flow - Feature # 8774

Status:	Resolved	**Priority:**	Should have
Author:	Bastian Waidelich	**Category:**	Persistence
Created:	2010-07-09	**Assigned To:**	Bastian Waidelich
Updated:	2010-10-20	**Due date:**	
PHP Version:			
Has patch:			
Complexity:			
Subject:	Query::execute() should return a Proxy instead of array		

Description

While discussing a widget concept (#8773) we realized, that it would be good to be able to access a query of a resultset before actually retrieving objects from the persistence.
The Query::execute() method could instead of an array return a **LazyQueryResultList** that contains the query object. Only if this list is accessed, the objects would be loaded. But it should be possible to retrieve the related query object without fetching the objects (e.g. through **LazyQueryResultList::getQuery()** or **persistenceManager->getQueryByResult($myLazyQueryResultList)**).

That would enable us for example to copy and modify a query without actually executing it.

Implications

- This will be a breaking change, cause existing array type hints will break

Related issues:

related to TYPO3.Flow - Feature # 6609: Implement joins for queries	**Closed**	
related to TYPO3.Fluid - Major Feature # 8773: Implement support for Widgets	**Resolved**	**2010-07-09**
related to TYPO3.Flow - Task # 9710: Refactor QueryResultProxy behaviour	**Resolved**	

Associated revisions

Revision 60fadf5c - 2010-08-30 14:15 - Bastian Waidelich

[!!!][+FEATURE] FLOW3 (Persistence): Added QueryResultProxy which is now the default return type of Query::execute().

This is a breaking change if you worked on the result set with an array_* function. To work around this issue, you can set the fetch mode to FETCH_ARRAY or convert the result via iterator_to_array().

Change-Id: Iba2e78e9df39f9098b7571e1747d775c37f44c38
Resolves: #8774

Revision 6ebe5814 - 2010-08-30 14:41 - Bastian Waidelich

[!!!][+FEATURE] FLOW3 (Persistence): Added QueryResultProxy which is now the default return type of Query::execute().

This is a breaking change if you worked on the result set with an array_* function. To work around this issue, you can set the fetch mode to FETCH_ARRAY or convert the result via iterator_to_array().

Change-Id: Iba2e78e9df39f9098b7571e1747d775c37f44c38
Resolves: #8774

Revision cb9c0461 - 2010-09-07 20:04 - Bastian Waidelich

[!!!][+FEATURE] FLOW3 (Persistence): Add QueryResultProxy

Query::execute() will return an QueryResultProxy by default now.
This is a breaking change if you worked on the result set with an
array_* function.
To work around this issue, you can set the fetch mode to
FETCH_ARRAY or convert the result via iterator_to_array().

Change-Id: Iba2e78e9df39f9098b7571e1747d775c37f44c38
Resolves: #8774

History

#1 - 2010-07-09 16:56 - Karsten Dambekalns

- Status changed from New to Accepted

- Assigned To set to Karsten Dambekalns

- Start date deleted (2010-07-09)

#2 - 2010-07-10 12:14 - Manuel Strausz

Just a few observations from me of what could break with existing code, since I recently implemented a similar class (it was more of an AR implementation, but the problem was similar):

- Like you already said, array type hints will break
- array_key_exists() will NEVER work on objects implementing the ArrayAccess interface (it doens't throw an error, but it will always return FALSE), you need to use isset(), which doesn't expose the same behavior unfortunately
- a lot of array-only functions obviously won't work at all

You probably already knew that, but just in case.. :D
Now, apart from that the real challenges of persistence-level pagination is the huge differences regarding the methods on the DB backends. It's relatively easy to do in MySQL, for SQL server it's pretty confusing and changed from version to version (originally they expected you to actually create a stored procedure for EVERY QUERY that uses pagination, horrific if you ask me...).

That said, the performance implications of actually implementing it on the persistence level can be staggering, a big improvement if done right. You will probably have to set priorities for which backend you support though.

Anyway, an interface for paginators that worked pretty well for me in the past looked like the following:

- setItemsPerPage(itemsPerPage) - needs to be set before actual query, reasonable default value is 10. Setting it to 0 could be made a trigger for returning the list without pagination.
- setCurrentPage(currentPage) - set before actual query, default value 1.
- getNumItemsTotal() - needs to implicitly query the number of items total, in mysql this can be done in a performant way if you did "SELECT SQL_CALC_FOUND_ROWS ..." on any select with a LIMIT clause ... of course, for other backends, it's a different story
- getNumPages() - can be calculated simply by ceil(numItemsTotal/itemsPerPage), obviously
- getResultsForPage(page) - to query specific pages directly
- setOrderBy(orderBy) - this heavily affects pagination since the list needs to be flushed, meaning you need to query again

Probably not the right place here for lengthy discussions, so I'll keep it at that for now.

#3 - 2010-07-10 14:47 - Sebastian Kurfuerst

Hey,

> - *Like you already said, array type hints will break*
> - *array_key_exists() will NEVER work on objects implementing the ArrayAccess interface (it doens't throw an error, but it will always return FALSE), you need to use isset(), which doesn't expose the same behavior unfortunately*
> - *a lot of array-only functions obviously won't work at all*

You are right, but I'd change it nevertheless :-) It will just enable lots of new functionality.
If anybody knew if it was possible to get an **unique ID** from an array which we can use to directly identify a given array, we could also use Arrays; but we would lose the "lazy-loading" features. Thus, I'd change that, altough it is a breaking change (after all, FLOW3 is not yet stable).

> *Now, apart from that the real challenges of persistence-level pagination is the huge differences regarding the methods on the DB backends. It's relatively easy to do in MySQL, for SQL server it's pretty confusing and changed from version to version (originally they expected you to actually create a stored procedure for EVERY QUERY that uses pagination, horrific if you ask me...).*

Well, we have the Query Object Model which is not bound to a particular storage-backend. Thus, at this point this is not a problem I think -- but has to be solved inside the particular storage backend.

> *Anyway, an interface for paginators that worked pretty well for me in the past looked like the following:*

Thanks for your input, we will however not directly implement this inside the persistence logic, but rather provide a separate Widget for that. See issue #8773 for details.

Thanks again for your input,
Sebastian

#4 - 2010-07-10 15:23 - Manuel Strausz

Sebastian Kurfuerst wrote:

> *You are right, but I'd change it nevertheless :-) It will just enable lots of new functionality.*
> *If anybody knew if it was possible to get an **unique ID** from an array which we can use to directly identify a given array, we could also use Arrays; but we would lose the "lazy-loading" features. Thus, I'd change that, altough it is a breaking change (after all, FLOW3 is not yet stable).*

I understand, and of course it's valid to do a breaking change at this stage. I just tried to think of the possible issues that could arise. In practical terms there are not many though - you usually don't do much else than loop over the returned data or access an element by key after a query. For that, a lazy loading proxy object should do just fine.

> *Well, we have the Query Object Model which is not bound to a particular storage-backend. Thus, at this point this is not a problem I think -- but has to be solved inside the particular storage backend.*

I was just wondering if the current query object model is able to "model" such non-standard queries on the database yet (be it things like the LIMIT/TOP clause or a SQL_CALC_ROWS statement that needs to be inserted, or a subsequent SELECT FOUND_ROWS() as a second query, which is very hard to model abstractly). Or if the whole tree structure of the CR makes such special constructs unnecessary. But since I don't nearly understand enough of the whole persistence layer of FLOW3 yet, it won't do much good to discuss here. It's just an interesting topic to me since I am curious as to how you will solve it; maybe I'll start a post on flow3-general about it ;)

#5 - 2010-08-12 14:35 - Bastian Waidelich

- Assigned To changed from Karsten Dambekalns to Bastian Waidelich

#6 - 2010-08-17 12:26 - Bastian Waidelich

- File 8774_v1.patch added

Attached is a **very first** spike, that introduces the feature to Extbase in order to test possible bottlenecks. I only upload this so it won't get lost. A clean test-driven version for FLOW3 will follow asap.

#7 - 2010-08-19 18:06 - Bastian Waidelich

We just decided that we probably should add a parameter to the execute() method to define the result type. something like Query::RESULT_TYPE_ARRAY, Query::RESULT_TYPE_PROXY and Query::RESULT_TYPE_OBJECT

#8 - 2010-08-30 14:30 - Bastian Waidelich

- Status changed from Accepted to Resolved

- % Done changed from 0 to 100

Applied in changeset commit:"60fadf5cf8cb1805607f0e2e12bd9f69afe37244".

Files

8774_v1.patch	4.8 kB	2010-08-17	Bastian Waidelich