Database - Feature #11723

Implement -prefetchNumRecs and -Mm database startup parameters

08/10/2026 02:27 AM - Artur Şcolnic

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Artur Şcolnic	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**reviewer:**	
vendor_id:	GCD	**production:**	No
case_num:		**env_name:**	
version_reported:		**topics:**	
version_resolved:			

Description

History

#2 - 08/10/2026 02:28 AM - Artur Şcolnic

- Status changed from New to WIP

- Assignee set to Artur Şcolnic

#3 - 08/10/2026 02:30 AM - Artur Şcolnic

I will start by writing a multi session test suite with harness to better understand the extent of the message buffer usage in OE, the documentation states that it is used for for each no-lock, but I suspect there are corner cases OE's documentation omits.

#5 - 08/10/2026 04:56 AM - Alexandru Lungu

I also found some extra information on this aspect:

- prefetchDelay, prefetchFactor, prefetchPriority are 3 other options beside prefetchNumRecs. Not sure if relevant at all.
- record phrases can be set with NO-PREFETCH.

How fit is this model for FWD from the POV of performance (scrolling)

I am not entirely sure how this OE prefetch would map to FWD ProgressiveResults. OE technique is to bring in fixed sized batches, while FWD brings exponentially bigger sized batches.
Presuming that OE is prefetching 16 records in 1 call, 32 records in 2 calls and 48 records in 3 calls, FWD is gathering 16 records in 3 calls (1 + 10 + 100), 32 records in 3 calls and 48 records also in 3 calls. In other words, queries in >=48 records should be faster in FWD **unless invalidation kicks in**.

Switching to a batched model like OE is wrong. OE is using a cursor that can gather 16 records at once very fast. In FWD, gathering N records **can be** very slow if the OFFSET is high (e.g. OFFSET 1000 LIMIT 16). So, the FWD model for progressively larger results is fit for OFFSET/LIMIT query method. I am strongly in favor of not honoring the prefetch for scrolling-mode queries

How fit is this model for FWD correctness

There is a certain edge case here. FWD is eagerly providing updated records, whereas OE presents stale data. This is a functional mismatch. What if the NO-LOCK session relies on **not seeing** the latest data? It sets a high prefetch window so that the session is basically **preselecting** with each FOR EACH NO-LOCK. It seems quirky, but possible (e.g. a reporting process that starts with a high -MM just to make FOR EACH NO-LOCK consistent).

How fit is this model for FWD from the POV of performance (dynamic)

This is **the most important** bullet. Invalidated queries in FWD gather records 1 by 1, whereas in OE the prefetch is honored so that records are gathered in batches (e.g. 16). If NO-LOCK, the buffer will be populated with a stale DMO (one we mark as COPY in FWD when gathered from dirty-share).

More exploration

- How does current session affect the prefetch? I understand that new/updated/deleted records from other sessions are invalidating the ProgressiveResults, but not seen by OE prefetch. However, what if the **same session** is actually creating/deleting/updating records on that table. Will the "prefetch" be dropped? **I am quite convinced that yes**.
  - I wonder if we can prefetch from persistent database and use that prefetch against the intra-session dirty-share database. In other words, if we have 2 records in dirty-share are 16 records prefetched (so 18 records in memory), can we rely on the fact that it is enough to work on that 16 pre-fetched records? Or are only 14 records pre-fetched?
  - What if we continue to gather new records with that query after create/update/delete in the same session? Will it "revalidate" the prefetching in OE?
  - TL;DR does creating/updating/deleting records in a table in the same-session invalidate the prefetch?
- I wonder how multi-table queries are honoring the pre-fetch. The NO-PREFETCH is a **record phrase** setting, which means that query components are independent and prefetching happens **per component**. This would be nice as it is straight-forward to implement in FWD.
- What happens in UNDO cases? I think this has the same answer as the first bullet. If the same session did create/update/delete records in that table, the prefetch is dropped, so that undo doesn't affect the prefetch.
- Does the prefetch happen for FOR EACH blocks only, or for OPEN QUERY as well? If it happens for OPEN QUERY, we need to understand the implication of PREVIOUS, FIRST, LAST as well + if the query is scrolling, what happens with REPOSITION (in SCROLLING or INDEXED-REPOSITION modes).

Idea sketch

I think we can add an intermediary PrefetchedResults that is an implementation of Results or sub-class of SimpleResults (maybe similar to BatchResults). It stays in between the query and the results implementation. When doing a next, it will automatically "cache" the next N results in a List. I guess it can simply mark the DMO kind as COPY, to avoid attaching to a live image of a record.

- For ScrollingResults it should work straight-forward. It will be used just to ensure correctness.
- For ProgressiveResults it should work quite easy, but there is a caveat: when changing the bracket, the ProgressiveResults **might** invalidate, so there might be some state we need to ensure is preserved.
  - For example, we prefetch 5 records, but on the 6th the query invalidates and we can't tell what will the 6th record actually be.
  - A trivial fix would be to make ProgressiveResults use base N, so that all brackets are multiples of N, so that prefetching would never occur across brackets. I am not entirely sure how this would work with PREV or reposition however.
  - PS: what happens if another session commits right in the mean-time of this whole ProgressiveResults prefetching?
- For DynamicResults, using a PrefetchedResults would be straight-forward, as long as the revalidation doesn't affect the prefetcing time. **However** this will still fetch records one-by-one from database.

Invalidation improvement idea - active bundle

The fix by now is only functional. In order to improve dynamic query performance, we would have to change RAQ so that it prefetches N records instead of 1 and resolves all these N records at once against the dirty database, instead of fetching them 1 by 1. For an index with (f1, f2), we can have the following situation:

- f1 = ? and f2 = ? and recid > ? -> finds 2 records
- f1 = ? and f2 > ? -> finds 10 records
- f1 > ? -> finds 10.000 records.

We would need to execute f1 = ? and f2 = ? and recid > ? with limit N, f1 = ? and f2 > ? with N - (number of records from first query e.g. 2) and f1 > ? with N - (number of records from first query e.g. 2) - (number of records from second query e.g. 10). In this specific case, the revalidation should have occurred after the first 12 records, so the prefetching here is tricky as we would need to stop after first 12 records, revalidate and get next 4 records to complete the prefetch.

PS: what happens if another session commits right in the mean-time of this whole process of fetching the active bundle and revalidation?

Invalidation improvement idea - dirty-share implication

We would need to only extract the first 16 and resolve against dirty-share. The though part would be to understand if prefetching N would be enough, considering that some of these N may have been updated/deleted in the dirty-share, so more than N would have been needed in the prefetch.

Invalidation improvement idea - #8388

I think the cleanest way to handle this would be to fix #8388 first, as that eliminated the need of intertwining the persistent database data with the dirty-share data at server-side and simply let the DB prefetch the next N records using a LIMIT 16. The UNION solution will properly resolve the updated/deleted records problem at the DB side and get us the correct next 16 records.

I think this last point is also subject to DB synchronization and will atomically get us the right prefetch in respect to other sessions work.

Conclusion

I think we will need more exploration first, especially in regard to intra-session changes visibility in prefetch and its invalidation.

#6 - 08/10/2026 07:04 AM - Greg Shah

Please also consider that the OE behavior might differ based on the reread-nolock startup parameter.

#7 - 08/10/2026 07:24 AM - Artur Școlnic

My understanding is that rereadnolock affects which version of cached records will be served, it does not fetch the latest true version from the DB. I confirmed with a test case that the message buffer is still used regardless of rereadnolock. Another point is that these parameters are for 2 distinct layers of the application, -Mm is a database parameter, any client connected to it will use the message buffer, whereas rereadnolock is a client parameter which affects only the client's management of the already fetched records.

#8 - 08/12/2026 07:21 AM - Artur Școlnic

Alex, I committed the batch fetching for invalidated AQ to 11723a.

Prefetched records in the window/batch are frozen against other sessions' commits; the next batch sees those commits; this session's own changes are always visible regardless of DML.

This is safe because RAQ navigation is keyset-based, not offset-based. Each single-record fetch was already its own keyset step at its own point in time, so batching widens an existing staleness window from 1 record to N. It introduces no anomaly class that record-at-a-time retrieval did not already permit.

- PrefetchWindow - holds frozen rows, delivery cursor, ramp and generation counter. Implements RecordChangeListener and registers once. ChangeBroker is context-local, so it reports only this session's changes, which is exactly the split the contract needs. Discards outright rather than overlaying the session's version of each record, because a same-session create landing inside the frozen range cannot be surfaced from records already fetched.
- canPrefetch() - gate: NEXT bundle, LockType.NONE, not unique, no join, no external buffers, not multiplexed, cross-session dirty-share off. Opt-in via setPrefetchEnabled, set only by createSimpleQuery(), so FindQuery and FOR FIRST/LAST are untouched.
- fillPrefetch() - greedy fill across the bundle cascade using Persistence.list(fql, parms, want - fetched, 0). A generation guard refuses to publish a batch if this session changed the table while the fetch was in flight.
- Any retrieval that bypasses the window discards it, so a partially drained window cannot sit behind the walk and replay records.
- The window is owned by the AdaptiveQuery, not the worker. createCompoundQuery() is deliberately left unbatched.

#9 - 08/12/2026 07:41 AM - Artur Școlnic

Alex, regarding the prefetched window in OE. When a reader walks a batch and another session alters records in that batch or inserts/deleted records in that batch, the reader does not see it. When the reader alters the prefetched records, the changes are visible, but if another record is inserted in the bracket in the same session, that record is not visible, it is never read. If the record is inserted in a batch that will be read, it will be visible. Basically the whole window is a snapshot that cannot be changed by other sessions in any way and can only be updated by the same session, no deletes or inserts.