Batching, Sorting and Filtering with ABL

Query and UltraWinGrid in OERA

Håvard Danielsen
Principal Software Engineer with OpenEdge
Summer 2009
This presentation assumes an architecture where presentation layer data requests are passed thru the layers to the data access.

This presentation discusses how batching can be used to allow users to work on large data amounts.

There are both simpler and more sophisticated alternatives to these approaches.
Agenda

- Managing Large Data Amounts
- Data Request Requirements
- Data Access Requirements
- Filter, Sorting and Batching with UltraGrid
- Sample Implementation
- Demo
- Questions
Managing Large Data Amounts

- Resolved by user
  - Limit amount
    - Filter or error
  - Paging
    - User selects a page of data to navigate on client

- Transparent batching
  - Forward batching
    - Append more data to client when navigating forward
  - Two-way batching (bidirectional)
    - Position anywhere and append when navigating in any direction
Managing Large Data Amounts
Limit Amount

- User Interface
  - Filter first
    - Remember Last Query
    - Stored Queries
    - Pseudo Query (A,B,C,D)
  - Error on too much data
    - Max number of records and timeout

- Performance not an issue
Managing Large Data Amounts

Paging

- **User Interface**
  - 1 – 100, 101 – 200, 201 – 300
  - 1, 2, 3, 4, 5

- **Performance**
  - No benefit from INDEXED-REPOSITION
    - Uses REPOSITION-TO-ROW
  - Very fast on cached data (PRESELECT)

- **Easy to adapt and implement**
Managing Large Data Amounts
Transparent Batching

- **User Interface**
  - Almost Transparent
  - Jumping or fixed scrollbar
  - Total number of records unknown

- **Performance**
  - Benefits from INDEXED-REPOSITION
    - Use set START-ROWID
Usability issues

- Search, Find and Last need special attention
- Resort and Refresh must start on first
Managing Large Data Amounts
Two-way batching

- Performance
  - Extra query open for Search, Find, Last and Previous
    - Return “look back” information

- Challenging to implement
Batching is used to manage large data amounts

Two way batching
- Position anywhere
- Can use indexed reposition

Paging
- Industry standard
- More efficient with data cache on server
- Relatively easy to add on two way batching
<table>
<thead>
<tr>
<th>Batch Context Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Expression</td>
</tr>
<tr>
<td>- Must be exactly the same for each request</td>
</tr>
<tr>
<td>- Tables, Filter and Sort</td>
</tr>
<tr>
<td>Transparent batching position context</td>
</tr>
<tr>
<td>- Prev position for two-way batching</td>
</tr>
<tr>
<td>- Next position</td>
</tr>
<tr>
<td>Paging context</td>
</tr>
<tr>
<td>- Start position</td>
</tr>
<tr>
<td>- Total num records</td>
</tr>
</tbody>
</table>
Batching Updatable Data

- Batching should only be used on read only data
  - Practice is different

- Updates on server can cause
  - Same record in next batch
  - Record already exists
    - Appending batch with unique index on client
Appending Batches and Sorting

- Sort on non unique index
  - Different sort on client
    - Add KeyFields to sort
## Batching Requirements

<table>
<thead>
<tr>
<th>Batching Requirements</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-way batching</td>
<td>Yes</td>
</tr>
<tr>
<td>Paging</td>
<td>?</td>
</tr>
<tr>
<td>Deal with record collision</td>
<td>No</td>
</tr>
<tr>
<td>Add key sort to non-unique sort</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Agenda

- Managing Large Data Amounts
- Data Request Requirements
- Data Access Requirements
- Filter, Sorting and Batching with UltraGrid
- Sample Implementation
- Demo
- Questions
Data Request Requirements
Request Granularity

- **Multiple Entities (datasets)**
  - Particularly important at start up
  - Separate receive from request

- **Table oriented requests**
  - Most requests after start up are table oriented
  - Keep lookup tables on subsequent requests
  - Use relation definitions
    - Empty (unless appending batch) and retrieve child tables
    - Keep tables that have a reposition relation
Data Request Requirements
Query Requests

- Open
  - Apply filter and sort
  - Could position to key

- Refresh
  - Position query to current key
  - Should return with batch size

- Resort
  - Done locally if not batching (or all batches)
  - Position query to current key
  - Should return with rows before and after
Data Request Types
Record Position Requests

- **Search (find first)**
  - Can search on client if first record available
  - Position query to first where
  - Should return with rows before and after

- **Find (unique)**
  - Can look on client first if unique index (or info)
  - Position query to key
  - Use batch size 1 for single row requests
Data Service Position Requirements

- **Position to**
  - Key (find unique)
    - Resort, Refresh, Find
  - Where (find first)
    - Search
  - Last

- **Keep find unique and find first separate**
  - No open necessary for key as order is irrelevant
Data Service Position Options

- **Ordinal Positioning**
  - Return rows before and after positioned row
    - Improve user experience with two way batching
  - Low cost - only when “look back” is already done

- **Fill Batch**
  - Always return enough rows to fill batch
    - When Search or Find positions to end of batch
    - Necessary for ABL GUI Browser scrollbars
Query (filter and sort) and batch size are implied

<table>
<thead>
<tr>
<th>Data Request Requirements</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple datasets in one request</td>
<td>Not shown – prepared APIs</td>
</tr>
<tr>
<td>Table oriented requests</td>
<td>Yes</td>
</tr>
<tr>
<td>Position to key</td>
<td>Yes</td>
</tr>
<tr>
<td>Position to where</td>
<td>Yes</td>
</tr>
<tr>
<td>Ordinal position</td>
<td>Yes – hard coded</td>
</tr>
<tr>
<td>Fill batch</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Agenda

- Managing Large Data Amounts
- Data Request Requirements

Data Access Requirements

- Filter, Sorting and Batching with UltraGrid
- Sample Implementation
- Demo
- Questions
Query Transformation

- Use data source field mapping
  - for each eOrder where eOrder.OrderNum > “20”
  - for each order where order.order-num > “20”

- Data source child query uses temp-table parent
  - for each eOrderLine where eOrderLine.OrderNum = eOrder.OrderNum
  - for each order-line where orderline.ordernum = eOrder.OrderNum
Data source query table order may vary

- for each eOrder where eOrder.OrderNum = “22”
- for each order where order.order-num = “22”

- for each eOrder,
  each eSalesRep
    where eSalesRep.Salesrep = eOrder.Salesrep
    and eSalesrep.SalesRep = “BBB”

- for each salesrep where salesrep.salesrep = “BBB”,
  each order where order.salesrep = salesrep.salesrep
Values in a query is interpreted according to session settings for date and numeric values

- Use quotes (quoter)
  - Must use same setting when executed
- Pass native data types
### Data Access Query Requirements

<table>
<thead>
<tr>
<th>Data Access Requirements</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query transformation</td>
<td>Yes</td>
</tr>
<tr>
<td>Variable table order in query</td>
<td>Yes</td>
</tr>
<tr>
<td>Base Query</td>
<td>Yes</td>
</tr>
<tr>
<td>Internationalization</td>
<td>No</td>
</tr>
</tbody>
</table>
Agenda

- Managing Large Data Amounts
- Data Request Requirements
- Data Access Requirements

Filter, Sorting and Batching with UltraGrid

- Sample Implementation
- Demo
- Questions
Grid keeps ordinal row **Selected**
  - Turn off before and set back to same row after

Open query activates first row
  - Set flag to turn off next `AfterRowActivate` (or Before)
    - `DisplayLayout.Override.ActiveRowAppearance.Reset()`
Sorting in UltraGrid

- Turn off default sort in DisplayLayout: Override
- HeaderClickAction: ExternalSortMulti (-Single)
- Improves performance for local sort also
- AfterSortChange event (or Before)
- Band: SortedColumns
- Let the Presenter/Model decide active row
Column Filtering in UltraGrid

- Filter UI is set in DisplayLayout:Override
  - Set FilterUIType = FilterUIType:FilterRow
- Population of drop down values from data (fires off end)
  - Use BeforeRowFilterDropDownPopulate event (e:Handled=true)
- Filter operators can be set per column
  - Set FilterOperatorDropDownItems to ABL friendly values
  - Set FilterOperatorDefaultValue to ABL friendly value
- Filter evaluation is controlled in DisplayLayout:Override
  - Set FilterEvaluationTrigger=FilterEvaluationTrigger:OnEnterKey
- Filters are evaluated per column (also on enter)
  - Cancel the BeforeRowFilterChanged event (e:Cancel = true)
  - Manage apply of filters to external source manually
Column Filtering in UltraGrid
Managing Filters for external data source

- Define local variables
  - mChangedFilterColumns as ArrayList – Not applied changed columns
  - mColumnFilters as SortedList – Applied column filters
  - mRemovedFilterColumn as logical – Flag if any filter was blanked

- Keep track of changes in FilterCellValueChanged
  - If non blank cell add column to mChangedFilterColumns
  - else remove it from both lists and set mRemovedFilterColumn true
  - Clear filters if no filters remain (needs improvement)

- Manage filters in BeforeRowFilterChanged
  - Add e:NewFilter to mColumnFilters
  - Remove e:NewFilter:Column from mChangedFilterColumns
  - Apply filters if mChangedFilterColumns became empty
  - or mChangedFilterColumns was empty and mRemovedFilterColumn

- Take over the dialog in BeforeCustomRowFilterDialog
  - filter = mColumnFilters:Item[..] or new ColumnFilter()
  - wait-for e:CustomRowFiltersDialog:ShowDialog(filter,?).
  - e:Cancel = true.
  - If dialog is ok apply filters.
Column Filtering in UltraGrid
Managing Filters for external data source

- Define local variables
  - Define a `SortedList` to track applied filters
  - Define an `ArrayList` to track columns with filter changes
  - Define a flag to set if any filter was blanked

- Keep track of changes in `FilterCellValueChanged`
  - Maintain the list of columns with changes
  - Also empty the applied list and set the flag when a cell is blanked

- Manage filters in `BeforeRowFilterChanged`
  - Add `e:NewFilter` to the `SortedList` and remove ref from `ArrayList`
  - Apply the `SortedList` if the `ArrayList` became empty
  - If there were no filters but any blanked apply the `SortedList`

- Take over the dialog in `BeforeCustomRowFilterDialog`
  - Give it a filter from the `SortedList` or create a new
  - Wait and apply filters if ok
Retrieving Data in a Batching UltraGrid

- OffEnd fires (as it is supposed to)
- Off home events fires (not always as it is supposed to)
- Events fires sequentially
  - Difficult to block batching during retrieval
  - KeyDown and KeyUp is helpful
- Events fires asynchronously (?)
  - Message statements does not always stop other events
Batching in UltraGrid

- **Forward batching**
  - Binding source **OffEnd** event

- **Backward batching**
  - Fetch prev batch in **BeforeRowRegionScroll**
    - If e:NewState:ScrollPosition = 1
  - Keep first row out of viewport when more batches exist
    - Control in **AfterSortChanged** (other data read events?)
    - Require service that can return rows before current on resort
  - On **KeyDown**
    - fetch batch on Home, End and Cursor events
Agenda

- Managing Large Data Amounts
- Data Request Requirements
- Data Access Requirements
- Filter, Sorting and Batching with UltraGrid

Sample Implementation

- Demo
- Questions
Sample Components
Query management with ABL in OERA

- Query sorting, filtering and batching
- Dataset and request management
- Working simulations

No update methods
Sample Classes
Query classes with QueryString class

**Query**
- BaseQuery: char
- KeyFields: char
- NumRecords: int {readOnly}
- QueryInfo: QueryString
- QuerySort: char
- QueryString: int {readOnly}
- Tables: char {readOnly}
- CloseQuery(): void
- CreateQuery(): logical
- # CreateQueryInfo(): void
- # DoRepositionQuery(rowid[]): logical
- GetCurrentRowKey(): char
- GetFirst(): logical
- GetLast(): logical
- GetNext(): logical
- GetPrev(): logical
- OpenQuery(): logical
- PrepareQuery(): logical
- RepositionQuery(rowid[]): logical
- RepositionQuery(char): logical
- RepositionQuery(int): logical

**QueryString**
- QueryMap: IQueryMap
- AddExpression(): void
- BuildQueryString(char): void
- BuildQueryString(handle): void
- SetQueryString(char): void
- SetSort(char): void

**DataView**
- BatchSize: int
- BindingHandle: handle
- BusinessEntityName: char
- HasFirst: logical
- HasLast: logical
- InstanceName: char
- LastBatchSize: int
- TableName: char
- CurrentChanged(): void
- DataRefresh(): void
- FetchLastBatch(): void
- FetchNextBatch(): void
- FetchPrevBatch(): void
- ReopenQuery(char): void
- ReopenQuery(rowid): void
- RepositionQuery(char): void
- RepositionQuery(int): void
- RepositionQuery(rowid): logical
- ResortData(char): void

**DataSource**
- BatchSize: int
- DataSourceHandle: handle
- FieldMapping: char
- FillMode: char
- NextPosition: char
- PrevPosition: char
- ColumnSource(): char
- PositionBatchRequest(): logical
- PositionToKey(): logical
- PositionToLastBatch(): logical
- PositionToWhere(): logical
- SetNextPosition(): logical
- SetPrevPosition(): logical
- # SetStartRowids(): logical

**IQueryMap**
- BatchSize: int
- # DataSourceHandle: handle
- # FieldMapping: char
- FillMode: char
- NextPosition: char
- PrevPosition: char
- ColumnSource(): char
- PositionBatchRequest(): logical
- PositionToKey(): logical
- PositionToLastBatch(): logical
- PositionToWhere(): logical
- SetNextPosition(): logical
- SetPrevPosition(): logical
- # SetStartRowids(): logical