OLAPRowNumber

As part of the OLAPOperations effort, we'd like to add support for the ROW_NUMBER function.

Rationale

The SQL 2003 standard specifies the ROW_NUMBER function. This function computes the sequential row number, starting with 1, of the row within its window partition according to the ordering of the window.

Syntax

In the SQL 2003 standard section 6.10 (and section 7.11 for window specification), the <window function> has the following (partial) grammar:

\[
<\text{window function}> ::= <\text{window function type}> \text{ OVER } <\text{window name or specification}>
\]

\[
<\text{window function type}> ::= <\text{rank function type}> \text{ <left paren> <right paren>}
\]

\[
\mid \text{ROW_NUMBER <left paren> <right paren>}
\]

\[
\mid \text{<aggregate function>}
\]

Thus the ROW_NUMBER function has a very simple syntax.

Examples

A nice example of the ROW_NUMBER function is provided in DERBY-581:

```sql
SELECT * FROM ( 
  SELECT  
    ROW_NUMBER() OVER (ORDER BY key ASC) AS rownumber, 
    columns 
  FROM tablename 
) AS foo
WHERE rownumber <= n
```

In this example the ROW_NUMBER function is used to limit the query as soon as the first N rows have been determined.

Note that the above example is not supported by the current implementation (Derby 10.4.1.3 / May 2008), since this implementation has some limitations.

Examples supported by the current implementation are included below, as well as on the Derby FAQ web page since the ROW_NUMBER function may be used as a replacement for the LIMIT and OFFSET keywords used in some other databases (and many users ask about this functionality).

For example, querying a table with 500000 rows, retrieving the first 5 rows only (using the IJ tool):

```sql
ij> SELECT * FROM ( 
  SELECT  
    ROW_NUMBER() OVER() AS rownum, myLargeTable.*  
  FROM myLargeTable 
) AS tmp 
WHERE rownum <= 5;
```

<table>
<thead>
<tr>
<th>ROWNUM</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>This is row number 1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>This is row number 2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>This is row number 3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>This is row number 4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>This is row number 5</td>
</tr>
</tbody>
</table>

5 rows selected

(corresponding to "SELECT * FROM myLargeTable LIMIT 5" in some other databases)

Example retrieving rows 200001 to 200005:
```sql
> SELECT * FROM (  
>     SELECT ROW_NUMBER() OVER() AS rownum, myLargeTable.*  
>     FROM myLargeTable  
>   ) AS tmp  
>   WHERE rownum > 200000 AND rownum <= 200005;
```

<table>
<thead>
<tr>
<th>ROWNUM</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>200001</td>
<td>200001</td>
<td>This is row number 200001</td>
</tr>
<tr>
<td>200002</td>
<td>200002</td>
<td>This is row number 200002</td>
</tr>
<tr>
<td>200003</td>
<td>200003</td>
<td>This is row number 200003</td>
</tr>
<tr>
<td>200004</td>
<td>200004</td>
<td>This is row number 200004</td>
</tr>
<tr>
<td>200005</td>
<td>200005</td>
<td>This is row number 200005</td>
</tr>
</tbody>
</table>

5 rows selected

(corresponding to "SELECT * FROM myLargeTable LIMIT 5 OFFSET 200000" in some other databases)

Related mailing list threads with examples:
- **Doubt in ROW_NUMBER function** - Topic: WHERE clause (started 2008-06-23 on derby-user)

## Current Implementation

An implementation of the ROW_NUMBER() window function is included in Derby starting with the 10.4.1.3 release. Limitations and usage description may be found in the Derby Reference Manual, **ROW_NUMBER built-in function**.

## Proposed Changes

**DRAFT**

### Overview

The changes involved to implement ROW_NUMBER() will touch several areas of the Derby engine. This work includes getting some of the framework needed for future window function extensions in place.

The largest changes will likely be:
- The new syntax must be added to the SQL grammar.
- New query tree node subclasses must be added for the window specification and window function columns.
- Optimization should avoid flattening and enable materialization.
- A new result set class implementing the window functions must be added.
- Statistics gathering must be implemented.

Implementation of the ROW_NUMBER() window function is likely to affect other areas as well but to a lesser extent, often only as helper methods.

For the first increment we will only implement support for empty, unnamed windows. This means a window spanning the complete result set.

### Query processing

The new ROW_NUMBER() syntax is added to the SQL grammar in `sqlgrammar.jj` to support empty, unnamed windows.

While walking the **ResultColumns** of a **SelectNode**, we add a **RowNumberColumnNode** to the **ResultColumnList** in the AST. The window specification is attached as a **WindowNode** below the **RowNumberColumnNode**. This is analogous to how a FROM list, a ORDER BY clause, etc, is done for the **SelectNode** itself. The **WindowNode** takes a set of parameters including the name, partition definition, ordering info, and the window frame definition. The **WindowNode** should serve as a basis for future support of named windows.

The interesting part of the AST looks like this for a **SelectNode** with a window function once parsing is completed:
Query Optimization

Avoid flattening

We avoid flattening queries with window functions during optimization. The main rationales behind this are

- avoid rewrite of the query

```
SELECT * FROM {
    SELECT row_number() over () as r, t.* FROM T
} AS tmp WHERE r <= 3;
```

into

```
SELECT row_number() over () as r, t.* FROM T WHERE R <= 3;
```

which will fail as r is not a column in table T.

- window function results that span multiple rows, like a moving average, will benefit from materialization. For the nested select query above we materialize the subquery select result, and have the outer SELECT pull rows from the materialized result.

Modification of access paths

At the last stages of optimization the AST is made into a queryplan by modification of access paths.

At this stage a SelectNode is exchanged for a ProjectRestrictNode (PRN) that handles projection and restriction, and possible index use is considered etc. The SelectNode clauses ORDER BY, GROUP BY, and FROM are made into their respective nodes in the queryplan. See example.

Given this select node in the AST

```
...  
  SelectNode  
  * ...  
  * From -> ...  
  * OrderBy -> ...  
  * GroupBy -> ...  
  * Distinct  
  * Where -> ...  
  * resultColumnList -> * ResultColumn: table T, column A  
  |  
  |  * ...  
  |  * RowNumberColumnNode -> WindowNode  
  |  * ...  
  ...  
```

this is made into the following queryplan:
A PRN is inserted on top of whatever is in the select FROM clause. This PRN does projection of the non-window function columns, and applies any restriction (where clause) on these columns. To the top of this PRN we add any GroupByNode, DistinctNode, and OrderByNode as needed.

In the case of a window function column we then pull the WindowNode up from under the RowNumberColumnNode, and merge this with the result we get from below. The WindowNode needs to go on top so that ordering, grouping etc are performed properly. We add one WindowNode for each window function column, and they are evaluated left to right in the result column list, with the right most column being the top WindowNode in the query plan.

Yet another PRN is used to top off the upper WindowNode. This PRN is effectively a no-op, and is eliminated during code generation. The PRN is mainly there due to other parts of the code expecting the top node to be a PRN once we are done with the access path modification.

**Code generation**

During code generation, we lay out code to generate a WindowResultSet, and pass the source resultset information as well as column info to it. The logic for ROW_NUMBER() over the empty window specification is simply a counter in the WindowResultSet, so this change should be seen as part of the window function framework. The intended use of this result set is mainly other window functions that may need row caching to calculate moving averages over a window or other features and functionality needed for the window function being implemented.

**Code execution**

During code execution the WindowResult behaves as a regular part of the ResultSet chain in the engine. Rows are fetched from this ResultSet with a call to getNextRowCore() which in turn fetches rows from the lower/child ResultSet in similar fashion. The window function column is added, and the row passed up the chain.