

**Logi Ad Hoc
Using ActiveSQL**



**Version 11
Last Updated: April 2014**

Table of Contents

What is ActiveSQL?	3
How is ActiveSQL implemented?.....	4
What reporting databases are supported?	6
ActiveSQL Performance Considerations	6
Are there other restrictions or considerations to using ActiveSQL?.....	7
Known Issues.....	10

What is ActiveSQL?

The original model for data retrieval in Logi applications involves retrieving *all* of the requested data from a data source and caching it, either in memory or to disk, for use by elements that will display the data. This generally works well and provides good performance when data volumes are modest.

However, there are situations where utilizing the power and functionality that exists in relational databases are a better option:

1. When SQL queries that return a large amount of data are displayed in a tabular form, it takes a long time to process and display the first page of data. In these situations, utilizing paging options that are supported by the SQL language and supported natively by the database is a much faster option.
2. When Charts are generated by AdHoc, reading a large number of records in the entirety, then sorting and grouping them in the Logi Engine is less efficient than using the data base to sort, group and return a small number of records usually required by charts.

ActiveSQL is a special type of datalayer designed for just such a situation. It differs from conventional datalayers in that it does *not* retrieve all of the rows in the initial request and, in response to runtime manipulations of the interface by users, it dynamically modifies and resends its initial SQL query. These helps the tabular display element work with much larger data sets and still provide good and linear performance. It also increases the number and frequency of SQL queries the database server must handle (which, generally, does not prove to be a burden on the database server) but dramatically improves performance for the user.

The datalayer generates SQL queries that limit the number of rows returned, helping with paging and sorting. With Interactive Paging in use, the query requests a number of pages of records instead of all of the data, thereby improving perceived performance. As the user moves beyond the first set of pages, the database is queried again to get the next set of pages.

ActiveSQL moves the bulk of the processing to the server instead of the client.

In that same vein, charts typically are based on aggregate information. **ActiveSQL** moves the data aggregation back to the reporting database instead of building large XML recordsets that would need to be processed by the engine with the traditional SQL approach. The **ActiveSQL** query would return just the aggregate information needed by the chart.

Starting with Version 11.2 of Ad Hoc (engine version 11.2.40), the Ad Hoc instance may be configured to take advantage of the benefits offered by **ActiveSQL**.

Additional technical information about the **ActiveSQL** datalayer may be found in [ActiveSQL Datalayer Documentation](#). This document discusses the datalayer from a Logi Info perspective. All of the implementation details found in that paper are managed by the Ad Hoc application, making them transparent to the end user.

How is ActiveSQL implemented?

The generation of **ActiveSQL** datalayers is controlled by the `ahUseActiveSQL` attribute found in the `<Ad Hoc>` element in the `_Definitions/_Settings.lgx` file of an Ad Hoc instance. If this attribute is set to "True", Ad Hoc will generate **ActiveSQL** datalayers and the engine will wrap the traditional SELECT statements with SQL that returns a requested block of records. If the attribute is set to "False", the traditional SQL datalayers will be generated by Ad Hoc and the engine will pass the SELECT statement unchanged to the data provider.

Note:

In version 11.2 of Ad Hoc, the `ahUseActiveSQL` attribute controls the generation and behavior of all datalayers for all reporting connections used in the Ad Hoc instance. If you are using reporting databases that are not in the list of supported databases for ActiveSQL, the `ahUseActiveSQL` value should be set to "False". By default, the attribute will be set to "True" for new Ad Hoc instances created in Version 11.2. The attribute will be set to "False" for instances upgraded to version 11.2. The `ahUseActiveSQL` attribute value may only be adjusted by manually editing the `Settings.lgx` file.

All of the following examples use a simple query from the Northwind database in SQL Server.

Example 1: Demonstrates the attribute disabled, the datalayer element generated by Ad Hoc, and the resultant SQL statement passed to the SQL Server database.

From the `_Settings.lgx` file:

```
<AdHoc ahUseActiveSQL="False" ... />
```

From the report definition file:

```
<DataLayer Type="SQL" ID="ahDataLayer" Source="SET ROWCOUNT 64000; SELECT
&#xD;&#xA;[O1].[CategoryID] AS [C1], &#xD;&#xA;[O1].[CategoryName] AS [C2],
&#xD;&#xA;[O1].[Description] AS [C3]&#xD;&#xA; FROM [Categories] [O1]; SET ROWCOUNT 0; "
ConnectionID="1" HandleQuotesInTokens="True" />
```

From a debug page:

```
SET ROWCOUNT 64000; SELECT
[O1].[CategoryID] AS [C1],
[O1].[CategoryName] AS [C2],
[O1].[Description] AS [C3]
FROM [Categories] [O1]; SET ROWCOUNT 0;
```

Example 2: For the same query, demonstrates the attribute enabled, the datalayer generated by Ad Hoc, and the resultant SQL statement passed to the SQL Server database.

From the _Settings.lgx file:

```
<AdHoc ahUseActiveSQL="True" ... />
```

From the report definition file:

```
<DataLayer Type="ActiveSQL" ID="ahDataLayer" Source="SELECT &#xD;&#xA;[O1].[CategoryID] AS [C1], &#xD;&#xA;[O1].[CategoryName] AS [C2], &#xD;&#xA;[O1].[Description] AS [C3]&#xD;&#xA; FROM [Categories] [O1]" ConnectionID="1" HandleQuotesInTokens="True" SkipRowCount="True" FirstRow="1" RowCount="15" rdResultsetGuid="e853b2a227414782b3c8709f8111c4d5" rdSortColumn="" rdSortDirection="" />
```

From a debug page:

```
WITH PagedMembers AS (SELECT TOP 5000 [ahDataLayer].[C1], [ahDataLayer].[C2], [ahDataLayer].[C3], ROW_NUMBER() OVER (ORDER BY [ahDataLayer].[C1]) as [IgxLogiRowNumber] FROM (SELECT [O1].[CategoryID] AS [C1], [O1].[CategoryName] AS [C2], [O1].[Description] AS [C3] FROM [Categories] [O1]) [ahDataLayer]) SELECT * FROM PagedMembers WHERE [IgxLogiRowNumber] >= 1 ORDER BY [IgxLogiRowNumber];
```

Example 3: For the same query with a sort sequence specified via the Ad Hoc interface, demonstrates the datalayer generated by Ad Hoc and the resultant SQL statement passed to the SQL Server database:

From the report definition file:

```
<DataLayer Type="ActiveSQL" ID="ahDataLayer" Source="SELECT &#xD;&#xA;[O1].[CategoryID] AS [C1], &#xD;&#xA;[O1].[CategoryName] AS [C2], &#xD;&#xA;[O1].[Description] AS [C3]&#xD;&#xA; FROM [Categories] [O1]" ConnectionID="1" HandleQuotesInTokens="True" SkipRowCount="True" FirstRow="1" RowCount="15" rdResultsetGuid="efda98c1f7d145958560663f1d72c3b8" rdSortColumn="" rdSortDirection=""><SqlSort SortColumn="C1" SortSequence="ASC" ID="" ahActiveSqlConverted="True" /></DataLayer>
```

From a debug page:

```
WITH PagedMembers AS (SELECT TOP 5000 [ahDataLayer].[C1], [ahDataLayer].[C2], [ahDataLayer].[C3], ROW_NUMBER() OVER (ORDER BY [ahDataLayer].[C1]) as [IgxLogiRowNumber] FROM (SELECT [O1].[CategoryID] AS [C1], [O1].[CategoryName] AS [C2], [O1].[Description] AS [C3] FROM [Categories] [O1]) [ahDataLayer]) SELECT * FROM PagedMembers WHERE [IgxLogiRowNumber] >= 1 ORDER BY [IgxLogiRowNumber];
```

What reporting databases are supported?

ActiveSQL *only works with*

- **Oracle**
- **MySQL**
- **Microsoft SQL Server 2005+**
- **PostgreSQL**

ActiveSQL Performance Considerations

ActiveSQL depends on a well designed and tuned database for its efficiency. For example, for aggregate queries usually necessary for charts, absence of supporting indices for aggregation will result in table scans and poor database performance. ActiveSQL uses the “ORDER BY <column>” clause in SQL as opposed the SORT being executed by the Logi engine. Lack of indices could result in the database using temp tables and increasing the time to sort and display the data.

Are there other restrictions or considerations to using ActiveSQL?

Yes, there are.

The primary restriction when **ActiveSQL** is enabled is that Stored Procedures may not be used as a data objects.

When **ActiveSQL** is enabled, the philosophy of **not** returning all rows initially has an impact on the Ad Hoc user interface and behavior. It may also be locally disabled when certain reporting constructs are used that require a full recordset.

With **ActiveSQL** disabled, a *Configuration / Report Configuration / Report Settings* page section appears as:

i The settings below set limits and features of new reports. You can apply these settings to an existing report by rebuilding it.

Max Records:	<input type="text" value="64000"/>	?
Show message if maximum records reached:	<input checked="" type="checkbox"/>	?
Show message if no records returned:	<input checked="" type="checkbox"/>	?
Row count message:	<input type="text"/>	[row count] <input type="text" value="rows returned."/> ?
Row count message style:	<input type="text"/>	?
Apply data format to Excel exports:	<input type="text" value="Always"/>	?
Max List Records:	<input type="text" value="100"/>	?

With **ActiveSQL** enabled, the same section of the page appears as:

i The settings below set limits and features of new reports. You can apply these settings to an existing report by rebuilding it.

Apply data format to Excel exports:	<input type="text" value="Always"/>	?
Max List Records:	<input type="text" value="100"/>	?

The missing attributes when **ActiveSQL** is enabled reflect the fact that not all of the data may be returned, consequently information related to row counts and messaging around row counts would be disabled.

Max Records – since the purpose of **ActiveSQL** is to return blocks of records as requested rather than all of the data, a limit on the total number of records to be returned is unnecessary and has been removed for instances with **ActiveSQL** enabled. Similarly, since the **Max Records** attribute is removed the messages around the limits have been removed.

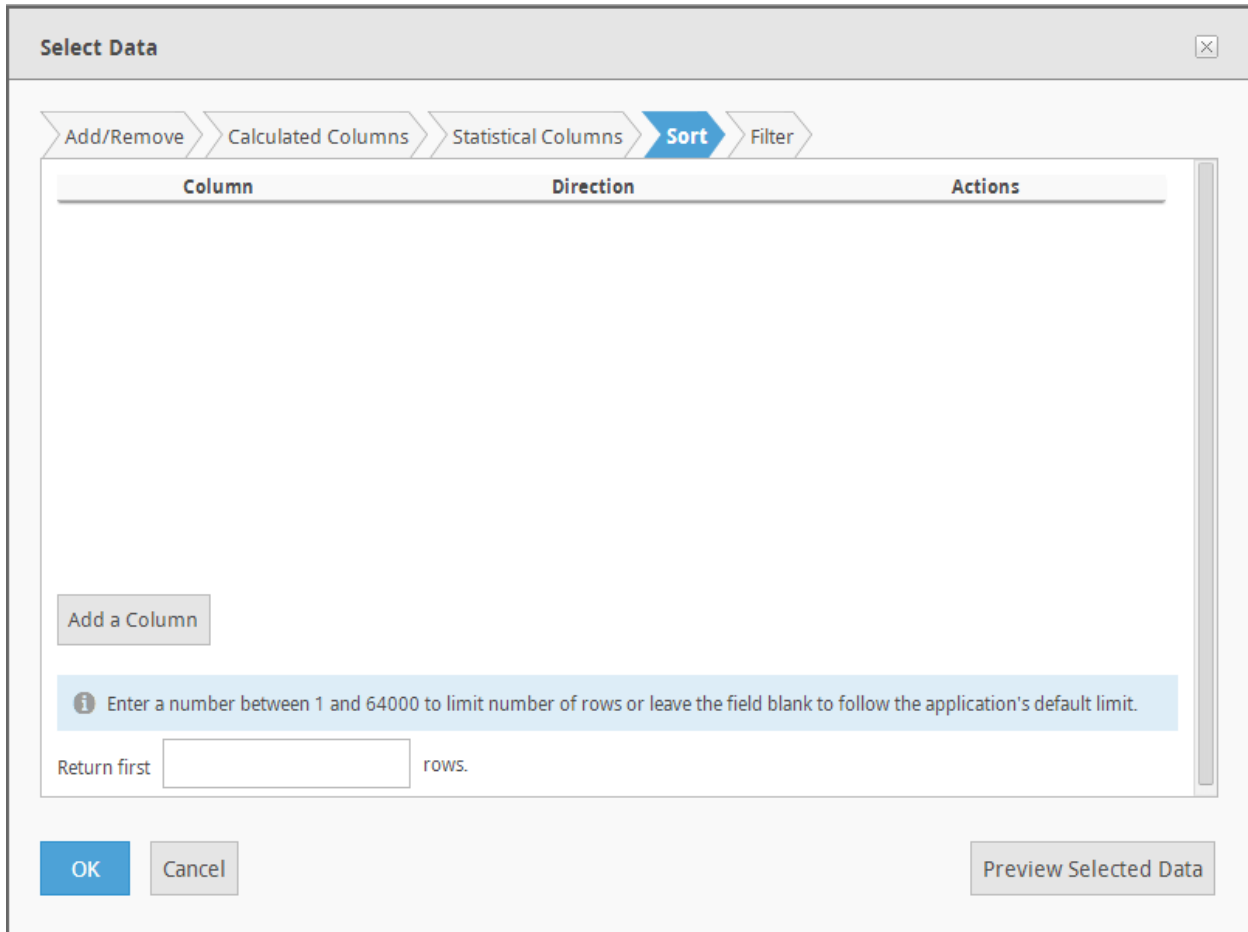
Attributes related to the number of records returned have been removed for the same reason.

In the *Report Builder / Table Settings* tab, the attributes **Include Row Number** and **Show record count** are removed when **ActiveSQL** is enabled.

The screenshot shows the 'Table Settings' tab selected in a report builder. The settings are as follows:

- Title:** [Empty text input field]
- Show record count:**
- Include row number:**
- Paging Style:** Interactive Paging ▼

In the **Select Data** dialog, the *Sort* tab will not show the **Return first ___ rows** attribute since **ActiveSQL** processing is already returning a limited set of records to the report.



In the same **Select Data** dialog, the inclusion of *Statistical Columns* in the report will cause Ad Hoc to generate a SQL datalayer instead of an **ActiveSQL** datalayer since the statistics may require a full set of records. When a Statistical Column is added to the report, a confirmation message will be issued.

If multiple data values, multiple label columns, summary row or summary columns are requested in a crosstab, Ad Hoc will generate a SQL datalayer instead of an **ActiveSQL** datalayer

Known Issues

With **ActiveSQL** enabled, there may be an issue with grouped/drill style reports if the last sub-report shows only a “text/memo” column. A simple replicator using the same data source as our examples would be:

- 1) Select a data table display element or template
- 2) Select the Categories object as a data source
- 3) Include the three Category columns in the data table
- 4) Build a group drill report and create a grouping level with the Category ID and Category Name columns. This will leave the Category Description column as the detail column

When the report is run, the main report will appear as:

Category Grouping Error Date: Friday, April 04, 2014
Time: 1:55:36 PM

Category ID	Category Name	Details
1	Beverages	1 Rows
2	Condiments	1 Rows
3	Confections	1 Rows
4	Dairy Products	1 Rows
5	Grains/Cereals	1 Rows
6	Meat/Poultry	1 Rows
7	Produce	1 Rows
8	Seafood	1 Rows

If you click on any drill label (1 Rows), the sub-report will show:

Logi Debugger Trace Report

There was an error while processing your request.

The error was:

The text, ntext, and image data types cannot be compared or sorted, except when using IS NULL or LIKE operator.

[Click here for more detailed information.](#)