XML element - ObjectsViewDef

Only in DbVisualizer Pro

This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

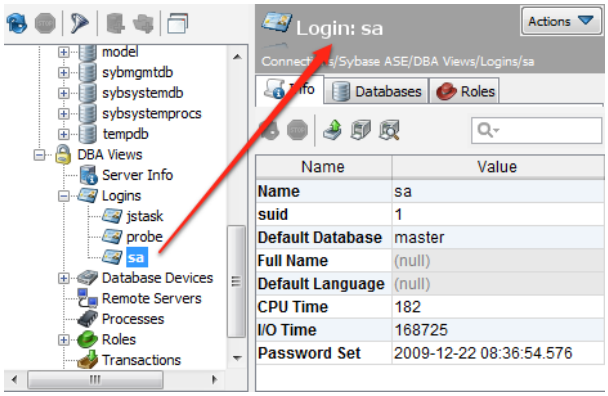
The ObjectsViewDef element define all views for the object types in the objects tree. These views are displayed in the Object View area for the selected object. Which views should appear when selecting a node in the tree is based on the object type for the tree node and the corresponding object view definition.


```
<ObjectsViewDef extends="true">
   <ObjectView type="xxx">
      <DataView id="yyy" label="yyy">
         ...
      </DataView>
   </ObjectView>
</ObjectsViewDef>
```

The **extends="true"** attribute specifies that this definition will extend the ObjectsViewDef definition in the database profile being extended.

When an object is opened in the database tree (**sa** in the screenshot below) a corresponding object view tab is created (right in the sample). Each of the DataView elements in the ObjectView will appear as sub tabs in the object view tab. The selected object and its information is passed to each of the data views for processing and presentation. The following example show the Object View in DbVisualizer and its ObjectView element definition.

Representation in DbVisualizer



XML definition

```
<ObjectView type="Logins">
  <DataView type="Logins" label="Logins"
            viewer="grid">
    <Command idref="sybase-ase.getLogins"/>
  </DataView>
</ObjectView>

<ObjectView type="Login">
  <DataView type="Info" label="Info"
            viewer="node-form"/>
  <DataView type="Databases" label="Databases"
            viewer="grid">
    <Command idref="sybase-ase.getLoginDatabases"/>
  </DataView>
  <DataView type="Roles" label="Roles"
            viewer="grid">
    <Command idref="sybase-ase.getLoginRoles"/>
  </DataView>
</ObjectView>
```

The screenshot and the database tree show both the **Logins** node and its child nodes, **jstask**, **probe** and **sa**. These nodes are instances of the object types **Logins** (labeled Logins in the screenshot) and **Login** (the three sub nodes: **jstask**, **sa** and **probe**).

The ObjectView XML definitions above shows the data views for these two types, **Logins** and **Login**. Opening the node labeled **Logins** in the tree will show the object view for the **<ObjectView type="Logins">** definition while opening the node labeled **jstask**, **probe** or **sa** will show the object view for the **<ObjectView type="Login">** .

The example shows **sa** being selected. Its DataView definitions displayed as tabs in the object view are (by label):

- Info
- Databases
- Roles

XML element - ObjectView

The ObjectView element is associated with an object type and groups all DataView elements that appear when the object type is selected in the database objects tree. Here follows the ObjectView definition for the Login object type.

```
<ObjectView type="Login">
  ...
</ObjectView>
```

The **type** attribute value is used when a node is clicked in the database objects tree to map with the corresponding ObjectView definition. The following lists the attributes for ObjectView:

Attribute	Value	Description
type		The type of the ObjectView as declared in the GroupNode and DataNode elements in the ObjectsTreeDef section
action	drop	**drop** is useful when extending another database profile to remove the ObjectView and all its child views

XML element - DataView

The DataView element is comparable with the DataNode element in the ObjectsTreeDef. It defines what SQL (command) should be executed, labeling, viewer type (presentation form) and other characteristics. The following is the DataView definitions for the **Login** object type. (The ObjectView element is part of the sample just for clarification).

```
<ObjectView type="Login">
  <DataView type="sybasease-login-info" icon="Info" label="Info" viewer="node-form"/>
  <DataView type="sybasease-login-databases" icon="Databases" label="Databases" viewer="grid">
    <Command idref="sybase-ase.getLoginDatabases"/>
  </DataView>
  <DataView type="sybasease-login-roles" icon="Roles" label="Roles" viewer="grid">
    <Command idref="sybase-ase.getLoginRoles"/>
  </DataView>
</ObjectView>
```

All three DataView elements have a **viewer** attribute identifying how the data in the view should be be presented, e.g., as a grid or a form. See the next sections for a list of viewers. The following lists all attributes for DataView:

Attribute	Value	Description
id		Every DataView element must have a unique id which is not only unique in the current profile but also with all id's in extended profiles To make sure the id is unique use the following recommended format: **profileName-objectViewType-viewerLabel**. Ex: sybasease-login-databases (The id should not contain any empty space or special characters other than dash ("-")).
label		The label for the viewer as it will appear in the tab
icon		The icon as defined in the icons.prefs file(s)
viewer		One of: grid, text, form, node-form, table-refs, tables-refs, table-date, table-rowcount, message, navigator, ddl, ProcedureViewer. See the viewers section in this document for more information

drop-label-not-equal		Drop the viewer if its label is not equal to the value of this attribute
class		Used to specify a custom Java class used as the viewer
classargs		Used to pass arguments to a custom viewer
action	drop	**drop** is useful when extending another database profile to remove the DataView
doclink		Relative HTML link to the related chapter in the users guide
order-before		Specifies the order of this DataView among a collection of viewers having the same parent ObjectView. It can either be an index starting at 0 (first) or a node type. Ex. **order-before="sybasease-login-databases"** will order this DataView before viewers defined by the **id="sybasease-login-databases"** attribute
order-after		Specifies the order of this DataView among a collection of viewers having the same parent ObjectView. It can either be an index starting at 0 (first) or a node type. Ex. **order-after="sybasease-login-databases"** will order this DataView after viewers defined by the **id="sybasease-login-databases"** attribute

Viewers

The viewer attribute for a DataView define how the data for the viewer should be presented. The following sections walk through the supported viewers.

The following sample illustrates the viewer attribute.

```
<ObjectView type="Login">
  <DataView type="Info" label="Info" viewer="node-form"/>
</ObjectView>
```

DataView definitions may be nested and the viewers are then presented with the nested DataView in the lower part of the screen.

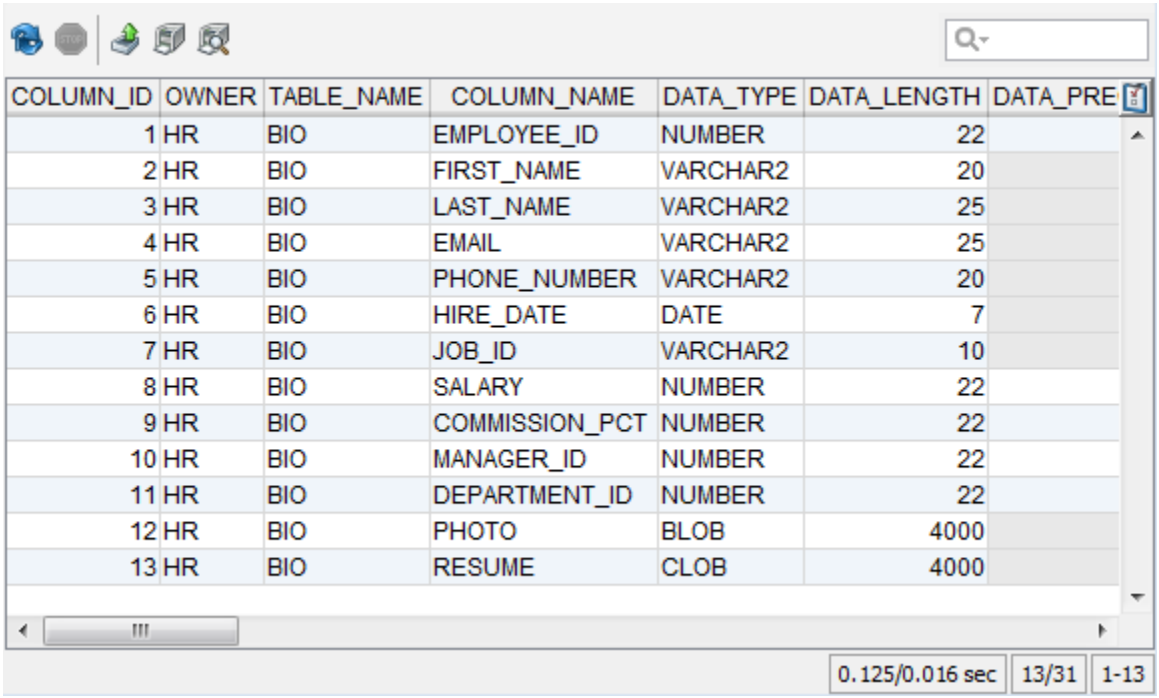
Viewer - grid

The **grid** viewer presents a result set in a grid with standard grid features such as search, copy, fit columns, export and so on. The result set is presented exactly as it is produced by the associated **Command** and any optional **Output** processing.

Here is a sample of the XML for the grid viewer:

```
<DataView type="oracle-columns-columns" icon="Columns" label="Columns" viewer="grid">
  <Command idref="oracle.getColumns">
    <Input name="owner" value="${schema}"/>
    <Input name="table" value="${objectname}"/>
  </Command>
</DataView>
```

And here is a screenshot of the standard grid viewer created from the above definition.



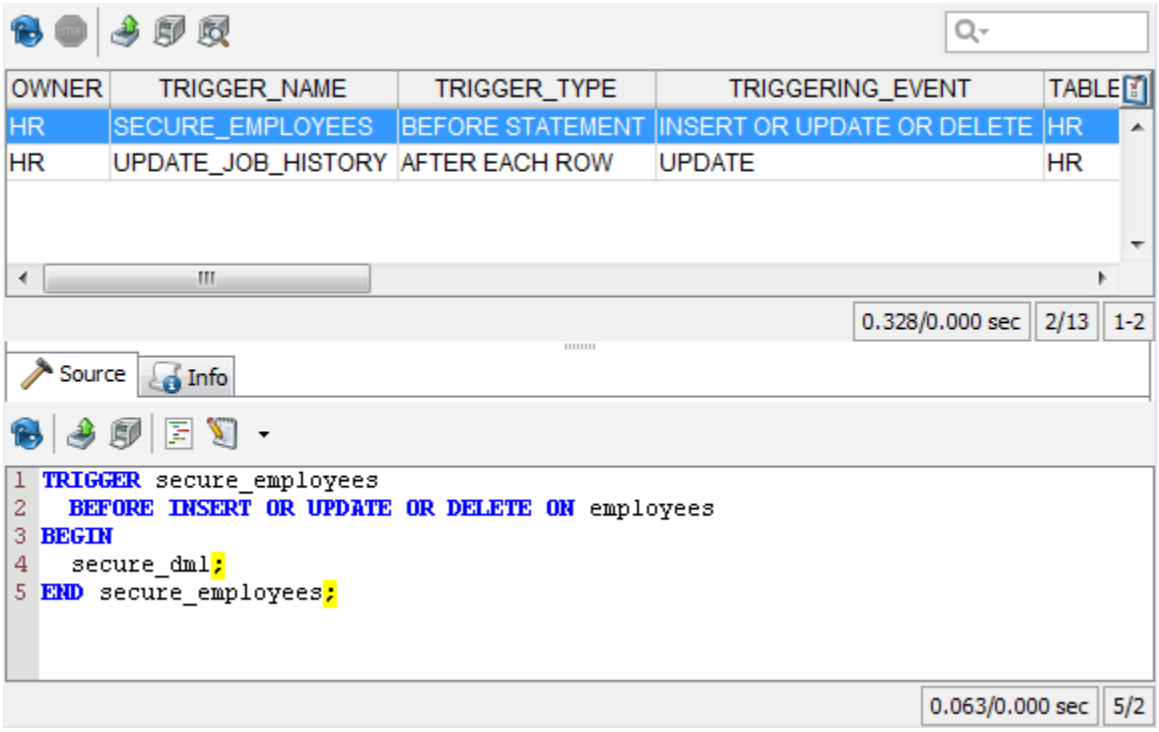
COLUMN_ID	OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRE
1	HR	BIO	EMPLOYEE_ID	NUMBER	22	
2	HR	BIO	FIRST_NAME	VARCHAR2	20	
3	HR	BIO	LAST_NAME	VARCHAR2	25	
4	HR	BIO	EMAIL	VARCHAR2	25	
5	HR	BIO	PHONE_NUMBER	VARCHAR2	20	
6	HR	BIO	HIRE_DATE	DATE	7	
7	HR	BIO	JOB_ID	VARCHAR2	10	
8	HR	BIO	SALARY	NUMBER	22	
9	HR	BIO	COMMISSION_PCT	NUMBER	22	
10	HR	BIO	MANAGER_ID	NUMBER	22	
11	HR	BIO	DEPARTMENT_ID	NUMBER	22	
12	HR	BIO	PHOTO	BLOB	4000	
13	HR	BIO	RESUME	CLOB	4000	

The nesting capability for grid viewers is really powerful, as it can be used to create a **drill-down** view of the data. Consider the scenario with a grid viewer showing all Trigger objects. Wouldn't it be nice to offer the user the capability to display the trigger source when selecting a row in the list? This is easily accomplished with the following definition:

```
<DataView id="oracle-table-triggers" icon="Trigger" label="Triggers" viewer="grid">
   <Command idref="oracle.getTriggers">
      <Input name="owner" value="${schema}"/>
      <Input name="condition" value="${triggersCondition}"/>
   </Command>
   <DataView id="oracle-table-triggers-source" icon="Source" label="Source" viewer="text">
      <Input name="dataColumn" value="text"/>
      <Input name="formatSQL" value="true"/>
      <Command idref="oracle.getTriggerSource">
         <Input name="owner" value="${OWNER}"/>
         <Input name="name" value="${TRIGGER_NAME}"/>
      </Command>
   </DataView>
   <DataView id="oracle-table-triggers-info" icon="Info" label="Info" viewer="node-form"/>
</DataView>
```

- The first DataView element define the top grid viewer labeled **Triggers** and the command to get the result set for it
- The next DataView is the **nested text viewer** labeled **Source**, specifying various input parameter for the viewer along with the command to get the source for the trigger. The difference here is that the input parameters for this command reference column names in the top grid. Since this viewer is nested, it will automatically be notified whenever an entry in the top grid is selected
- The third DataView labeled **Info** is presented as a tab next to the **Source** viewer, and presents additional information about the selected trigger

The following screenshot illustrates the above sample:



Adding custom menu items in the grid

The grid right-click menu contain a lot of standard actions. Custom commands can be defined in the DataView element and these will appear last in the menu.

```
<Input name="menuItem" value="Open in New Tab...">
   <Input name="action" value="open-object-in-new-tab-command ${schema||OWNER}${object||TABLE_NAME}"/>
</Input>
<Input name="menuItem" value="Open in Floating Tab...">
   <Input name="action" value="open-object-in-floating-tab-command ${schema||OWNER}${object||TABLE_NAME}"/>
</Input>
   <Input name="menuItem" value="Script: SELECT ALL">
   <Input name="command" value="select * from ${schema||OWNER}${object||TABLE_NAME}"/>
</Input>
   <Input name="menuItem" value="Script: DROP TABLE">
   <Input name="command" value="drop table ${schema||OWNER}${object||TABLE_NAME}"/>
</Input>
```

The **<Input name="menuItem">** element define a menu item entry that should appear in the grid right-click menu. The **value** for the menuItem is the **label** for the item as it will appear in the menu while the child **Input** element with **name="command"** is the **SQL command** that should be produced for all selected rows when the menu item is selected. Invoking a custom menu item will **not execute** the produced SQL directly but rather **copy the statements to a SQL Editor**. In the SQL Editor you will then need to manually execute the script and track the result.

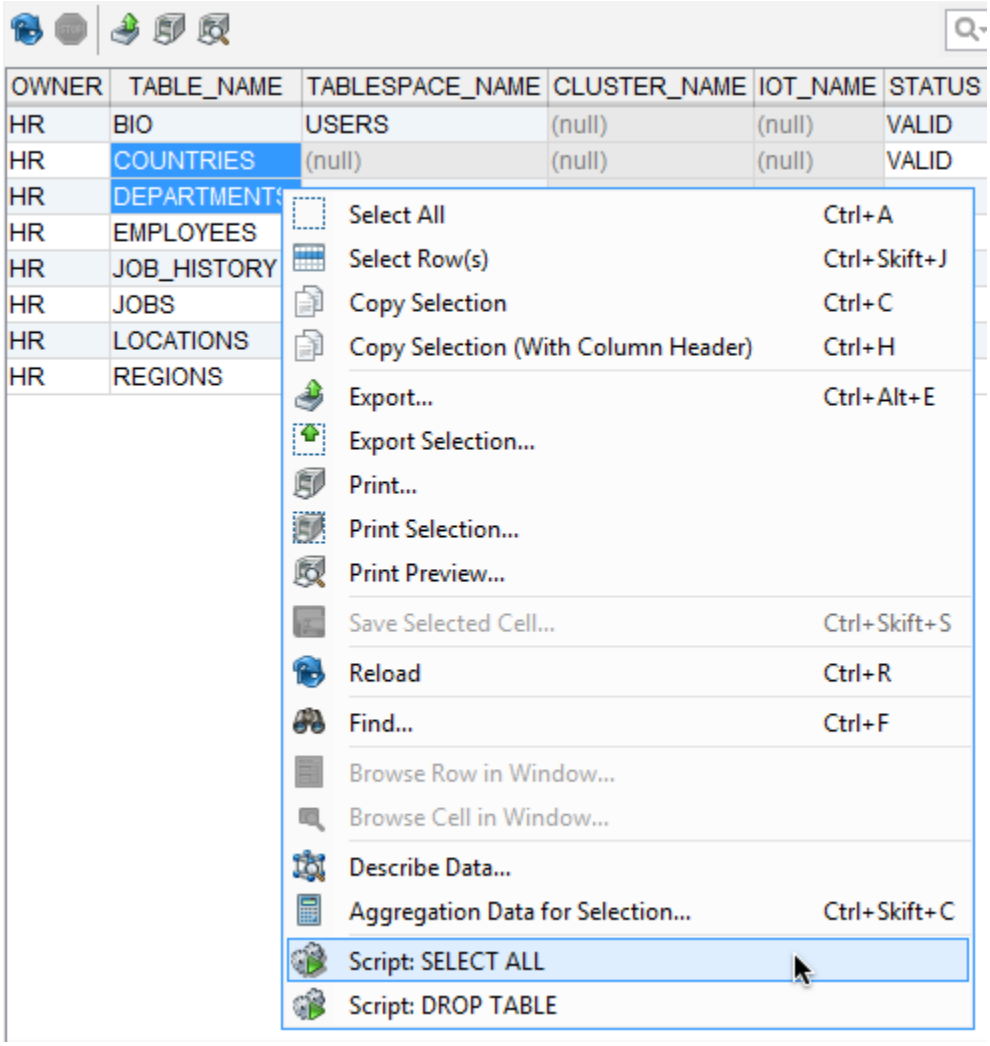
The **name="action"** attribute declare that the value is a pre-defined action. Valid actions are:

- open-object-in-new-tab-command
- open-object-in-floating-tab-command

Any variables in the SQL statement should identify column names in the result set. The user may select any cells in the grid and choose a custom menu item. It is only the actual rows that are picked from the selection as the columns are predefined by the menuItem declaration.

The variables specified in these examples starts with **${schema=...}** and **${object=...}**. These define that the first variable represents a schema variable while the second defines an object. This is needed for DbVisualizer to determine whether delimited identifiers should be used and if identifiers should be qualified, as defined in the connection properties for the database.

Here is a sample:



Setting initial max column width

Some result sets may contain wide columns. The following parameter sets an initial maximum width for all columns in the grid.

```
<Input name="columnWidth" value=""/>
```

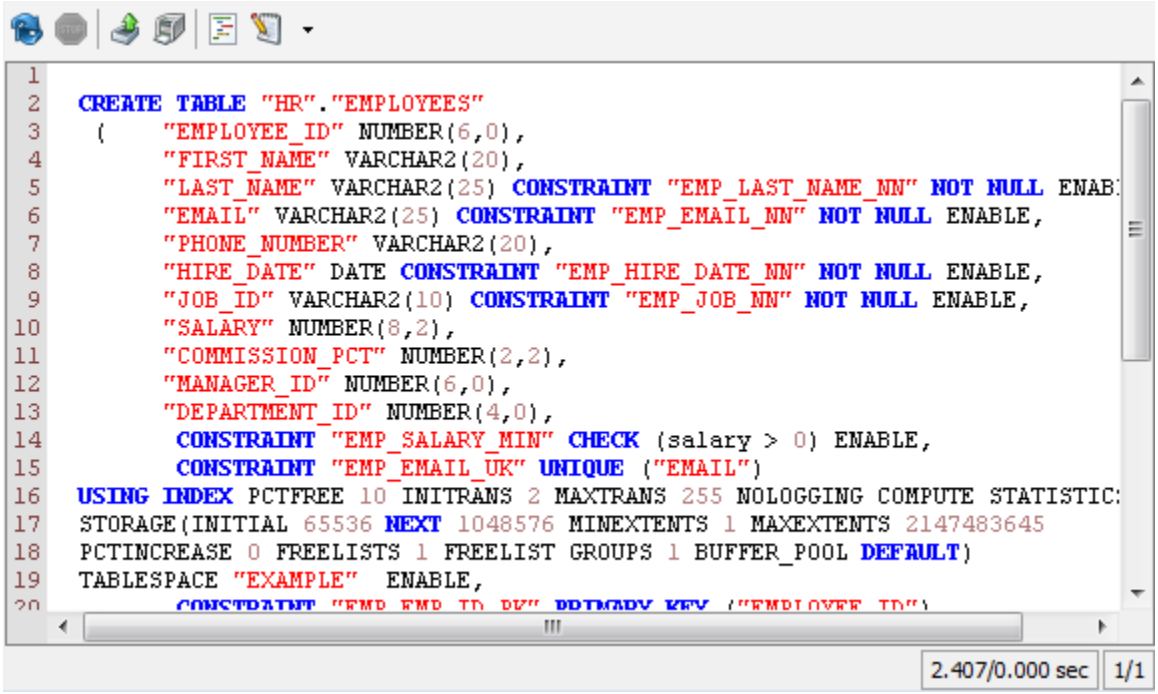
Viewer - text

The **text** viewer presents data from **one column** in a result set in a text browser (read only). This viewer is typically used to present large chunks of data, such as source code, SQL statements, etc. If the result set contain several rows, the text viewer reads the data in the column for each row and present the combined data.

Here is a sample of the XML for the text viewer:

```
<DataView id="oracle-table-triggers-source" icon="Source" label="Source" viewer="text">
   <Input name="dataColumn" value="text"/>
   <Input name="formatSQL" value="true"/>
   <Input name="newline" value""/>
   <Command idref="oracle.getTriggerSource">
      <Input name="owner" value="${OWNER}"/>
      <Input name="name" value="${TRIGGER_NAME}"/>
   </Command>
</DataView>
```

And here is a screenshot of the Source tab based on the previous definition.



Specify what column to browse

By default, the text viewer uses the data in first column. This behavior can be controlled by using the dataColumn input parameter. Simply specify the name of the column in the result set or its index (starting at 1 from the left).

```
<Input name="dataColumn" value=""/>
```

Enable SQL formatting of the data

The text viewer have the **SQL Formatting** function, which when invoked formats the SQL buffer in the viewer. The **formatSQL** input parameter is used to control whether formatting should be made automatically when the data first displayed. If formatSQL is not specified, no initial formatting is made.

```
<Input name="formatSQL" value=""/>
```

Adding newline to each row

For a result set containing multiple rows and all rows should be displayed in a text viewer, the **newline** parameter define the character(s) that should separate the rows in the viewer. A **\n** somewhere in the value will be converted to a platform dependent newline sequence in the viewer. By default there is no newline sequence between multiple rows.

```
<Input name="newline" value="\n"/>
```

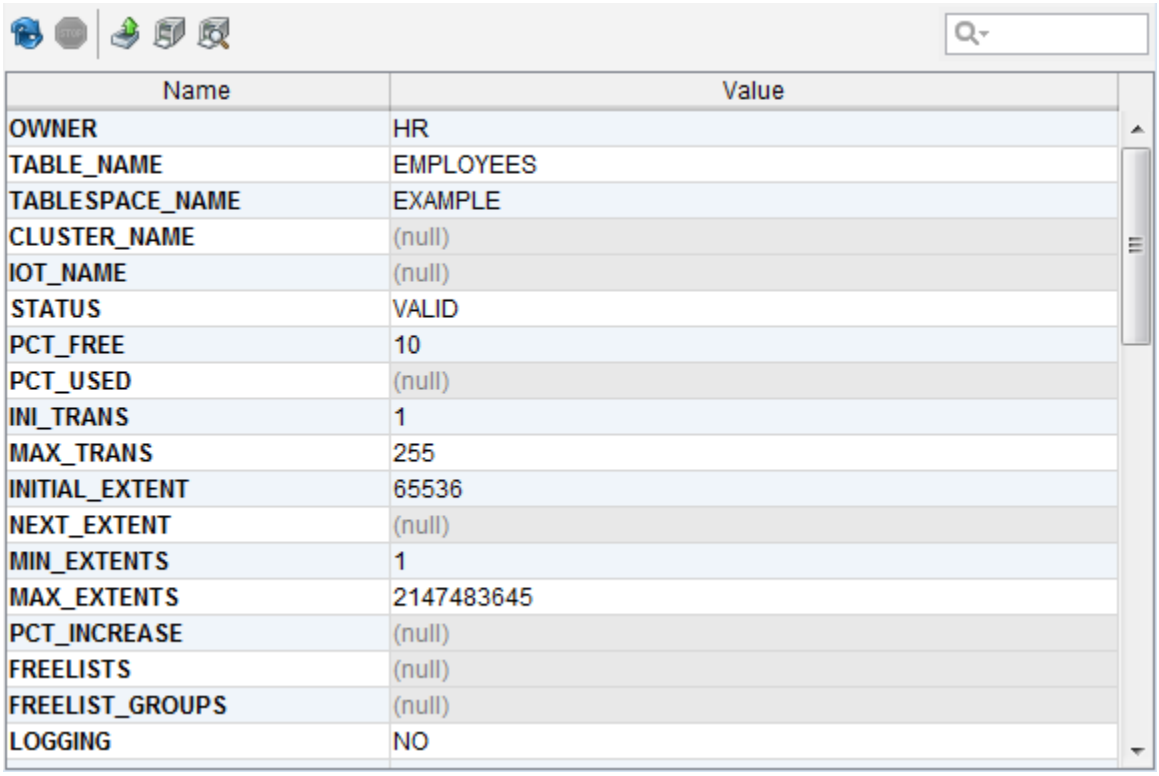
Viewer - form

The **form** viewer displays row(s) from a result set in a form. If several rows are in the result, they are presented in a list. Selecting one row from the list presents all columns and data for that row in a form.

Here is a sample of the XML for the form viewer:

```
<DataView id="oracle-table-info" icon="Info" label="Info" viewer="form" order-before="0">
   <Command idref="oracle.getTable">
      <Input name="owner" value="${schema}"/>
      <Input name="table" value="${objectname}"/>
   </Command>
</DataView>
```

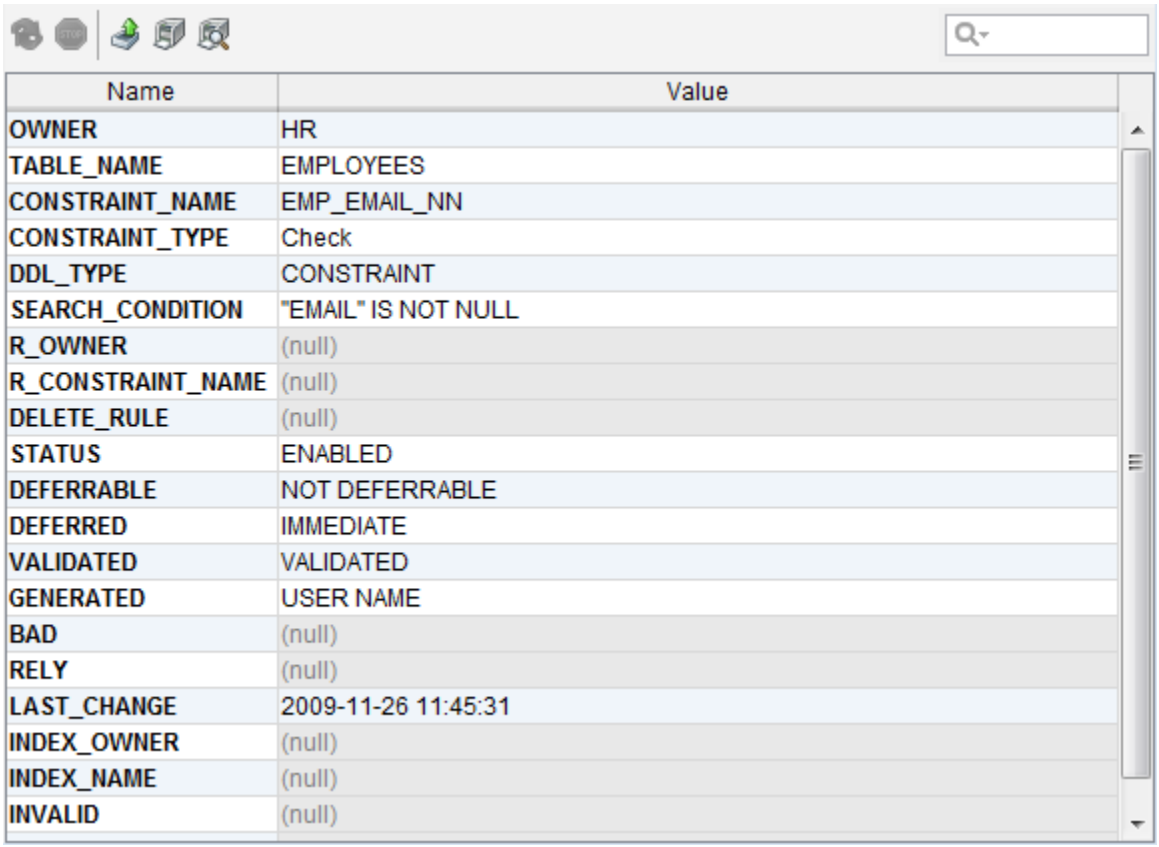
And here is a screenshot of the Info tab based on the previous definition.



Name	Value
OWNER	HR
TABLE_NAME	EMPLOYEES
TABLESPACE_NAME	EXAMPLE
CLUSTER_NAME	(null)
IOT_NAME	(null)
STATUS	VALID
PCT_FREE	10
PCT_USED	(null)
INI_TRANS	1
MAX_TRANS	255
INITIAL_EXTENT	65536
NEXT_EXTENT	(null)
MIN_EXTENTS	1
MAX_EXTENTS	2147483645
PCT_INCREASE	(null)
FREELISTS	(null)
FREELIST_GROUPS	(null)
LOGGING	NO

Viewer - node-form

The **node-form** viewer presents all data associated with the selected DataNode (variables). Here is a sample of the XML for the node-form viewer:



Name	Value
OWNER	HR
TABLE_NAME	EMPLOYEES
CONSTRAINT_NAME	EMP_EMAIL_NN
CONSTRAINT_TYPE	Check
DDL_TYPE	CONSTRAINT
SEARCH_CONDITION	"EMAIL" IS NOT NULL
R_OWNER	(null)
R_CONSTRAINT_NAME	(null)
DELETE_RULE	(null)
STATUS	ENABLED
DEFERRABLE	NOT DEFERRABLE
DEFERRED	IMMEDIATE
VALIDATED	VALIDATED
GENERATED	USER NAME
BAD	(null)
RELY	(null)
LAST_CHANGE	2009-11-26 11:45:31
INDEX_OWNER	(null)
INDEX_NAME	(null)
INVALID	(null)

Hiding columns

There may be data associated with the object that you don't want to present in the node form. The **hidecolumn** input parameter control what data for the object that should be invisible and you may repeat this option as many times you like to handle multiple variables that shouldn't be displayed.

```
<Input name="hidecolumn" value="oracle.getKeys.TABLE_OWNER"/>
```

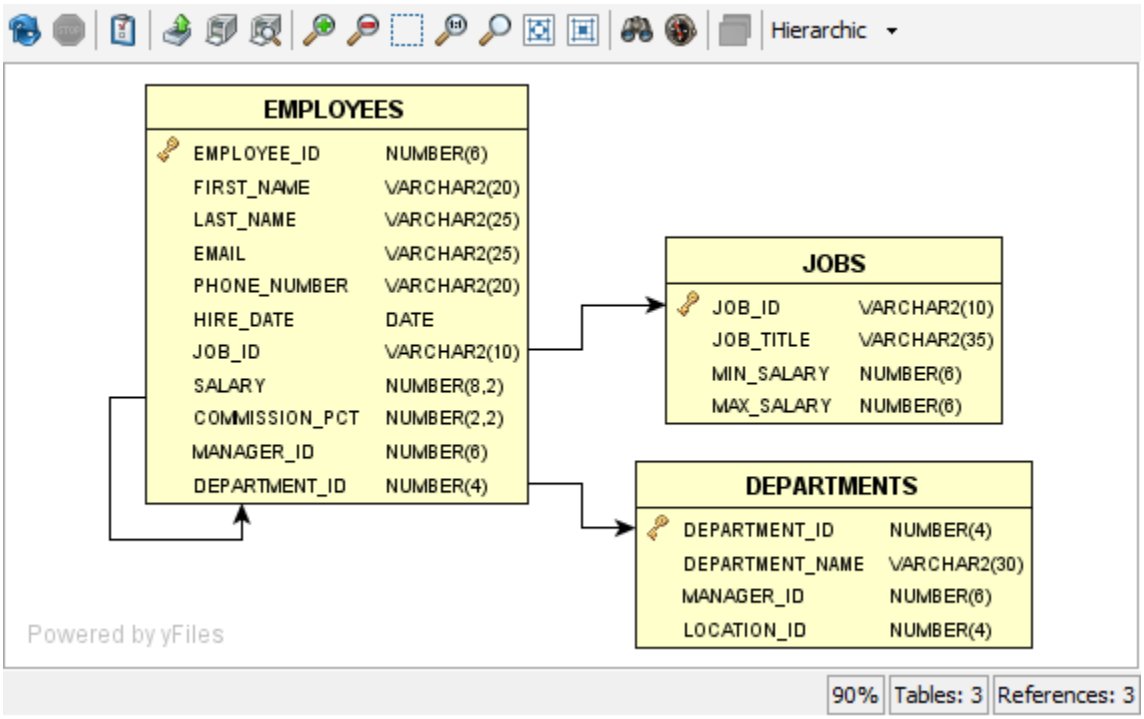
Viewer - table-refs

The **table-refs** viewer shows the references graph for the current object (this must be an object supporting referential integrity constraints, such as a Table),

Here is a sample of the XML for the table-refs viewer:

```
<DataView id="generic-table-references" icon="References" label="References" viewer="table-refs"/>
```

And here is a screenshot of the References tab based on the previous definition.

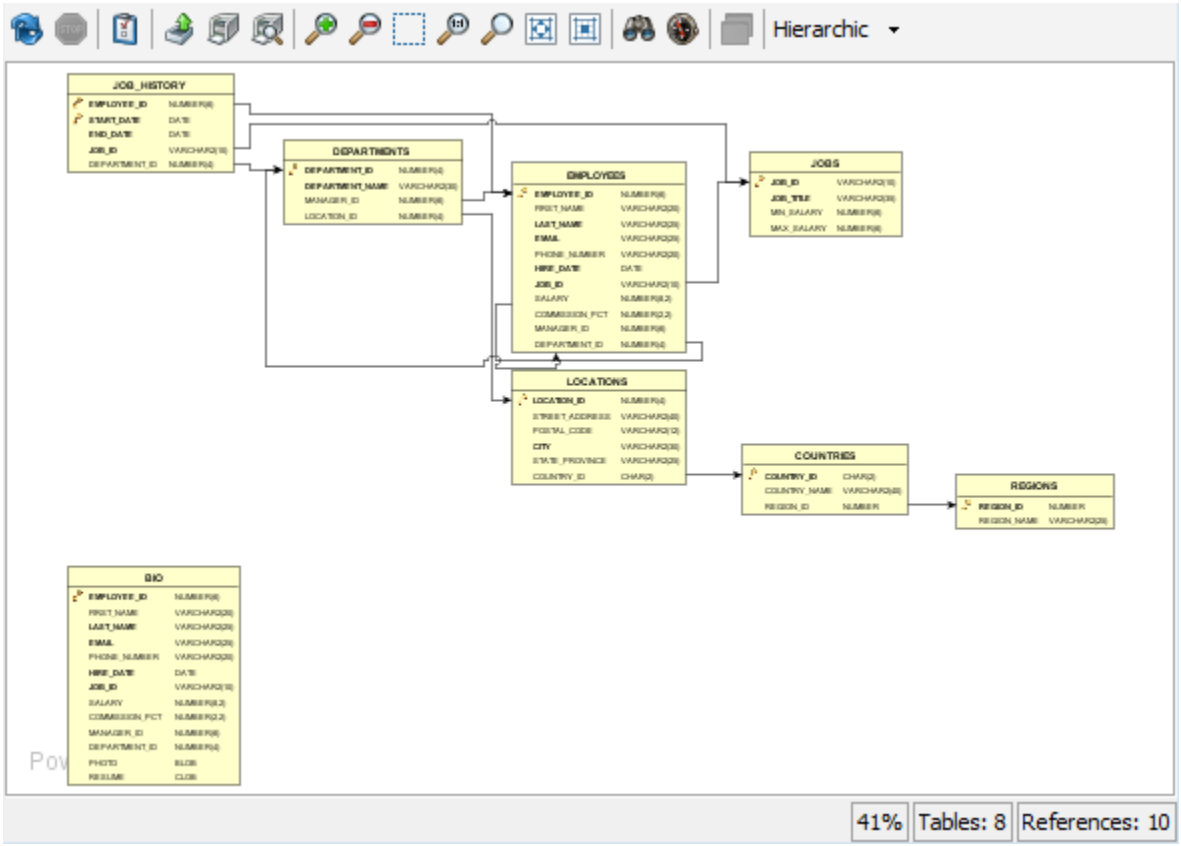


Viewer - tables-refs

The **tables-refs** viewer shows the references graph for **several tables** in the result set (the result set must contain objects supporting referential integrity constraints, such as a Table). Here is a sample of the XML for the tables-refs viewer:

```
<DataView id="oracle-tables-references" icon="References" label="References" viewer="tables-refs">
   <Command idref="oracle.getTables">
      <Input name="owner" value="${schema}"/>
      <Output modelaction="rename" index="OWNER" name="TABLE_SCHEM"/>
      <Output modelaction="rename" index="TABLE_NAME" name="TABLE_NAME"/>
   </Command>
</DataView>
```

And here is a screenshot of the References tab based on the previous definition.



Viewer - table-data

The **table-data** viewer shows the data for a table in a grid with various features such as filtering and editing (if licensed) functionality.

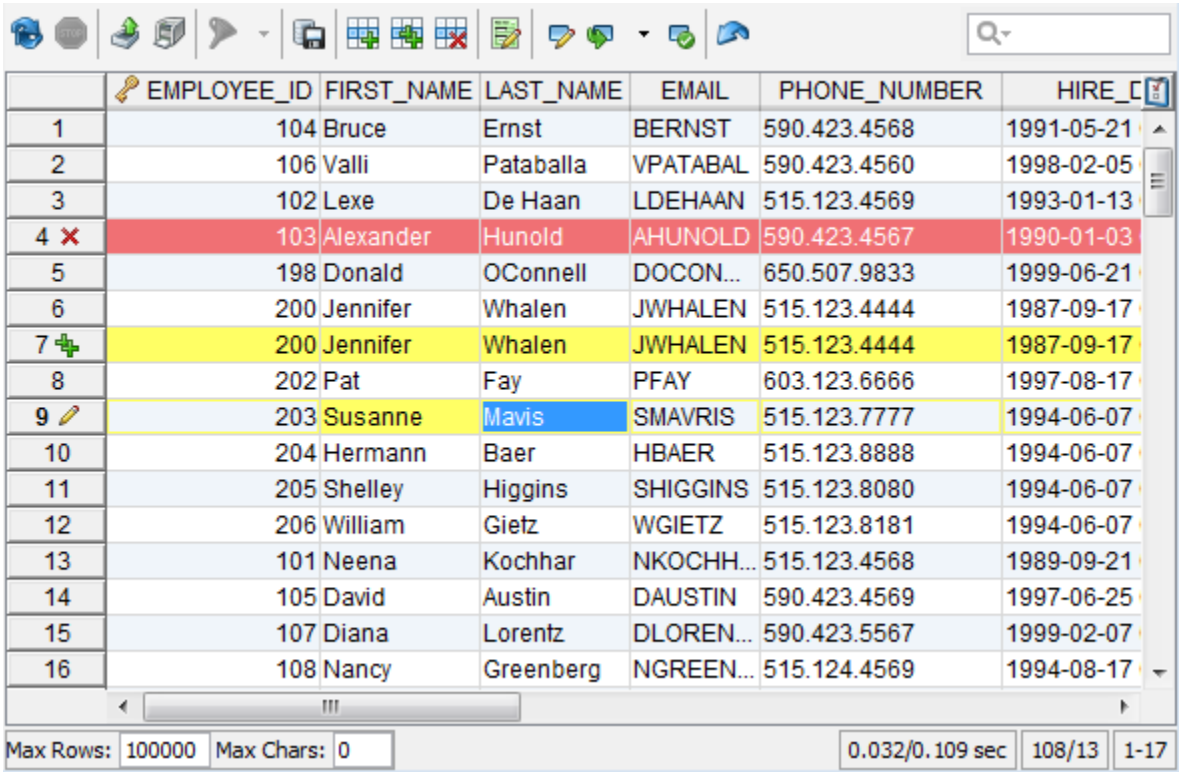
Information presented in the grid is obtained automatically by the viewer via a standard **SELECT * FROM table** statement, i.e., the object type having this viewer defined must be able to support getting a result set via this SQL statement.

It is important that the **Database Type** in the connection setup is properly set to match the database being accessed. The reason is that the identifiers (schema, database, table) are delimited automatically. Delimiters are database specific and if having the wrong database type set it may result in an error getting the result.

Here is a sample of the XML for the table-data viewer:

```
<DataView id="oracle-view-data" icon="Data" label="Data" viewer="table-data"/>
   <Input name="disableEdit" value="false"/>
</DataView>
```

And here is a screenshot of the **Data tab** based on the previous definition.



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_D
1	104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21
2	106	Valli	Pataballa	VPATABAL	590.423.4560	1998-02-05
3	102	Lexe	De Haan	LDEHAAN	515.123.4569	1993-01-13
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03
5	198	Donald	OConnell	DOCON...	650.507.9833	1999-06-21
6	200	Jennifer	Whalen	JWHALEN	515.123.4444	1987-09-17
7	200	Jennifer	Whalen	JWHALEN	515.123.4444	1987-09-17
8	202	Pat	Fay	PFAY	603.123.6666	1997-08-17
9	203	Susanne	Mavis	SMAVRIS	515.123.7777	1994-06-07
10	204	Hermann	Baer	HBAER	515.123.8888	1994-06-07
11	205	Shelley	Higgins	SHIGGINS	515.123.8080	1994-06-07
12	206	William	Gietz	WGIETZ	515.123.8181	1994-06-07
13	101	Neena	Kochhar	NKOCHH...	515.123.4568	1989-09-21
14	105	David	Austin	DAUSTIN	590.423.4569	1997-06-25
15	107	Diana	Lorentz	DLOREN...	590.423.5567	1999-02-07
16	108	Nancy	Greenberg	NGREEN...	515.124.4569	1994-08-17

Max Rows: 100000 Max Chars: 0 0.032/0.109 sec 108/13 1-17

Disable data editing

The default strategy for the table-data viewer is to automatically check whether the data can be edited or not. If editing is allowed a few related buttons will appear in the toolbar. However, sometimes you may want to disable editing completely for the **table-data** viewer. Do this with the following input element:

```
<Input name="disableEdit" value="true"/>
```

Viewer - table-rowcount

The table-rowcount viewer shows the row count for a (table) object.

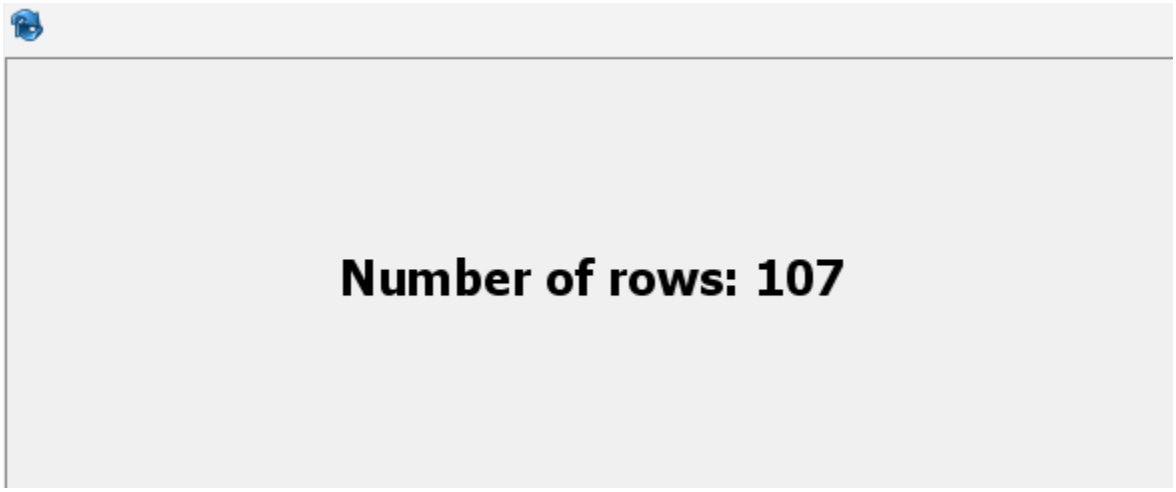
The row count is obtained automatically by the viewer via a traditional **SELECT COUNT(*) FROM table** statement, i.e., the object type having this viewer defined must be able to support getting a result set via this SQL statement.

It is that the **Database Type** in the connection setup is properly set to match the database being accessed. The reason is that the identifiers (schema, database, table) are delimited automatically. Delimiters are database specific and if having the wrong database type set it may result in an error getting the result.

Here is a sample of the XML for the table-rowcount viewer:

```
<DataView id="generic-table-rowcount" icon="RowCount" label="Row Count" viewer="table-rowcount"/>
```

And here is a screenshot of the Row Count tab based on the previous definition.



XML element - Command

Check the commands section for more information.

XML element - Input

The **Input** element is supported for some of the data viewers. Check the viewer sections for more information.

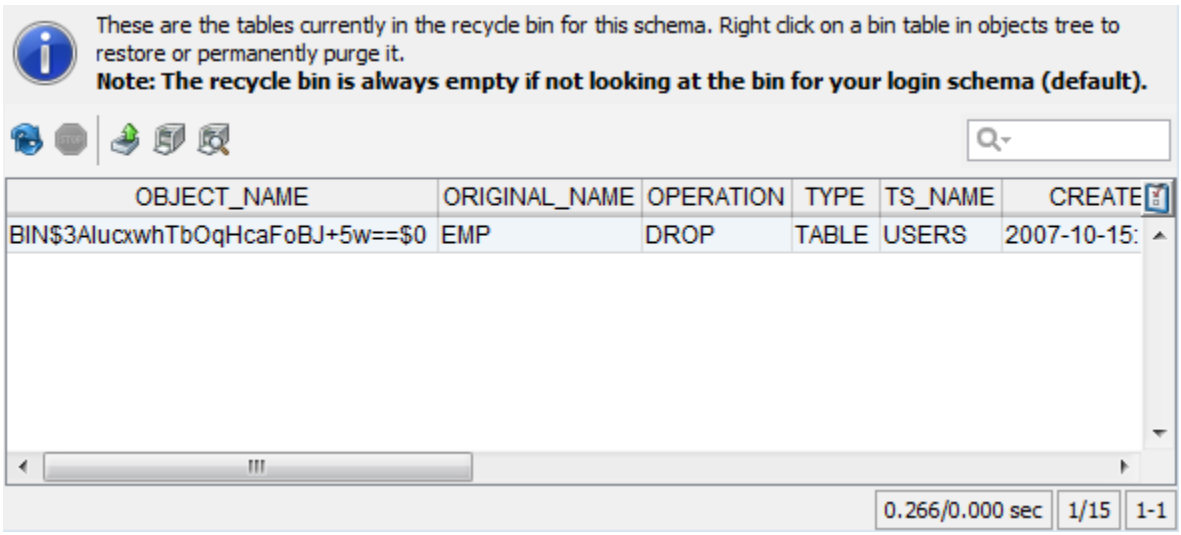
XML element - Message

The **Message** element is very simple as it just define a message that should appear at the top of the viewer. The text in the message may contain HTML tags such as <b> (bold), <i> (italic),
 (line break), etc.

Here is a sample of the XML for using the message element in a grid viewer:

```
<ObjectView type="RecycleBin">
   <DataView id="oracle-recyclebin-recyclebin" icon="RecycleBin" label="Recycle Bin" viewer="grid">
      <Command idref="oracle.getRecycleBin">
         <Input name="schema" value="${schema}"/>
         <Input name="login_schema" value="${dbvis-defaultCatalogOrSchema}"/>
      </Command>
      <Message>
         <![CDATA[
<html>
These are the tables currently in the recycle bin for this schema. Right click on a bin
table in objects tree to restore or permanently purge it.<br>
<b>Note: The recycle bin is always empty if not looking at the bin for your
login schema (default).</b>
</html>
         ]]>
      </Message>
   </DataView>
</ObjectView>
```

And here is a screenshot of the **Recycle Bin tab** based on the previous definition.


These are the tables currently in the recycle bin for this schema. Right click on a bin table in objects tree to restore or permanently purge it.
Note: The recycle bin is always empty if not looking at the bin for your login schema (default).
OBJECT_NAME
ORIGINAL_NAME
OPERATION
TYPE
TS_NAME
CREATE
BIN$3AlucxwhTbOqHcaFoBJ+5w==$0
EMP
DROP
TABLE
USERS
2007-10-15:
0.266/0.000 sec
1/15
1-1