CA ERwin® Data Modeler

ODBC Reporting Guide

Release 8.2.1

CA Technologies Product References

This document references the following CA Technologies products:

- CA ERwin® Data Modeler (CA ERwin DM)

Contact CA Technologies

Understanding your Support

Review support maintenance programs and offerings.

Registering for Support

Access the CA Support online registration site to register for product support.

Accessing Technical Support

For your convenience, CA Technologies provides easy access to "One Stop" support for all editions of CA ERwin Data Modeler, and includes the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

For information about other Home Office, Small Business, and Enterprise CA Technologies products, visit http://ca.com/support.

Provide Feedback

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

If you would like to provide feedback about CA Technologies product documentation, complete our short customer survey, which is also available on the CA Support website, found at http://ca.com/docs.

CA ERwin Data Modeler News and Events

Visit www.erwin.com to get up-to-date news, announcements, and events. View video demos and read up on customer success stories and articles by industry experts.

Documentation Changes

The following documentation updates have been made since the last release (r8 GA) of this documentation:

- DGM Schema (see page 50)—This section was added to the Schemas chapter to describe the tables included in this new schema.

Contents

Chapter 1: Introduction

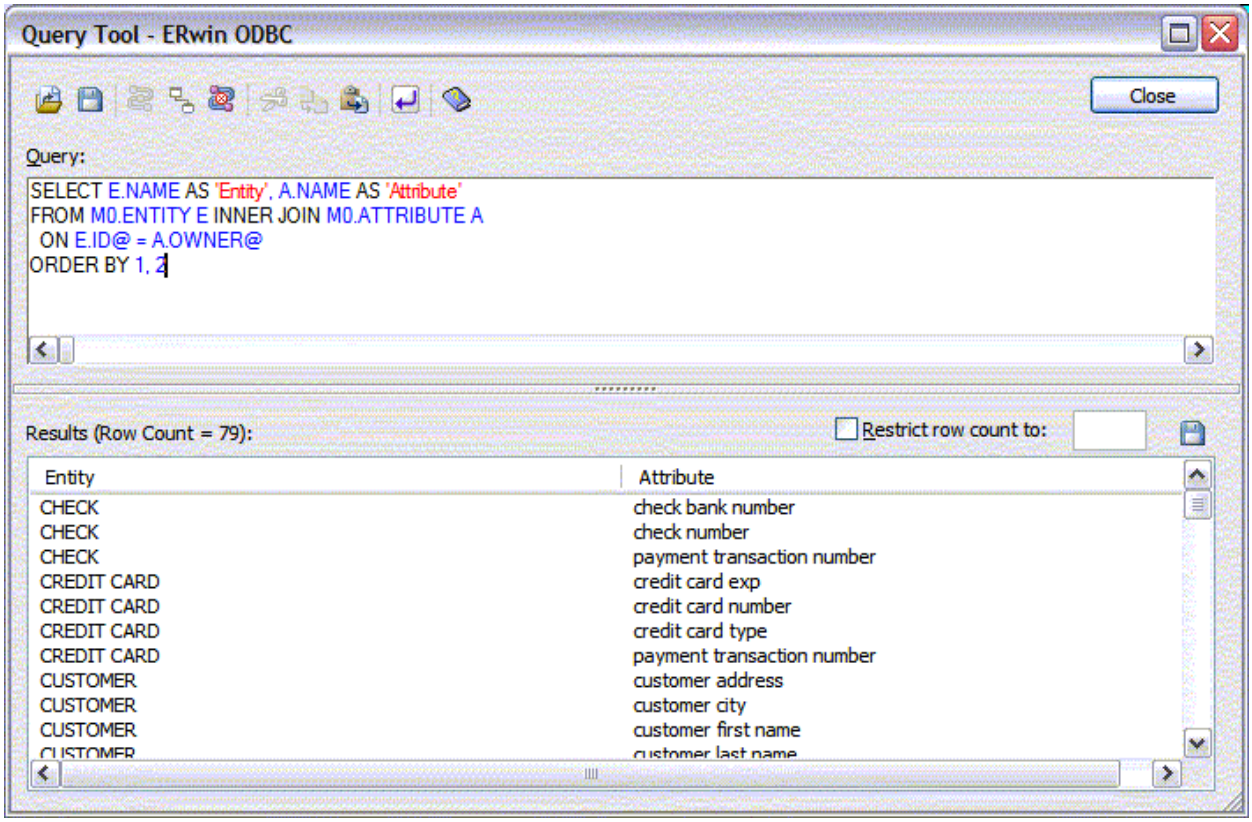
CA ERwin DM r8 implements an ODBC interface that allows SQL-based reporting or query tools to retrieve information from the product. You can report on the objects and properties in your models, the metadata defining those objects and properties, and the action log that records what changes were made to the model during the session.

This section contains the following topics:

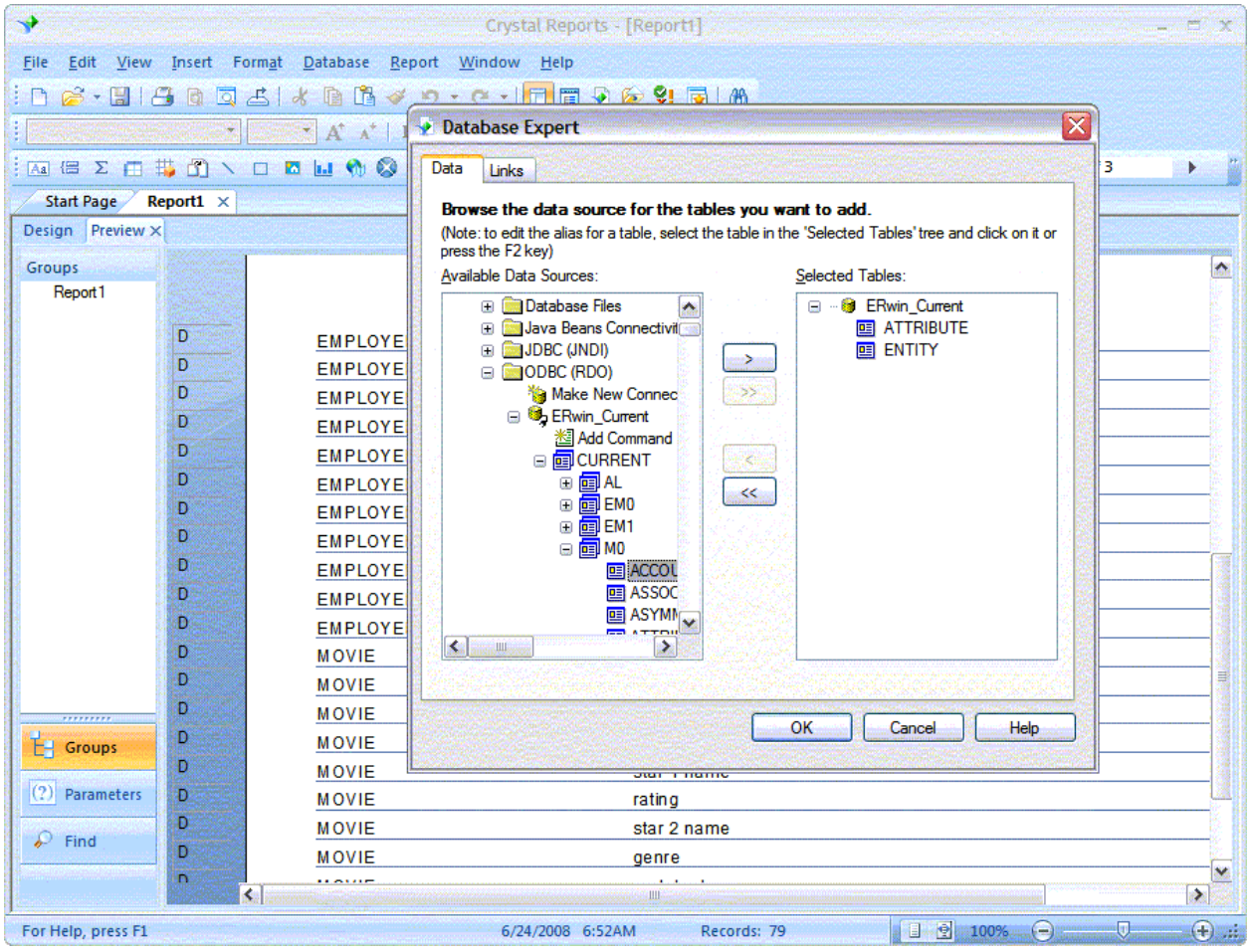
Query Tool

Any ODBC-compliant tool can be used to communicate with this component, such as a banded report writer, spreadsheet program, or ad hoc query tool. This includes Business Objects™ Crystal Reports® 2008, which is installed with CA ERwin DM. You also can use the built-in query tool with CA ERwin DM. To launch it, click Query Tool on the Tools menu.

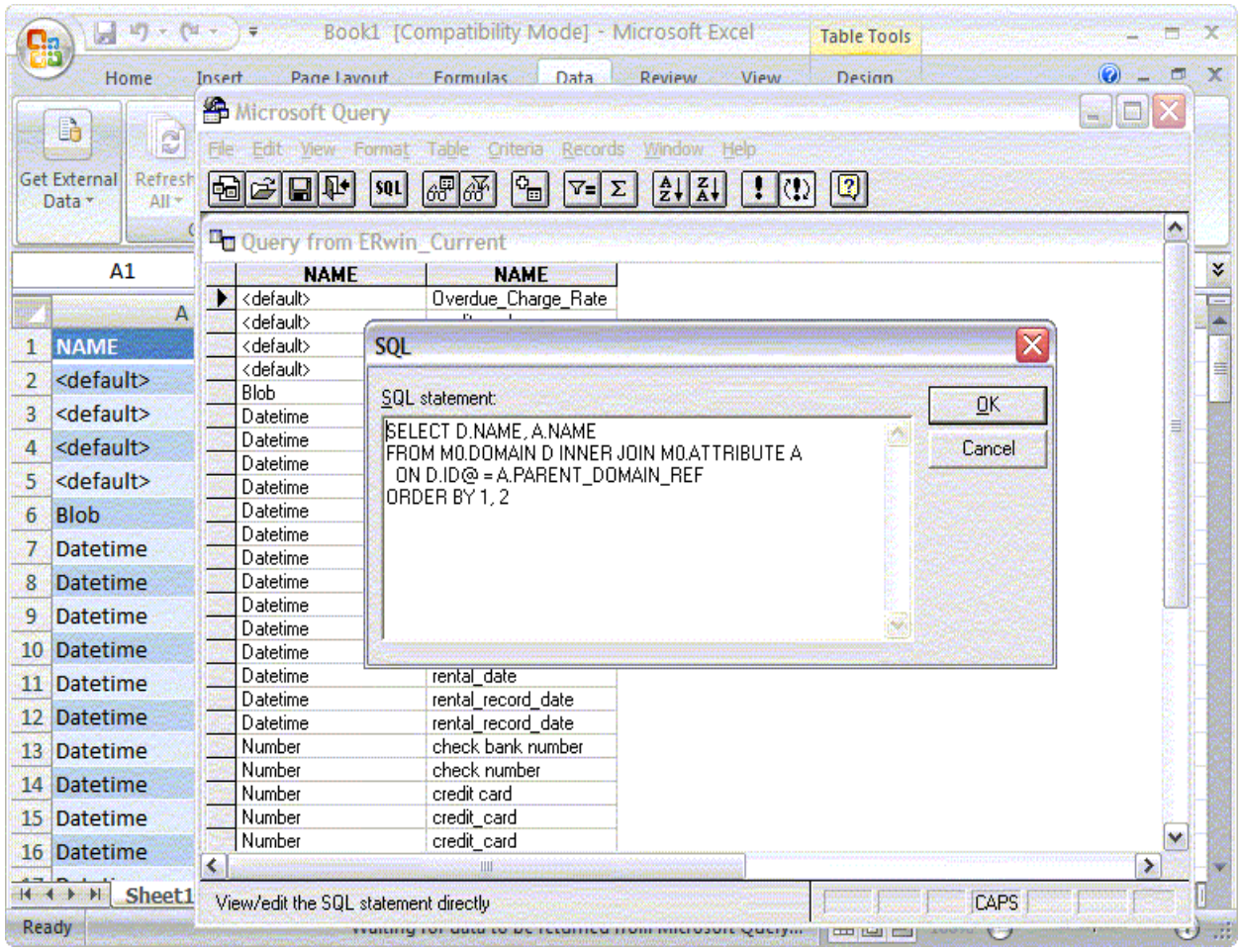
The following illustration demonstrates how CA ERwin DM's Query Tool reports against CA ERwin DM:



The following illustration demonstrates how Crystal Reports 2008 reports against CA ERwin DM:



The following illustration demonstrates how Microsoft® Office Excel® reports against CA ERwin DM:

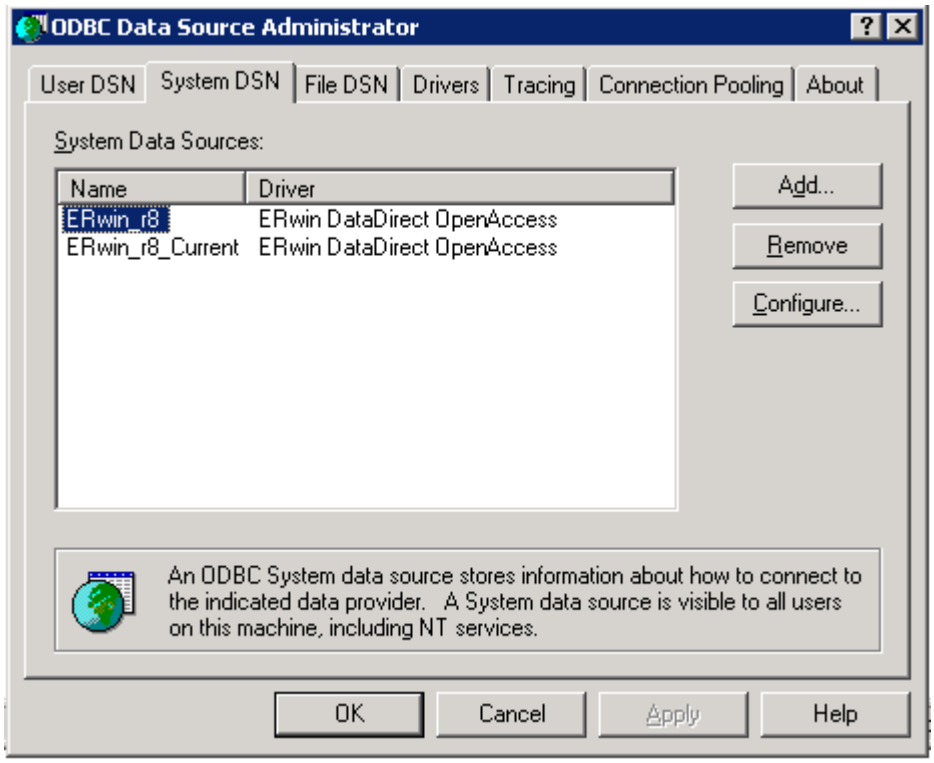


Chapter 2: ODBC Data Sources

This section contains the following topics:

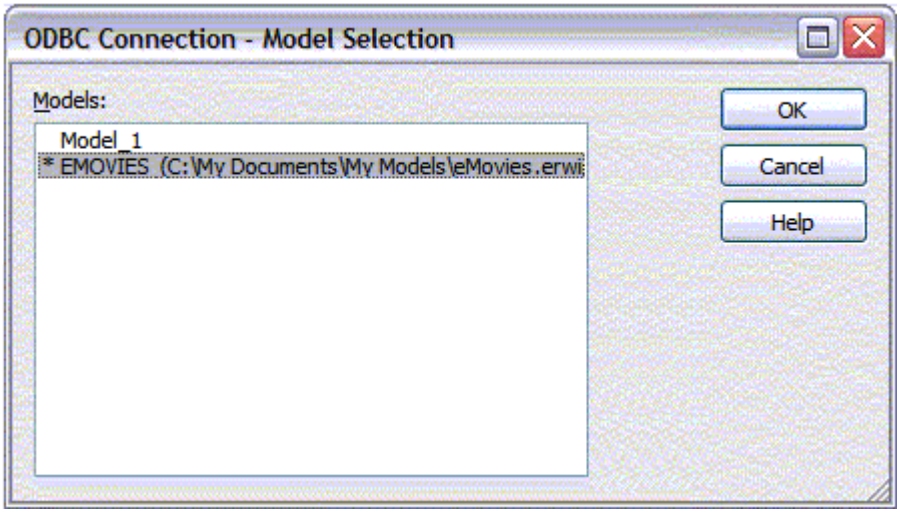
Using ODBC Data Sources

The installation of CA ERwin DM r8 creates two ODBC data sources on your machine. One is named *ERwin_r8* and the other *ERwin_r8_Current*. If you want to confirm that these are present, you can invoke the Windows ODBC Data Source Administrator and go to the System DSN tab page, as shown in the following illustration:

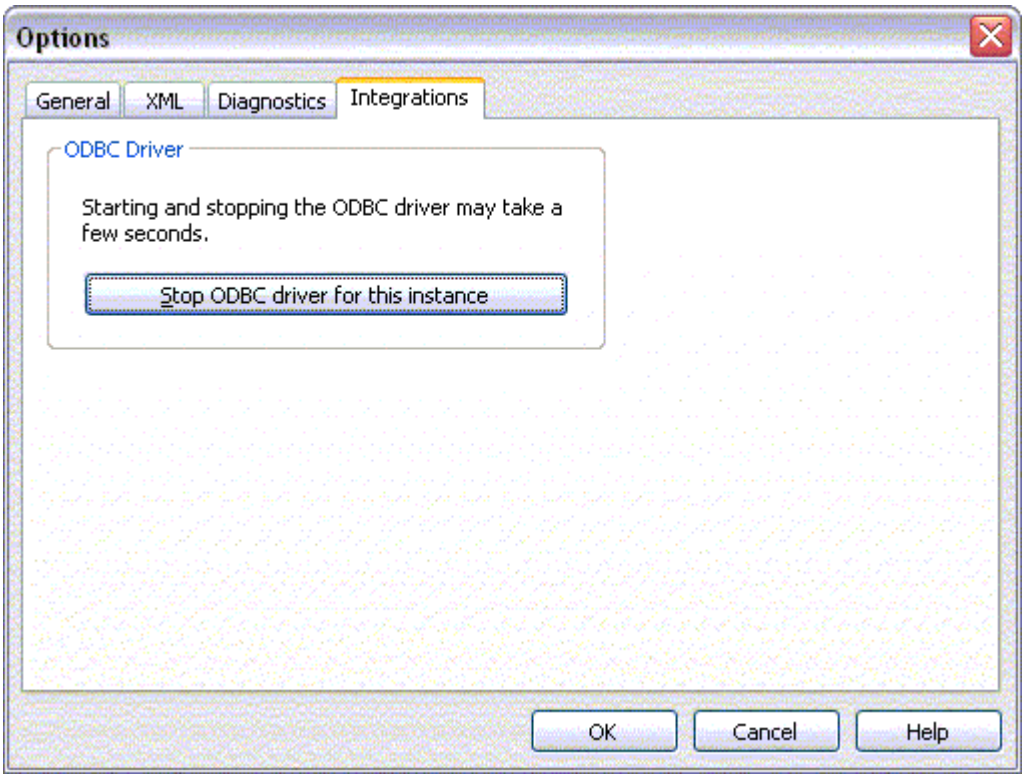


Use the connection mechanism of your query tool to connect to one of these data sources. No user ID or password is required by CA ERwin DM.

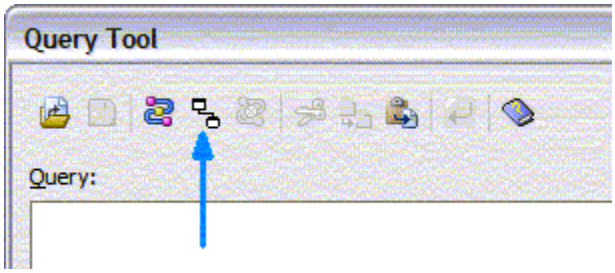
A connection is made to a specific model loaded into the product. The *ERwin_r8_Current* data source connects to the current model in CA ERwin DM at the time of connection. The *ERwin_r8* data source presents a dialog that allows you to select the target model for the connection. The current model is denoted with an asterisk, as shown in the following illustration of the Model Selection dialog:



If you have multiple copies of CA ERwin DM running, the ODBC driver can serve data from only one instance. By default, this is the first one started. If you want to change which instance of CA ERwin DM is handling the ODBC connection, first stop the driver on the one that is running using the Integrations page on the Options dialog, then start the driver on the second copy of CA ERwin DM from the same place:

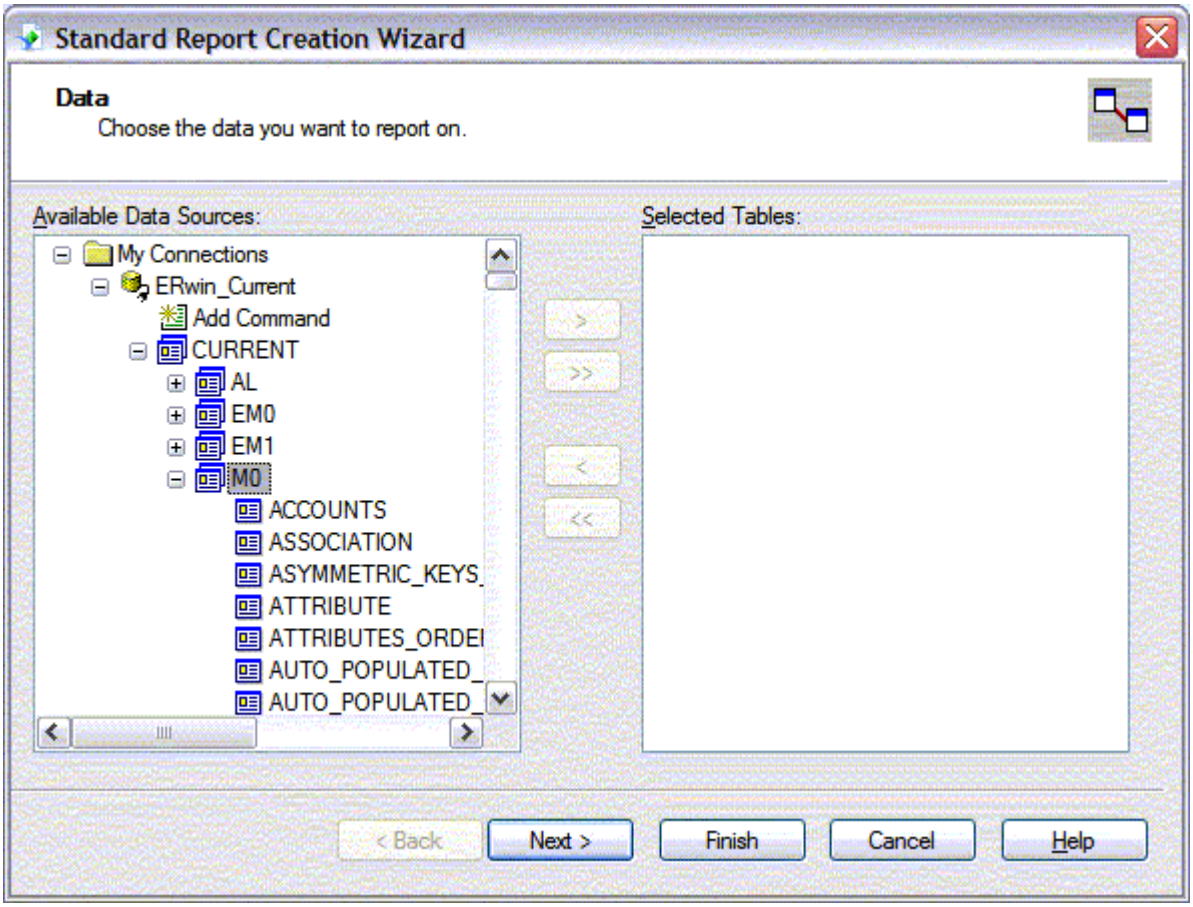


For the ERwin Query Tool, use the Connect to ERwin ODBC toolbar item as quick way to make this connection:



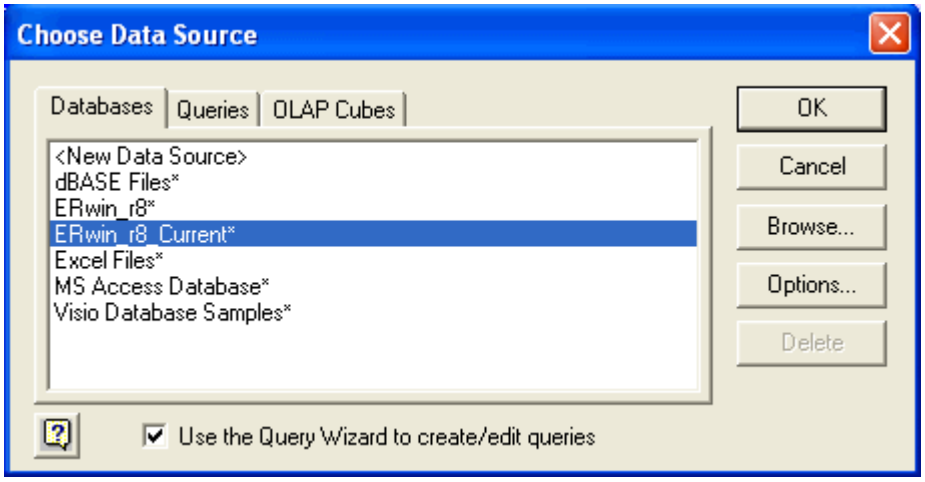
To connect to CA ERwin DM using Crystal Reports 2008, follow this process:

- Ensure CA ERwin DM is running with a model loaded.
- Click Report Wizard on the startup pane. If you have not already created a report against CA ERwin DM in Crystal Reports 2008, you will not see *ERwin_r8_Current* under My Connections.
- Click Create a New Connection in the Available Data Sources tree.
- Expand the ODBC (RDO) entry, select *ERwin_r8_Current* in the popup window, and click Finish. You should now see *ERwin_r8_Current* in My Connections.
- Expand *ERwin_r8_Current*, and then expand *CURRENT*. You should now see a list of tables available to you and you can start creating a report using the wizard.

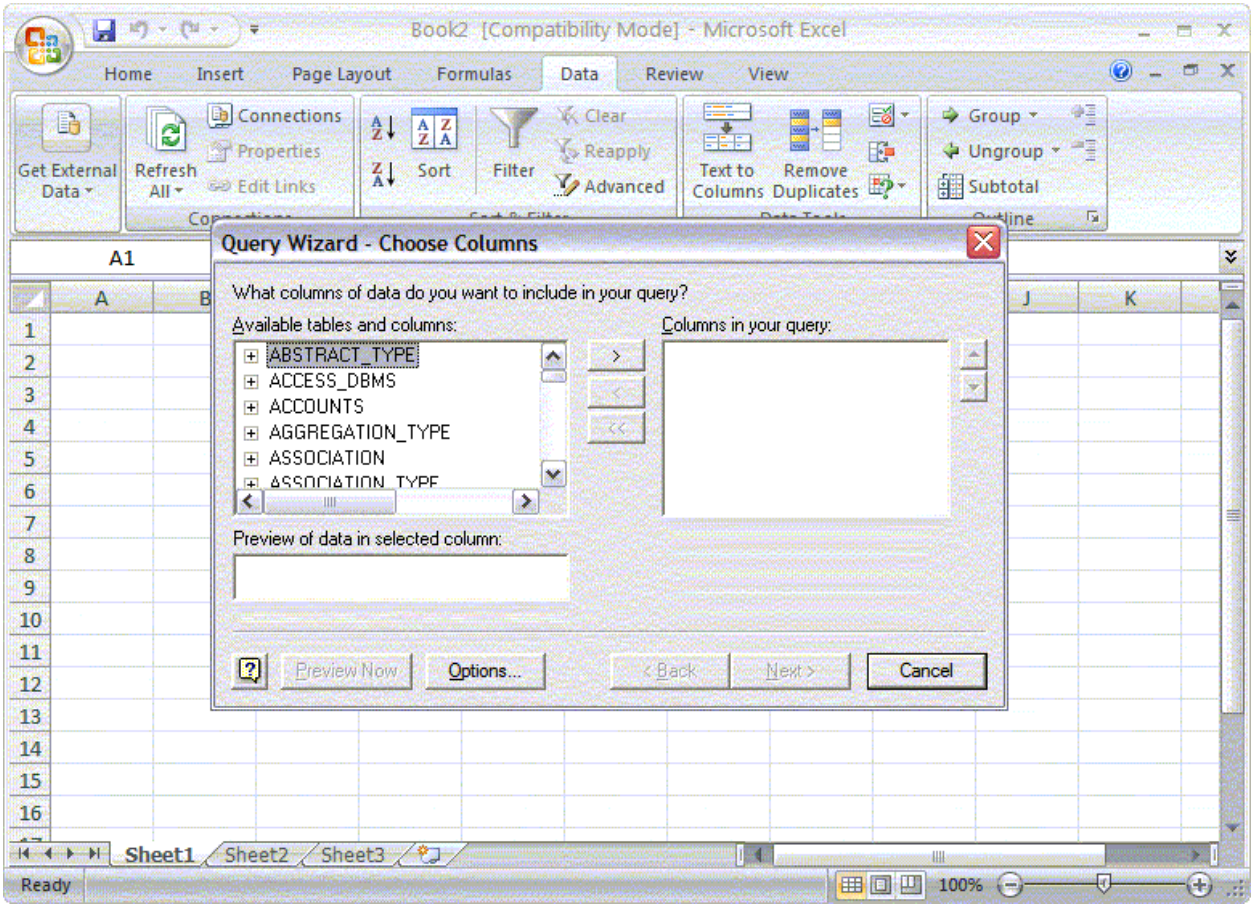


To connect to CA ERwin DM using Microsoft Office Excel 2007, follow this process:

- Ensure CA ERwin DM is running with a model loaded.
- Click the Data menu item.
- Click the From Other Data Sources tool and select the From Microsoft Query option. You are asked to choose a data source:



- Select the *ERwin_r8_Current* data source, click OK in the Login dialog, and you are connected to CA ERwin DM and ready to start constructing a query. You can either use the Wizard or click Cancel and answer 'Yes' to "Do you want to continue editing this query in Microsoft Query?" to get more direct access to SQL.



Chapter 3: Schemas

This section contains the following topics:

Exposing Data

Various components of CA ERwin DM expose data using the ODBC driver. Each of these areas represents itself as a schema. For example, the tables representing information in the model are in the M0 schema, while the tables representing information from the Action Log are in the AL schema.

Note: For CA ERwin DM r8, all table names are unique, so it is not necessary to qualify them with a schema in a query. However, it is possible that future releases of CA ERwin DM will have tables in one schema with the same name as tables in another schema.

Changes to Column Names

Beginning with r7.3.10, column names in the underlying tables are suffixed with '@' (if they do not already have one) and prefixed as described below:

- Foreign key attribute columns: **CHILD_ATTRIBUTE_**
- Entity owning the foreign key attribute columns: **CHILD_ENTITY_**
- Parent attribute columns: **PARENT_ATTRIBUTE_**
- Entity owning the parent attribute: **PARENT_ENTITY_**
- Contributing relationship: **RELATIONSHIP_**

For example, the NAME column for the foreign key attribute becomes CHILD_ATTRIBUTE_NAME@ and the ID@ column for the parent attribute would be PARENT_ATTRIBUTE_ID@.

M0 Schema

The M0 schema contains the modeling data in a CA ERwin DM model.

The various classes of objects and properties in a CA ERwin DM model form the definitions of tables and columns exposed using the ODBC interface. To find the class names used in CA ERwin DM's metamodel, see the document *ERwin Metamodel Reference*. All spaces and periods in class names are replaced with underscores. The ODBC interface capitalizes the names, but the driver is not case-sensitive. For example, the *Entity* class becomes the *ENTITY* table and the *Long_Id* property becomes the *LONG_ID* column.

M0 Schema Tables

This section describes the tables in the M0 schema.

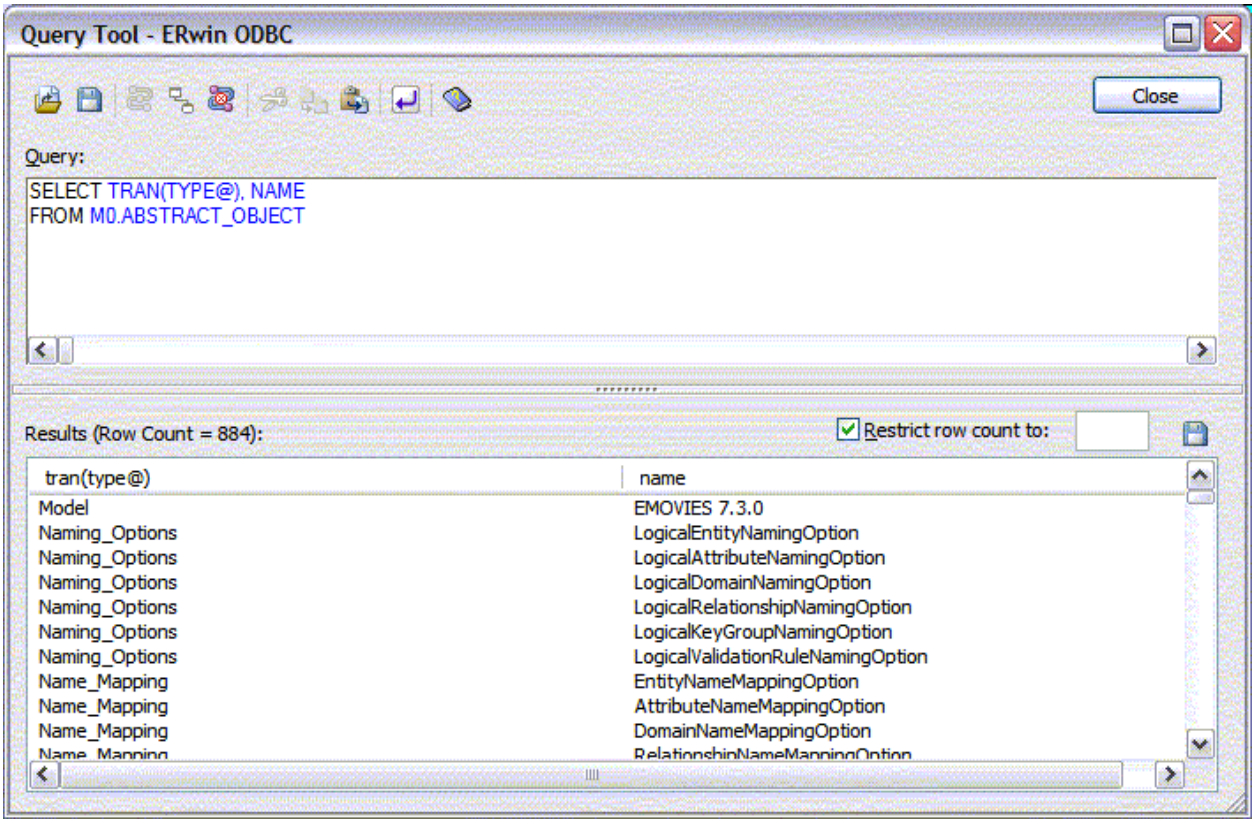
ABSTRACT_OBJECT Table

The M0 schema contains the *ABSTRACT_OBJECT* table. This table, based on the *Abstract_Object* class, has a row for every object in the model that is not marked as *Tag_Is_Internal*. The following table describes the columns in the *ABSTRACT_OBJECT* table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. This is the short ID of an object, which is unique in the model, but may change from session to session.
TYPE@	INTEGER	The class ID of the object.
OWNER@	INTEGER	The ID of the owning object.
NAME	VARCHAR	The name of the object. For a dual object, this will be the logical name.
LONG_ID	CHAR(67)	The long ID of the object. This is the permanent ID assigned to each object in a CA ERwin DM model.

You can use a query similar to the following to see a list of the types and names of all objects in the model.

Example: Query for returning the list of types and names of all model objects



Note: The *TRAN()* function is described in the Functions section.

Concrete Object Tables

The M0 schema also contains a table for every concrete type of object that can be created in CA ERwin DM, excluding types that are marked as *Tag_Is_Internal*. For example, there will be an *ENTITY* table that holds rows describing *Entity* objects in the model, an *ATTRIBUTE* table that holds rows describing *Attribute* objects, and so on.

In these tables, each row represents an instance of an object of that type in the model. There are three columns that are present on each instance of this type of table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model.
TYPE@	INTEGER	The class ID of the object.
OWNER@	INTEGER	The ID of the owning object.

There is also a column for each possible property on the object that has a scalar data type and is not flagged as *Tag_Is_Internal*. For example, there will be a *NAME* column, a *LONG_ID* column, and so on. The data types of these columns will depend upon the underlying data types of the properties represented.

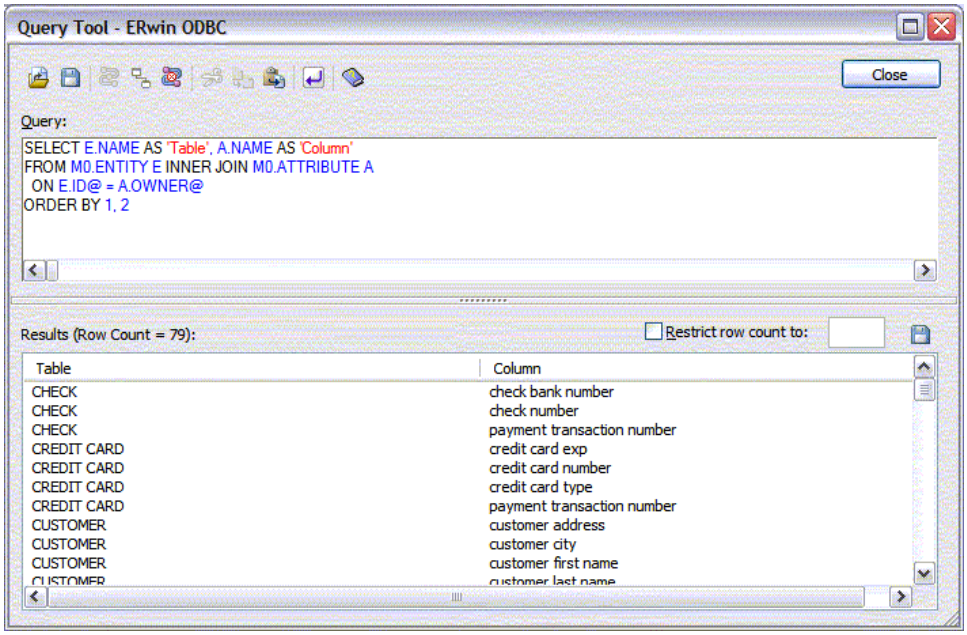
The following table maps the property data types to those exposed using the ODBC interface:

Property Data Type	Column Data Type	Comments
Integer	INTEGER	(None)
String	VARCHAR	(None)
Long Id	CHAR(67)	The string will be formatted with the standard Long Id formatting. For example: {51A4D991-46CE-4530-AC4F-63757E46A494}+00000000
Binary	LONG VARBINARY	(None)
Time	TIMESTAMP	(None)
Boolean	CHAR(1)	The value will be set to *T* or *F*. **Note:** Many Boolean properties in CA ERwin DM are actually three-state: *True*, *False*, or *Not Set*. If the property is *Not Set*, it will appear as a NULL.
Point	CHAR(23)	The string will be formatted as the *X* and *Y* components, separated by a period. For example: -2134.898

Property Data Type	Column Data Type	Comments
Rectangle	CHAR(47)	The string will be formatted as the *Left*, *Top*, *Right*, and *Bottom* components, separated by periods. For example: -2134.898.1394839.193839
Resource	CHAR(23)	The string will be formatted as the *Id* and *Resource Selector*, separated by a comma. For example: 1,0c138988
64-bit Integer	INTEGER	(None)
Short Id	INTEGER	(None)
Size	CHAR(23)	The string will be formatted as the width and height components, separated by a period. For example: 20.30

To illustrate this, the *ATTRIBUTE* table would have columns such as *NAME*, *DEFINITION*, *DATATYPE*, *NULL_OPTION*, and so on. You can use a query similar to the following to determine the names of all the *Attribute* objects in your model.

Example: Query for returning the list of names of all the *Attribute* model objects



Vector Property Tables

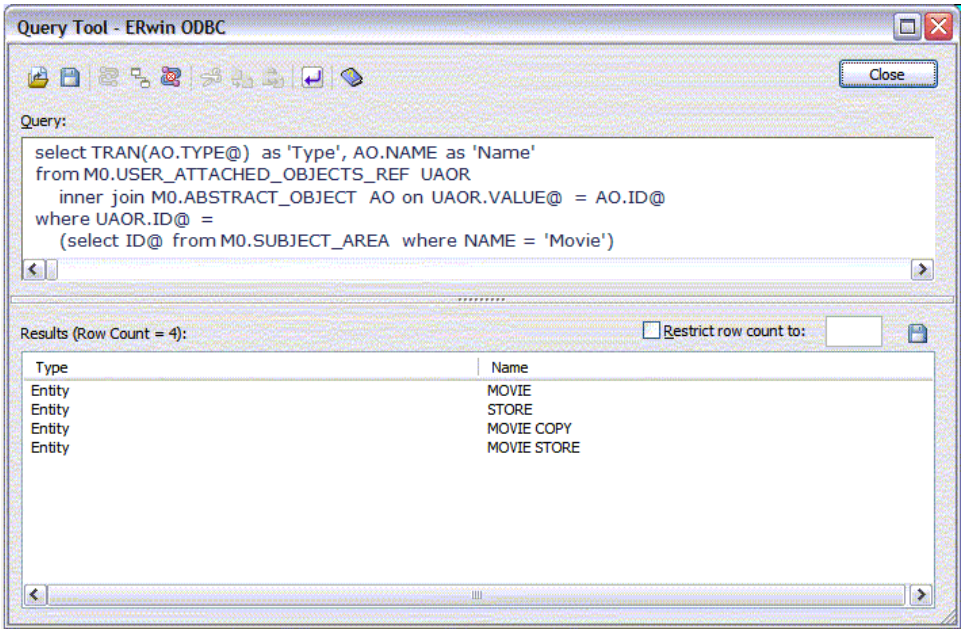
The M0 schema also contains one table for each type of property that has a vector data type, excluding properties flagged as *Tag_Is_Internal*. In these tables, each row represents a single value in an instance of the property.

There are three columns present on each instance of this type of table. The following table describes the columns found on all Vector Property type tables:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object holding the property.
SEQUENCE_NUMBER@	INTEGER	The zero-based index of the value in the property's vector.
VALUE@	See *Description*	This column will contain the value. The data type of this column will depend upon the data type of the underlying property (see the previous table that describes the data types exposed using the ODBC interface).

For example, a *Subject_Area* object holds a vector property that contains the IDs of all *Entity* objects that are members. This property has a class name of *Referenced_Entities_Ref*, which becomes the *REFERENCED_ENTITIES_REF* table. You can use the following query if you wanted to see the names and types of all members of the *Movie* subject area in the *eMovies.ERWIN* model.

Example: Query for returning the members of a subject area



Functions

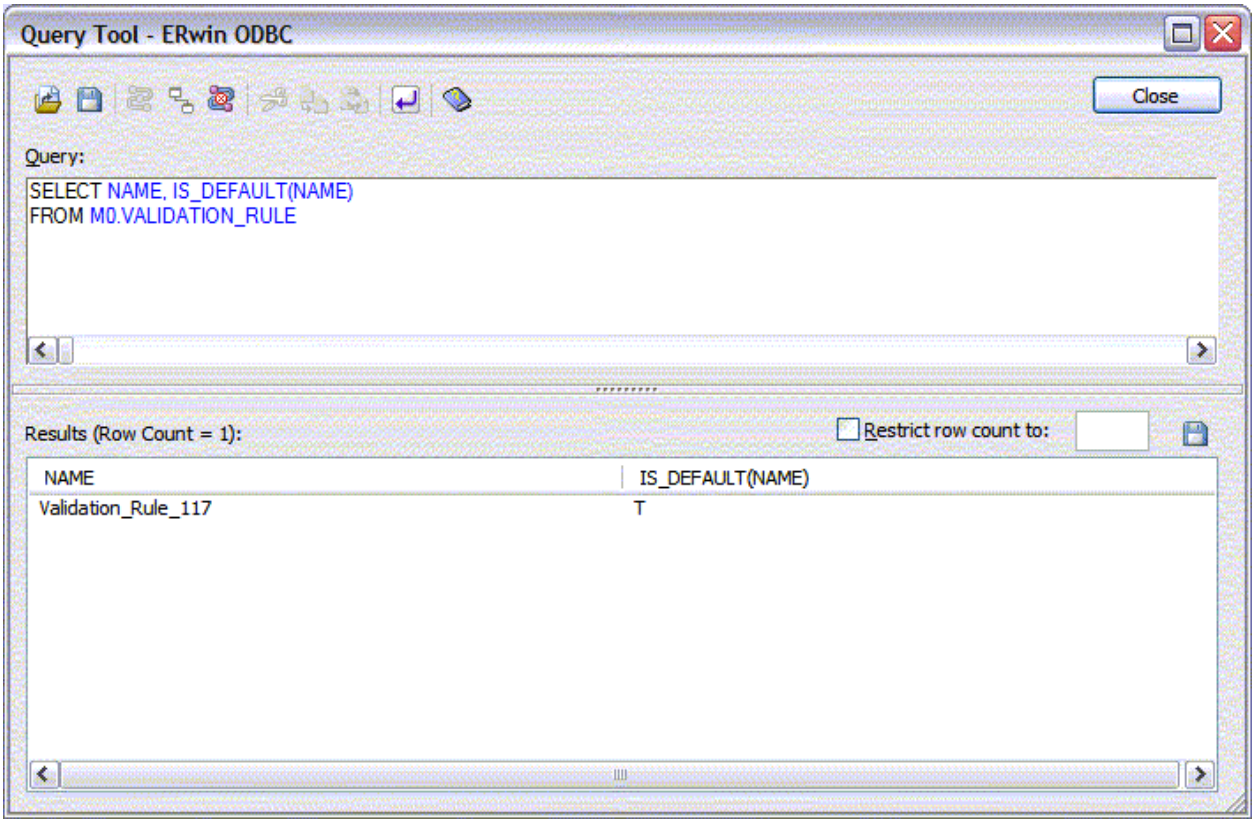
The M0 schema supports the following scalar functions in addition to the standard ODBC functions.

IS_DEFAULT

This function evaluates to a single character and is applied to columns that contain property values. It will evaluate to *T* if the value in the property is a default value assigned by CA ERwin DM, otherwise, it will evaluate to *F*.

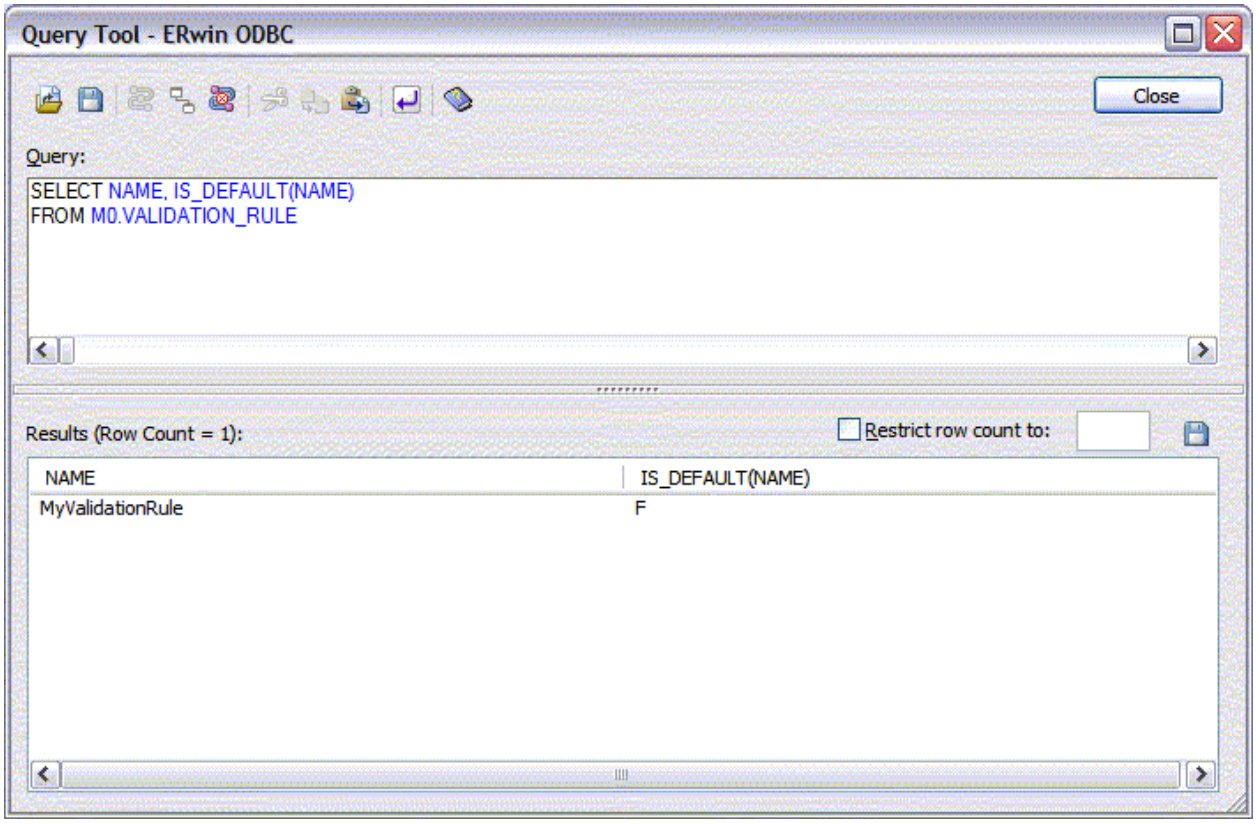
For example, when you create a new *Validation_Rule* object in CA ERwin DM, it is assigned an application-generated name of the form *Validation_Rule_XX* where *XX* is the ID. A query would show that this is a default name.

Example: Query for returning the default name of an object



If you change the name and re-execute the query, the status changes:

Example: Query for returning the name of an object showing the change in status from its default



IS_HARDENED()

This function evaluates to a single character and is applied to columns that contain property values that can be hardened.

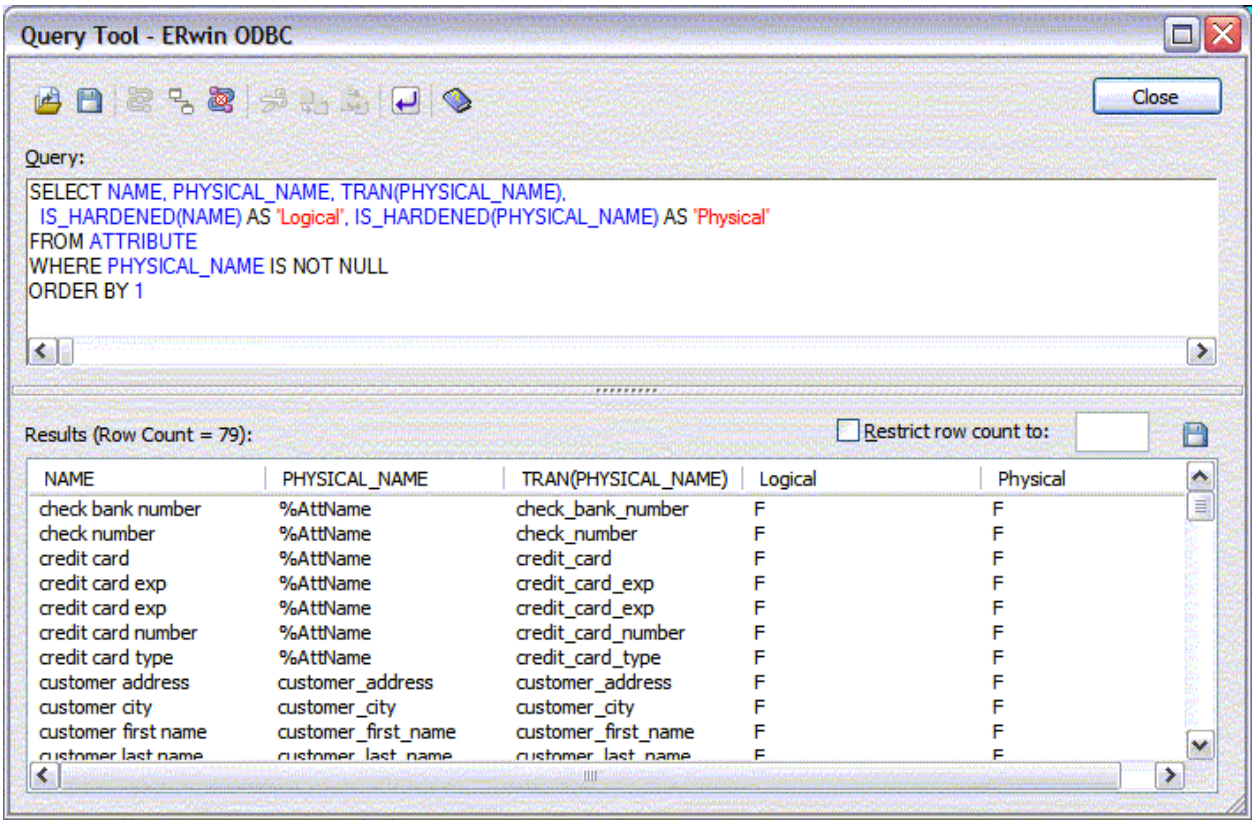
Note: CA ERwin DM r8 only allows hardening of name values, so this function only has meaning when applied to the *NAME* column and the *PHYSICAL_NAME* column of the *ATTRIBUTE*, *DEFAULT*, *DOMAIN*, *ENTITY*, *KEY_GROUP*, *RELATIONSHIP*, and *VALIDATION_RULE* tables. Future releases of CA ERwin DM may extend hardening to more property types.

It will evaluate to *T* if the value in the property has been hardened, otherwise, it will evaluate to *F*.

For example, suppose you use the Name Hardening Wizard to harden the *Physical_Name* of all *Attribute* objects in the *eMovies.ERWIN* model. The following query shows the results of the *IS_HARDENED()* function.

Note: The row restriction is to filter out view columns, which are not type-split from table columns as of CA ERwin DM r8.

Example: IS_HARDENED() function applied to objects



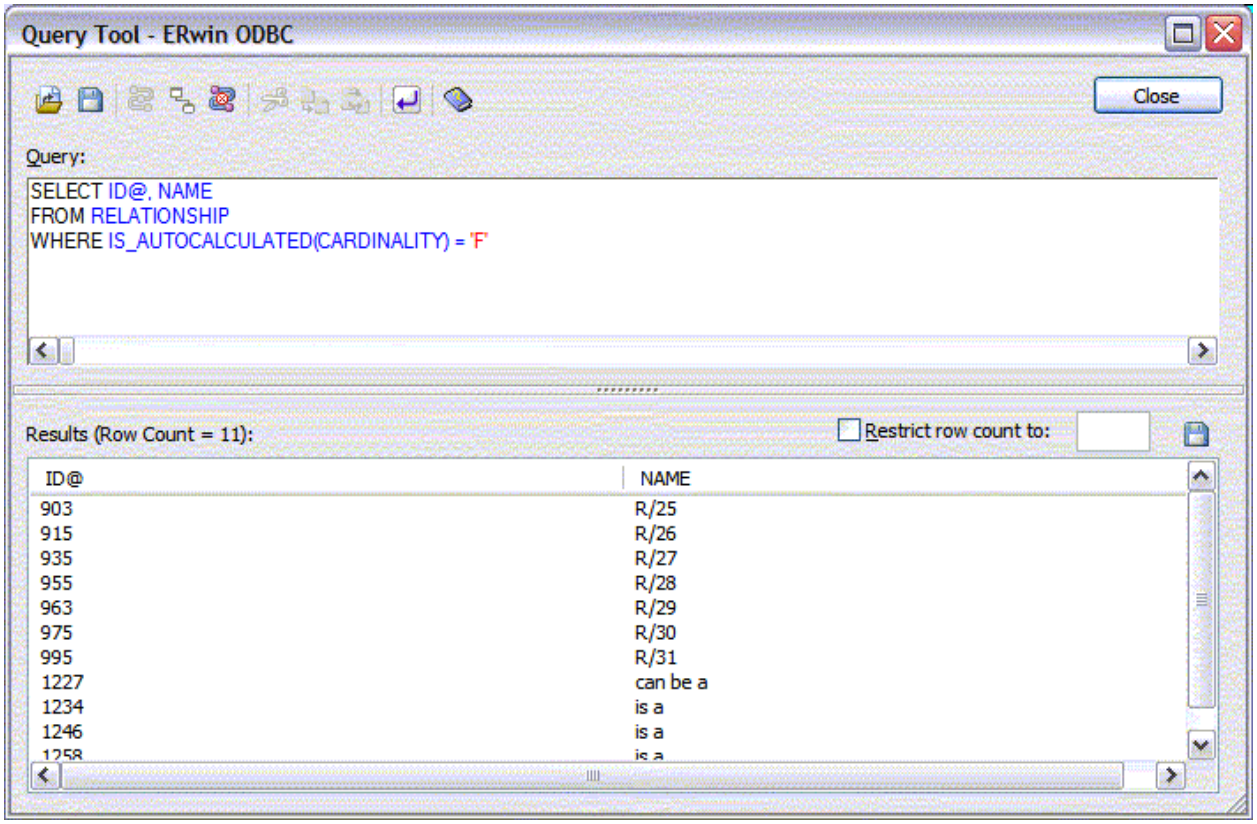
IS_AUTOCALCULATED()

This function evaluates to a single character and is applied to columns that contain property values. It evaluates to *T* if the value in the property is in an auto-calculate state, and to *F* if it is not.

Note: CA ERwin DM r8 only applies this state to the *Cardinality* property on a *Relationship* object. Future releases of CA ERwin DM may extend the auto-calculation state to more property types.

You can use a query similar to the following to see which *Relationship* objects in the model have explicit cardinality set.

Example: Query for returning *Relationship* objects in the model that have explicit cardinality set

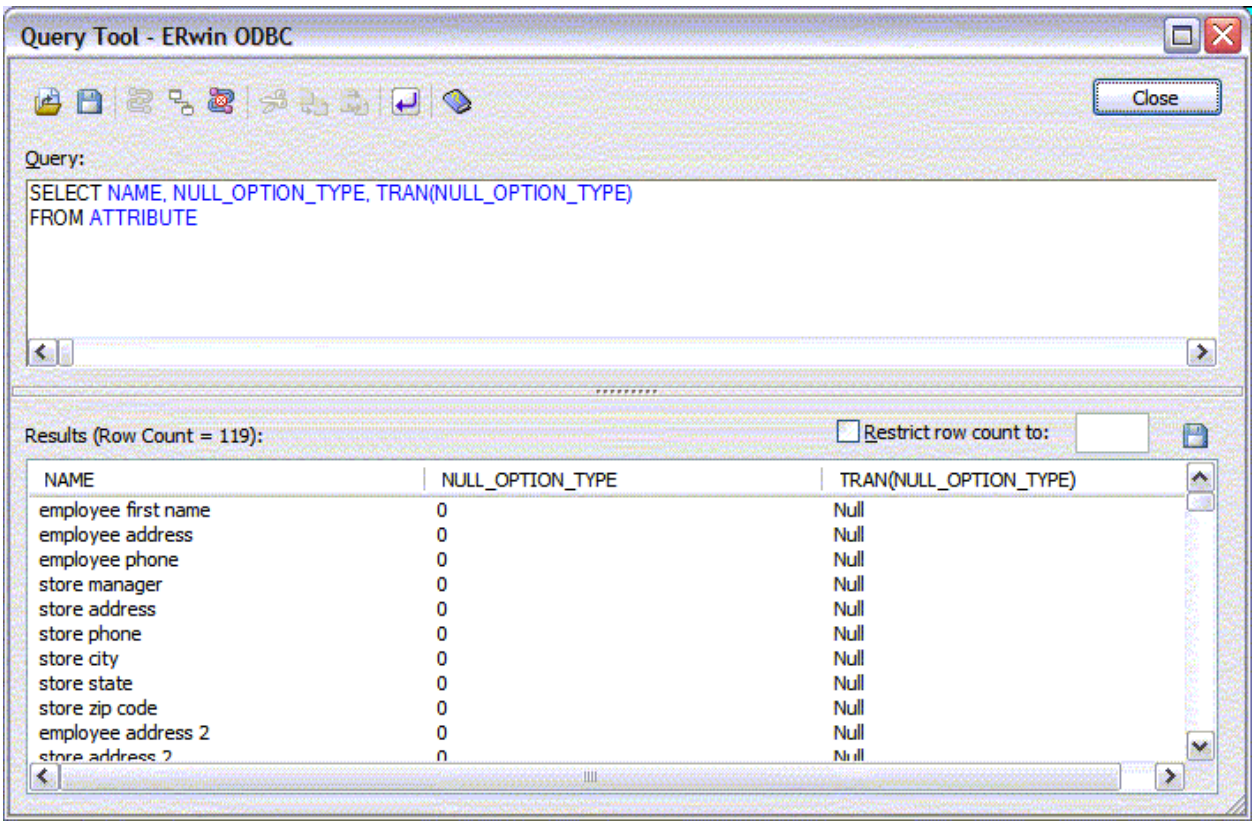


TRAN()

CA ERwin DM stores a number of property values in an encoded form that is not friendly for display purposes. Additionally, some properties contain macro code that is expanded to form a display value. If you retrieve the value from the property directly, you will get the *raw* value.

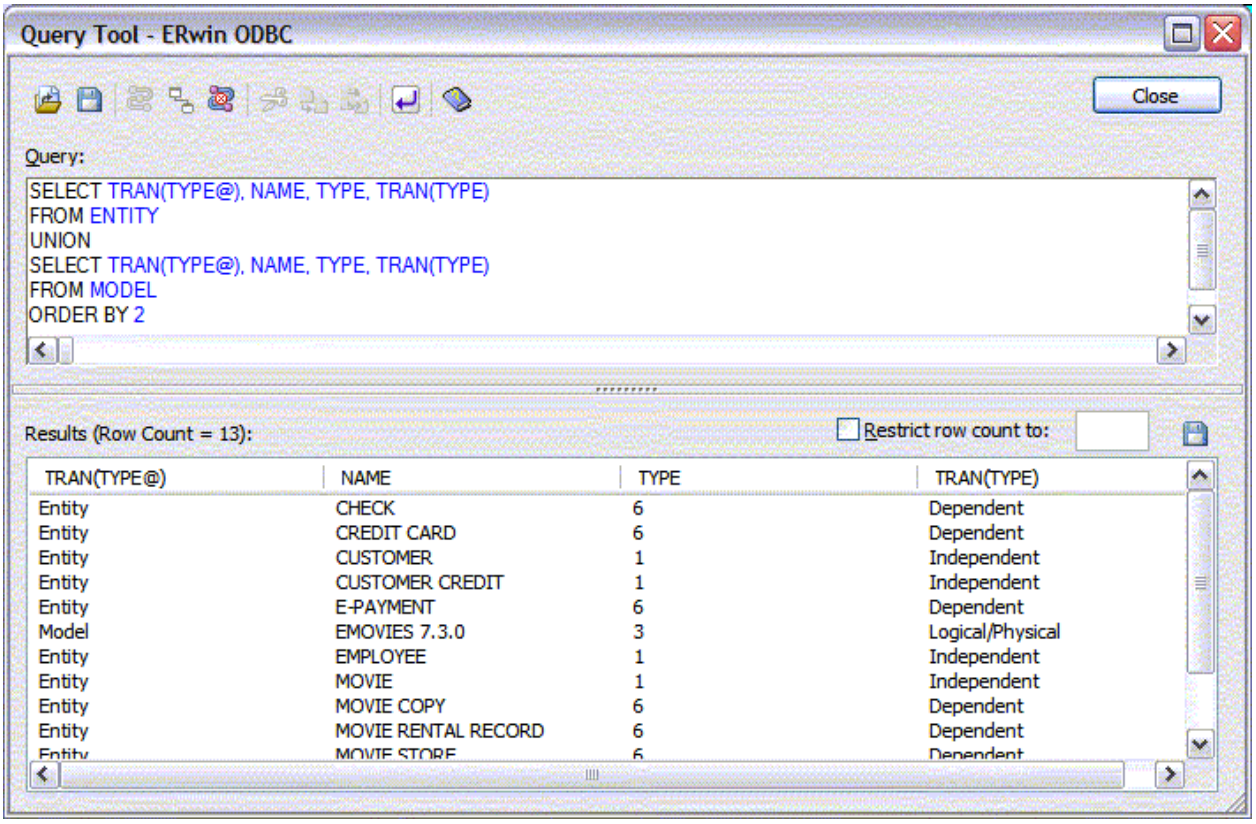
If the *TRAN()* function is applied to the column, the raw value is *translated* into a display value. The result is always a string. If this function is applied to a column that has no translated value, a string representation of the raw value is returned.

Example: TRAN() function applied to a column



The *TRAN()* function can also be applied to the *TYPE@* column to retrieve an English version of the type identifier, such as the class name:

Example: TRAN() function applied to the TYPE@ column



M1 Schema

The M1 schema contains the metadata in a CA ERwin DM model.

The various classes of objects and properties in a CA ERwin DM model form the definitions of tables and columns exposed using the ODBC interface. To find the class names used in CA ERwin DM's metamodel, see the document *ERwin Metamodel Reference*. All spaces and periods in class names are replaced with underscores. The ODBC interface capitalizes the names, but the driver is not case-sensitive. For example, the *Object_Type* class becomes the *OBJECT_TYPE* table and the *Long_Id* property becomes the *LONG_ID* column.

Several of the table descriptions below indicate the base columns exposed. These represent the fundamental columns of the table that will be present across various releases of CA ERwin DM. These tables will also expose columns showing tag values for the item being described. A tag is a characteristic of an item such as "does it show up in the Explorer?" (*TAG_IS_EXPLORER_SUPPRESSED*). Tags can change from release to release of CA ERwin DM as the needs of the product change. You should consult the *ERwin Metamodel Reference* document for a description of the tags for a given release.

M1 Schema Tables

This section describes the tables in the M1 schema.

OBJECT_TYPE Table

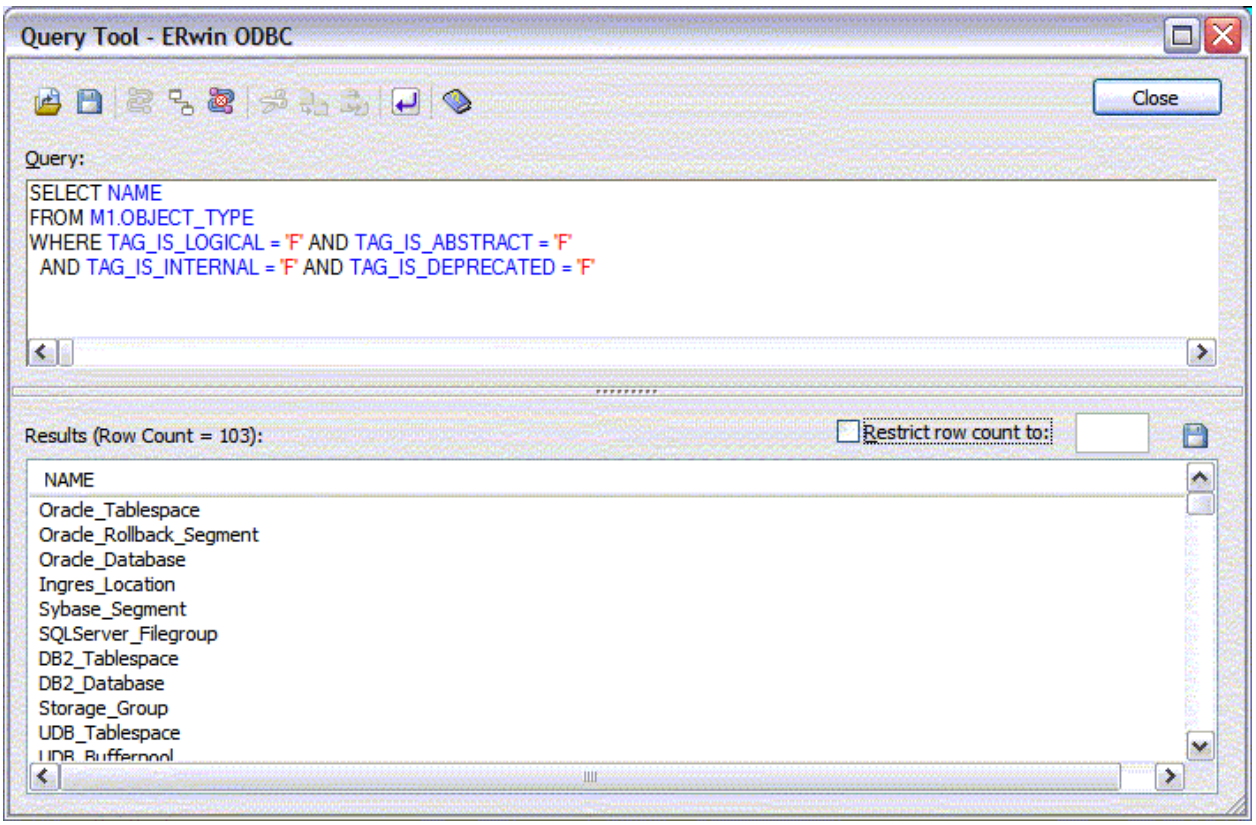
The *OBJECT_TYPE* table contains information about the types of objects permitted in a CA ERwin DM model. In other words, rows in this table will correspond to tables exposed by the M0 schema. The following table describes the base columns in the *OBJECT_TYPE* table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. Though this is a short ID, it never changes.
NAME	VARCHAR	The name of the type.
LONG_ID	CHAR(67)	The long ID of the type. This is provided for consistency with M0.
DEFINITION	VARCHAR	The definition of the type.
TAG_IS_ABSTRACT	CHAR(1)	Does the object type represent an abstract object (one that cannot be created in the model)? Abstract object types are not exposed by the M0 schema.

Column Name	Data Type	Description
TAG_IS_INTERNAL	CHAR(1)	Is the object type marked as Internal? Internal types are not exposed by the M0 schema.
PARENT_REF	INTEGER	If the object type is derived from an abstract type, this is the abstract type's ID.
TAG_IS_DEPRECATED	CHAR(1)	Has this type been deprecated?
TAG_DEPRECATION_LEVEL	INTEGER	A value of *2* indicates that the type is still in the metamodel, but is no longer active. A value of *3* indicates the type is no longer in the metamodel. A value of *-1* indicates the type is for a future release.
TAG_RELEASE_DEPRECATED	VARCHAR	The release in which the type was deprecated. If the type is deprecated, but this value is NULL, then the release was 7.0.
TAG_RELEASE_ADDED	VARCHAR	The release in which the type was added.

You can use a query similar to the following to find the active, physical-only object types.

Example: Query for returning the active, physical-only object types



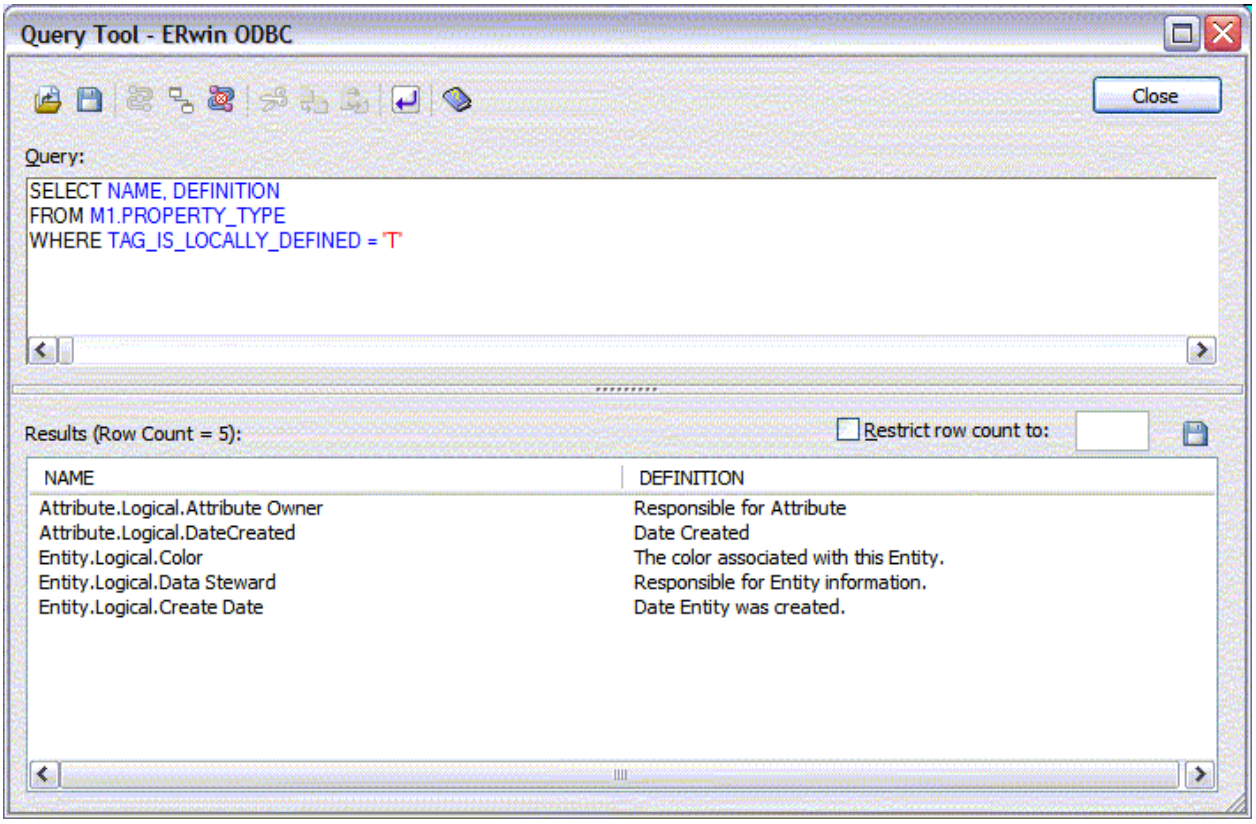
PROPERTY_TYPE Table

The *PROPERTY_TYPE* table contains information about the types of properties permitted in a CA ERwin DM model. In other words, rows in this table will correspond to columns on M0 tables (for scalar properties) or to M0 tables (for vector properties). The following table describes the base columns in the *PROPERTY_TYPE* table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. Though this is a short ID, it never changes.
NAME	VARCHAR	The name of the type.
LONG_ID	CHAR(67)	The long ID of the type. This is provided for consistency with M0.
DEFINITION	VARCHAR	The definition of the type.
TAG_IS_INTERNAL	CHAR(1)	Is the object type marked as Internal? Internal types are not exposed by the M0 schema.
TAG_IS_DEPRECATED	CHAR(1)	Has this type been deprecated?
TAG_DEPRECATION_LEVEL	INTEGER	A value of *2* indicates that the type is still in the metamodel, but is no longer active. A value of *3* indicates the type is no longer in the metamodel. A value of *-1* indicates the type is for a future release.
TAG_RELEASE_DEPRECATED	VARCHAR	The release in which the type was deprecated. If the type is deprecated, but this value is NULL, then the release was 7.0.
TAG_RELEASE_ADDED	VARCHAR	The release in which the type was added.
TAG_IS_LOCALLY_DEFINED	CHAR(1)	Is this metadata defined in the model? For example, is this a UDP?
DATA_TYPE	INTEGER	The data type of the property.
TAG_IS_REFERENCE	CHAR(1)	Is this a property that holds a reference to another object?
TAG_IS_SYSTEM_MAINTAINED	CHAR(1)	Is this property maintained by CA ERwin DM instead of by the user?
REFERENCED_TYPE_REF	INTEGER	If this is a reference property, the type of object referenced.
TAG_IS_ORDERED_VECTOR	CHAR(1)	If this is a vector property, is the order of elements significant?
TAG_IS_ATOMIC_VECTOR	CHAR(1)	If this is a vector property, must the entire set of values always be set as a unit?

You can use a query similar to the following to find all the user-defined properties (UDPs) in a model.

Example: Query for returning all the UDPs in a model



ASSOCIATION_TYPE Table

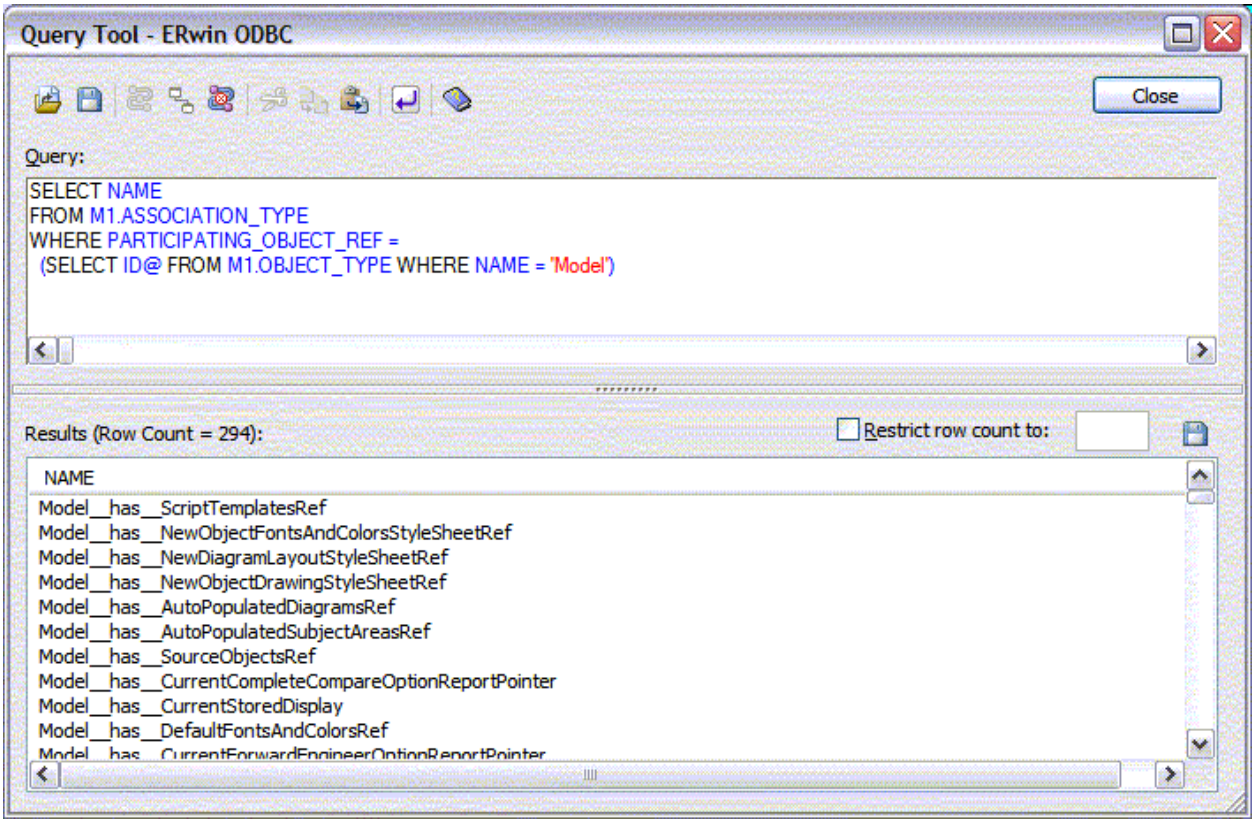
The *ASSOCIATION_TYPE* table exposes the associations between rows in the *OBJECT_TYPE* table and rows in the *PROPERTY_TYPE* table. The following table describes the base columns in the *ASSOCIATION_TYPE* table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. Though this is a short ID, it never changes.
NAME	VARCHAR	The name of the type.
LONG_ID	CHAR(67)	The long ID of the type. This is provided for consistency with M0.
DEFINITION	VARCHAR	The definition of the type.
TAG_IS_DEPRECATED	CHAR(1)	Has this type been deprecated?

Column Name	Data Type	Description
TAG_DEPRECATION_LEVEL	INTEGER	A value of *2* indicates that the type is still in the metamodel, but is no longer active. A value of *3* indicates the type is no longer in the metamodel. A value of *-1* indicates the type is for a future release.
TAG_RELEASE_DEPRECATED	VARCHAR	The release in which the type was deprecated. If the type is deprecated, but this value is NULL, then the release was 7.0.
TAG_RELEASE_ADDED	VARCHAR	The release in which the type was added.
PARTICIPATING_OBJECT_REF	INTEGER	The ID of the object type instance.
PARTICIPATING_PROPERTY_REF	INTEGER	The ID of the property type instance.
TAG_IS_PREFETCH	CHAR(1)	Is the value of the property sometimes calculated when read instead of stored in the model?

You can use a query similar to the following to find all the properties that may be applied to a model object.

Example: Query for returning all the properties that may be applied to a model object



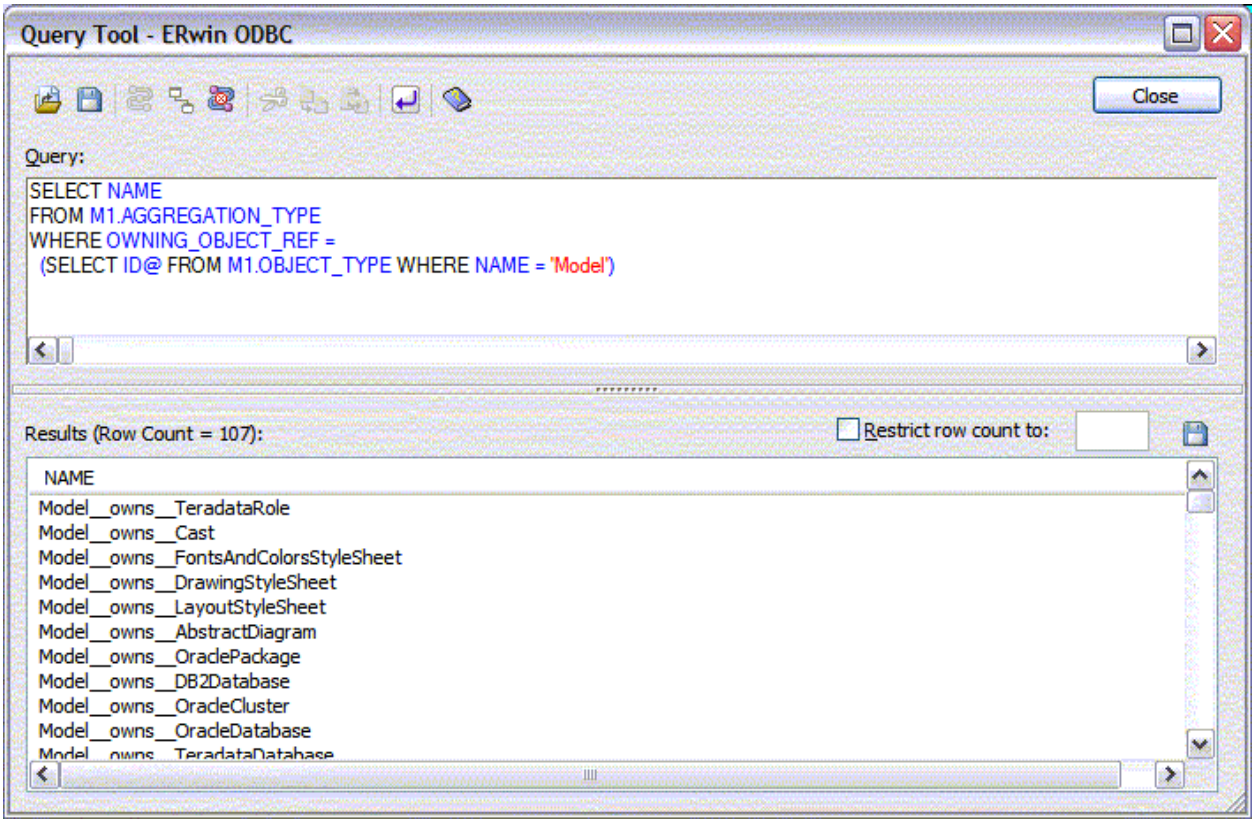
AGGREGATION_TYPE Table

The *AGGREGATION_TYPE* table exposes the ownership relationships between rows in the *OBJECT_TYPE* table. The following table describes the base columns in the *AGGREGATION_TYPE* table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. Though this is a short ID, it never changes.
NAME	VARCHAR	The name of the type.
LONG_ID	CHAR(67)	The long ID of the type. This is provided for consistency with M0.
DEFINITION	VARCHAR	The definition of the type.
TAG_IS_DEPRECATED	CHAR(1)	Has this type been deprecated?
TAG_DEPRECATION_LEVEL	INTEGER	A value of *2* indicates that the type is still in the metamodel, but is no longer active. A value of *3* indicates the type is no longer in the metamodel. A value of *-1* indicates the type is for a future release.
TAG_RELEASE_DEPRECATED	VARCHAR	The release in which the type was deprecated. If the type is deprecated, but this value is NULL, then the release was 7.0.
TAG_RELEASE_ADDED	VARCHAR	The release in which the type was added.
OWNING_OBJECT_REF	INTEGER	The ID of the owning object type.
OWNED_OBJECT_REF	INTEGER	The ID of the owned object type.

You can use a query similar to the following to show what objects are owned directly by the model.

Example: Query for returning a list of objects that are directly owned by a model



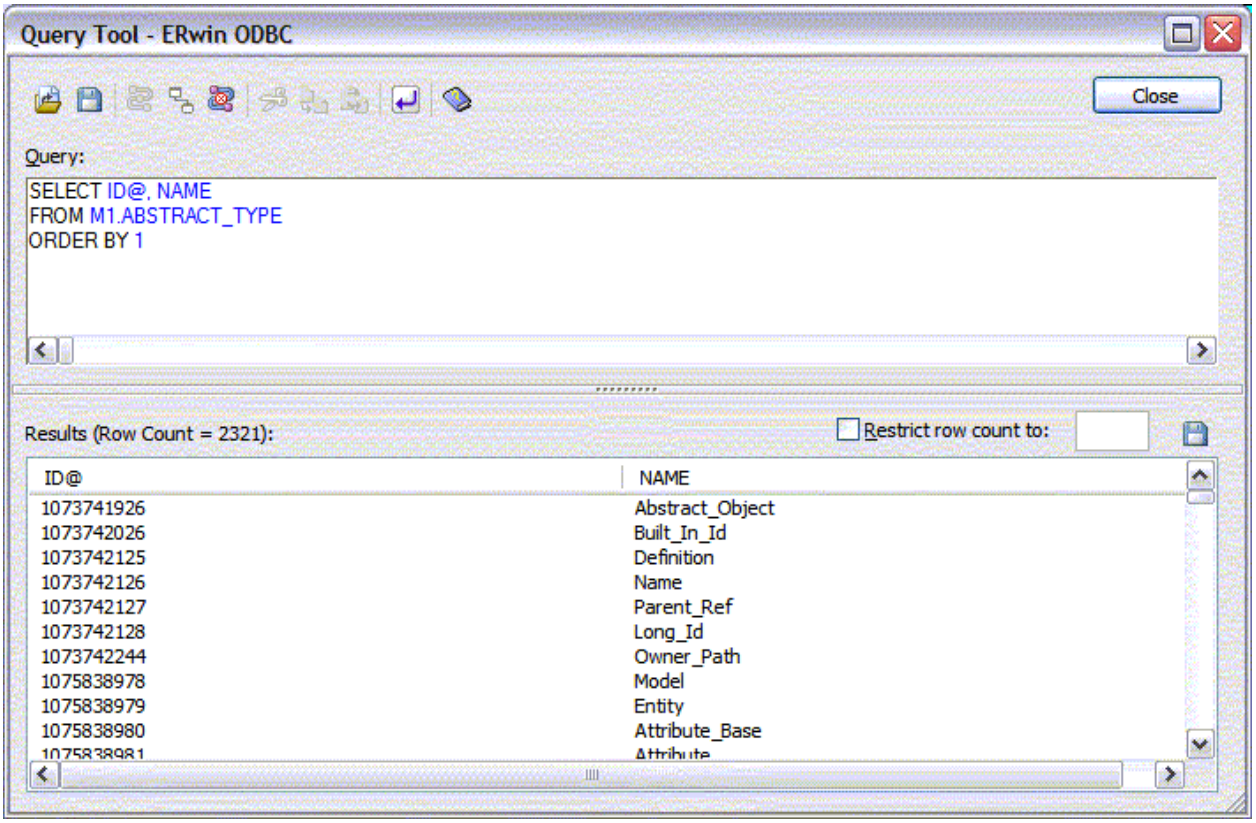
ABSTRACT_TYPE Table

The *ABSTRACT_TYPE* table contains rows for each entry in the *OBJECT_TYPE* and the *PROPERTY_TYPE* table. The following table describes the columns in the *ABSTRACT_TYPE* table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. Though this is a short ID, it never changes.
TYPE@	INTEGER	The class ID of the object.
NAME	VARCHAR	The name of the type.

You can use a query similar to the following to show all the class names that are associated with the types used to define the M0 schema.

Example: Query for returning a list of all the class names associated with the types used to define the M0 schema



CATEGORY_TYPE and CATEGORY_MEMBERSHIP Tables

The *CATEGORY_TYPE* table exposes the category definitions found in CA ERwin DM's metamodel. The *CATEGORY_MEMBER* table defines the members of the categories. Categories are collections of object and/or property types that are similar in some way significant to CA ERwin DM. The following table describes the columns in the *CATEGORY_TYPE* table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. Though this is a short ID, it never changes.
NAME	VARCHAR	The name of the type.
LONG_ID	CHAR(67)	The long ID of the type. This is provided for consistency with M0.

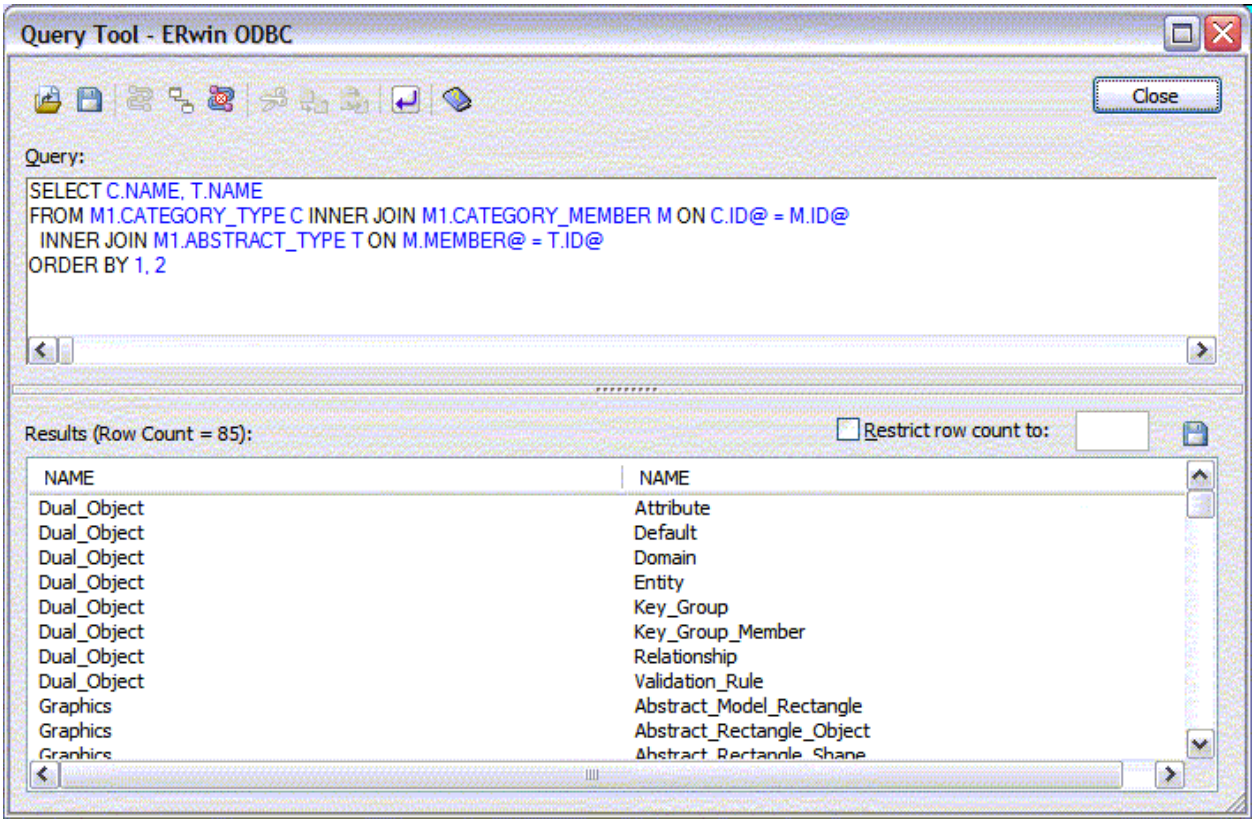
Column Name	Data Type	Description
DEFINITION	VARCHAR	The definition of the type.
TAG_IS_DEPRECATED	CHAR(1)	Has this type been deprecated?
TAG_DEPRECATION_LEVEL	INTEGER	A value of *2* indicates that the type is still in the metamodel, but is no longer active. A value of *3* indicates the type is no longer in the metamodel. A value of *-1* indicates the type is for a future release.
TAG_RELEASE_DEPRECATED	VARCHAR	The release in which the type was deprecated. If the type is deprecated, but this value is NULL, then the release was 7.0.
TAG_RELEASE_ADDED	VARCHAR	The release in which the type was added.

The following table describes the columns in the *CATEGORY_MEMBER* table:

Column Name	Data Type	Description
ID@	INTEGER	The category's ID.
MEMBER@	INTEGER	The ID of the member object.

You can use a query similar to the following to show the members of the various categories as they are used in the Action Log Metamodel Filter dialog.

Example: Query for returning the members of the various categories



EM0 Schema

The EM0 schema exposed by CA ERwin DM is an extension to the M0 schema exposed by the modeling engine. The M0 schema exposes the data in a CA ERwin DM model, including the dual objects-those objects that may exist simultaneously in both the logical and physical models.

Some of these objects may have their presence in one model or the other blocked. For example, an *Entity* might be marked as *Is_Logical_Only*. Alternatively, for historical reasons, CA ERwin DM maintains hidden copies of an *Attribute* when attribute unification occurs. For example, the hidden copy of an *Attribute* unified on the logical side would have the *Hide_In_Logical* set.

This schema exposes views of these types of objects that filter out the instances that are hidden.

MV_FOREIGN_KEY_ATTRIBUTE@ can be used to help speed processing of queries. It contains rows for all foreign key attributes, joined to the following:

- Entity owning the foreign key attribute
- Parent attribute
- Entity owning the parent attribute
- Contributing relationship

EM0 Schema Tables

The following table lists the tables exposed by EM0, as well as the underlying M0 table to which it corresponds.

The *MV_VISIBLE_ATTRIBUTE@* table will be identical to the *ATTRIBUTE* table, except that unified copies of *Attribute* objects are filtered out.

Note: There is no filtering on *Is_Logical_Only* or *Is_Physical_Only* in this table.

For the other tables exposed by this schema, the EM0 table will be identical to the M0 table except that objects having the *Is_Physical_Only* or *Hide_In_Logical* properties set will not occur in the logical versions of the table, and objects having the *Is_Logical_Only* or *Hide_In_Physical* properties set will not occur in the physical versions of the table.

The following table describes the tables in the EM0 schema:

EM0 Table	Underlying M0 Table
MV_LOGICAL_ATTRIBUTE@	ATTRIBUTE
MV_PHYSICAL_ATTRIBUTE@	ATTRIBUTE
MV_LOGICAL_DEFAULT@	DEFAULT
MV_PHYSICAL_DEFAULT@	DEFAULT
MV_LOGICAL_DOMAIN@	DOMAIN
MV_PHYSICAL_DOMAIN@	DOMAIN
MV_LOGICAL_ENTITY@	ENTITY
MV_PHYSICAL_ENTITY@	ENTITY
MV_LOGICAL_KEY_GROUP@	KEY_GROUP
MV_PHYSICAL_KEY_GROUP@	KEY_GROUP
MV_LOGICAL_RELATIONSHIP@	RELATIONSHIP
MV_PHYSICAL_RELATIONSHIP@	RELATIONSHIP
MV_LOGICAL_VALIDATION_RULE@	VALIDATION_RULE
MV_PHYSICAL_VALIDATION_RULE@	VALIDATION_RULE
MV_VISIBLE_ATTRIBUTE@	ATTRIBUTE

EM1 Schema

The EM1 schema exposed by CA ERwin DM is an extension to the M1 schema exposed by the modeling engine. The M1 schema exposes the entire metamodel, including the raw information about what objects and properties apply to what databases. However, the raw format of the database information can be difficult to interpret. The EM1 schema exposes views of that same information in a more readable format.

EM1 Schema Tables

This section describes the tables in the EM1 schema.

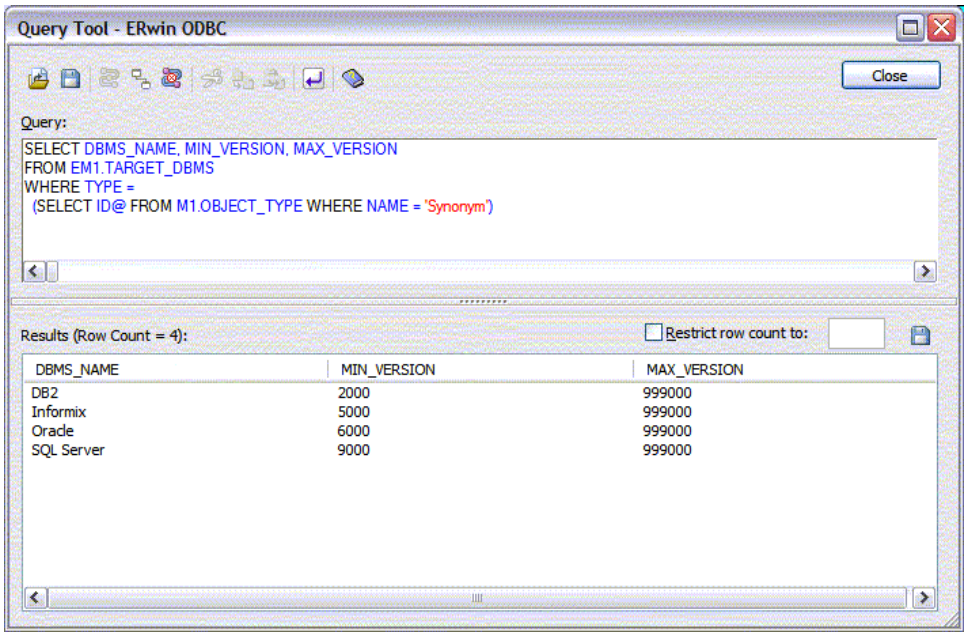
TARGET_DBMS Table

This table categorizes the rows of the *OBJECT_TYPE*, *PROPERTY_TYPE*, *ASSOCIATION_TYPE*, and *AGGREGATION_TYPE* tables by DBMS. The following table describes the columns in the *TARGET_DBMS* table:

Column Name	Data Type	Description
TYPE	INTEGER	The ID of an object type, property type, association type, or aggregation type.
DBMS	VARCHAR	A constant indicating the DBMS in which the object is found. This value will correspond to the DBMS tag's ID defined in the *ERwin Metamodel Reference* document. A value of zero indicates any DBMS.
DBMS_NAME	VARCHAR	The name of the DBMS.
MIN_VERSION	INTEGER	The earliest version for which the type is defined. The format is the version number multiplied by 1000. For example, Version 7.2 would be 7200. A value of zero indicates all versions less than *MAX_VERSION*.
MAX_VERSION	INTEGER	The latest version for which the type is defined. The format is the version number multiplied by 1000. For example, Version 7.2 would be 7200. A value of 999000 indicates all versions greater than *MIN_VERSION*.

You can use a query similar to the following to see which CA ERwin DM DBMS targets define synonym objects.

Example: Query for returning synonym object definition by CA ERwin DM target database



DBMS-Specific Tables

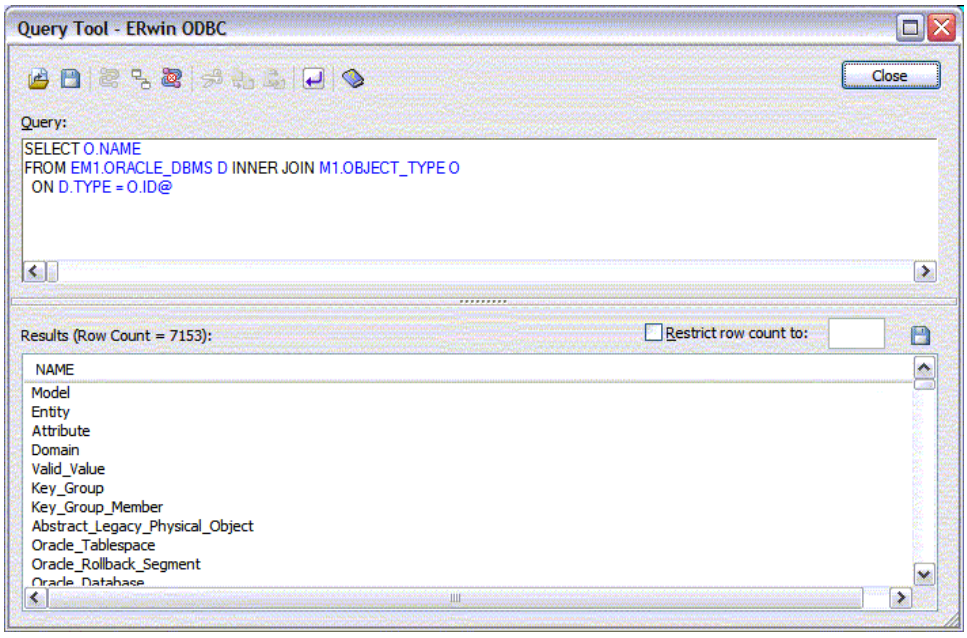
The information from *TARGET_DBMS* is broken down into DBMS-specific tables to make some queries easier. Each of these tables is named in the form *XX_DBMS* where *XX* is the name of the DBMS.

The following table describes the columns in the DBMS-specific tables:

Column Name	Data Type	Description
TYPE	INTEGER	The ID of an object type, property type, association type, or aggregation type.
MIN_VERSION	INTEGER	The earliest version for which the type is defined. The format is the version number multiplied by 1000. For example, Version 7.2 would be 7200. A value of zero indicates all versions less than *MAX_VERSION*.
MAX_VERSION	INTEGER	The latest version for which the type is defined. The format is the version number multiplied by 1000. For example, Version 7.2 would be 7200. A value of 999000 indicates all versions greater than *MIN_VERSION*.

You can use a query similar to the following to determine the objects that are defined for Oracle.

Example: Query for returning a list of the objects defined for Oracle



AL Schema

This schema exposes the contents of the Action Log in CA ERwin DM. Querying these tables allows you to retrieve the same type of information that is found in the Activity Summary file that can be generated when a CA ERwin DM model is closed.

AL Schema Tables

This section describes the tables in the AL schema.

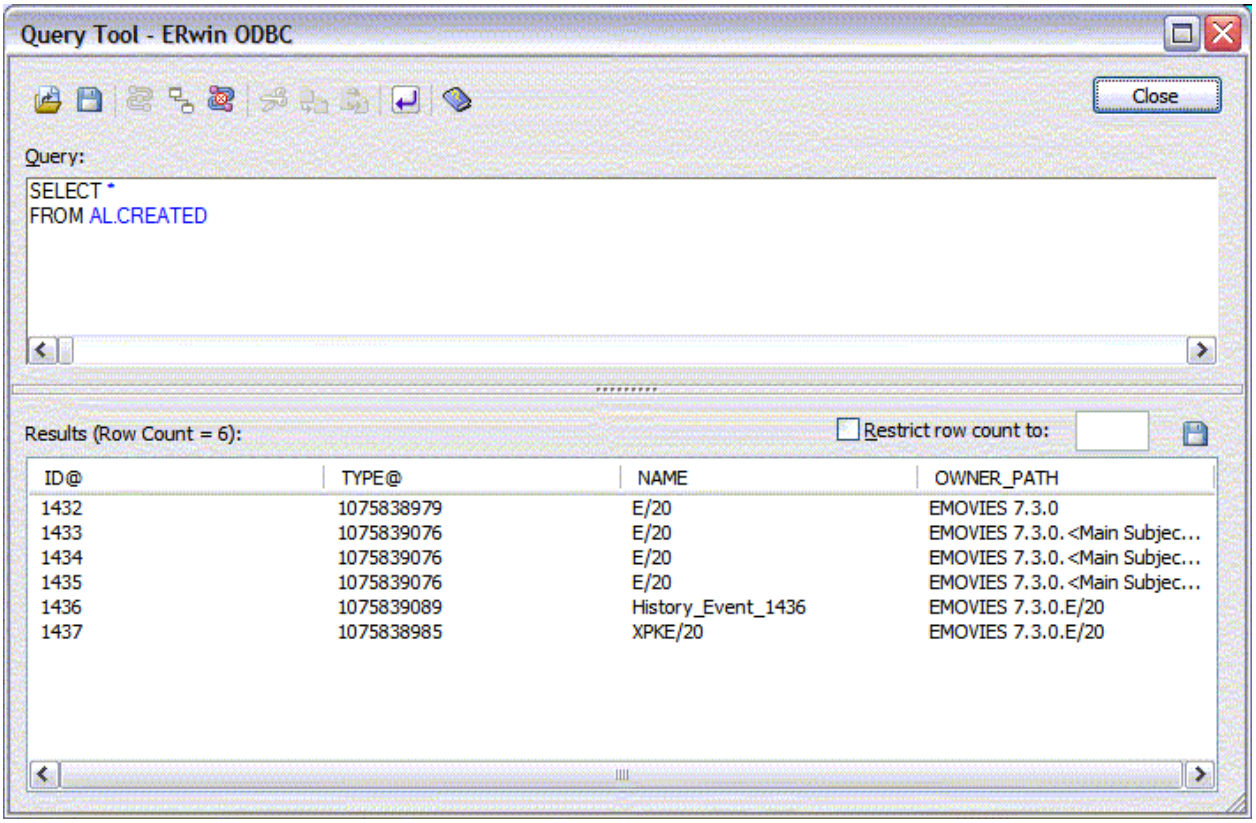
CREATED Table

This table exposes all the objects that were created during the session. The following table describes the columns in the *CREATED* table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. This is the short ID of an object, which is unique in the model, but may change from session to session.
TYPE@	INTEGER	The class ID of the object.
NAME	VARCHAR	The name of the object. For a dual object, this will be the logical name.
OWNER_PATH	VARCHAR	A period-separated list of the ownership chain for the object.

You can use a query similar to the following to see all the objects created in the session.

Example: Query for returning a list of all the objects created in a modeling session



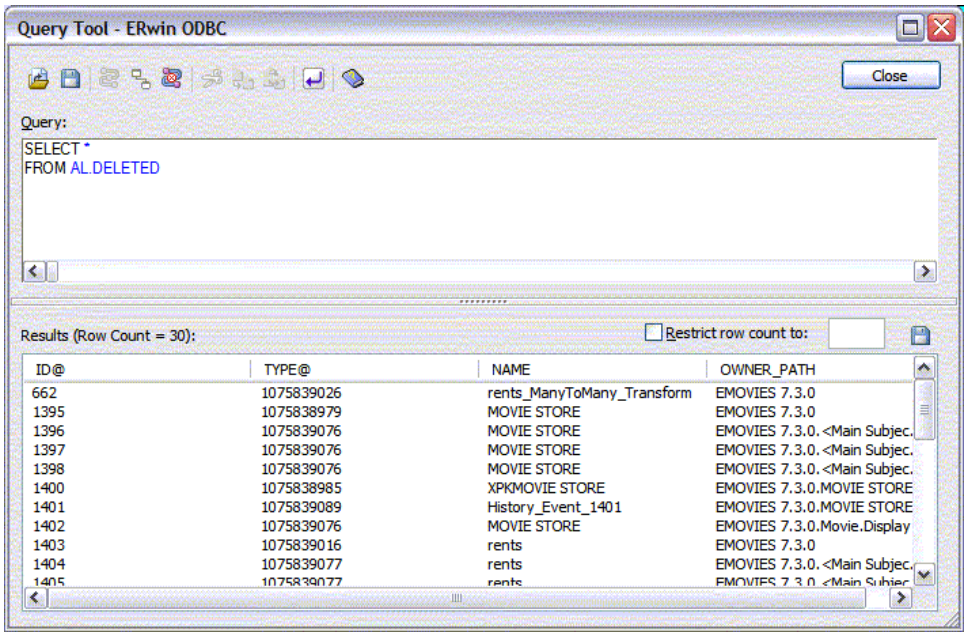
DELETED Table

This table exposes all objects that were deleted during the session. The following table describes the columns in the *DELETED* table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. This is the short ID of an object, which is unique in the model, but may change from session to session.
TYPE@	INTEGER	The class ID of the object.
NAME	VARCHAR	The name of the object. For a dual object, this will be the logical name.
OWNER_PATH	VARCHAR	A period-separated list of the ownership chain for the object.

You can use a query similar to the following to see all the objects deleted in the session.

Example: Query for returning a list of all the objects deleted in a modeling session



MODIFIED and PROPERTIES Tables

The *MODIFIED* table exposes all objects that were modified during the session. The *PROPERTIES* table lists the old and new values for the changed properties on those objects. The following table describes the columns in the *MODIFIED* table:

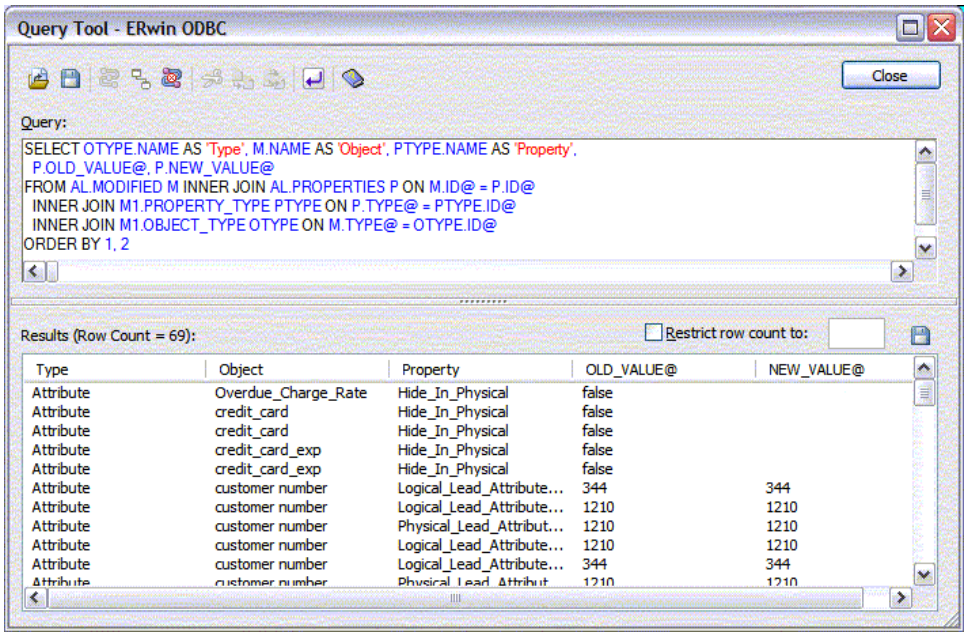
Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. This is the short ID of an object, which is unique in the model, but may change from session to session.
TYPE@	INTEGER	The class ID of the object.
NAME	VARCHAR	The name of the object. For a dual object, this will be the logical name.
OWNER_PATH	VARCHAR	A period-separated list of the ownership chain for the object.

The following table describes the columns in the *PROPERTIES* table:

Column Name	Data Type	Description
ID@	INTEGER	The ID of the object in the model. This is the short ID of an object, which is unique in the model, but may change from session to session.
TYPE@	INTEGER	The class ID of the property modified.
OLD_VALUE@	VARCHAR	A string representation of the old value of the property.
NEW_VALUE@	VARCHAR	A string representation of the new value of the property.

You can use a query similar to the following to see all changed properties during a session.

Example: Query for returning a list of all the properties that changed during a modeling session



DGM Schema

The DGM schema exposes information about the diagrams in CA ERwin DM.

DGM Schema Tables

This section describes the tables in the DGM schema.

CURRENT_DIAGRAM Table

The *CURRENT_DIAGRAM* table includes information about the current diagram. The following table describes the columns in the *CURRENT_DIAGRAM* table:

Column Name	Data Type	Description
Entire_MF	VARCHAR	Default path of the Windows metafile image of the current diagram.

Note: The default folder for Windows XP is: C:\Documents and Settings\<user name>\Local Settings\Temp. The default folder for Windows 7 is: C:\Users\<user name>\AppData\Local\Temp.

Example: Query to retrieve information about the current diagram

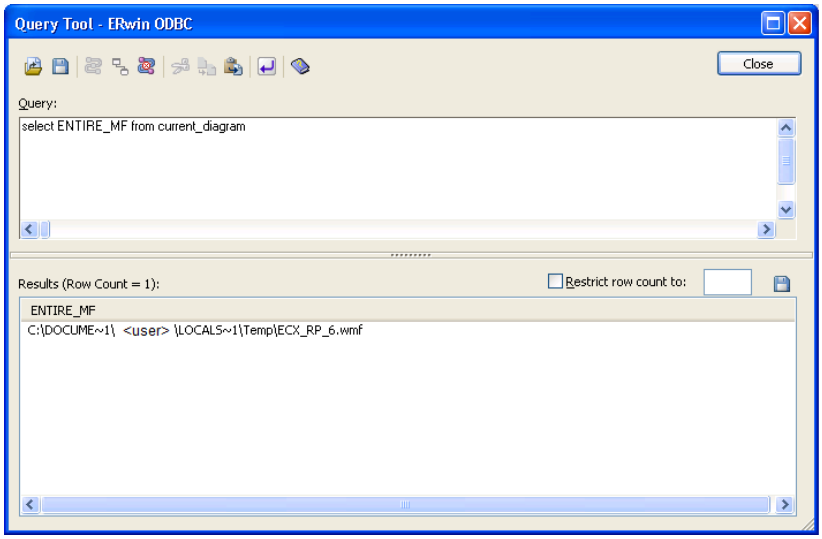


DIAGRAM Table

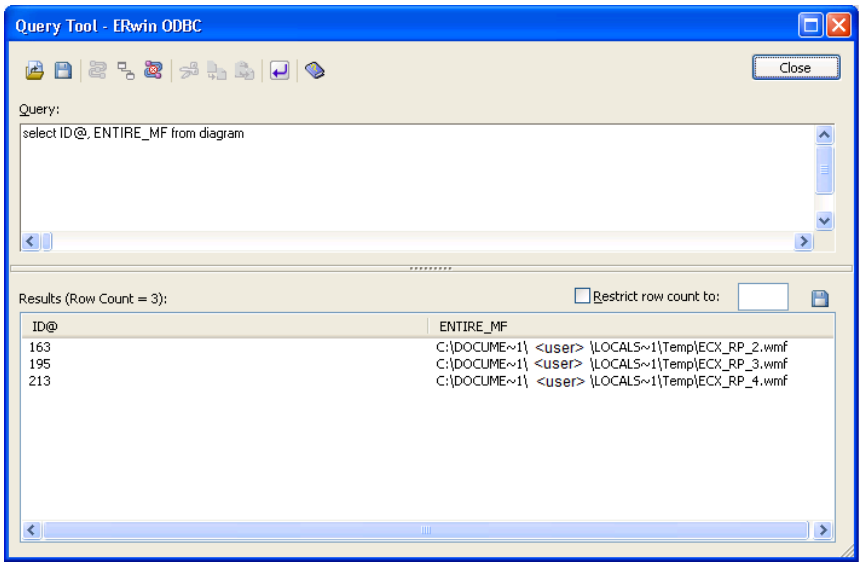
The *DIAGRAM* table includes information about all the diagrams in a model. The following table describes the columns in the *DIAGRAM* table:

Column Name	Data Type	Description
ID@	INTEGER	ID of the diagram.

Column Name	Data Type	Description
Entire_MF	VARCHAR	Default path of the Windows metafile images of the diagrams in the model.

Note: The default folder for Windows XP is: C:\Documents and Settings\<user name>\Local Settings\Temp. The default folder for Windows 7 is: C:\Users\<user name>\AppData\Local\Temp.

Example: Query to retrieve information about all diagrams



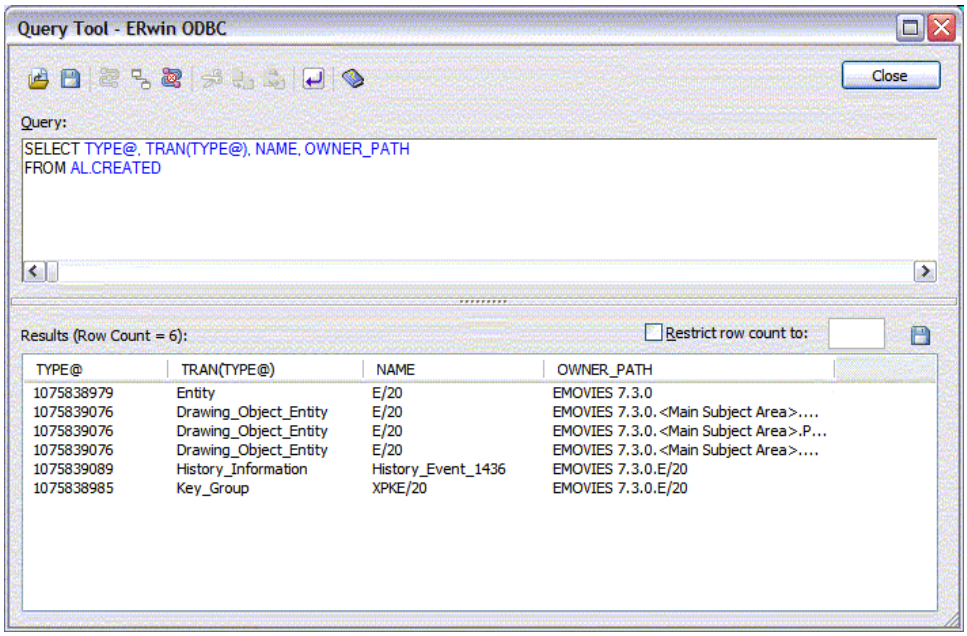
Functions

This section describes the functions used to retrieve property values.

TRAN()

The *TRAN()* function can be applied to the *TYPE@* column to retrieve an English version of the type identifier, such as the class name.

Example: TRAN() function applied to the TYPE@ column



Appendix A: Useful Queries

The following queries may be useful as written, or they can be used as the starting point for developing a query of your own.

This section contains the following topics:

Logical and Physical Names for all Entities

Example: Query for returning the logical and physical names for all entities

```
SELECT TRAN(NAME) AS 'LOGICAL', TRAN(PHYSICAL_NAME) AS 'PHYSICAL'
FROM M0.ENTITY
ORDER BY 1
```

Names of Logical Entities

Example: Query for returning the names for logical entities

```
SELECT NAME
FROM EM0.MV_LOGICAL_ENTITY@
ORDER BY 1
```

Qualified Names for Physical Entities (Tables)

Example: Query for returning a list of the qualified names for physical entities (tables) for target servers that use the *DB Owner* property, such as SQL Server 2000

```
SELECT SCHEMA_NAME AS 'SCHEMA', TRAN(PHYSICAL_NAME) AS 'TABLE'
FROM EM0.MV_PHYSICAL_ENTITY@
ORDER BY 1, 2
```

Example: Query for returning a list of the qualified names for physical entities (tables) for target servers that use separate schema objects, such as SQL Server 2005

```
SELECT S.NAME AS 'SCHEMA', TRAN(E.PHYSICAL_NAME) AS 'TABLE'
FROM EM0.MV_PHYSICAL_ENTITY@ E LEFT JOIN SCHEMA S
ON E.SCHEMA_REF = S.ID@
ORDER BY 1, 2
```

Definitions for All Entities

Example: Query for returning the definitions for all entities

```
SELECT NAME, DEFINITION
FROM M0.ENTITY
ORDER BY 1
```

Logical and Physical Names of Attributes

Example: Query for returning the logical and physical names for all attributes, unified or not

```
SELECT TRAN(E.NAME) || '.' || TRAN(A.NAME) AS 'LOGICAL',
    TRAN(E.PHYSICAL_NAME) || '.' || TRAN(A.PHYSICAL_NAME) AS 'PHYSICAL'
FROM M0.ENTITY E INNER JOIN M0.ATTRIBUTE A
    ON E.ID@ = A.OWNER@
ORDER BY 1, 2
```

Example: Query for returning the logical and physical names for all attributes, filtering out unified attributes

```
SELECT TRAN(E.NAME) || '.' || TRAN(A.NAME) AS 'LOGICAL',
    TRAN(E.PHYSICAL_NAME) || '.' || TRAN(A.PHYSICAL_NAME) AS 'PHYSICAL'
FROM M0.ENTITY E INNER JOIN EM0.MV_VISIBLE_ATTRIBUTE@ A
    ON E.ID@ = A.OWNER@
ORDER BY 1, 2
```

Attributes with Data Types

Example: Query for returning all attributes and their data types

```
SELECT E.NAME AS 'ENTITY', A.NAME AS 'ATTRIBUTE',
    TRAN(A.LOGICAL_DATA_TYPE) AS 'DATA TYPE'
FROM M0.ENTITY E INNER JOIN EM0.MV_VISIBLE_ATTRIBUTE A
    ON E.ID@ = A.OWNER@
ORDER BY 1, 2
```

Logical-Only Attributes

Example: Query for returning the names of logical-only attributes

```
SELECT E.NAME AS 'ENTITY', A.NAME AS 'ATTRIBUTE'
FROM M0.ENTITY E INNER JOIN EM0.MV_LOGICAL_ATTRIBUTE A
    ON E.ID@ = A.OWNER@
WHERE A.IS_LOGICAL_ONLY = 'T'
```

Logical-Only and Physical-Only Status of Attributes

Example: Query for returning the logical-only and physical-only status of all attributes

```
SELECT TRAN(E.NAME) || '.' || TRAN(A.NAME) AS 'NAME',
        ISNULL( A.IS_LOGICAL_ONLY, 'F') AS 'LOGICAL ONLY',
        ISNULL( A.IS_PHYSICAL_ONLY, 'F') AS 'PHYSICAL ONLY'
FROM M0.ENTITY E INNER JOIN M0.ATTRIBUTE A
ON E.ID@ = A.OWNER@
ORDER BY 1
```

Domain Usage

Example: Query for returning information on domain usage

```
SELECT D.NAME AS 'DOMAIN', A.NAME AS 'ATTRIBUTE'
FROM M0.DOMAIN D INNER JOIN M0.ATTRIBUTE A
    ON D.ID@ = A.PARENT_DOMAIN_REF
ORDER BY 1, 2
```

Domains Not Used by Attributes

Example: Query for returning a list of domains not used by attributes

Note: This report will contain domains that are used indirectly through domain inheritance.

```
SELECT NAME
FROM DOMAIN
WHERE ID@ NOT IN (SELECT PARENT_DOMAIN_REF FROM ATTRIBUTE)
```

Example: Query for returning a list of domains not used by attributes, excluding domains that are used by other domains

```
SELECT NAME
FROM DOMAIN
WHERE ID@ NOT IN
(
        SELECT PARENT_DOMAIN_REF FROM ATTRIBUTE
        UNION
        SELECT PARENT_DOMAIN_REF FROM DOMAIN
)
```

Key Group Types

Example: Query for returning information on key group types

```
SELECT E.NAME AS 'ENTITY', KG.NAME AS 'KEY GROUP',
        DECODE( LEFT(KEY_GROUP_TYPE, 2),
                'PK', 'PRIMARY KEY',
                'IF', 'FOREIGN KEY',
                'IE', 'INVERSION ENTRY',
                'AK', 'ALTERNATE KEY' )
        AS 'TYPE'
FROM M0.ENTITY E INNER JOIN M0.KEY_GROUP KG
        ON E.ID@ = KG.OWNER@
ORDER BY 1,2
```

Subject Area Membership

Example: Query for returning information on subject area membership

```
SELECT SA.NAME AS 'SUBJECT AREA', E.NAME AS 'MEMBER'
FROM M0.USER_ATTACHED_OBJECTS_REF RE
    INNER JOIN M0.SUBJECT_AREA SA ON RE.ID@ = SA.ID@
    INNER JOIN M0.ENTITY E ON RE.VALUE@ = E.ID@
UNION
SELECT SA.NAME, V.NAME
FROM M0.USER_ATTACHED_OBJECTS_REF RE
    INNER JOIN M0.SUBJECT_AREA SA ON RE.ID@ = SA.ID@
    INNER JOIN M0.VIEW V ON RE.VALUE@ = V.ID@
UNION
SELECT SA.NAME, CV.NAME
FROM M0.USER_ATTACHED_OBJECTS_REF RE
    INNER JOIN M0.SUBJECT_AREA SA ON RE.ID@ = SA.ID@
    INNER JOIN M0.CACHED_VIEW CV ON RE.VALUE@ = CV.ID@
ORDER BY 1, 2
```

Model UDP Definitions

Example: Query for returning all UDP definitions in the model

```
SELECT NAME
FROM M1.PROPERTY_TYPE
WHERE TAG_IS_LOCALLY_DEFINED = 'T'
```

Objects Using a Default Name

Example: Query for returning all objects that have a default name

```
SELECT NAME
FROM M0.ABSTRACT_OBJECT
WHERE IS_DEFAULT(NAME) = 'T'
```

Non-Subtype Relationships

Example: Query for returning a list of non-subtype relationships

```
SELECT P.NAME AS 'PARENT', R.PARENT_TO_CHILD_VERB_PHRASE AS 'VERB',
      C.NAME AS CHILD
FROM M0.RELATIONSHIP R INNER JOIN M0.ENTITY P
      ON R.PARENT_ENTITY_REF = P.ID@
            INNER JOIN M0.ENTITY C
                  ON R.CHILD_ENTITY_REF = C.ID@
```

All Objects Created During a Modeling Session

Example: Query for returning all objects that were created in a session

```
SELECT TRAN(TYPE@) AS 'OBJECT', NAME, OWNER_PATH
FROM AL.CREATED
```

All Objects Deleted During a Modeling Session

Example: Query for returning all objects that were deleted in a session

```
SELECT TRAN(TYPE@) AS 'OBJECT', NAME, OWNER_PATH
FROM AL.DELETED
```

All Objects Modified During a Modeling Session

Example: Query for returning all objects that were modified in a session

```
SELECT TRAN(TYPE@) AS 'OBJECT', NAME, OWNER_PATH
FROM AL.MODIFIED
```

All Objects and Properties Modified During a Modeling Session

Example: Query for returning all objects and properties that were modified in a session

```
SELECT TRAN(O.TYPE@) AS 'OBJECT', O.NAME, O.OWNER_PATH,
        TRAN(P.TYPE@) AS 'PROPERTY',
        ISNULL(P.OLD_VALUE@, '<NULL>') AS 'OLD VALUE',
        ISNULL(P.NEW_VALUE@, '<NULL>' ) AS 'NEW VALUE'
FROM AL.MODIFIED O INNER JOIN AL.PROPERTIES P
        ON O.ID@ = P.ID@
ORDER BY O.TYPE@, O.ID@
```

Entities Not Having Primary Key Columns

Example: Query for returning all entities that do not have primary key columns

```
SELECT NAME
FROM M0.ENTITY
WHERE ID@ NOT IN
    (
        SELECT E.ID@
        FROM M0.ENTITY E INNER JOIN M0.KEY_GROUP KG
                ON E.ID@ = KG.OWNER@
                INNER JOIN M0.KEY_GROUP_MEMBER M
                        ON KG.ID@ = M.OWNER@
        WHERE KG.KEY_GROUP_TYPE = 'PK'
    )
```

Migrated Physical Attributes

Example: Query for returning all migrated physical attributes with a different data type than their parent

```
SELECT PE.NAME || '.' || PA.NAME AS 'PARENT NAME',
        PA.PHYSICAL_DATA_TYPE AS 'PARENT DATA TYPE',
        CE.NAME || '.' || CA.NAME AS 'CHILD NAME',
        CA.PHYSICAL_DATA_TYPE AS 'CHILD DATA TYPE',
        R.NAME AS 'MIGRATING RELATIONSHIP'
FROM EM0.MV_PHYSICAL_ATTRIBUTE@ PA INNER JOIN M0.ENTITY PE
        ON PA.OWNER@ = PE.ID@
        INNER JOIN EM0.MV_PHYSICAL_ATTRIBUTE@ CA
                ON CA.PARENT_ATTRIBUTE_REF = PA.ID@
                INNER JOIN M0.ENTITY CE
                        ON CA.OWNER@ = CE.ID@
                        INNER JOIN M0.RELATIONSHIP R
                                ON CA.PARENT_RELATIONSHIP_REF = R.ID@
WHERE PA.PHYSICAL_DATA_TYPE <> CA.PHYSICAL_DATA_TYPE
```

Oracle Model Object Types

Example: Query for returning all the object types that can appear in an Oracle model

```
SELECT TRAN(A.TYPE)
FROM EM1.MV_ORACLE_DBMS A INNER JOIN M1.OBJECT_TYPE B
    ON A.TYPE = B.ID@
WHERE B.TAG_IS_ABSTRACT IS NULL OR B.TAG_IS_ABSTRACT = 'F'
ORDER BY 1
```