Teiid - Salesforce Connector Guide

1

6.2.0

Preface

This document shows how to model the metadata of an hosted Salesforce application with Teiid Designer, and details the SQL semantics supported by the connector in Teiid Embedded. This document contains an overview of the process, including:

- Using the Salesforce Importer to create relational models representing your Salesforce application.
- Querying the Salesforce application.

Commercial development support, production support, and training for Teiid is available through JBoss. Teiid is a Professional Open Source project and a critical component of the JBoss Enterprise Data Services Platform.

Note

Please read *Federation Basics* [http://www.jboss.org/teiid/basics.html] and understand different terminologies used, resources needed and artifacts that need to be generated before developing a successful application. This example takes advantage of only a minimal set of features from Teiid Embedded for the sake of simplicity and time.

Importing Metadata

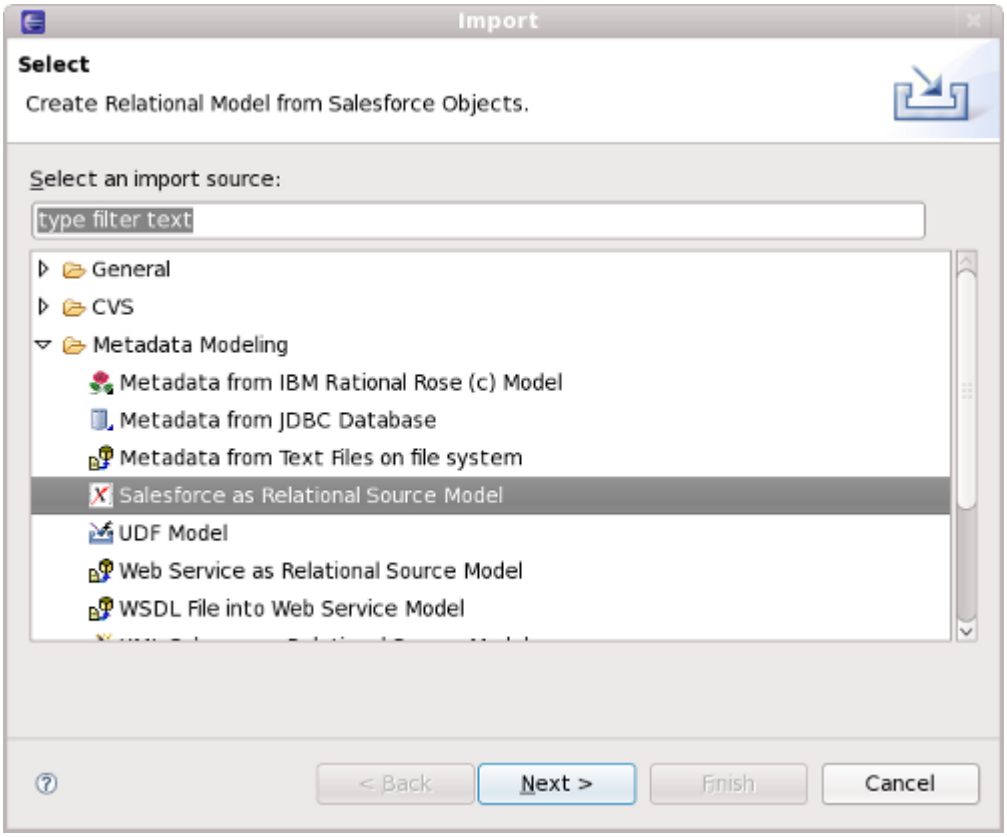
1.1. Overview

Salesforce metadata is structured relationally. Salesforce Objects and the fields on those objects are analogous to relational tables and columns. Because of this similarity, the modeling of Salesforce metadata and interaction with the data through standard SQL is easily accomplished within Teiid. However, the properties of the objects in a Salesforce application are not an exact match to the properties that are available on the standard Teiid Relational Table and Column. In order to bridge the gap between these metadata properties, integration with Salesforce requires extensions to the standard Teiid Relational Base Table and Column. The extensions required are created within the Model Project by the Salesforce Importer and appropriate values are set into these properties by the importer.

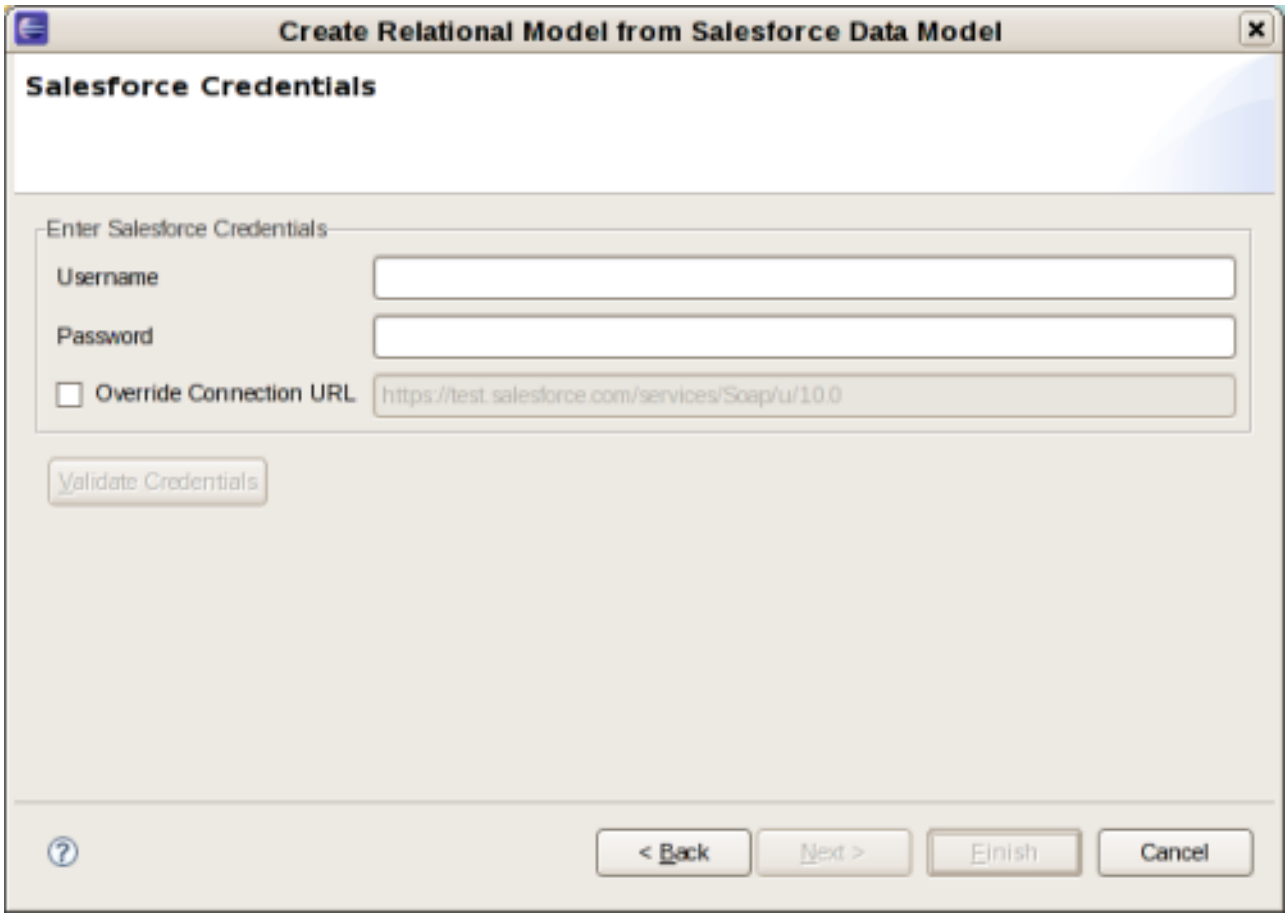
The Salesforce Importer will model the unique metadata that is exposed to it by the Salesforce API as determined by the credentials used in the importer. Users with different profiles within the same Salesforce instance may be authorized to see different subsets of the applications metadata and data. Keep this in mind when binding a connector to the model. You must ensure that the credentials used in the binding have access to all of the data that is exposed in the model, or exceptions will occur. The importer also understands customizations that have been made to applications and will model them as well.

1.2. Running the Importer

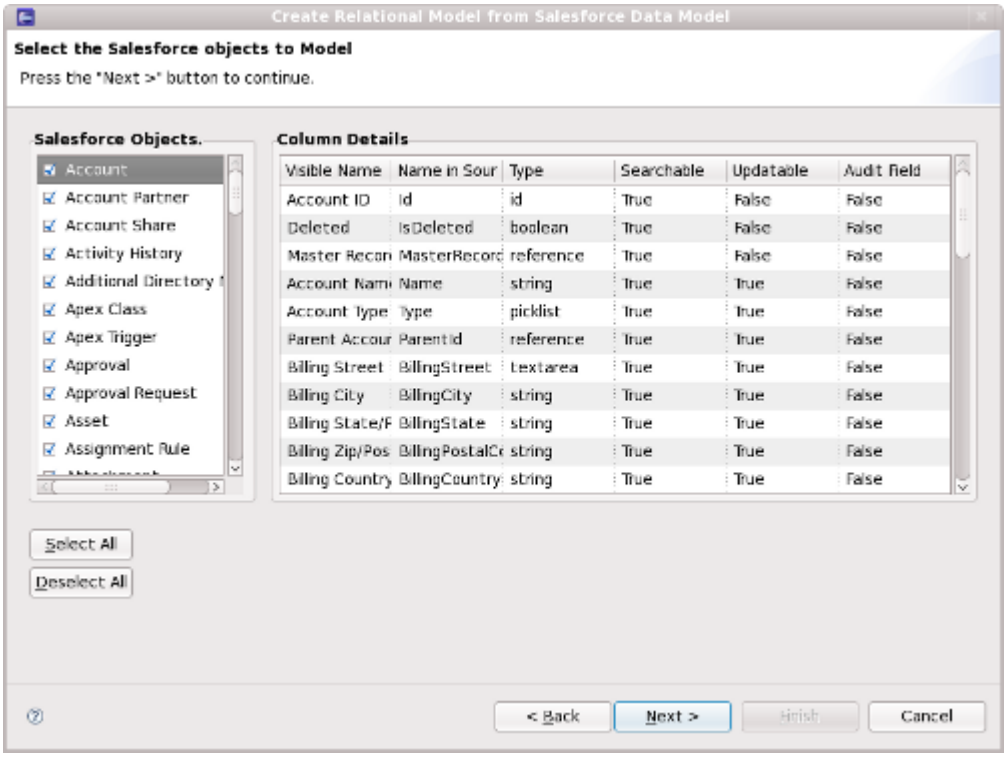
Select File->Import to display the Wizard Selection Dialog.



Select Salesforce as Relational Source Model from in the list displayed and click Next. The Salesforce Importer is displayed.

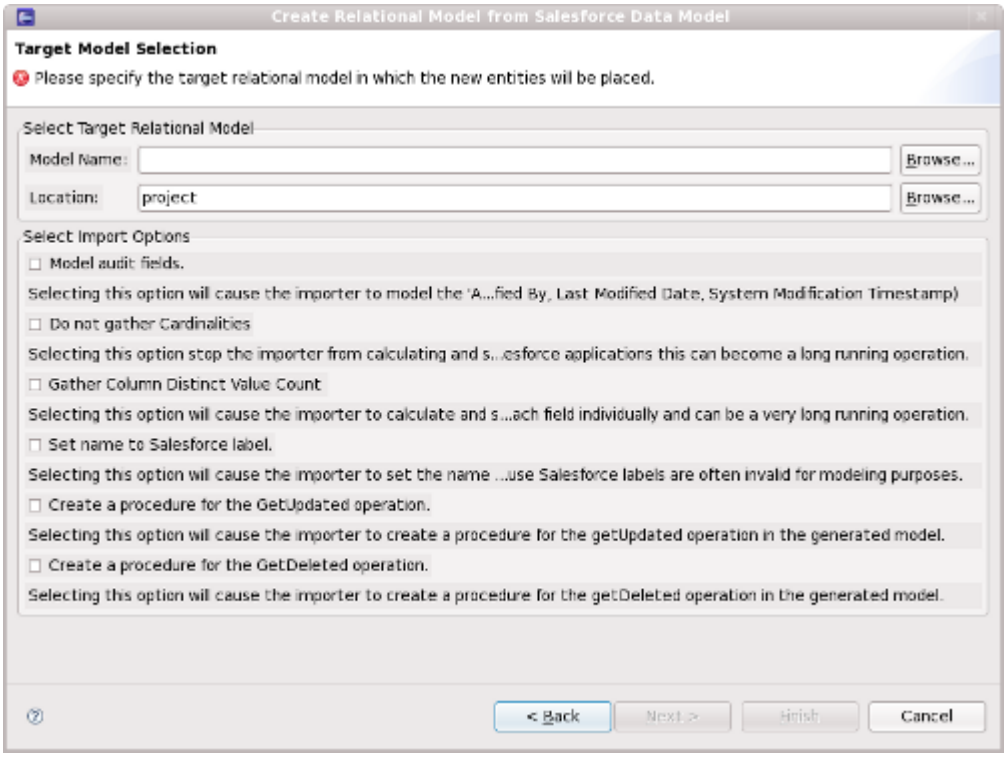


Enter a valid Username and Password into the text fields and click Validate Credentials. If you are connecting to a sandbox instance of a Salesforce application, enable the Override Connection URL textbox and enter the URL to the sandbox instance. The default value will probably be correct in most instances. After the Username and Password are supplied, the Validate Credentials button will be come active and you click this button to validate the connection parameters and enable the Next Button. After clicking Next the Salesforce Objects page is displayed.



In the left column the Salesforce objects in the system are listed and can be selected for modeling. Each object selected will be modeled as a Table. In the Column Details table metadata from the fields on the currently selected object are displayed, and these will be modeled as columns on the tables.

When you have selected all of the Salesforce objects that you wish to model, click Next. The Target Model Selection page will be displayed.



Select the model name, location, and options.

- **Model Name:** type the name of the new model to be created, or browse to select an preexisting model to update.
- **Location:** type or browse to the location of the model project to create the model in.
- **Model audit fields:** each Salesforce object has fields that track changes to the object, such as LastModifiedDate and CreateById. These fields are know as audit fields. Because each object in the application has these fields, and some of these fields have relations to other Salesforce objects, modeling these fields can cause a model to be congested with relations. If you do not plan to use a model to query audit data, you probably want to suppress modeling of these fields. Checking this box will override the default behavior and will model the audit fields.
- **Do not gather Cardinalities:** by default the importer will calculate the table cardinalites. Cardinality metadata can greatly improve query performance, but it may take the importer some time calculate this information. Checking this box will suppress the calculation of cardinalities.
- **Gather Column Distinct Value Count:** by default the importer does not calculate column distinct value counts. Checking this box will cause the importer to calculate these values. This can be an **extremely long** running operation.
- **Set name to Salesforce Label:** each Salesforce object has a label (user visible name) that can be modified, and names (the fixed internal name). By default the importer will use the name when modeling objects. Checking this box will cause the importer to use the label as the table name.

- **Create a procedure for the GetUpdated operation:** checking this box will cause the importer to create a procedure for the getUpdated operation in the generated model.
- **Create a procedure for the GetDeleted operation:** checking this box will cause the importer to create a procedure for the getDeleted operation in the generated model.

Click Finish.

After the Salesforce Importer has finished, you will find two new models created in your model project. One will have the name you supplied in the last page of the importer, and the other will be named SalesforceExtensions.xmi.

The SalesforceExtensions.xmi model contains the extensions to the standard relational model. These extensions enable the model to provided enhanced data to the Salesforce Connector at query time. For example, some objects in Salesforce do not support the query operation. In order to enforce this limitation the extension model defined a Supports Query property, and the Salesforce Importer enters the correct value into this property for each Table. The Salesforce Connector validates that each table included in a query operation has the value true for this property before executing a query and will throw and exception in the event that the table is not querable.

The other model created by the importer contains the model of the data in the Salesforce instance that you selected in the importer. This is the model you will execute your SQL statements against.

Using the Connector

The Salesforce Connector supports the SELECT, DELETE, INSERT and UPDATE operations.

2.1. SQL Processing

Salesforce does not provide the same set of functionality as a relational database. For example, Salesforce does not support arbitrary joins between tables. However, working in combination with the Teiid Query Planner, the Salesforce connector supports nearly all of the SQL syntax supported by the Teiid.

The Salesforce Connector executes SQL commands by “pushing down” the command to Salesforce whenever possible, based on the supported capabilities. Teiid will automatically provide additional database functionality when the Salesforce Connector does not explicitly provide support for a given SQL construct. In these cases, the SQL construct cannot be “pushed down” to the data source, so it will be evaluated in Teiid, in order to ensure that the operation is performed.

In cases where certain SQL capabilities cannot be pushed down to Salesforce, Teiid will push down the capabilities that are supported, and fetch a set of data from Salesforce. Then, Teiid will evaluate the additional capabilities, creating a subset of the original data set. Finally, Teiid will pass the result to the client.

```
SELECT sum(Reports) FROM Supervisor where Division = 'customer support';
```

Neither Salesforce nor the Salesforce Connector support the sum() scalar function, but they do support CompareCriteriaEquals, so the query that is passed to Salesforce by the connector will be transformed to this query.

```
SELECT Reports FROM Supervisor where Division = 'customer support';
```

The sum() scalar function will be applied by the Teiid Query Engine to the result set returned by the connector.

In some cases multiple calls to the Salesforce application will be made to support the SQL passed to the connector.

```
DELETE From Case WHERE Status = 'Closed';
```

The API in Salesforce to delete objects only supports deleting by ID. In order to accomplish this the Salesforce connector will first execute a query to get the IDs of the correct objects, and then delete those objects. So the above DELETE command will result in the following two commands.

```
SELECT ID From Case WHERE Status = 'Closed';
DELETE From Case where ID IN (<result of query>);*
```

*The Salesforce API DELETE call is not expressed in SQL, but the above is an SQL equivalent expression.

It's useful to be aware of unsupported capabilities, in order to avoid fetching large data sets from Salesforce and making you queries as performant as possible. See the Supported Capabilities section in the appendix.

2.2. Selecting from Multi-Select Picklists

A multi-select picklist is a field type in Salesforce that can contain multiple values in a single field. Query criteria operators for fields of this type in SOQL are limited to EQ, NE, includes and excludes. The full Salesforce documentation for selecting from mullti-select picklists can be found at the following link. *Querying Mulit-select Picklists* [http://www.salesforce.com/us/developer/docs/api/index_Left.htm#StartTopic=Content%2Fsforce_api_calls_soql_querying_multiselect_picklists.htm|SkinName=webh

Teiid SQL does not support the includes or excludes operators, but the Salesforce connector provides user defined function definitions for these operators that provided equivalent functionality for fields of type multi-select. The definition for the functions is:

```
boolean includes(Column column, String param)
boolean excludes(Column column, String param)
```

For example, take a single multi-select picklist column called Status that contains all of these values.

- current
- working
- critical

For that column, all of the below are valid queries:

```
SELECT * FROM Issue WHERE true = includes (Status, 'current, working' );
SELECT * FROM Issue WHERE true = excludes (Status, 'current, working' );
SELECT * FROM Issue WHERE true = includes (Status, 'current;working, critical' );
```

EQ and NE criteria will pass to Salesforce as supplied. For example, these queries will not be modified by the connector.

```
SELECT * FROM Issue WHERE Status = 'current';
SELECT * FROM Issue WHERE Status = 'current;critical';
SELECT * FROM Issue WHERE Status != 'current;working';
```

2.3. Selecting All Objects

The Salesforce connector supports the calling the queryAll operation from the Salesforce API. The queryAll operation is equivalent to the query operation with the exception that it returns data about **all current and deleted** objects in the system.

The connector determines if it will call the query or queryAll operation via reference to the isDeleted property present on each Salesforce object, and modeled as a column on each table generated by the importer. By default this value is set to False when the model is generated and thus the connector calls query. Users are free to change the value in the model to True, changing the default behavior of the connector to be queryAll.

The behavior is different if isDeleted is used as a parameter in the query. If the isDeleted column is used as a parameter in the query, and the value is 'true' the connector will call queryAll.

```
select * from Contact where isDeleted = true;
```

If the isDeleted column is used as a parameter in the query, and the value is 'false' the connector perform the default behavior will call query.

```
select * from Contact where isDeleted = false;
```

2.4. Selecting Updated Objects

If the option is selected when importing metadata from Salesforce, a GetUpdated procedure is generated in the model with the following sturcture:

```
GetUpdated (ObjectName IN string,
  StartDate IN datetime,
  EndDate IN datetime,
  LatestDateCovered OUT datetime)
returns
 ID string
```

See the description of the *GetUpdated* [http://www.salesforce.com/us/developer/docs/api/Content/sforce_api_calls_getupdated.htm] operation in the Salesforce documentation for usage details.

2.5. Selecting Deleted Objects

If the option is selected when importing metadata from Salesforce, a GetDeleted procedure is generated in the model with the following sturcture:

```
GetDeleted (ObjectName IN string,
  StartDate IN datetime,
```

```
  EndDate IN datetime,
  EarliestDateAvailable OUT datetime,
  LatestDateCovered OUT datetime)
returns
 ID string,
 DeletedDate datetime
```

See the description of the *GetDeleted* [http://www.salesforce.com/us/developer/docs/api/Content/sforce_api_calls_getdeleted.htm] operation in the Salesforce documentation for usage details.

2.6. Relationship Queries

Salesforce does not support joins like a relational database, but it does have support for queries that include parent-to-child or child-to-parent relationships between objects. These are termed Relationship Queries. The SalesForce connector supports Relationship Queries through Outer Join syntax.

```
SELECT Account.name, Contact.Name from Contact LEFT OUTER JOIN Account
on Contact.Accountid = Account.id
```

This query shows the correct syntax to query a SalesForce model with to produce a relationship query from child to parent. It resolves to the following query to SalesForce.

```
SELECT Contact.Account.Name, Contact.Name FROM Contact
```

```
select Contact.Name, Account.Name from Account Left outer Join Contact
on Contact.Accountid = Account.id
```

This query shows the correct syntax to query a SalesForce model with to produce a relationship query from parent to child. It resolves to the following query to SalesForce.

```
SELECT Account.Name, (SELECT Contact.Name FROM
Account.Contacts) FROM Account
```

See the description of the *Relationship Queries* [http://www.salesforce.com/us/developer/docs/api/index_Left.htm#StartTopic=Content/sforce_api_calls_soql_relationships.htm] operation in the SalesForce documentation for limitations.

Appendix

3.1. Supported Capabilities

The following are the the connector capabilities supported by the Salesforce Connector. These SQL constructs will be pushed down to Salesforce.

- SELECT command
- INSERT Command
- UPDATE Command
- DELETE Command
- CompareCriteriaEquals
- InCriteria
- LikeCriteria - Supported for String fields only.
- RowLimit
- AggregatesCountStar
- NotCriteria
- OrCriteria
- CompareCriteriaOrdered
- OuterJoins with join criteria KEY