FME Desktop®

Database (Oracle) Pathway Training

FME 2013-SP2 Edition

SAFE SOFTWARE

Safe Software Inc. makes no warranty either expressed or implied, including, but not limited to, any implied warranties of merchantability or fitness for a particular purpose regarding these materials, and makes such materials available solely on an “as-is” basis.

In no event shall Safe Software Inc. be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of purchase or use of these materials. The sole and exclusive liability of Safe Software Inc., regardless of the form or action, shall not exceed the purchase price of the materials described herein.

This manual describes the functionality and use of the software at the time of publication. The software described herein, and the descriptions themselves, are subject to change without notice.

Copyright

Revisions

Every effort has been made to ensure the accuracy of this document. Safe Software Inc. regrets any errors and omissions that may occur and would appreciate being informed of any errors found. Safe Software Inc. will correct any such errors and omissions in a subsequent version, as feasible. Please contact us at:

Safe Software Inc.
Suite 2017, 7445 – 132nd Street
Surrey, BC
Canada
V3W1J8

www.safe.com

Safe Software Inc. assumes no responsibility for any errors in this document or their consequences, and reserves the right to make improvements and changes to this document without notice.

Trademarks

Documentation Information

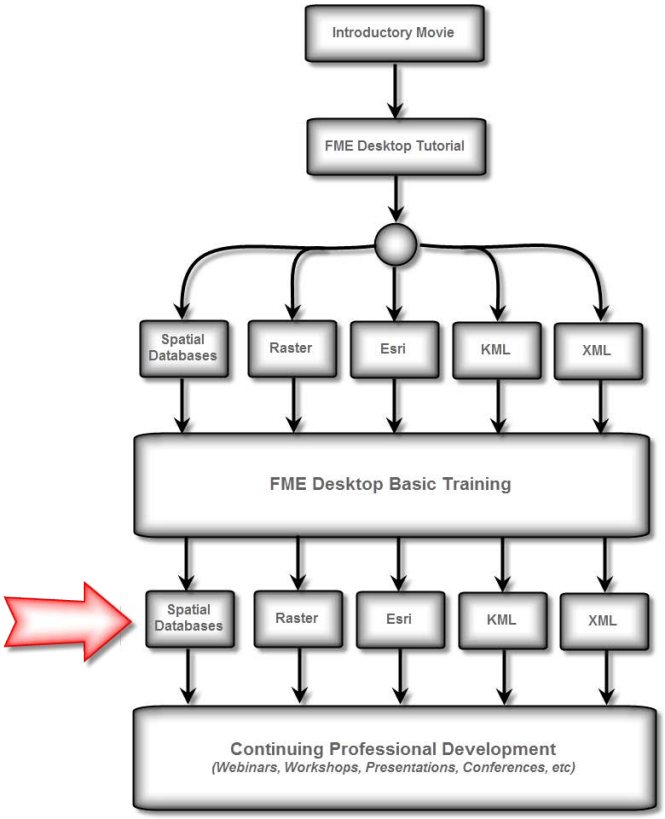
Document Name: FME Desktop Database Pathway Training Manual
FME Version: FME 2013-SP2 (Build 13499) 32-bit
Operating System: Windows 7 SP-1, 64-bit
Database: Oracle Database Express 11g Release 2 (11.2), 32-bit
Updated: June 2013

Introduction

This training material is part of the FME Training Pathway system.

Database Pathway
This training material is part of the FME Training Database Pathway.

It contains advanced content and assumes that the user is familiar with all of the concepts and practices covered by the FME Database Pathway Tutorial, and the FME Desktop Basic Training Course.



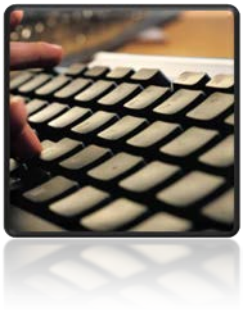
FME Version
This training material is designed specifically for use with FME2013-SP2. You may not have some of the functionality described if you use an older version of FME.

Sample Data
The sample data required to carry out the examples and exercises in this document can be obtained from:

www.safe.com/fmedata

Supported Database
For the purposes of simplicity, this training includes documented steps for Oracle 11g only. In particular it was created using Oracle Express 11g Release (11.2)

Connecting to a Spatial Database



Connecting to the database is the one step all FME operations must perform.

Connecting to a database is slightly different to selecting a file for a file/folder-based format. The operation relies much more on format specific parameters.

Client Software

Use of an Oracle Reader or Writer in FME requires the presence of Oracle database and/or client software.

The version of the client required is determined by the version of FME being used. 32-bit FME requires a 32-bit Oracle Client (regardless of whether you are using a 64-bit computer). 64-bit FME requires a 64-bit Oracle Client.

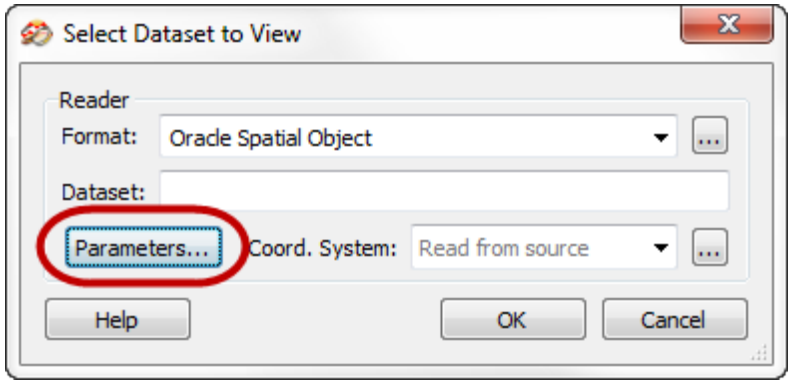
If you have both a 32-bit and a 64-bit FME installed then you will need to install both versions of Oracle Client, ensuring that ORACLE_HOME is **not** defined, as that would cause problems.

Basic Connection Parameters

The basic connection parameters for an Oracle database are:

- Host (Server) Name
- Database (Service/SID) Name
- Username
- Password
- Network Port Number

These parameters may differ slightly for each format, but will always be found in any Reader/Writer dialog by clicking on the “Parameters…” button.



Connecting to Oracle

Connection is possible through either *tnsnames.ora* or a direct connection.

tnsnames.ora

tnsnames.ora is a file that usually resides in the Oracle client installation folder. It is a text file that consists of a number of service definitions of the form:

```
<net_service_name> =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = <hostname>)(PORT = <1521>))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = <oracle_sid>)
    )
  )
```

When you have set up connection information in tnsnames.ora then in FME you can connect simply by specifying the service to connect to, a username, and a password.

Oracle Direct Connection

Direct Connection is when a single string is supplied that includes all the parameters required to connect to the database. This string should include all of the connection parameters, including server name and port number.

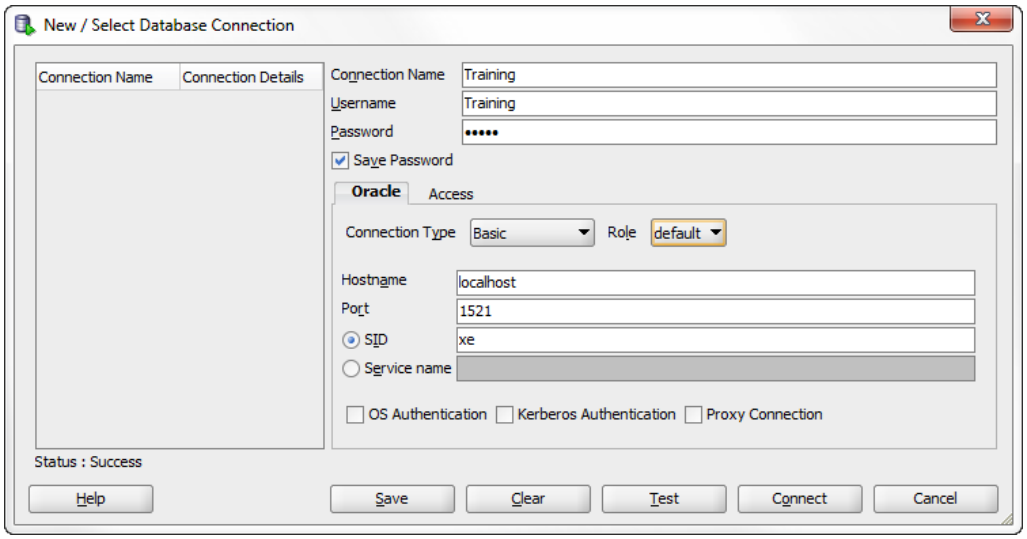
The connection string is of the form: `user/password@//hostname:port/sid`
For example: `training/training@//localhost:1521/xe`

When the database is on the same machine as you are connecting from, then you might be able to use simply: `user/password@sid`, for example `training/training@xe`

Testing a Connection

A connection can be tested using any of FME's Oracle Readers. The Non-Spatial Reader is preferred for a virgin database that won't have any spatial tables.

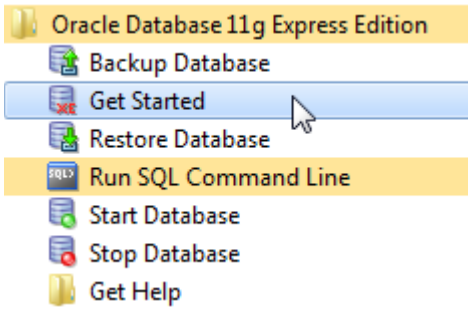
A connection can also be tested using an alternative tool such as Oracle's SQL Developer or Toad™ (Tool for Oracle Application Developers). If these products can connect to your database then there is no reason why FME should not be able to.



Example 1: Connection Parameters	
Scenario	FME user; City of Interopolis, Planning Department
Data	Census Tracts (Esri Shape)
Overall Goal	Test connection parameters by writing Census Tracts data to Oracle
Demonstrates	Connection parameters and database writing
Starting Workspace	None
Finished Workspace	C:\FMEData\Workspaces\PathwayManuals\Database1(Oracle)-Complete.fmw

Follow these steps to test the connection to your Oracle database:

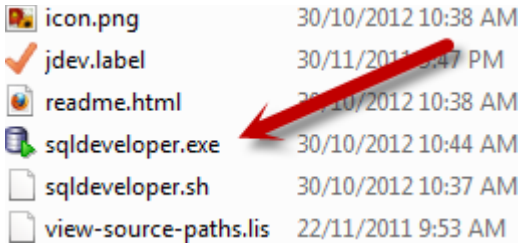
1) Open SQL/Database Tool
Open a tool with which to connect to the database.



Oracle has one built in that you can open by selecting **Start** > **All Programs** > **Oracle Database** > **Get Started**

Get Started *opens a basic interface to the Oracle database.*

Alternatively, locate and start up Oracle SQL Developer if it is installed on your system:



2) Enter Authentication Details
When prompted enter your authentication details.

In SQL Devloper you will need to create a new Connection. In the Get Started page you'll need to click on the Storage Tab to be prompted.

Either click **Login** or **Connect**.

In FME training, the authentication parameters are usually:

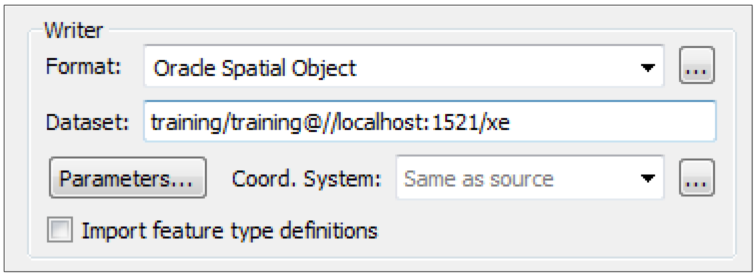
Username training
Password training

The Get Started authentication process is really just a test of the username and password. Success here does not guarantee a successful connection from FME.
The best test is to use a tns option in a third-party application like SQL Developer.

3) Start FME Workbench
Assuming a successful test, start FME Workbench, and open the Generate Workspace dialog. Set up a translation as follows:

Reader Format	Esri Shape
Reader Dataset	*C:\FMEData\Data\GovtBoundaries\TravisCounty\CensusTracts.shp*
Writer Format	Oracle Spatial Object

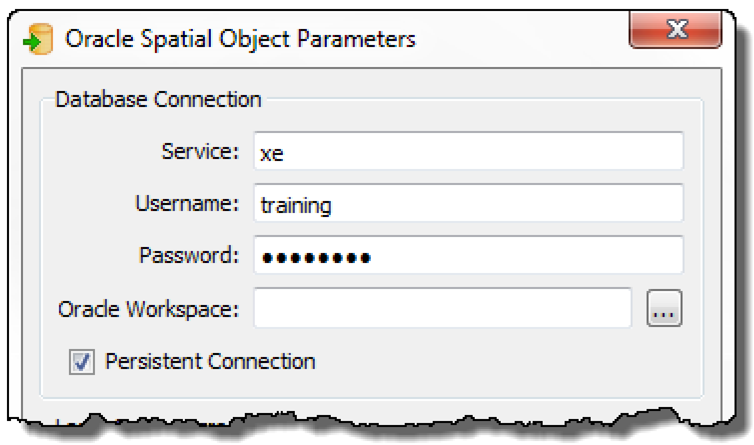
For the connection parameters either enter a direct connection into the Dataset field:

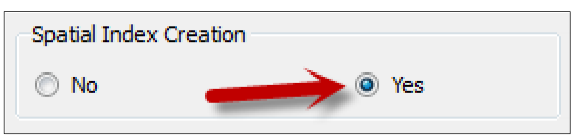


…or click on the Parameters button and enter the connection parameters in this dialog:

In FME training this is usually:

Service:	XE
Username:	training
Password:	training





Also set Spatial Index Creation to Yes.

Now click **OK**, and then **OK** again, to create the workspace.

4) Run Workspace
Run the workspace. The foot of the log file will report success as follows:

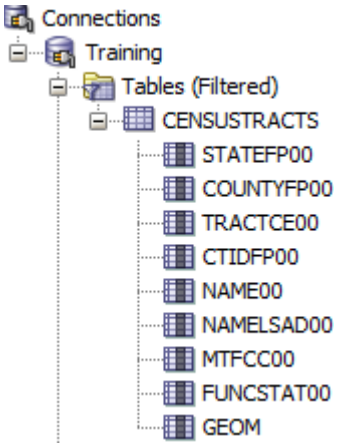
```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                           Features Written Summary
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
CENSUSTRACTS                                                                 181
================================================================================
Total Features Written                                                       181
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Translation was SUCCESSFUL with 1 warning(s) (181 feature(s) output)
```

If the log reports a message like one of these:

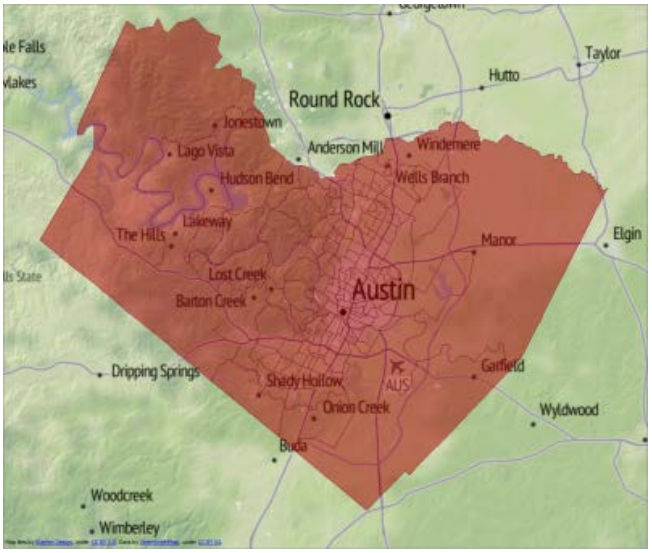
- `ORA-01017: invalid username/password; logon denied`
- `ORA-12154: TNS:could not resolve the connect identifier specified`

…then you should check the connection parameters entered into the dialog and, if necessary, re-check these against the parameters entered into the Oracle Database Home Page.

5) Check Result
In SQL Developer the result of the translation will appear as a new table, CENSUSTRACTS:



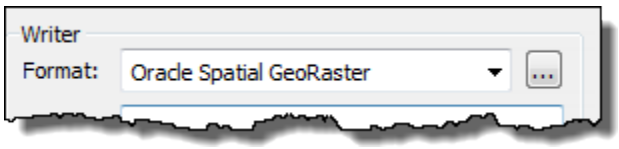
In the FME Data Inspector (with a background map and the area fill set to 50% transparency) the data looks like this:



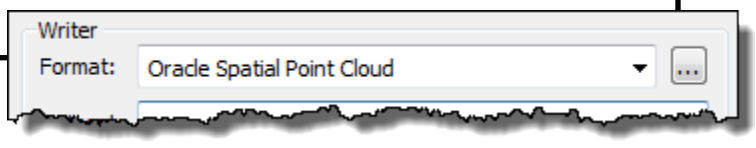
Advanced Task
As an advanced task, try converting the following dataset to Oracle:

Reader Format LizardTech MrSID
Reader Dataset *C:\FMEData\Data\Raster\130105.sid*

As a raster dataset you'll need to use the Oracle Spatial GeoRaster writer. You'll also need the full Oracle database – Oracle XE and Locator do not support raster.

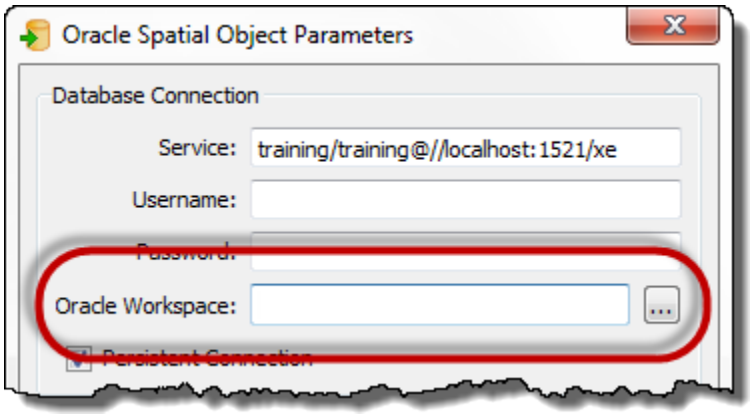


If you are interested in less-common types of data, then be aware that both Oracle and FME support Point Cloud datasets. In fact, like raster, FME has a specific format for this type of data.



Oracle Workspace Manager
The Oracle Workspace Manager is basically a form of versioning for an Oracle database.

FME lets you select an Oracle workspace to read from, and decide which to write to. The parameter is found in the same location on the Reader and Writer parameters dialog:



If this value is left empty then the default LIVE workspace will be used.

From a Reader, you can determine which features came from which Oracle Workspace by using the format attributes fme_dataset and fme_feature_type

Updating Features

Updating entire tables is simple enough, but updating individual features is a task that requires a little more finesse.

Once information is stored in a database, it's unlikely it will stay static. Changes will occur.

Sometimes the update will involve reloading an entire set of data, totally replacing the existing data. Sometimes the table will also be replaced, and sometimes just emptied and refilled. Feature Type parameters such as 'Drop Table' and 'Truncate Table' can help with this task.

However, most often only certain features need to be updated. The source data may indicate which features require updates, but sometimes it is necessary to go through a process of change detection to decide what to update.

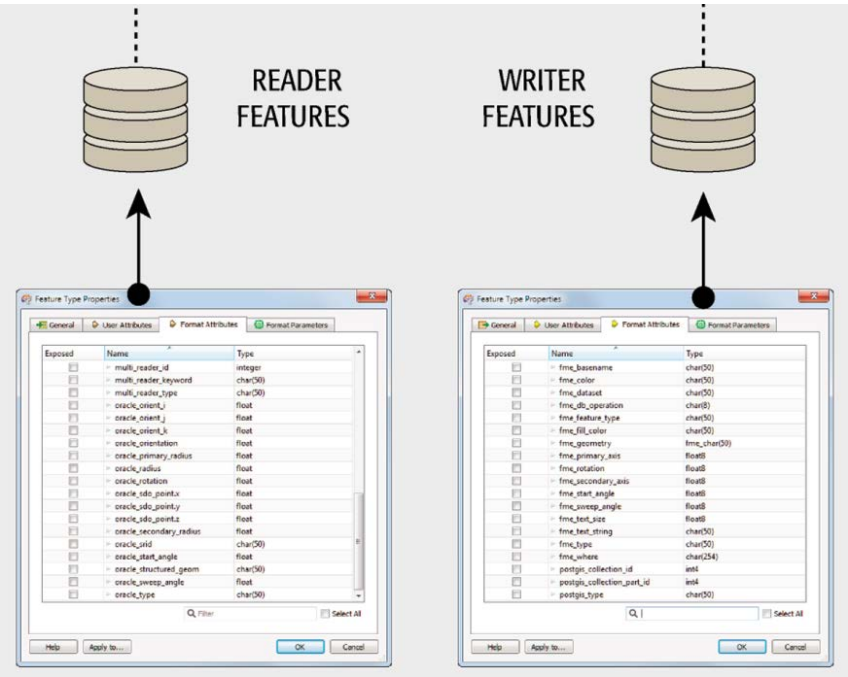
Database Format Attributes

To carry out specific actions on specific features, **Format Attributes** must be used.

You'll recall that operations on a Reader or Writer (i.e. at the database level) are carried out using Reader/Writer Parameters, located in the Navigator window.

Operations on a feature type (i.e. at the database table level) are carried out using Feature Type Parameters, located both in the Navigator window and the Feature Type Properties dialog.

However, operations on individual features are carried out using Format Attributes. For databases there are two particular format attributes that control updates on individual features.



fme_db_operation
fme_db_operation is a format attribute whose value denotes how a database writer should handle that feature. Very simply, it may take the value DELETE, INSERT or UPDATE.

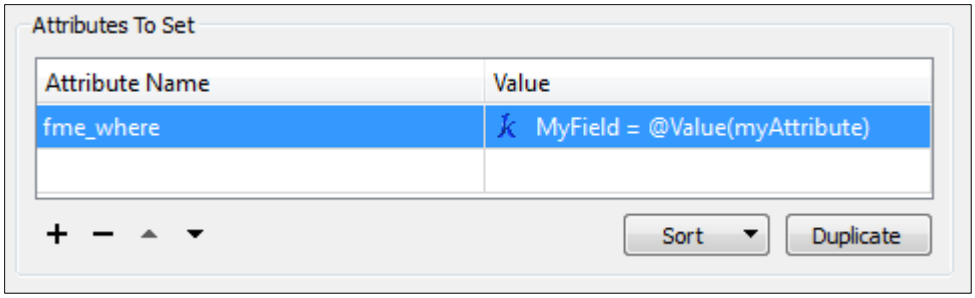
fme_where
fme_where is a format attribute whose value denotes a match that identifies which database record(s) this feature should update.

The structure is usually:

```
<database field> <operator> <value>
```

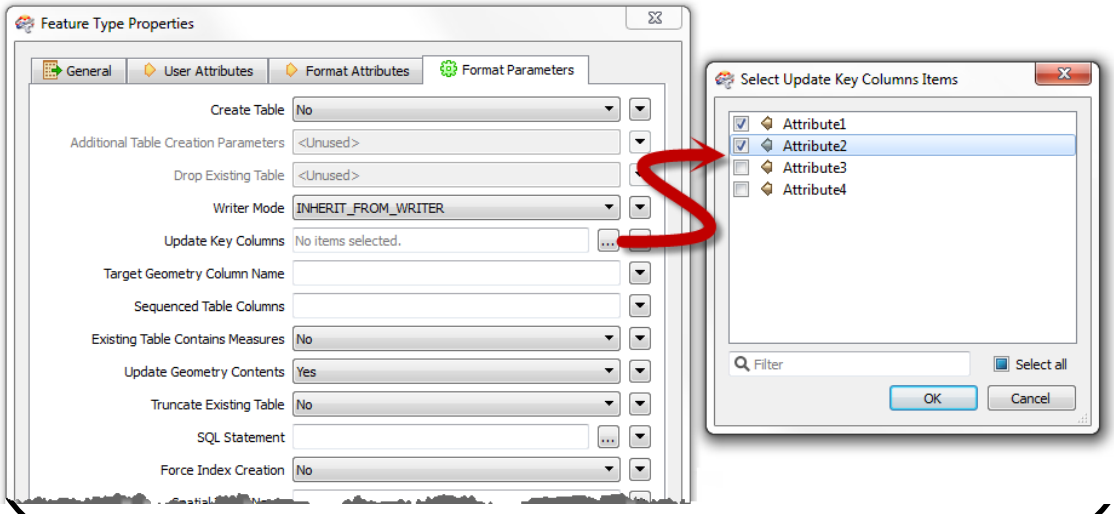
For example an fme_where statement of `MyField = 4` says to update features where the database column named "MyField" has a value of 4.

fme_where can be constructed in FME using either a StringConcenator or AttributeCreator transformer. Integrated attribute handling allows a where clause to be constructed that matches the value of an attribute on a feature, for example:



If the value part of the where clause is a string, then the string part must have single quotes around it, for example `MyField = 'abc'`, or `MyField = '@Value(myAttribute)'`

As an alternative to using the fme_where format attribute, you can use the "Update Key Columns" parameter on the writer feature type. One advantage there is that you can easily select multiple columns

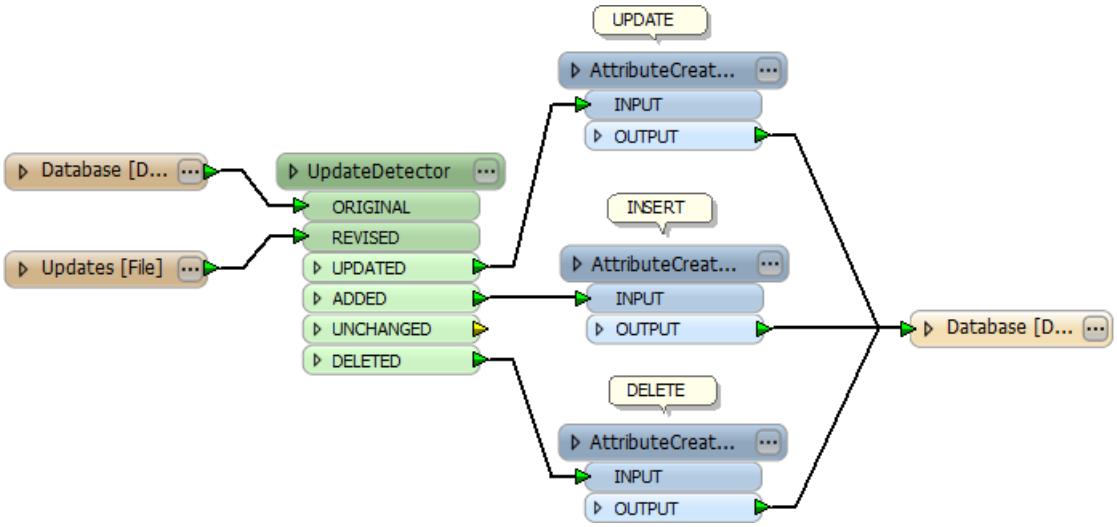


Identifying Features

When an entire database or table needs updating, it's easy to identify which features are to be processed in which way. However, when only certain features need to be updated it's important to be able to identify which features they are.

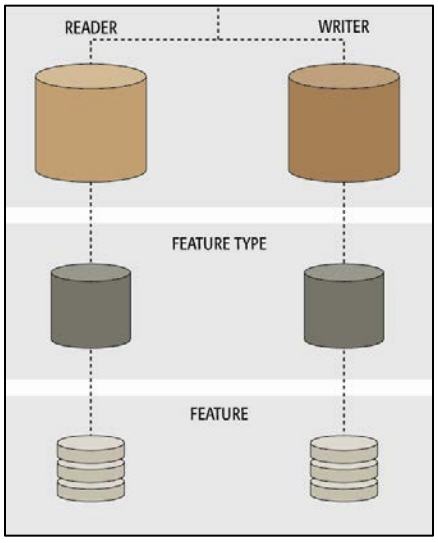
In this scenario the source data sometimes indicates which features require updates. On other occasions it's necessary to go through a process of change detection.

A typical Change Detection workspace uses a ChangeDetector or Matcher transformer (sometimes in a Custom Transformer like the UpdateDetector) and will look like this:



Parameter Priority

The basic rule for parameters is that any higher-level parameter affects every component below it. For example, a Reader Parameter affects all Feature Types that belong to that particular reader.



Database writing mode does work in this way in general. For example, if the writer level is set to INSERT then ALL features are written to tables as an insert.

However, this mode can be set not only at the Writer level, but also at the Feature Type level, or on individual features with a format attribute; and this causes a different effect.

When the same parameter exists at multiple levels, the higher-up parameter only applies when the lower-down parameters are not set (or are set to "INHERIT FROM WRITER"). When the same parameter is set at different levels, then the lower-level parameter wins out.

For example, a Writer might be set as INSERT mode; but a table is set to UPDATE mode. In that case the Feature Type level parameter wins out, and features are written to that table as an update.

Example 2: Feature Updates	
Scenario	FME user; City of Interopolis, Planning Department
Data	Address Data (GeoMedia Access Warehouse)
Overall Goal	Load address data and updates
Demonstrates	Feature-level updates
Starting Workspace	None
Finished Workspace	C:\FMEData\Workspaces\PathwayManuals\Database2(Oracle)Loader-Complete.fmw C:\FMEData\Workspaces\PathwayManuals\Database2(Oracle)Updater-Complete.fmw

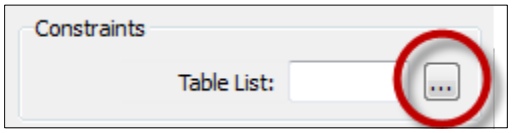
The city has a set of address data, plus a set of updates. This example will show how to load the data (with a Loader workspace) and apply the updates (with an Updater workspace). It also highlights some of the pitfalls and problems that can occur during such a process.

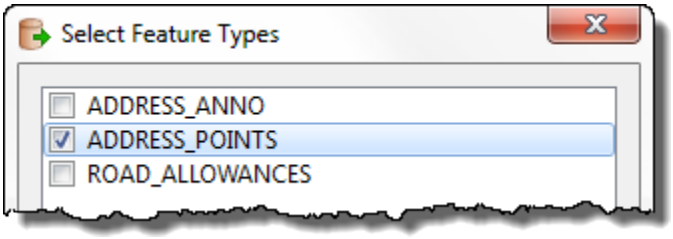
Loader

1) Create Loader Workspace
Start Workbench if necessary and open the Generate Workspace dialog.
Set up a translation as follows:

Reader Format Intergraph GeoMedia Access Warehouse
Reader Dataset *C:\FMEData\Data\Addresses\roadAllowancesAndAddressPoints.mdb*

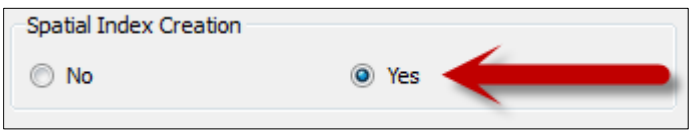
Reader Parameters
Table List Click the Table List button





Choose ADDRESS_POINTS as the source table to read

Writer Format Oracle Spatial Object
Writer Parameters Enter the database connection parameters as before
Spatial Index Creation Yes



A spatial index makes it more efficient for querying and updating data in the database table. In particular, it can really speed up a spatial query such as an envelope search.

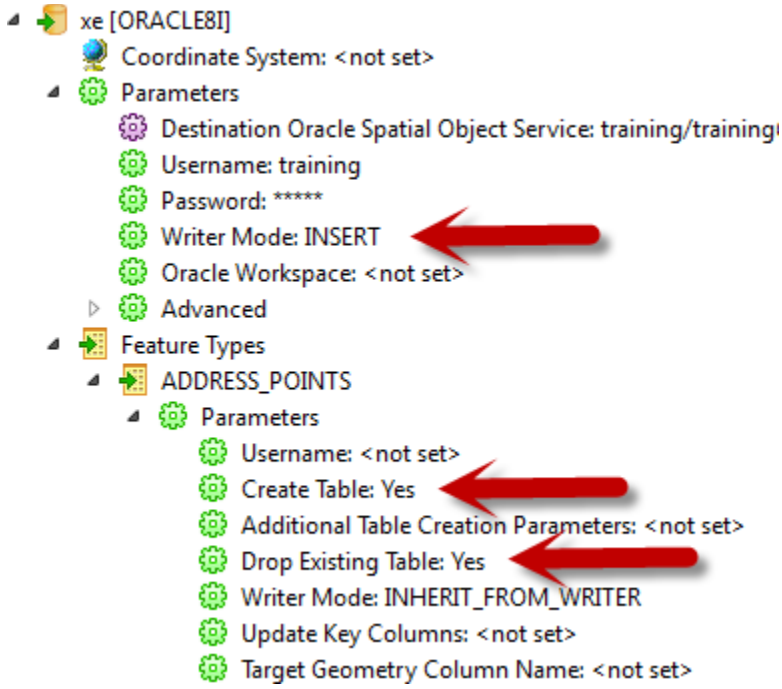
Click **OK**, and **OK** again, to create the workspace.

Once you have set the connection parameters a few times, and are sure they are correct, then choose the "Save as My Defaults" option at the foot of the parameters dialog.

This will save the parameters and prevent them having to be entered again and again.

2) Check Parameters
Check that the writer parameter 'Writer Mode' is set to INSERT.

Next check that the Feature Type parameters 'Create Table' and 'Drop Existing Table' are both set to **Yes**.



3) Save and Run Workspace
Save the workspace, and then run the workspace.

The process will take about 10 seconds to load 12,292 features.

Use the FME Data Inspector to inspect the contents of the newly written table.
Notice that the attribute PRIMARYINDEX is a unique ID number for each address.

PRIMARYINDEX	ADDRESS	PRE	STREET_NAM	STREET_TYP
1	4201		BROOKVIEW	RD
2	1718		SCHIEFFER	AVE
3	1707		SCHIEFFER	AVE
4	1709		SCHIEFFER	AVE
5	4018		CRESCENT	DR
6	1711		SCHIEFFER	AVE
7	4016		CRESCENT	DR
8	1713		SCHIEFFER	AVE
9	4011		CRESCENT	DR
10	4019		BROOKVIEW	RD
11	4009		CRESCENT	DR

You can rearrange the order of columns in the Data Inspector Table View by dragging a column into a new position.

Updater

4) Inspect Updates File
In a text editor open the file *C:\FMEData\Data\Addresses\AddressUpdates\AddressUpdates.txt*

This file contains updates to the main address database. It has almost exactly the same schema, but has an additional field called UPDATE_TYPE, which records the type of edit to carry out:

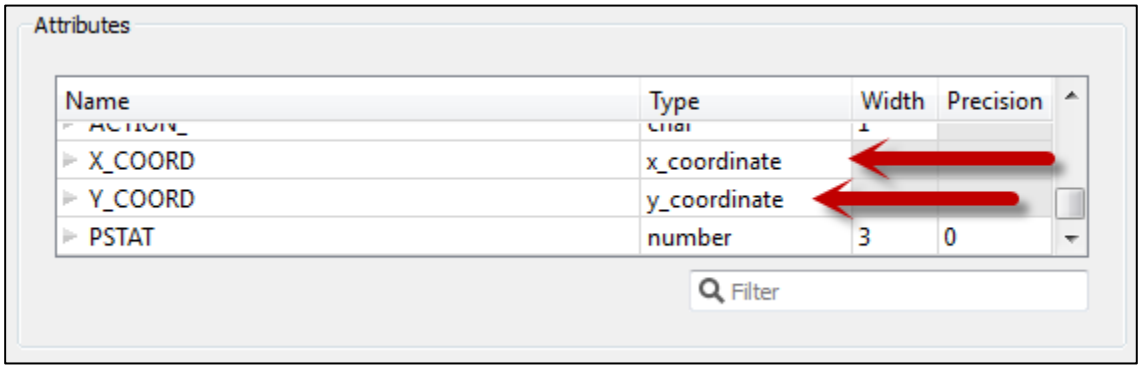
I	[I]nsert this record to the database as a new address
D	[D]elete this record from the database as an old address
U	[U]pdate this record as a changed address

5) Create Updating Workspace
Back in FME Workbench (it's easiest to start a new instance of Workbench) create a workspace to translate the updates file into Oracle.

Reader Format Comma Separated Value (CSV)
Reader Dataset *C:\FMEData\Data\Addresses\AddressUpdates\AddressUpdates.txt*

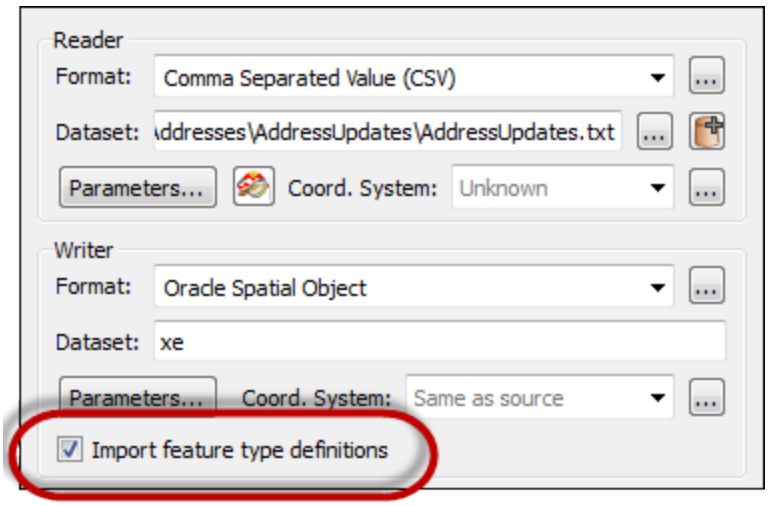
Reader Parameters
File Has Field Names Yes **Lines to Skip (Header)** 1

Attributes
For the Attributes section, locate X_COORD and Y_COORD and change their attribute types to x_coordinate and y_coordinate. This will set the geometry of the features being read.



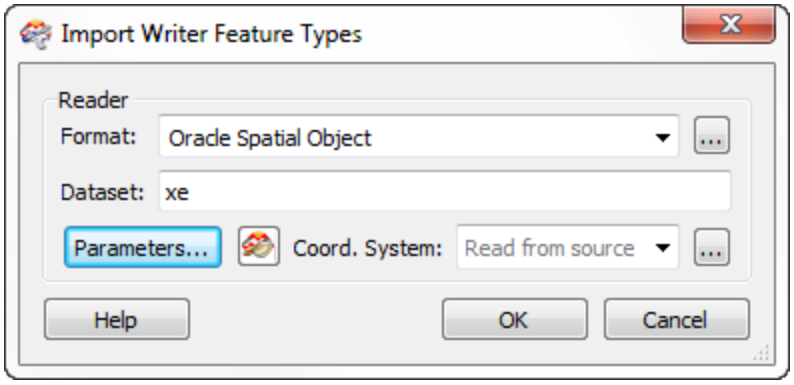
Writer Format Oracle Spatial Object

Check the option to "Import Feature Type Definitions" and click **OK**.

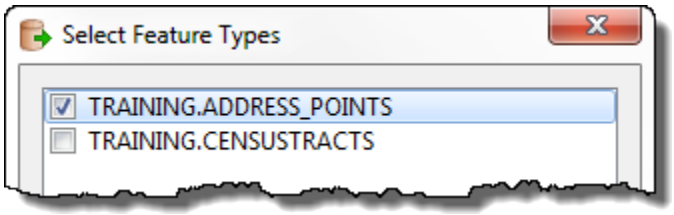


6) Select Writer Feature Types

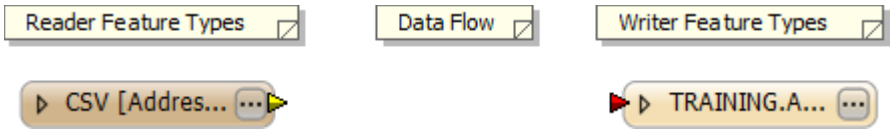
When prompted, we should set the format and dataset where the feature types can be imported from. By default this should be the Oracle database, so the fields may not need much editing:



Click the Parameters button, and use the table selection tool to select the newly created ADDRESS_POINTS table (it will be named TRAINING.ADDRESS_POINTS or possibly FMEDATA.ADDRESS_POINTS depending on the Oracle database definition):



Click **OK** and then **OK** again. A workspace will be created that looks like this:



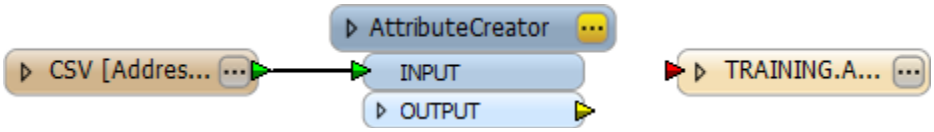
7) Assign Operation Type
The different action types defined in the updates file need to have different values for *fme_db_operation* in order to carry out the different action for each.

The workspace must be set up to assign the following:

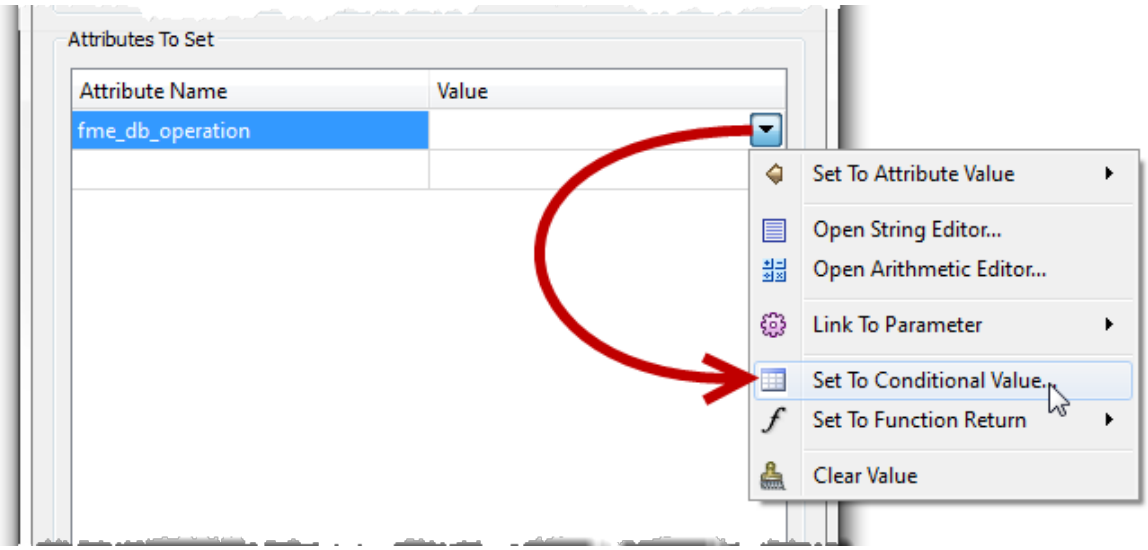
Update Type	fme_db_operation
I	INSERT
D	DELETE
U	UPDATE
N	<none>

In FME2013-SP1 (or newer) this can be done with an AttributeCreator transformer using new Conditional Mapping functions.

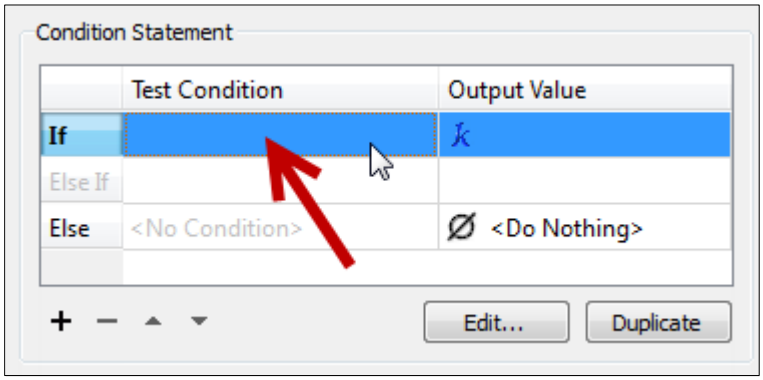
Place an AttributeCreator transformer and connect it to the Reader Feature Type, like so:

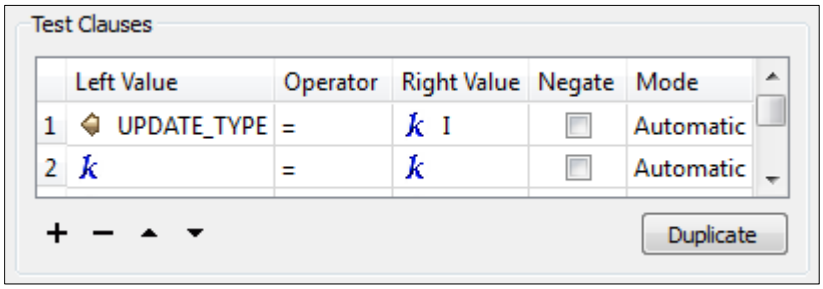


8) Set up INSERT
Open the AttributeCreator parameters dialog. Enter fme_db_operation as the Attribute Name to be created. Open the Value drop-down menu and choose “Set to Conditional Value”.



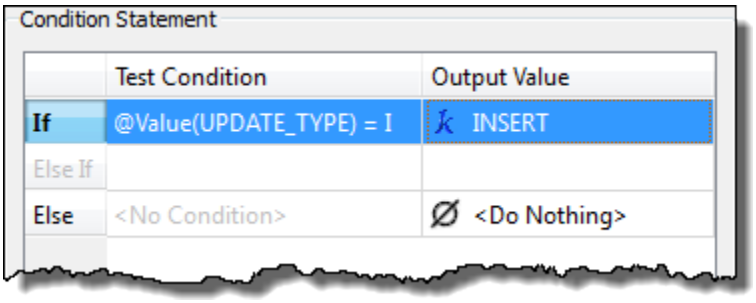
In the Condition Definition dialog there is a line for each test to be carried out. To start with double-click in the first “If” condition.

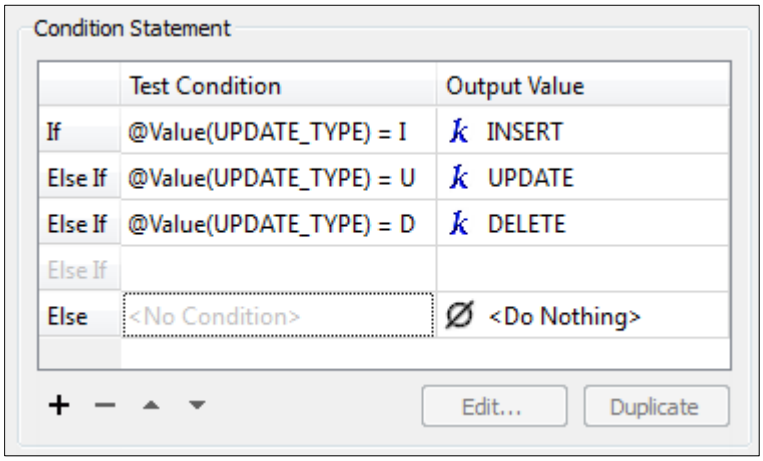




This opens up a Tester-like dialog. In here enter a test to check whether the incoming feature is an INSERT:

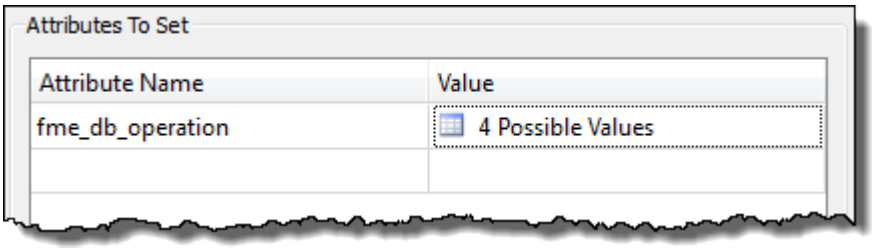
Back in the Condition Definition dialog, set the output value (remember we are setting fme_db_operation here) to INSERT





9) Set up UPDATE/DELETE
Now repeat this process to set up Conditional Mapping for the U (UPDATE) and D (DELETE) update types.

10) Assign WHERE Clause
The AttributeCreator dialog will now show there are 4 possible values for fme_db_operation (INSERT, UPDATE, DELETE, <Do Nothing>).



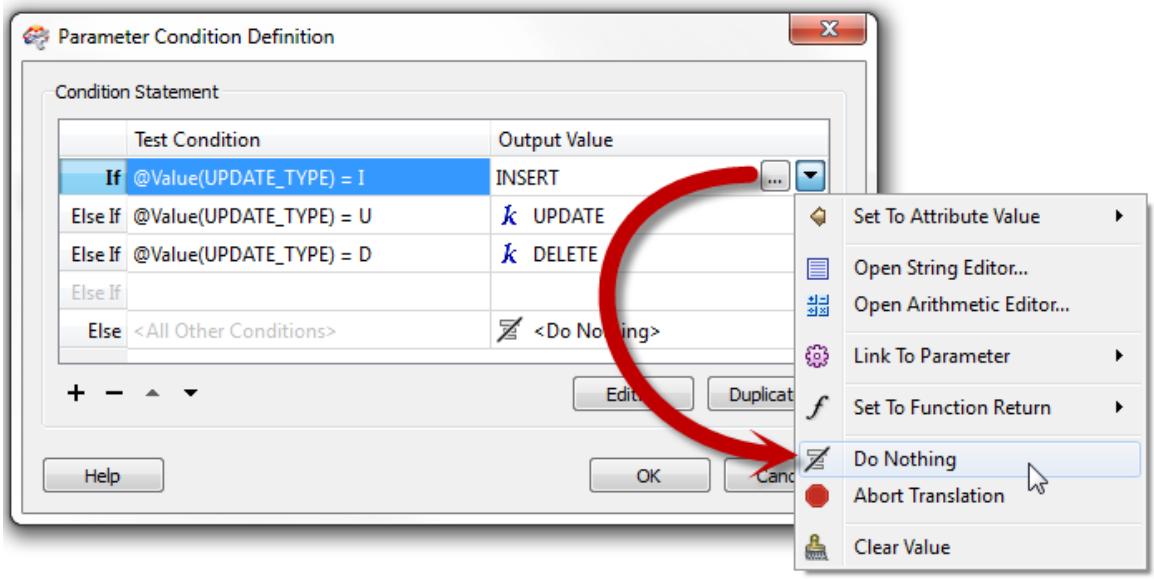
Now we can set up a WHERE clause by creating fme_where.

In the AttributeCreator parameters dialog, click on the entry for fme_db_operation and then click the Duplicate button. This sets up a duplicate set of conditions/values and is the easiest method to use where all of the tests (here for Update Type) will be the same.

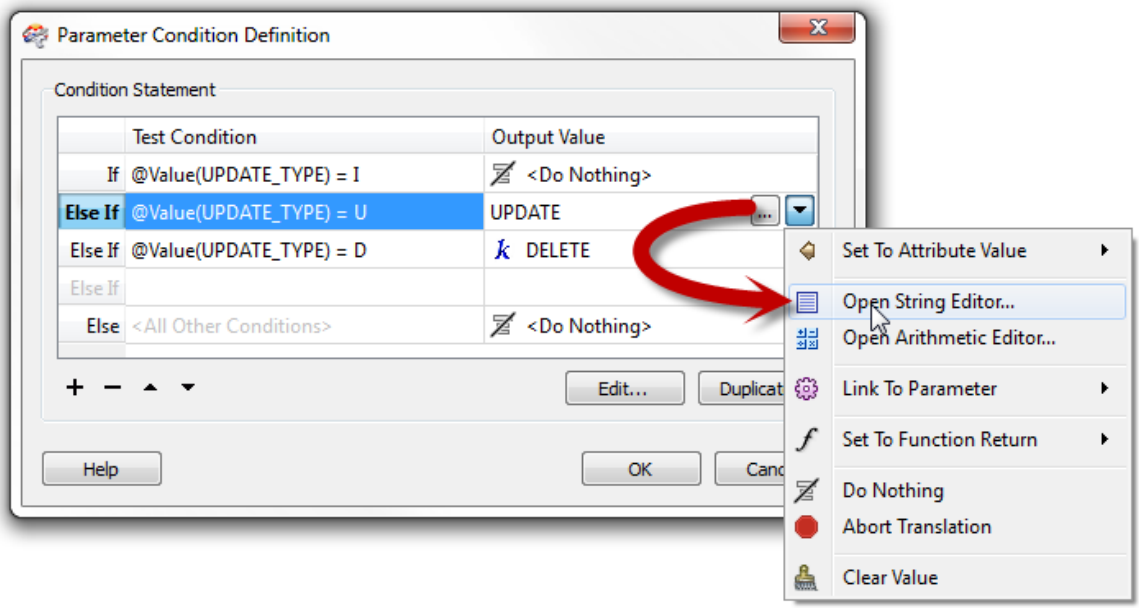
Change the newly created attribute name from fme_db_operation(1) to fme_where and select "Set to Conditional Value".

Because this is a set of duplicate conditions, in the Condition Definition dialog we can leave the Test Conditions to be the same and now only need to start editing the output values.

Where UPDATE_TYPE is "I", the output value can be left empty, as a WHERE clause is not required for an INSERT. So set the output value to "Do Nothing":



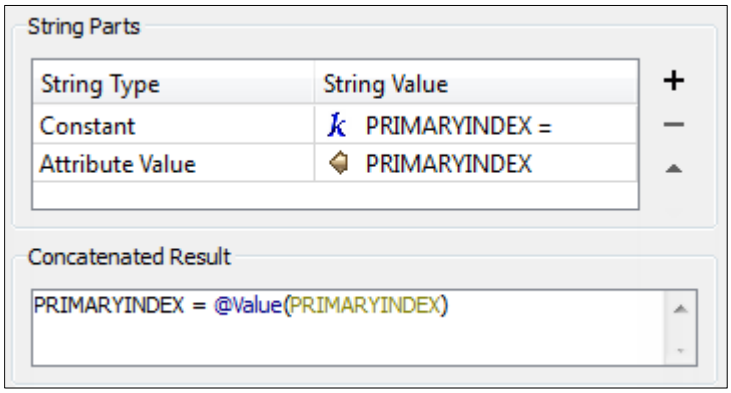
Where UPDATE_TYPE is “U” the output value needs to be a SQL WHERE clause. So click the drop-down menu and select Open String Editor.



In the String Editor dialog, use either the Basic or Advanced version to create a string matching a field called PRIMARYINDEX to the value of the attribute called PRIMARYINDEX.

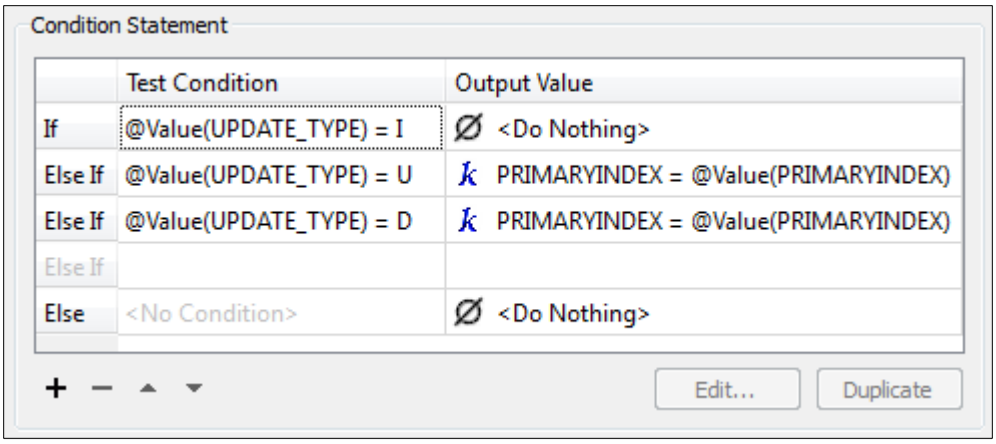
In the basic editor it will look something like this:

The Advanced Editor will merely show the string: PRIMARYINDEX = @Value(PRIMARYINDEX)

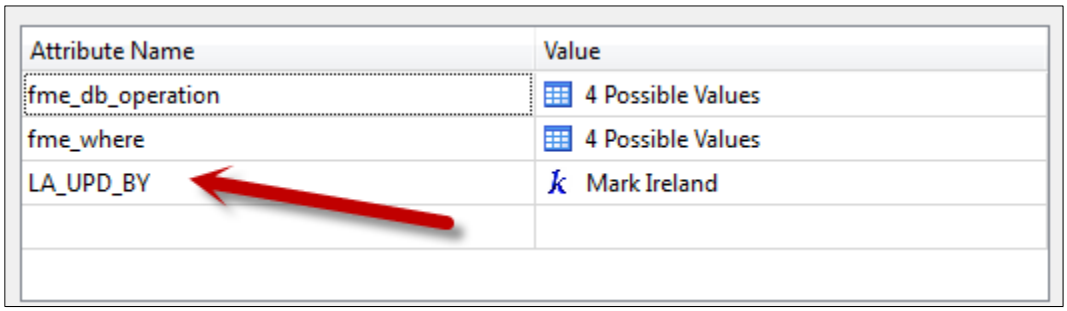


Now repeat the process for the DELETE update type (it will have the exact same WHERE clause):

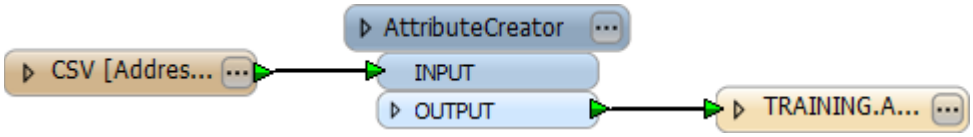
You could, of course, delete the existing DELETE condition and duplicate the UPDATE one.



11) Set Last Update Field
So that it's possible to recognize the updated data, add another attribute to the *AttributeCreator* transformer to create LA_UPD_BY (Last Updated By) and set its value to be a constant such as your own name.

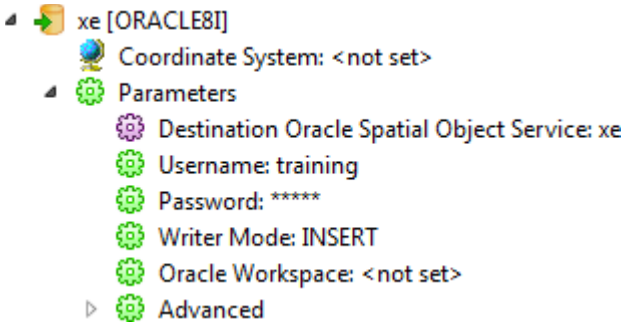


Connect the *AttributeCreator* to the output feature type and your workspace now looks like this:

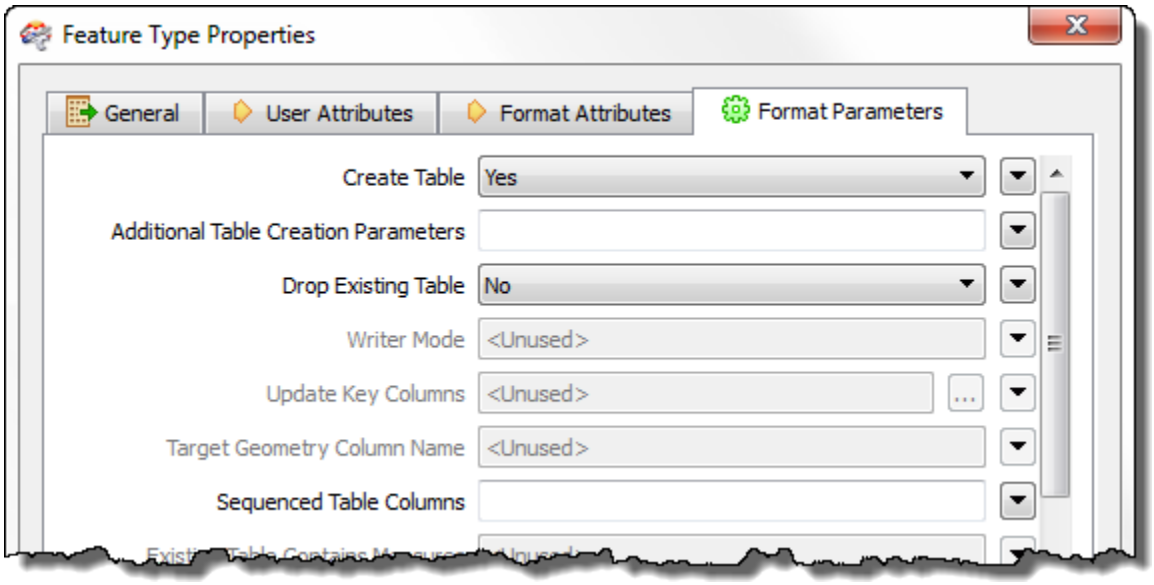


12) Check Parameters
The workspace is now complete, but the parameters need attention.

At this point the Writer (database) parameters probably look like this:



…and the writer Feature Type (table) parameters like this:



Obviously we don't want to drop or re-create the table, merely update selected records.

So set:

Create Table = *No*.

When you do this, Drop Existing Table is automatically grayed-out, and Writer Mode changes from <Unused> to INHERIT_FROM_WRITER.

The question now is one of Writer Mode. There are parameters on both the Writer (seen in the Navigator screen shot) and on the Feature Type. Currently:

Writer: Writer Mode	INSERT
Feature Type: Writer Mode	INHERIT_FROM_WRITER

We have already defined writer mode (through fme_db_operation) to be a separate action (INSERT, UPDATE, DELETE) for individual features. According to the FME Readers and Writers Manual, this will work *"unless the parameter at the feature type level is set to INSERT"!*
In other words, to force FME to use fme_db_operation you need to set Writer Mode: UPDATE

There are two options here:

- Leave the writer:Writer Mode parameter as-is, and set the feature type:Writer Mode parameter to UPDATE.
- Set the writer:Writer Mode parameter to UPDATE, and leave the feature type:Writer Mode parameter as-is.

Choose one of these methods to change writer mode to UPDATE. *Q: Are there any obvious benefits to choosing one over the other?*

13) Save and Run Workspace

Save the workspace, and then run it. The workspace will take about 25 seconds to complete. There are 1465 features processed and, counting the updates file, I find out that there are:

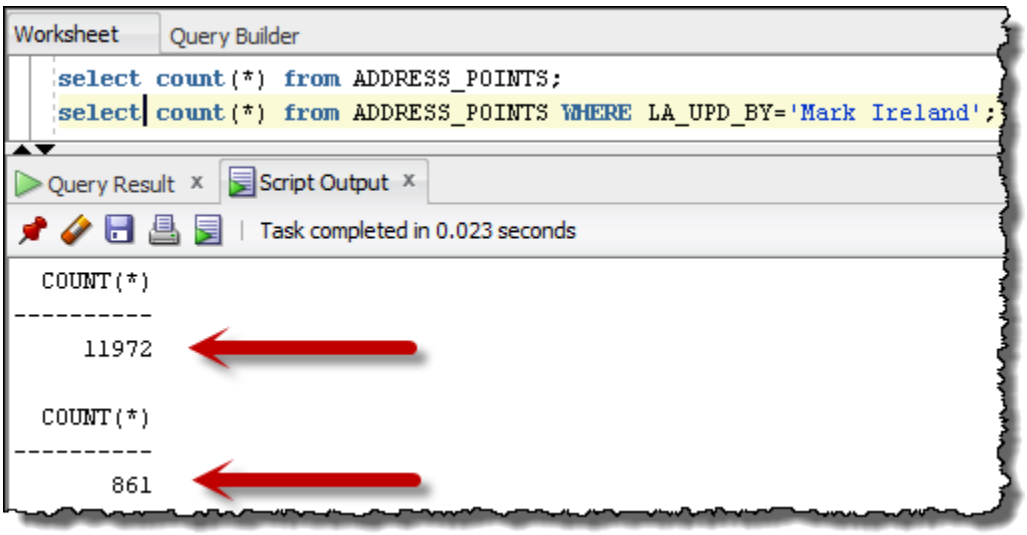
INSERTS	284
DELETES	604
UPDATES	577

So, two calculations are important:

Original Features (12,292) + INSERTS (284) – DELETES (604) = 11972
There should now be 11,972 records in the database.

INSERTS (284) + UPDATES (577) = 861
There should be 861 records now tagged with LA_UPD_BY = <YourName>

Query the database (using SQL Developer or a similar tool) to find out if these calculations are correct:



Advanced Tasks
Can you use the FME Universal Viewer to check whether there are now 861 records tagged with your name as the LA_UPD_BY?

Can you use a transformer to set a value for LA_UPD_DAT as the update date/time?

What would happen if the X_COORD or Y_COORD attributes were the part that had changed in a "U" (Updated) record? The CSV Reader will use those coordinates in the new feature, but what Writer (Format Parameter) would you need to ensure is set to update the geometry?

Lessons to Learn!
Besides how to apply updates to a database, there are two important lessons to take from this exercise:

- Always check your writer parameters carefully. Then check them again.
  And again.

- Don't assume because the translation was successful, that the data was written correctly!
  Always inspect the output to make sure it is what you were expecting.

Time and Date Attributes in Spatial Databases

Time and Date Attributes are among the more tricky to get into, and out of, a database.

Time and date attributes are complicated territory because each different database format may have its own unique structure for dates.

Oracle
Oracle expects DATE values in the format YYYYMMDDHHMMSS even though when you display a date field from an Oracle table it shows something like this: **01-JAN-08 12:00:00**

Microsoft SQL Server
DateTime fields represent date and time data from January 1, 1753 to December 31, 9999.
For example, a value of 20061231235959 represents 11:59:59PM on December 31, 2006.
When writing to the database, the writer expects the date attribute to be in the form YYYYMMDDHHMMSS

SmallDateTime fields represent date and time data from January 1, 1900 to June 6, 2079.
For example, a value of 20060101101000 represents 10:10:00AM on January 1, 2006.
When writing to the database, the writer expects the date attribute to be in the form YYYYMMDDHHMMSS

IBM Informix
The Informix reader returns two attributes for each DATE field.

The first attribute has the name of the database column, and the form YYYYMMDD
The second attribute has a suffix of .full and is of the form YYYYMMDDHHMMSS

For example, if the date field is called UPDATE_DATE, the attributes are UPDATE_DATE and UPDATE_DATE.full

The Informix writer looks for both attributes when a DATE or DATETIME column is being output. Either may be specified. If both attributes are specified, then <name>.full takes precedence.

Formatting Date Attributes with Transformers
To write dates to a database DATE or DATETIME field you can use the *TimeStamper* or *DateFormatter* transformer to get the date into the correct format.

A format string of ^Y^m^d^H^M^S will return a date-time in the form YYYYMMDDHHMMSS
A format string of ^Y^m^d will return a date in the form YYYYMMDD

New for FME2013, the DateFormatter now also allows you to specify the source date format using the same format strings. Also provided are default output strings of the most common output formats.

Creative Feature Reading

Rather than a plain reader, there are some quite creative ways by which database features can be read using Workbench.

Using FME to read from a database should be carefully planned and considered. Frequently not every feature in every table is required, and yet that is what a user might be doing.

The fewer features that are read from the database, the quicker the read will be, the less system resources are used, and the faster the overall translation will be.

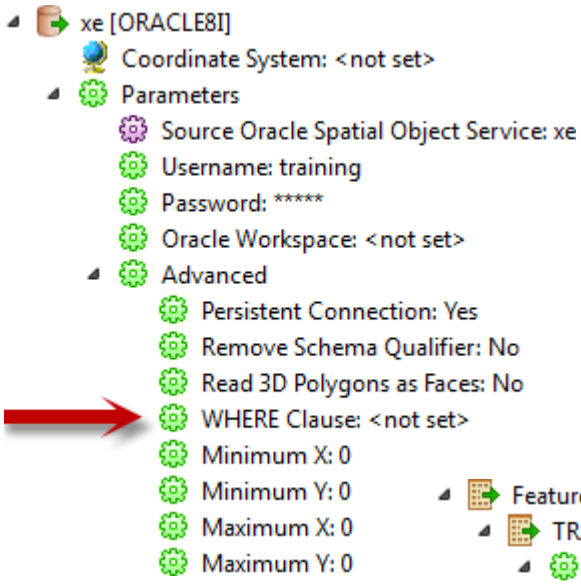
Chef Bimm says…

'Think of a database like a restaurant. A sensible person would browse the menu, and order just the dishes that they want. A foolish person would order everything and waste the food they didn't really need.

Like a restaurant, it's expensive and time-consuming to order all data from a database just to discard most of it. Far better to order only the data you intend to consume in the current session!'

There are a number of items of functionality that can improve the performance of database reading in this way:

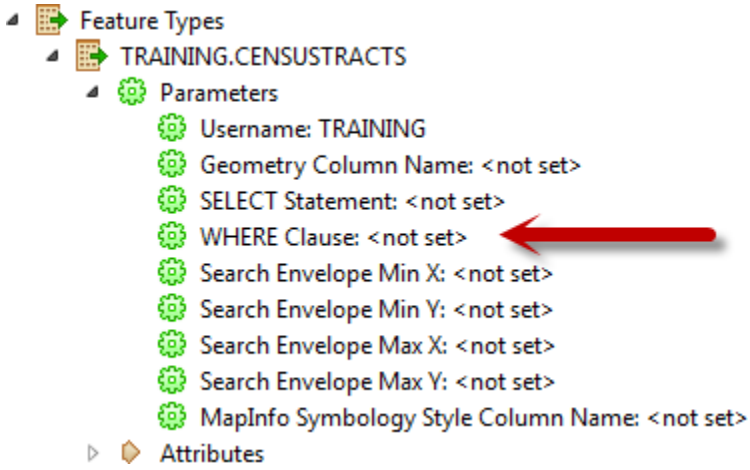
- WHERE Clause
- Search Envelope
- "Records to Read at a Time" parameter
- Concatenated Parameters
- FeatureReader transformer
- Oracle Workspace Manager



WHERE Clause
Most database readers will have a WHERE clause parameter. Here a query can be set, so that only features that pass the query will be returned to FME.

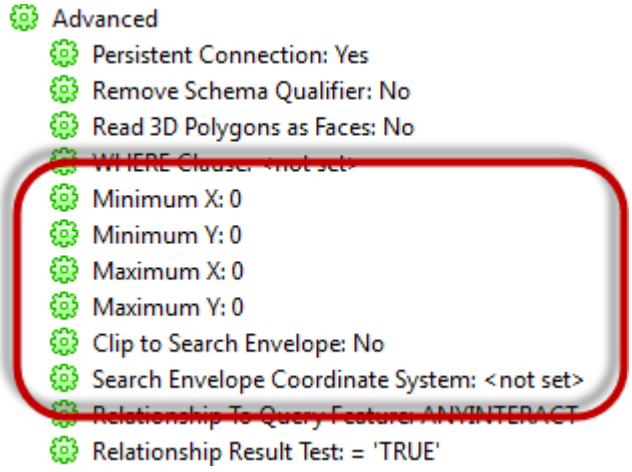
This employs database query tools – which in turn make use of database indices – and is a lot more efficient than reading an entire table and then filtering it with a *Tester* transformer.

The WHERE clause can also be set on each individual Reader Feature Type:



Search Envelope
Similar to a WHERE clause, "search envelope" parameters set a spatial query; only features that fall inside the specified extents will be returned to FME.

Again, this employs native database functionality, and is more efficient than reading the entire table and then clipping it with a *Clipper* transformer.



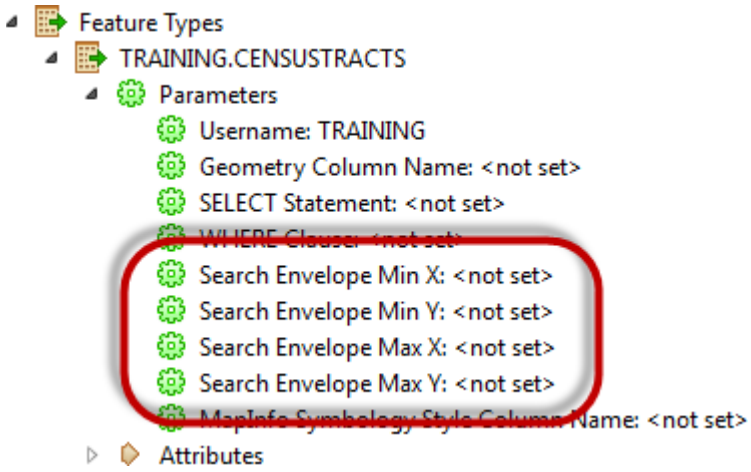
An optional "Clip to Search Envelope" parameter defines whether features will be clipped where they cross the defined extents, or be allowed to pass completely where at least a part of them falls inside the extents.

Of course the limitation here is that the four parameters only define a rectangular envelope.

For convenience, each reader also has a parameter called “Search Envelope Coordinate System”. This allows the search envelope coordinates to be provided in a different coordinate system to that of the actual data.

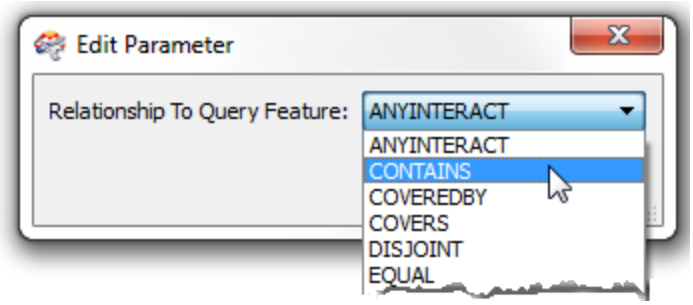
For example, the coordinates could be provided in Lat/Long format (degrees) to use as a search envelope for data stored in State Plane (metres/feet).

As with the WHERE clause, the Search Envelope can also be set on each reader feature type.



Search Envelope Relationship

The Search Envelope is also used by a parameter called “Relationship to Query Feature”.

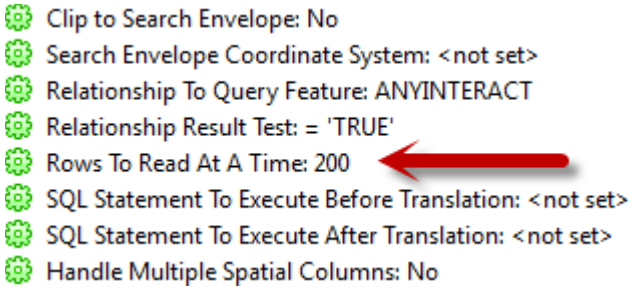


This allows the user to query features based on their spatial relationship to the Search Envelope. For example the user may wish to retrieve features that are entirely within the envelope, or that intersect the search envelope.

To use a search envelope requires a spatial index to exist for that table; FME can't do a spatial query without one. If a format doesn't support a spatial index being created on a view, then FME will not be able to do spatial queries on that view.

“Rows to Read at a Time” Parameter

This parameter defines how many records will be fetched at a time from the database. Tweaking it allows the user to tune the performance of the reader.



Too low a number and FME will spend excessive time making read requests. Too high a number and database performance might be slowed down for other users.

Example 3: WHERE Clause	
Scenario	FME user; City of Interopolis, Planning Department
Data	Oracle Spatial Object (Address Point Data)
Overall Goal	Read address data with a query
Demonstrates	WHERE Clause parameters
Starting Workspace	None
Finished Workspace	C:\FMEData\Workspaces\PathwayManuals\Database3(Oracle)-Complete.fmw

To test the previous example, it's time to read back some data – but only for a specific zipcode.

1) Start FME Workbench
Start Workbench if necessary and begin with a blank workspace.

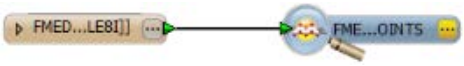
2) Add Reader
Add a Reader using **Readers** > **Add Reader**. Set it up as follows:

Reader Format Oracle Spatial Object

Reader Parameters
Database Connection Enter the database connection parameters as before
Table List Select the ADDRESS_POINTS table

Click **OK**, and **OK** again to close the dialogs and add the new reader.

3) Add Inspector
Connect an *Inspector* transformer to the reader feature type.



4) Save and Run Workspace
Save and run the workspace. Note how many features were read and how long it took.

5) Set WHERE Clause
Locate the WHERE Clause parameter in the Navigator Window, and double-click it.
In the Edit Parameter box, enter:

```
ZIPCODE=78723
```

6) Re-run Workspace
Re-run the workspace. Note how many features were read now and compare how long it took.

Advanced Task
As a side-example, why not try reading back the census data loaded in example 1?

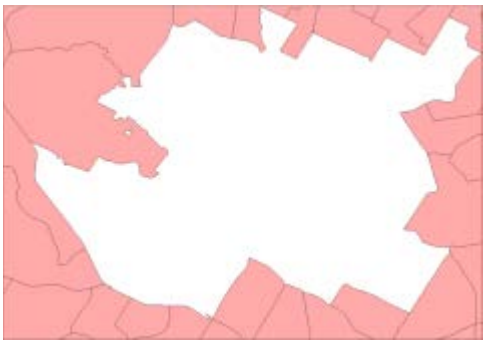
Because it was created with a spatial index you can use a Search Envelope to read it back. Experiment with all of the different parameters and then try to match the following screenshots to the parameters used:

- Minimum X: -97.828
- Minimum Y: 30.217
- Maximum X: -97.681
- Maximum Y: 30.319
- Clip to Search Envelope: No
- Search Envelope Coordinate System: <not set>
- Relationship To Query Feature: ANYINTERACT

Clip: No
Relationship: Any

Clip: Yes
Relationship: Overlap/Intersect

Clip: Yes
Relationship: Any



Clip: Yes
Relationship: Inside

Concatenated Parameters
The problem with the WHERE clause – as with similar parameters – is that it is difficult to get user input and apply it to the clause.

Simply publishing the parameter is not useful because the user would have to enter the full clause (<field> = <value>), when often only the <value> part is required as input.

This is where a concatenated parameter comes in. It is a parameter that is built of a constant string (the <field> part) and a user-defined value (the <value> part).

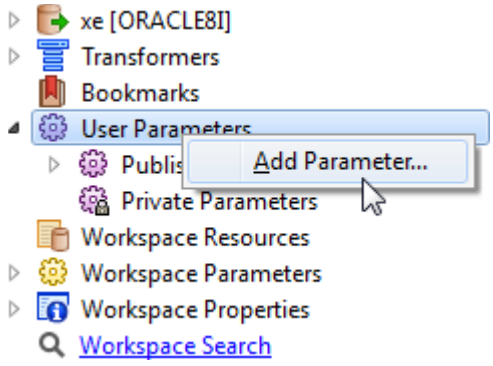
***NB**: Scripted Parameters are a similar tool, but would be used in scenarios where the value needs more complex processing that requires using a Python or Tcl script. For example, a scripted parameter would be useful for manipulating the search envelope parameter values.*

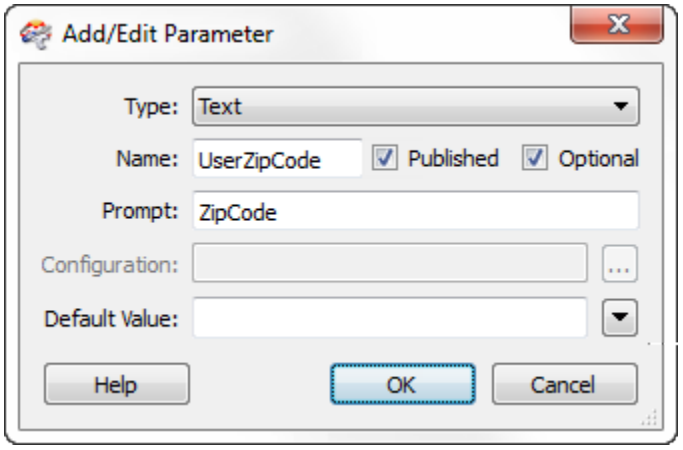
Example 4: Concatenated Parameter	
Scenario	FME user; City of Interopolis, Planning Department
Data	Oracle Spatial Object (Address Point Data)
Overall Goal	Read address data with a user-defined query
Demonstrates	Concatenated parameters for a database
Starting Workspace	C:\FMEData\Workspaces\PathwayManuals\Database4(Oracle)-Begin.fmw
Finished Workspace	C:\FMEData\Workspaces\PathwayManuals\Database4(Oracle)-Complete.fmw

Continue on from the previous example….

1) Start FME Workbench
Start Workbench if necessary and open the workspace from the previous example. Or open
C:\FMEData\Workspaces\PathwayManuals\Database4(Oracle)-Begin.fmw

2) Add Parameter
Right-click on "User Parameters" in the Navigator window and choose **Add Parameter**.

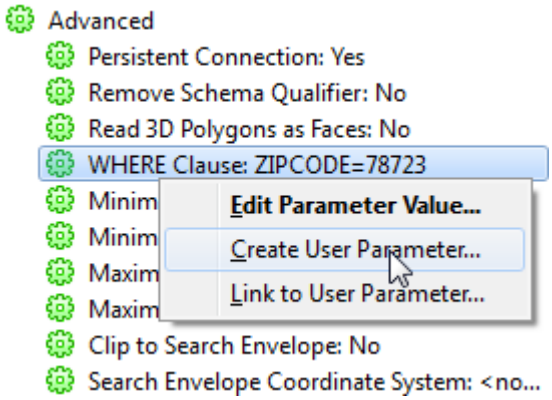




When prompted, choose a parameter of type Text.

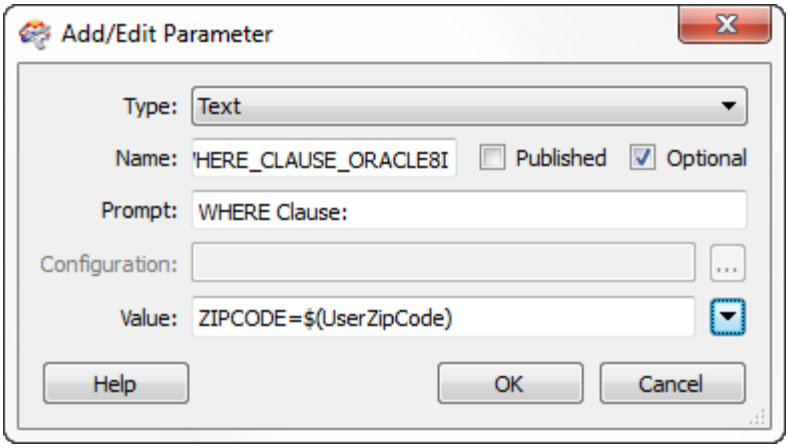
Set the name to UserZipCode and the prompt to ZipCode:

3) Add Parameter
In the Navigator window, right-click the Oracle Reader WHERE Clause parameter and choose Create User Parameter.



This time create the parameter with the following settings:

Type Text
Published Uncheck this flag, so the user is not prompted to set this private parameter
Value `ZIPCODE=$(UserZipCode)`

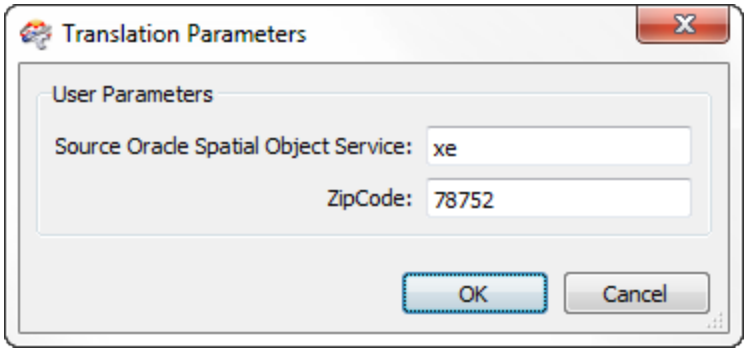


Notice how this published parameter makes a reference to the one previously created (UserZipCode).

4) Run Workspace
Run the workspace using **Prompt and Run Translation**.
When prompted, enter a zipcode into the field provided.

Valid values are:

- 78722
- 78723
- 78724
- 78751
- 78752
- 78753
- 78754



Each different zipcode will change the database WHERE clause and return a different set of data.

Advanced Task
If you have a little time, recreate the *UserZipCode* parameter to be a choice, rather than a text parameter. The choices should reflect the above list of valid zipcodes for Interopolis.

But if you did have to find the list of unique zipcodes in the data, how would you do it?!

Mr. Saif-Investor says…

"Notice that a WHERE clause on the reader applies the same query to ALL tables being read. This can be useful, but also inefficient.

So be aware that there is also a WHERE clause parameter on each feature type, so different queries can be applied to individual tables."

FeatureReader
The *FeatureReader* transformer is one that acts – as the name suggests – as a reader in its own right. *FeatureReader* will read data, from a database, and use incoming features as the basis for a spatial or non-spatial query.

Incoming features are known as **initiators**. Each input feature causes a single query to be carried out through the reader configured in the *FeatureReader*. The query can use the geometry of the incoming feature as a base against which to test a spatial predicate, and the reader returns one or more features as the result of the query.

For example, an incoming line feature (maybe a road) can be used to define the base for an intersection query against linear database features such as rivers or rail. Or a polygon feature could be used as a boundary within which to select point features.

***NB:** This transformer is designed to replace both the OracleQuerier and ArcSDEQuerier.*

The FeatureReader works best when there is a spatial index on the table being read. For that reason it's best to not use this transformer with a View (that does not support spatial indexing) but only with a Materialized View (that does).

Example 5: FeatureReader	
Scenario	FME user; City of Interopolis, Planning Department
Data	Census Data (Oracle Spatial Object) Address Point Data (Oracle Spatial Object)
Overall Goal	Read address data within a user-defined census area
Demonstrates	FeatureReader Transformer
Starting Workspace	None
Finished Workspace	C:\FMEData\Workspaces\PathwayManuals\Database5(Oracle)-Complete.fmw

1) Start FME Workbench
Start Workbench if necessary and begin with a Blank workspace.

2) Add Reader
Add a reader (using **Readers** > **Add Reader**) to read the following dataset:

Reader Format Oracle Spatial Object

Reader Parameters
Database Connection Enter the database connection parameters as before
Table List Select the CENSUSTRACTS table from the first example.

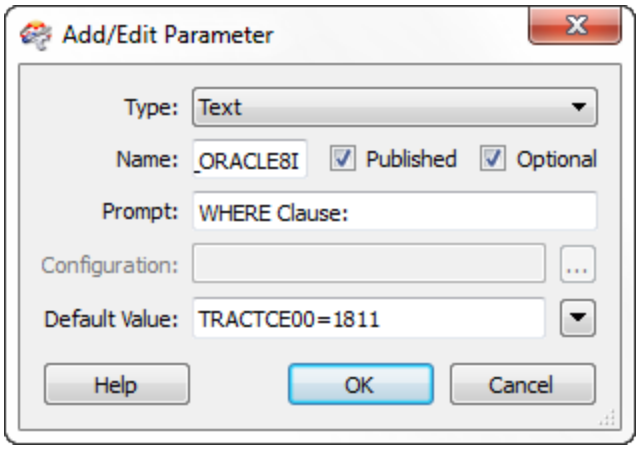
Click **OK**, and **OK** again to close the dialogs and add the new reader.

3) Set WHERE Clause
Turn the Reader WHERE Clause into a Published Parameter.

For the default value enter:

```
TRACTCE00 = 1811
```

Advanced Task: As in the previous example, try to create a concatenated Choice-type parameter with the choices 1811, 1812 and 1813 for the WHERE clause.

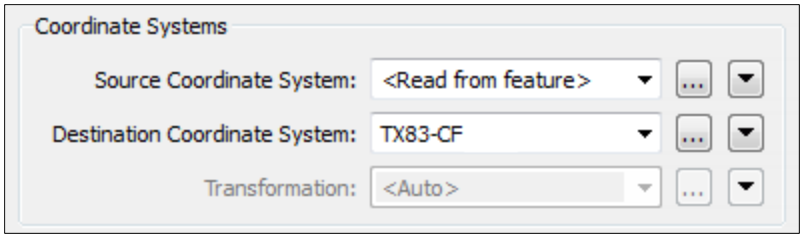


4) Add CsmapReprojector
The Oracle CENSUSTRACT data loaded in an earlier example is in a Lat/Long coordinate system, but the ADDRESS_POINTS data is in TX83-CF.

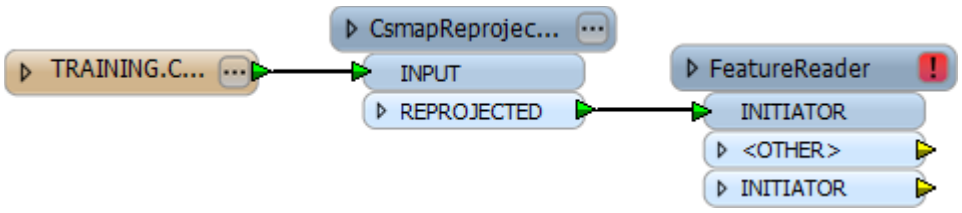
If we wish to use one dataset to act as an Initiator for the other, the CENSUSTRACTS data must be reprojected to match the ADDRESS_POINTS.

Add a *CsmapReprojector* transformer connected to the CENSUSTRACT feature type.
Open the properties dialog. Set:

Source Coordinate System	Read from feature
Destination Coordinate System	TX83-CF



5) Add FeatureReader
Add a *FeatureReader* transformer connected to the CsmapReprojector:REPROJECTED port.



6) Set Parameters
Open the parameters dialog for the *FeatureReader* and set the parameters as follows:

Panel 1:

Reader Format	Oracle Spatial Object
Reader Parameters	
Database Connection	Enter the database connection parameters as before
Table List	Select the ADDRESS_POINTS table

Panel 2:

Feature Types	Query the Feature Types specified on the previous page
WHERE Clause	<none>

A nice feature here is that the Feature Types (tables) to read can be obtained from the contents of an attribute. For example, you could get the list of tables from a published parameter, and extract that into an attribute for use here.

Panel 3:

Spatial Interaction	Select only features that satisfy "INITIATOR geometry… etc
Spatial Test	ORACLE:INSIDE_INITIATOR

This will return all address features INSIDE the Initiator (Census Tract) feature.

Note that the first set of predicates (those prefixed by ORACLE:) will be processed using a query to the underlying database. Those without the prefix are internal predicates that use FME functionality.

Panel 4:

Attribute Handling	Keep result attributes only
Geometry Handling	Keep result geometry only

Click **Finish**. With only one Initiator feature there is no need to use a cache.

7) Add Inspectors and Run Workspace
Add *Inspectors* to the *FeatureReader* output ports.

Run the workspace using **File** > **Prompt and Run Translation**.
When prompted, enter census tract 1811.

The workspace will read addresses from the Oracle database, only where they fall inside this census tract.

Advanced Task
Experiment with a WHERE clause on the *FeatureReader*. For example:

```
LA_UPD_BY = 'name'
```

Be aware that only tracts 0303, 2201, 2112, and 2106 have records with LA_UPD_BY set.

Coordinate System Granularity in Spatial Databases

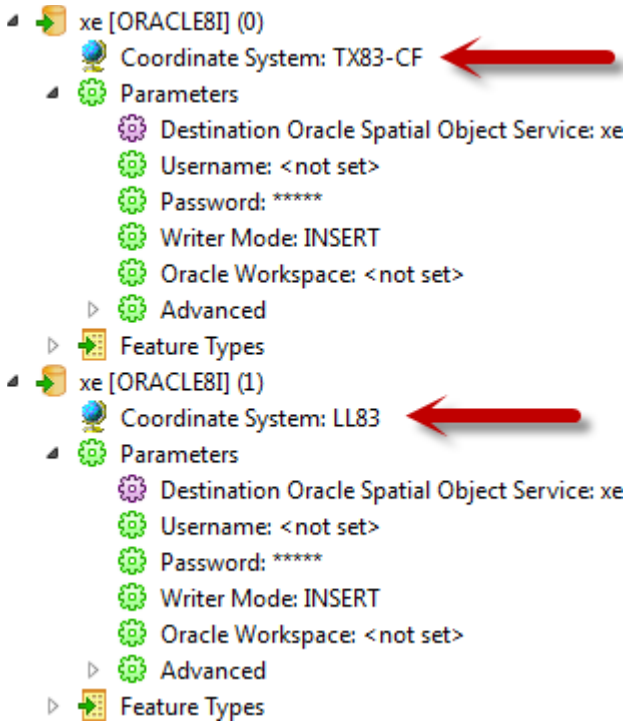
Granularity refers to the level at which different features can be written to different coordinate systems

Some formats of database support improved coordinate system granularity in FME. This means different tables may be written simultaneously, each with a different coordinate system.

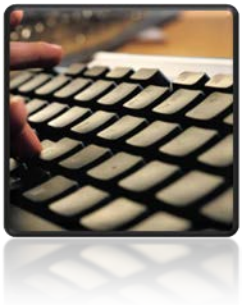
Supported Formats
This functionality is supported in the following database formats:

- Geodatabase and SDE
- SQL Server
- Informix
- Teradata
- IBM DB2

…however it is not yet supported in Oracle (or GeoMedia). This means that each Oracle Writer can only write to a single coordinate system. If you need to write multiple tables, each in a different coordinate system, then you need to use multiple Oracle Writers.



Multiple Geometries



Multiple Geometries are permitted where supported by the database, usually in the form of multiple geometry columns per table

Most databases include the ability to have multiple geometry columns per table, and FME supports this too. However, the table must exist beforehand – FME cannot create multiple geometry tables.

Multiple Geometry Writing
There are two multiple-geometry writing scenarios:

- Reading AND Writing multiple geometries
- Reading single geometry features and converting them to multiple geometries

Reading and Writing Multiple Geometries
In a **Multiple -> Multiple** translation FME handles the reading and writing automatically.

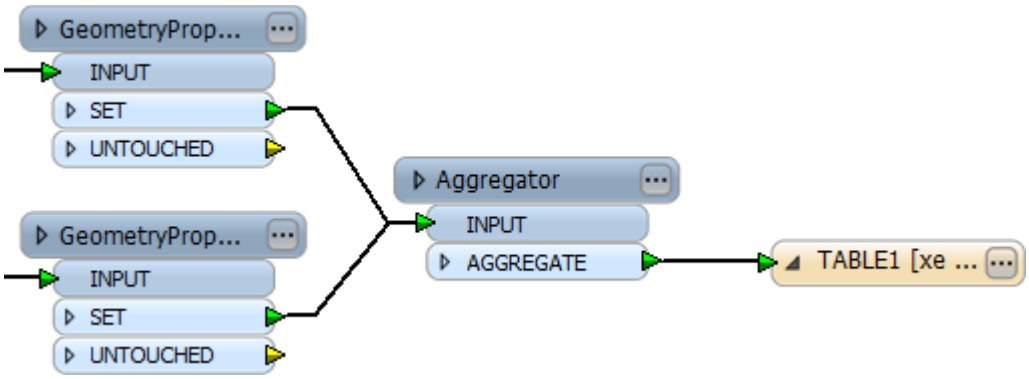
Reading Single and Writing Multiple Geometries
When converting single geometries to multiple, the key is in how to identify two features that are related, and how to assign each of them to the appropriate geometry column.

Because FME doesn't yet handle **Single -> Multiple** translations automatically within Workbench, the setup for each database record to be written must be defined manually. It will be composed of two or more features, each of which contributes its geometry to the final record.

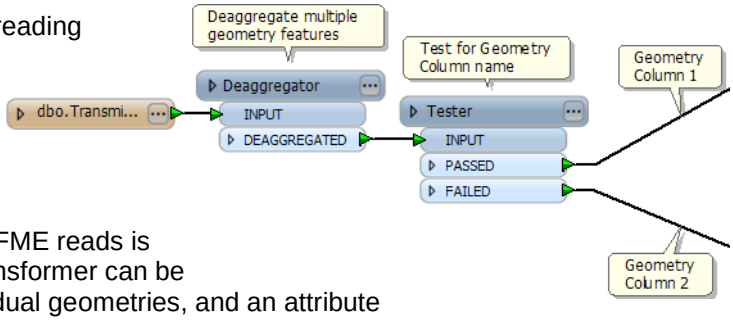
The functionality used to do this involves geometry names and aggregates.

1) A geometry name is applied to each feature with a *GeometryNameSetter* and this geometry name identifies the geometry columns to write to.

2) The features are grouped together as an aggregate—usually with an *Aggregator* transformer—and this identifies which features form a particular database record.

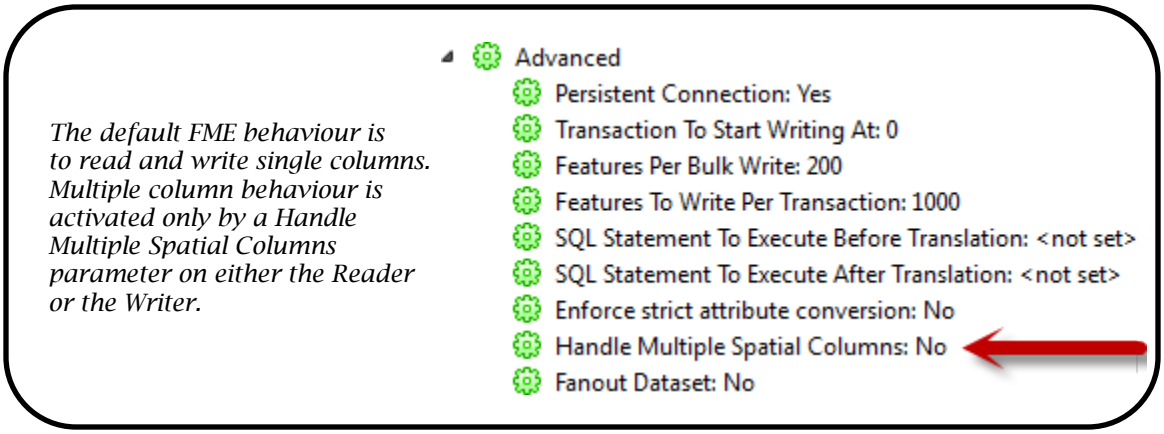


Multiple Geometry Reading
Similar to writing, multiple geometry reading involves aggregates.



Each multiple geometry feature that FME reads is an aggregate. The *Deaggregator* transformer can be used to split up the record into individual geometries, and an attribute (_geometry_name) used to determine which geometry came from which column.

The default FME behaviour is to read and write single columns. Multiple column behaviour is activated only by a Handle Multiple Spatial Columns parameter on either the Reader or the Writer.



Creating Multiple Geometry Tables
FME isn't yet able to create tables with multiple geometry columns. You may either use an existing table with multiple geometry columns, or you can use a SQL script within the FME translation to create one.

The goals of the SQL script are to:

- Check if the table already exists and, if so, drop (delete) it
- Delete any existing metadata records for the table
- Create the new table
- Insert new metadata records for the table

There are two example SQL scripts you can use as examples:

- *C:\FMEData\Resources\Databases\OraclePLSQL.txt*
- *C:\FMEData\Resources\Databases\OracleSimpleSQL.txt*

PL/SQL Script Example:
An example PL/SQL script (OraclePLSQL.txt) is as follows:

```
FME_SQL_DELIMITER !
DECLARE
   cnt NUMBER;
BEGIN
   SELECT COUNT(*)
   INTO cnt
   FROM ALL_TABLES
   WHERE TABLE_NAME = 'CITY_PARKS' AND OWNER = 'TRAINING';
IF ( cnt > 0 )
   THEN
       EXECUTE IMMEDIATE 'DROP TABLE CITY_PARKS';
       EXECUTE IMMEDIATE 'DELETE FROM USER_SDO_GEOM_METADATA WHERE
       TABLE_NAME=''CITY_PARKS'' ';
END IF;
END;
!
CREATE table CITY_PARKS (
      "PARK_ID"            INTEGER,
      "NAME"               CHAR(64),
      "NAME_ALT"           CHAR(64),
      "POINTS"      SDO_GEOMETRY,
      "POLYGONS"    SDO_GEOMETRY
   )!
INSERT into USER_SDO_GEOM_METADATA (table_name,column_name,diminfo,srid) VALUES
('CITY_PARKS','POINTS',
sdo_dim_array(sdo_dim_element('X',3000000,10000000,0.005),sdo_dim_element('Y',32
00000,10100000,0.005)), 2277)!
INSERT into USER_SDO_GEOM_METADATA (table_name,column_name,diminfo,srid) VALUES
('CITY_PARKS','POLYGONS',
sdo_dim_array(sdo_dim_element('X',3000000,10000000,0.005),sdo_dim_element('Y',32
00000,10100000,0.005)), 2277)!
```

Notes:
The script is divided into four sections:

- An FME keyword to define the SQL delimiter
- A PL/SQL script to drop an existing table and delete its metadata
- A SQL command to create the CITY_PARKS table
- Two insert statements to create metadata

- The FME_SQL_DELIMITER keyword specifies a character to separate multiple SQL statements. Usually you would use a semi-colon (;) but here we have a section of PL/SQL that already uses semi-colon delimiters within itself, and so this script uses the exclamation mark (!) instead.

- The PL/SQL section is used in order to check if the table exists before dropping it. It is equivalent to – though not as elegant as – the SQL Server “IF EXISTS” function.

- In the line that deletes existing metadata records, CITY_PARKS is bracketed by two single quote characters (`''CITY_PARKS''`). This is a requirement of the FME parser.

- There are two lines to create metadata (one for each geometry column). Without these records FME would not be able to find the table to read it back!

Commits are performed automatically, and do not need to be included.

Alternate Script:
An alternative to the above (OracleSimpleSQL.txt) script is as follows:

```
FME_SQL_DELIMITER ;
-DROP TABLE CITY_PARKS;
-DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME='CITY_PARKS';
CREATE table CITY_PARKS (
      "PARK_ID"            INTEGER,
      "NAME"               CHAR(64),
      "NAME_ALT"           CHAR(64),
      "POINTS"      SDO_GEOMETRY,
      "POLYGONS"    SDO_GEOMETRY
   );
INSERT into USER_SDO_GEOM_METADATA (table_name,column_name,diminfo,srid) VALUES
('CITY_PARKS','POINTS',
sdo_dim_array(sdo_dim_element('X',3000000,10000000,0.005),sdo_dim_element('Y',32
00000,10100000,0.005)), 2277);
INSERT into USER_SDO_GEOM_METADATA (table_name,column_name,diminfo,srid) VALUES
('CITY_PARKS','POLYGONS',
sdo_dim_array(sdo_dim_element('X',3000000,10000000,0.005),sdo_dim_element('Y',32
00000,10100000,0.005)), 2277);
```

Notes:
The difference here is, rather than trap errors in the script, a specific FME device is used to ignore them. Notice the hyphen character that precedes the DROP and DELETE commands. This prompts FME to ignore any errors from these commands, such as would occur when trying to drop a table that does not exist.

Pros	A shorter, simpler script
Cons	Non-standard SQL

Executing SQL Scripts
There are various ways you can execute SQL Scripts within FME.

The next exercise uses *SQL Statement to Execute Before Translation* which is one of the database reader and writer parameters.

Later sections will introduce you to the *SQLCreator* and the *SQLExecutor*.

Example 6: Multiple Geometry Writing	
Scenario	FME user; City of Interopolis, Planning Department
Data	City Parks (MapInfo TAB, Oracle Spatial Object)
Overall Goal	Write to multiple geometry columns
Demonstrates	Multiple Geometry Writing
Starting Workspace	None
Finished Workspace	C:\FMEData\Workspaces\PathwayManuals\Database6(Oracle)-Complete.fmw

In this example, FME will be used to create multiple geometries where the two geometries are different representations (point and polygon) of the same park objects.

A workspace will then be created to read the data back, and filter it (either points or polygons) depending on the scale required for the output.

1) Start FME Workbench
Start FME Workbench, and open the Generate Workspace dialog.
Set up a translation as follows:

Reader Format MapInfo TAB (MFAL)
Reader Dataset *C:\FMEData\Data\Parks\city_parks.tab*

Writer Format Oracle Spatial Object

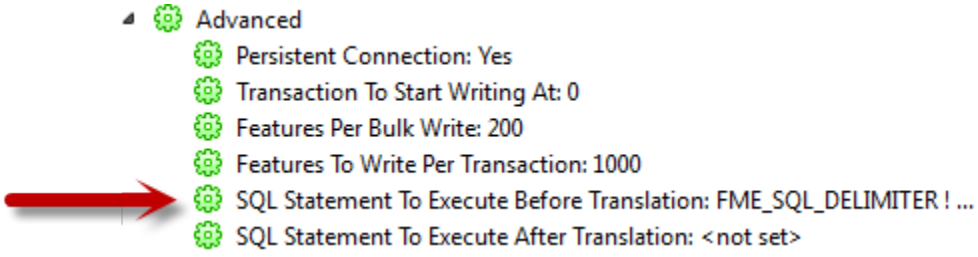
Writer Parameters
Database Connection Enter the database connection parameters

Click **OK**, and then **OK** again, to create the workspace.

2) SQL Statement to Execute Before Translation
Because FME isn't yet able to create tables with multiple geometry columns you will use a SQL script to create one. There are two example SQL scripts you can use:

C:\FMEData\Resources\Database\OraclePLSQL.txt
C:\FMEData\Resources\Database\OracleSimpleSQL.txt

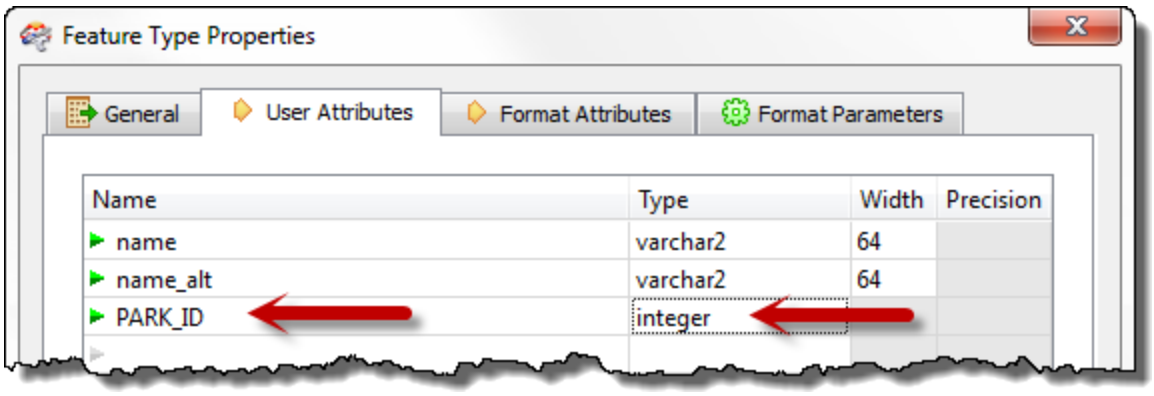
Locate and double-click the writer parameter 'SQL Statement to Execute Before Translation'.



Open one of the example SQL scripts from the resources folder in a text editor and cop/ypaste the contents into the FME SQL edit dialog.

3) Edit Schema
Notice that the new CITY_PARKS table (as defined in the SQL script) has columns *NAME*, *NAME_ALT*, and *PARK_ID* the latter of which is not defined in the workspace.

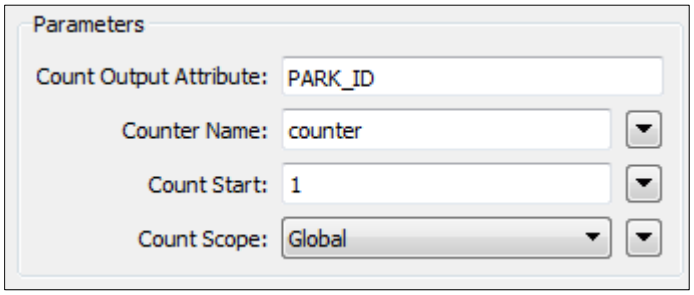
Edit the writer feature type and add the new *PARK_ID* field, with **integer** type.
Remove the unnecessary fields ROTATION, HEIGHT and TEXTSTRING



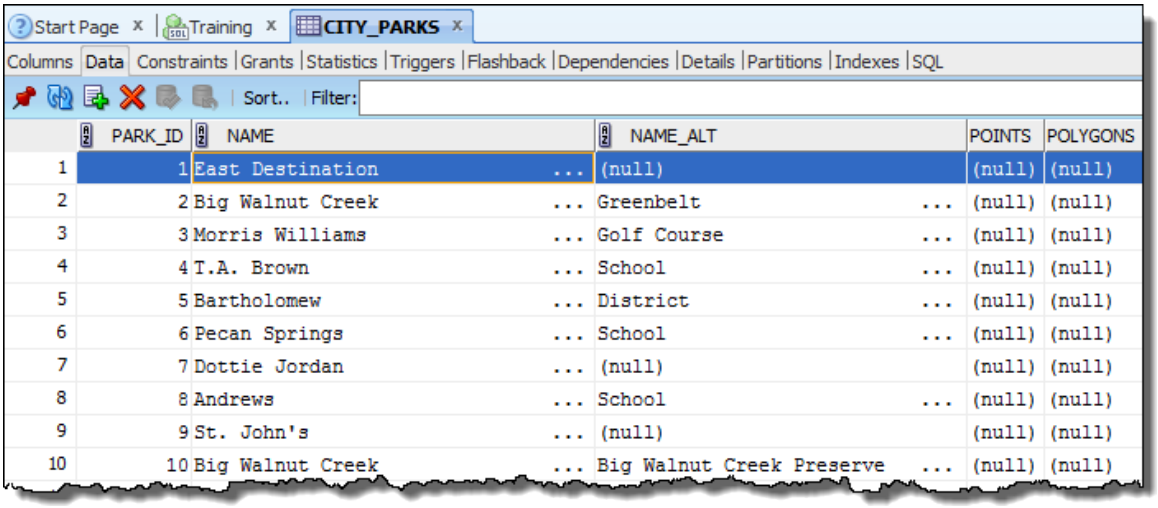
You may wish to rename the attributes name and name_alt to NAME and NAME_ALT. The Oracle writer will automatically do this to the output anyway, so this is not compulsory at this point.

4) Add Counter
To create *PARK_ID*, add a *Counter* transformer.

Open the properties dialog and set Count Output Attribute to *PARK_ID*
Change the Count Start parameter to start at 1 (instead of 0 because IDs should be non-zero)



If you run the workspace now you will get a table with attributes, but two null geometry fields:

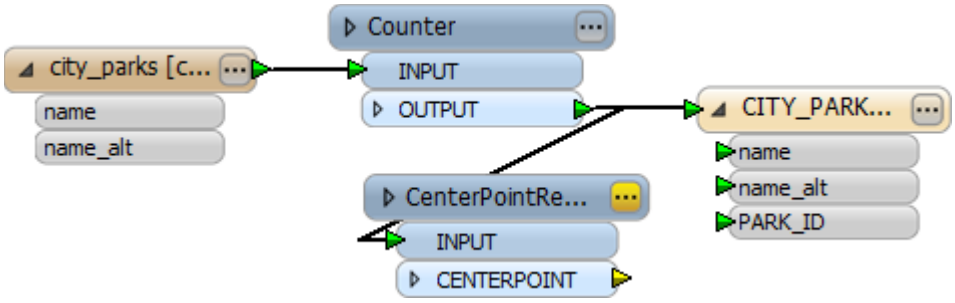


	PARK_ID	NAME	NAME_ALT	POINTS	POLYGONS
1	1	East Destination	(null)	(null)	(null)
2	2	Big Walnut Creek	Greenbelt	(null)	(null)
3	3	Morris Williams	Golf Course	(null)	(null)
4	4	T.A. Brown	School	(null)	(null)
5	5	Bartholomew	District	(null)	(null)
6	6	Pecan Springs	School	(null)	(null)
7	7	Dottie Jordan	(null)	(null)	(null)
8	8	Andrews	School	(null)	(null)
9	9	St. John's	(null)	(null)	(null)
10	10	Big Walnut Creek	Big Walnut Creek Preserve	(null)	(null)

5) Add CenterPointReplacer
Now to create the multiple geometries: The park features are originally polygons; to create points insert a *CenterPointReplacer* transformer, connected to a second output stream from the *Counter*.

The transformer has no parameters (except name) to worry about.

The workspace will now look something like this:



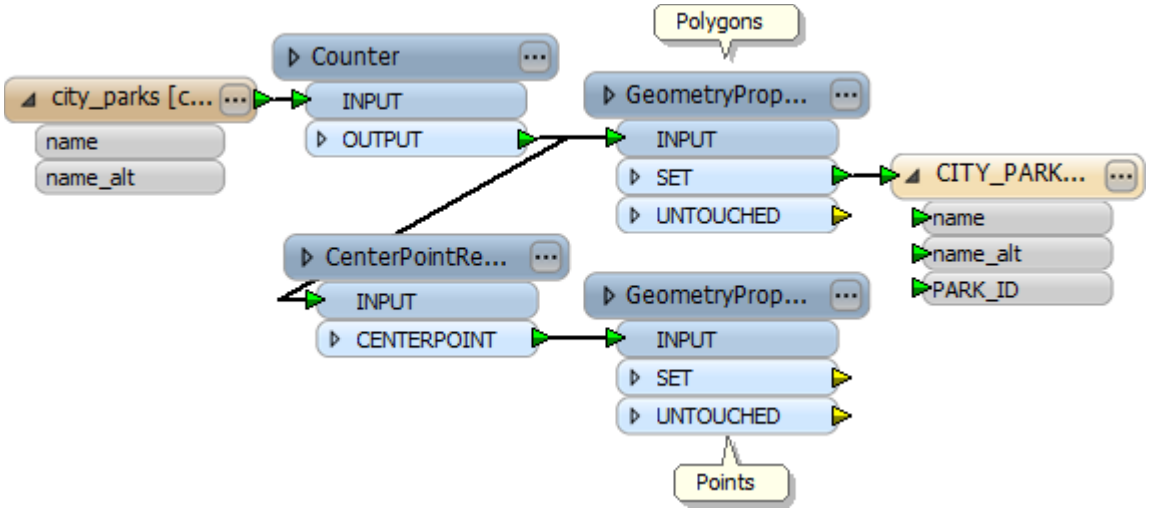
6) Add GeometryPropertySetter
Now there are two sets of geometries, and they must be given different names. The names should match the geometry column names in the database: POINTS and POLYGONS

Place two *GeometryPropertySetter* transformers. Open the parameters dialog for the first and set the parameters:

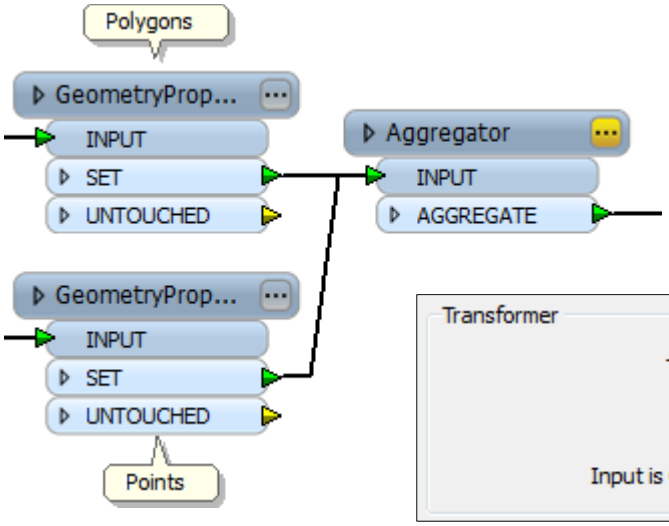
Property to Set Geometry Name
Geometry Name POLYGONS

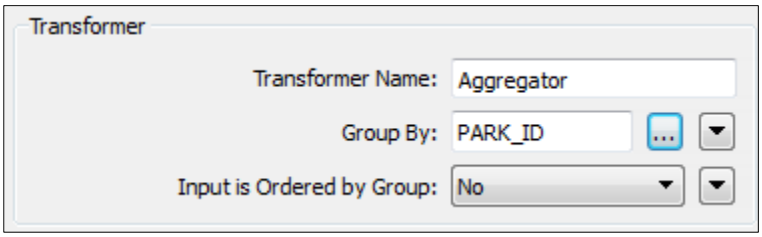
Repeat the process for the second transformer, except name the geometry as POINTS

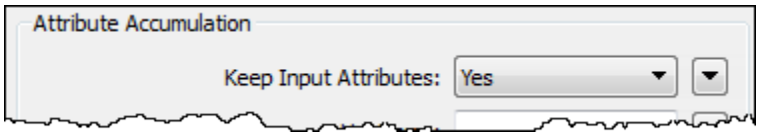
The workspace will now look like this:

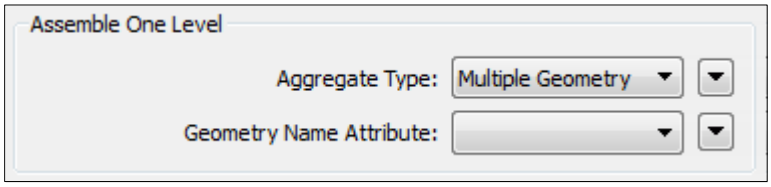


7) Add Aggregator
The next step is to identify the matching features. This is done with an *Aggregator* transformer. Place an *Aggregator* transformer and connect both streams of data to it.









In the parameters dialog set:

Group By = *PARK_ID*;

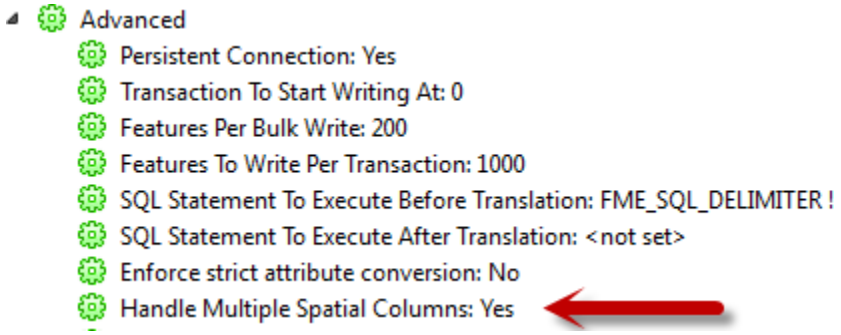
This is how the two sets of features are paired off and aggregated together.

Keep Input Attributes = *Yes*
This keeps all source attributes.

Aggregate Type = *Multiple Geometry*
This tags the aggregate features as actually being multiple geometries.

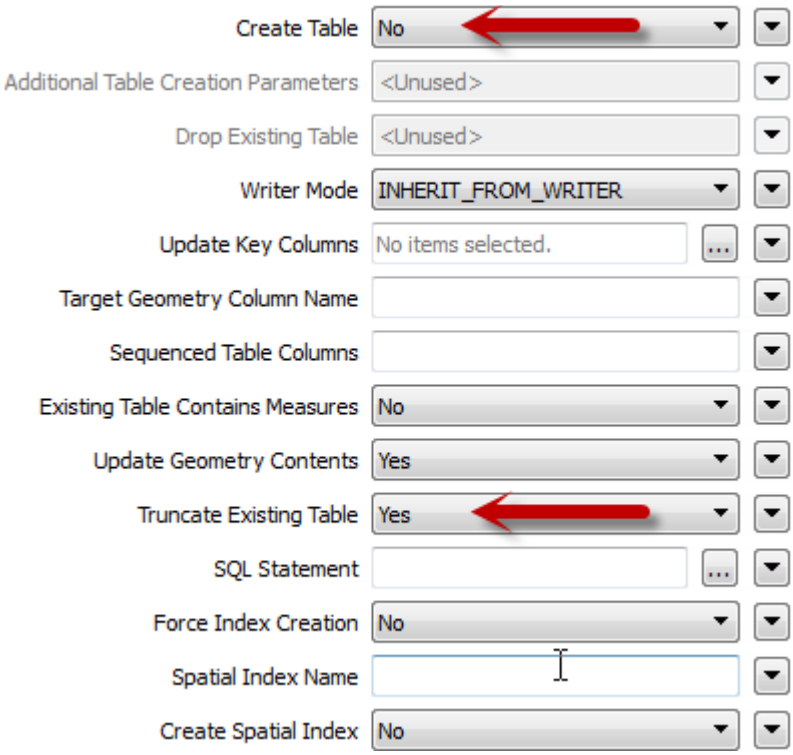
If you use a different method to merge features such as a transformer (like the FeatureMerger) or the source data is already in this form, then you can use the MultipleGeometrySetter transformer to tag the aggregate as representing multiple geometries.

8) Set Multiple Geometry Parameter
In the Navigator window, locate the writer parameter *Handle Multiple Spatial Columns*, and set it to Yes.



9) Set Create Table to No
As a final task, open the writer Feature Type properties. Set these parameters:

Create Table = *No.*
Truncate Existing Table = *Yes.*



10) Save and Run Workspace
Save the workspace and then run it. The City Park features will be written to Oracle with multiple geometries.

PARK_ID	NAME	NAME_ALT	POINTS	POLYGONS
12	Big Wa...	Chimney ...	[MDSYS.SDO_GEOMETRY]	[MDSYS.SDO_GEOMETRY]
3	Morris...	Golf Cou...	[MDSYS.SDO_GEOMETRY]	[MDSYS.SDO_GEOMETRY]
6	Pecan ...	School ...	[MDSYS.SDO_GEOMETRY]	[MDSYS.SDO_GEOMETRY]
8	Andrew...	School ...	[MDSYS.SDO_GEOMETRY]	[MDSYS.SDO_GEOMETRY]
10	Big Wa...	Big Waln...	[MDSYS.SDO_GEOMETRY]	[MDSYS.SDO_GEOMETRY]

The FME Data Inspector does not yet display multiple geometry columns. If you view the results of this translation in the Data Inspector you will only see one of the two geometries you have written, either points or polygons, depending on which geometry column is first in the database.

Example 7: Multiple Geometry Reading	
Scenario	FME user; City of Interopolis, Planning Department
Data	City Parks (Oracle Spatial Object, KML)
Overall Goal	Read and filter multiple geometry columns
Demonstrates	Multiple Geometry Reading
Starting Workspace	Create from scratch
Finished Workspace	C:\FMEData\Workspaces\PathwayManuals\Database7(Oracle)-Complete.fmw

In this example, FME will be used to read the previous data back, and filter it (either points or polygons) depending on the scale required for the output.

NB: *Workbench will be used to read the data back, as the FME Universal Viewer does not yet support the selection of geometry column.*

1) Start FME Workbench
Start FME Workbench, and open the Generate Workspace dialog.
Set up a translation as follows:

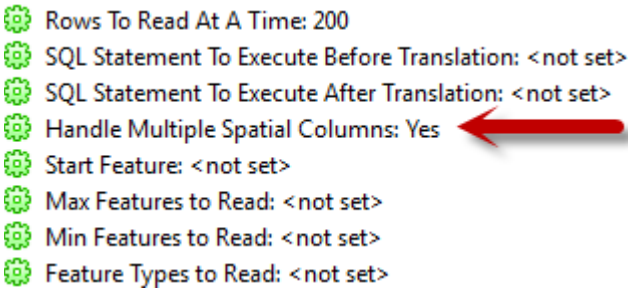
Reader Format Oracle Spatial Object

Reader Parameters
Database Connection Enter the database connection parameters as before
Table List Select CITY_PARKS as the table to read

Writer Format Google Earth KML
Writer Dataset *C:\FMEData\Output\DemoOutput\Parks.kml*

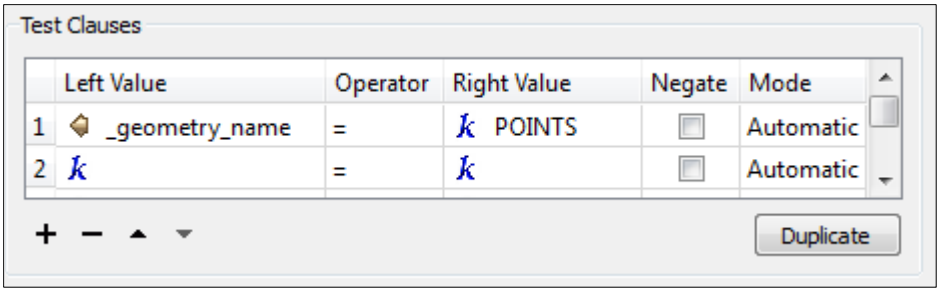
Click **OK** to create the workspace.

2) Set Multiple Geometry Parameter
In the Navigator window, set the reader parameter **Handle Multiple Spatial Columns** = *Yes.*

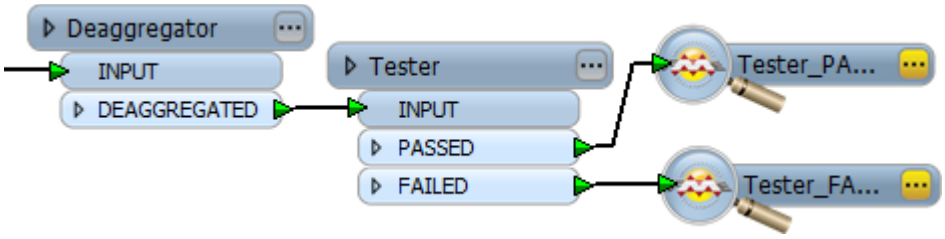


3) Add Deaggregator
Add a *Deaggregator* transformer. This will divide the data into its two geometry types. This dividing is based on the Geometry Name Attribute, by default called *_geometry_name*.

4) Add Tester
Add a *Tester* transformer. Set up a test to check if the geometry name attribute (*_geometry_name*) has a value of POINTS



To prove that the workspace is doing what is expected so far, disconnect the output feature type so that no data is written, and attach an *Inspector* transformer to each *Tester* output port



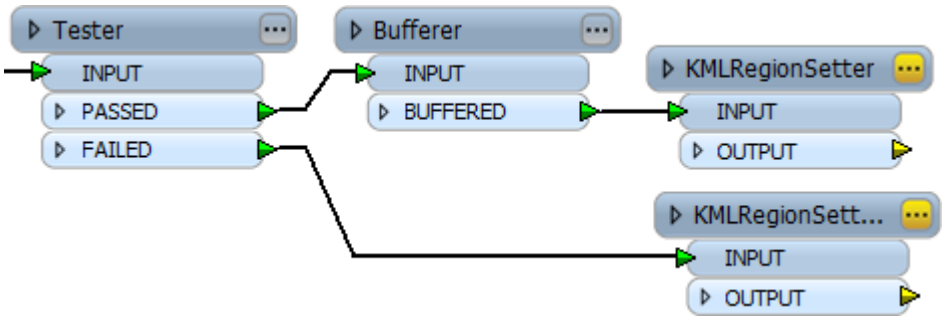
Run the translation. Points and polygons should get separated out.

Optional: Define the KML Output Styles
Because it's no longer database related, you can skip the rest of the steps if you desire. Otherwise, continue work to make the KML display the different features at different zoom levels.

5) Delete Inspectors. Add Bufferer.
Delete the *Inspectors* that were attached to the *Tester* transformer.

Add a *Bufferer* transformer to *Tester*:PASSED to buffer the point features by 1000ft. This makes it easier to turn the point features on/off as we zoom.

6) Add KMLRegionSetters
Add two *KMLRegionSetter* transformers; one for the buffered points, one for the polygons.



The parameters for the buffered POINTS version should be set to:

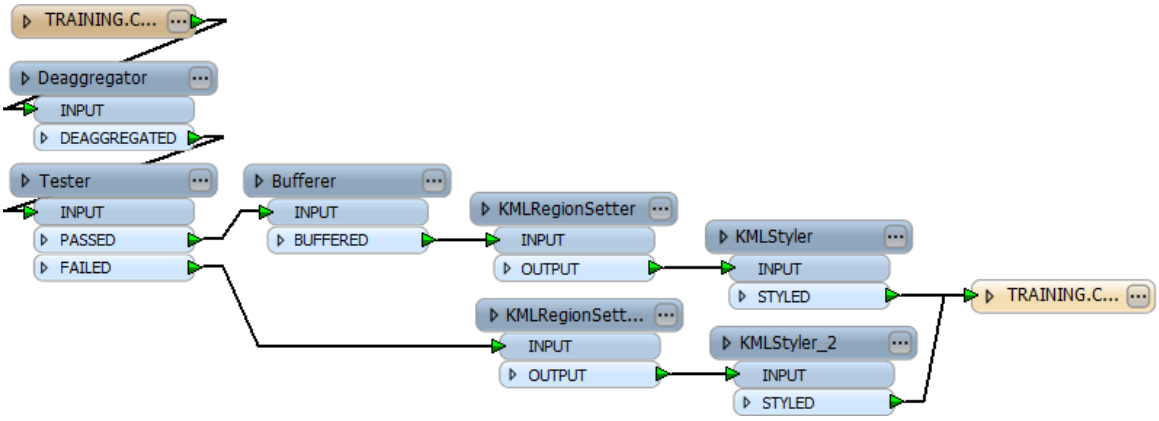
Bounding Box: Calculate: Yes – 2D
Minimum Display Size 10
Maximum Display Size 30

The POLYGONS version should be set to:

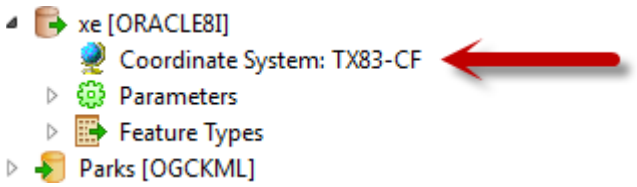
Bounding Box: Calculate: Yes – 2D
Minimum Display Size 50
Maximum Display Size -1

7) Add KMLStylers
Add two *KMLStyler* transformers; one for the buffered POINTS, one for the POLYGONS.
Assign the buffered POINTS a red fill color, and the POLYGONS green.
Connect the *KMLStylers* to the writer.

The workspace will now look like this:



8) Set Coordinate System
Set the source coordinate system for the reader to TX83-CF



9) Save and Run Workspace
Save the workspace and then run it.
If you like, open the output in Google Earth (it may be slow on a Virtual Machine)

The POINT geometry will show when the view is zoomed out; as you zoom into the view, the points will disappear to be replaced by the actual polygons.

Database Transformers

Besides the FeatureReader there are a number of database-related transformers for submitting SQL statements directly to the database.

There are several methods to submit SQL statements from FME to a database.

Firstly, SQL statements can be entered into parameters to run before and after a translation. These were used in the previous section.

Secondly, there is a Reader WHERE clause, a Feature Type WHERE clause, and a Feature Type SELECT Statement parameter.

Thirdly, there are SQL-related transformers such as the *FeatureReader*.

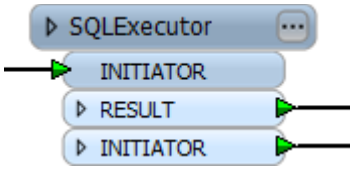
Such statements might be used to:

- Create, drop, modify or truncate a database table
- Carry out a database join
- Drop constraints prior to data loading
- Any other function that is usually carried out using a SQL statement.

This section focuses on the use of transformers to run SQL statements.

SQLExecutor

The *SQLExecutor* is a transformer for executing SQL statements against a database. Each incoming feature initiates the SQL statement that has been defined.

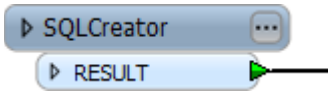


If the SQL is a query, and if features are returned from the database, those features form the output from the transformer. One feature will be output for each row of the results.

This transformer exposes result attributes and does not need to be followed by an *AttributeExposer*. The *SQLExecutor* can also return geometry columns if requested.

SQLCreator

The *SQLCreator* transformer is similar to the *SQLExecutor*, but does not rely on incoming features to initiate it. Therefore it has no INPUT port and the statement is executed once only.



The *SQLCreator* is executed before features are read from any Readers, so this is an alternative way you could have executed the table creation SQL used in an earlier example.

Like the *SQLExecutor*, there will be a feature output for each row of the results.

Uses
SQLExecutor can also be used for many things, including obtaining a foreign key value and:

Working with Sequence Numbers
A sequence number is a field for automatic numbering of features. The syntax <sequenceName>.NextVal is used (in SQL) to interrogate the database and return the next available sequence number.

If you wish to handle sequences manually in FME (rather than through a Reader/Writer) you can use a SQL transformer to read the next sequence number like so:

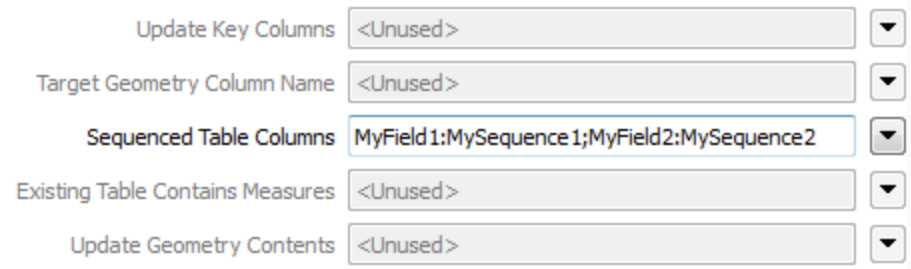
```
select MyField1.NextVal as MYFIELD from MyTable
```

…or to write a sequence ID like this:

```
insert into MyTable (MyField1, MyField2)
values (MySequence1.NextVal, MySequence2.NextVal)
```

When writing data through a Writer, an Oracle sequence number can also be specified in the Feature Type Properties dialog. The syntax is:

<ColumnName>:<SequenceName>;<ColumnName>:<SequenceName>;etc

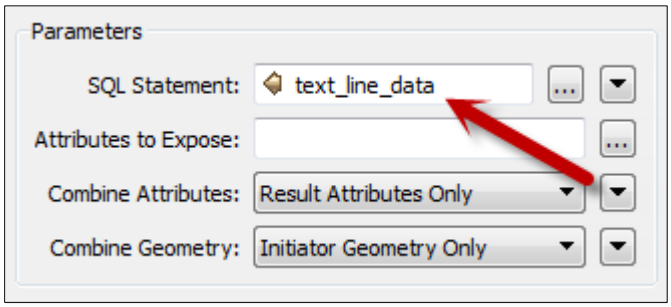


Carrying out a Join
The *SQLExecutor* can carry out multiple table joins very efficiently, for example:

```
SELECT parcel.*, Landuse.LANDUSE, Landuse.USES FROM parcel
      INNER JOIN Landuse ON (parcel.LULU = Landuse.Code)
      WHERE PARCEL_ID = '@Value(_key_field)';
```

This would require two *Joiner* transformers but can be done with a single *SQLExecutor*

Executing a file of SQL Statements
Because the transformer allows the SQL to come from an attribute, you can read a file of SQL statements into an attribute (with the Text File Reader) and then run them through the SQLExecutor.



Example 8: SQLExecutor	
Scenario	FME user; City of Interopolis, Planning Department
Data	Address Data (Oracle Spatial) Emergency Facilities (Access Database)
Overall Goal	Carry out a join using the SQLExecutor
Demonstrates	SQLExecutor transformer
Starting Workspace	Create from scratch
Finished Workspace	C:\FMEData\Workspaces\PathwayManuals\Database8(Oracle)-Complete.fmw

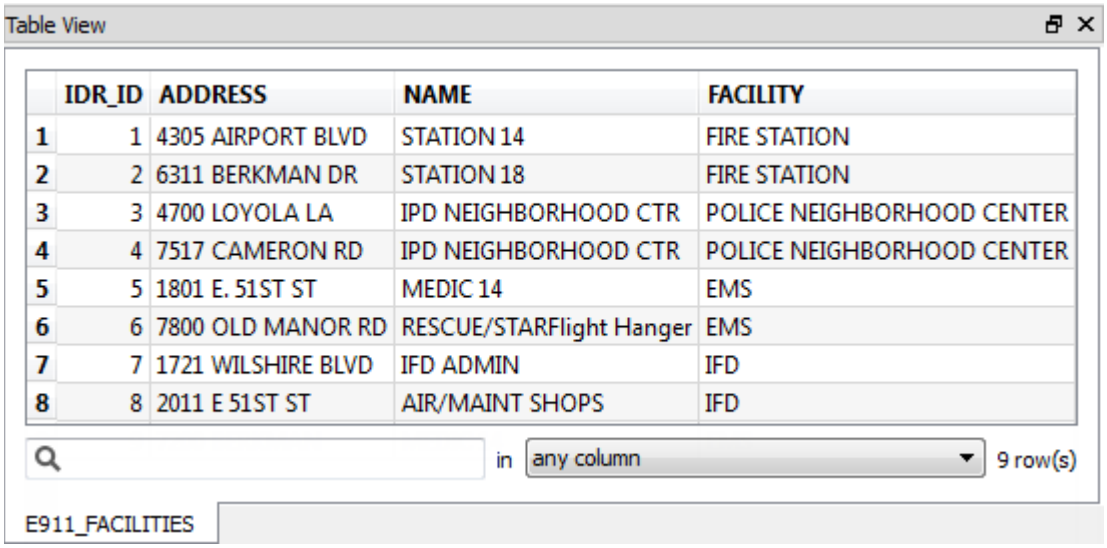
The city has a database of emergency facilities (in Microsoft Access format).

In this example, FME will be used to update an Oracle address database, to flag addresses that are an emergency facility. This will be done by using a SQLExecutor transformer to do a database join.

1) Inspect Source Data
Use the FME Data Inspector to open the source file:

Reader Format Microsoft Access
Reader Dataset *C:\FMEData\Data\Emergency\e911_facilities.mdb*

Inspect the text records in the Table View window.



Notice that one of the available fields is an address. It is this field that will be used to create a join between the Access and Oracle tables.

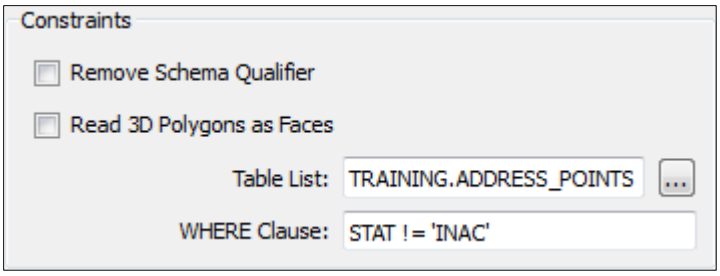
2) Start FME Workbench
Start Workbench if necessary and begin with a blank workspace.

3) Add Reader
Add a Reader using **Readers** > **Add Reader**. Set it up as follows:

Reader Format	Oracle Spatial Object
Reader Parameters	
Database Connection	Enter the database connection parameters as before
Table List	ADDRESS_POINTS

Also, because we don't want to work with inactive addresses, set a WHERE Clause:

```
STAT != 'INAC'
```



Click **OK**, and **OK** again to close the dialogs and add the new reader.

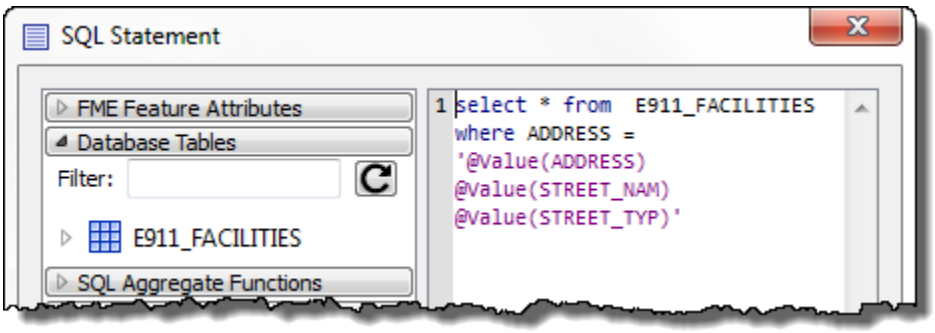
Connect an *Inspector* transformer to the Reader Feature Type and run the workspace.
If the reader and WHERE clause are functioning correctly, you should read 11,723 point features.

4) Add SQLExecutor
Add a *SQLExecutor* transformer to the workspace.
Open the *SQLExecutor* parameters dialog. Set up the parameters as follows:

Reader Format	Microsoft Access
Reader Dataset	*C:\FMEData\Data\Emergency\e911_facilities.mdb*
SQL Statement	`select * from E911_FACILITIES where ADDRESS = '@Value(ADDRESS) @Value(STREET_NAM) @Value(STREET_TYP)'`
Attributes to Expose	NAME
Combine Attributes	Keep Initiator Attributes if Conflict

The easiest method to create the SQL Statement is to use the builder dialog.

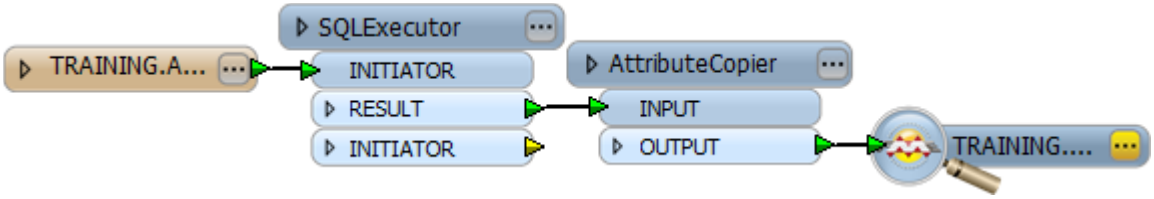
This dialog has shortcuts to database tables and FME attributes.



NB[1]: There needs to be space characters between the three attributes in the SQL Statement.

NB[2]: To populate Attributes to Expose from the SQL Query you will first need to set the SQL statement without the WHERE clause, and only add the WHERE clause once the list is populated.

5) Add AttributeCopier
Add an *AttributeCopier* transformer to copy *NAME* to *COMMENT_*
This will populate the name of the emergency facility in the comments field of the address table.



6) Save and Run Workspace
Save the workspace and then run it.

Inspect the output to confirm the COMMENT_ field
now includes the contents of the emergency data NAME attribute.

	ADDRESS	STREET_NAM	STREET_TYP	COMMENT_
1	4305	AIRPORT	BLVD	STATION 14
2	1721	WILSHIRE	BLVD	IFD ADMIN
3	6311	BERKMAN	DR	STATION 18
4	7517	CAMERON	RD	IPD NEIGHBORHOOD CTR
5	7800	OLD MANOR	RD	RESCUE/STARFlight Hanger

in any column 5 row(s)

SQLExecutor_INITIATOR | TRAINING.ADDRESS_POINTS

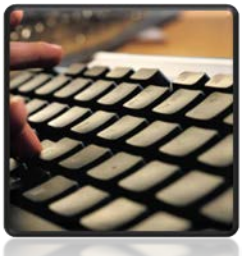
Advanced Task
Now this data could be written back to the address database table – using update mode and PRIMARYINDEX as a key – to flag database records that are also emergency facilities.

To do this you would need to:

- Add an Oracle Writer
- Import the Address Points table schema (or duplicate the Reader Feature Type)
- Set the Writer mode to UPDATE
- Set PRIMARYINDEX as the key field
- Set the attribute LA_UPD_BY to your own name.

Why not give this a try, as an advanced task.

Transactions

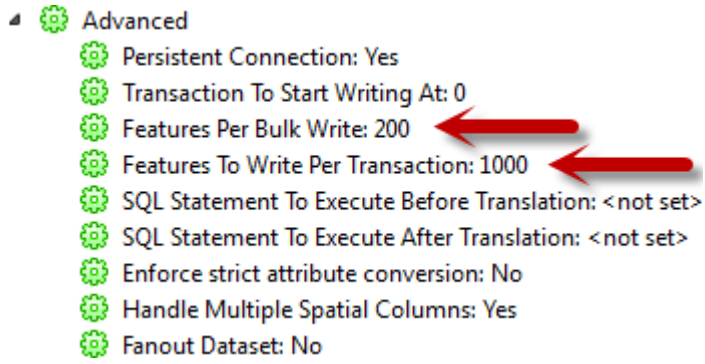


Transactions are an important piece of functionality, with several different uses, and are controlled by a number of parameters.

Transactions and Performance
One purpose of Writer transactions is to control performance. There are two parameters that are of use here: *Features to Bulk Write* and *Transaction Interval*.

Features to Bulk Write
In FME, features sent to a database writer are cached in memory until this number of features is reached, and then sent to the database. This is what is known in FME as CHUNK SIZE.

Transaction Interval
Transaction Interval specifies the number of features sent to the database before a **commit** is issued. Committing means data written to the database is confirmed and made permanent.



In this screenshot the Oracle Writer writes 200 features at a time (the Chunk Size) to the database. When 1000 features have been written, then the data is committed to the database.

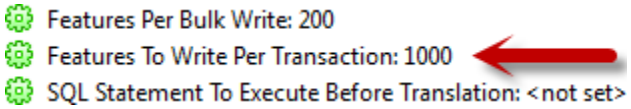
Increasing the interval of these two parameters can help performance because it takes fewer write operations and fewer database transactions to actually load a dataset.

However, after a certain point the amount of features cached in memory – either in FME or the database – is such that system resources become low and performance is adversely affected.

Any data written by the SQLExecutor is NOT considered to be part of the same transaction as that written by a writer.

Some database writers also support a format attribute called fme_db_transaction that can be used to control commits and rollbacks at the feature level. See the Readers and Writers manual for more documentation on this functionality.

Transactions and Recovery
A second use of transactions is to handle data loading failures.



In this screenshot 1,000 features are written to the database before they are committed.

If feature number 1,005 caused the process to fail, then the first 1,000 features have already been committed and are safe in the database. Only 5 features will be rolled-back.

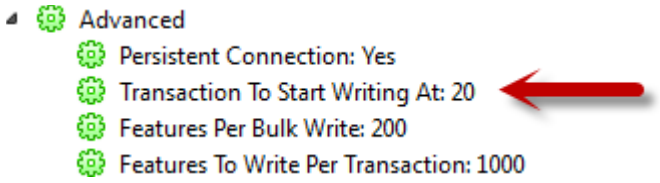
Therefore, for purposes of data recovery, it makes sense to decrease the interval of this parameter, because any failure will cause fewer features to be rolled-back.

Setting this parameter to 1 means that every single feature is committed as it is written. It means that only 1 feature can ever be rolled back, although performance can be adversely affected by the volume of transactions.

Setting this parameter to a very high number (greater than the number of features in the dataset) means that all features must be written before any are committed. This is useful where the user wishes to commit no data unless it is all correct. Correct setting of "Features Per Bulk Write" will ensure that performance is not impacted.

Transaction to Start Writing At
Whenever a translation fails mid-write, FME will report in the log window how many transactions had been applied up to that point. Features written but not yet committed will be rolled back.

If the problem can be easily fixed, then the translation can be re-run, but this time specifying which transaction to start at. This is done using the parameter "Transaction To Start Writing At".



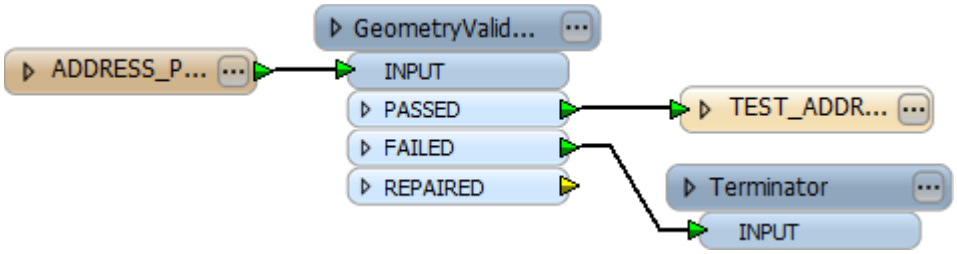
In this example the previous translation failed during transaction 20. The user has fixed the problem feature and is re-running the translation from that point onwards.

The process is then quicker because the entire dataset isn't being reloaded – only data after the failure point.

If the "Transaction to Start Writing At" parameter is set to zero – the default – then all data is written as usual.

Forcing a Transaction Failure

A translation can be made to fail mid-write by using a Terminator transformer to abort the translation.



In this case the user is testing data with the GeometryValidator transformer. If the data is found to be invalid then it is sent to a Terminator transformer.

When a translation is terminated, then FME reports which transaction to restart from:

```
================================================================================
Oracle Writer: Translation aborted -- rerun specifying "ORACLE8I_1_START_TRANSACTION 10"
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Feature output statistics for `ORACLE8I' writer using keyword `ORACLE8I_1':
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                              Features Written
================================================================================
Total Features Written                                                      105
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Creative Feature Writing

Sometimes, for complex scenarios, it's necessary to be creative in how features are written with an Oracle Writer.

There are a few writing scenarios that require careful consideration and an aspect of creativity.

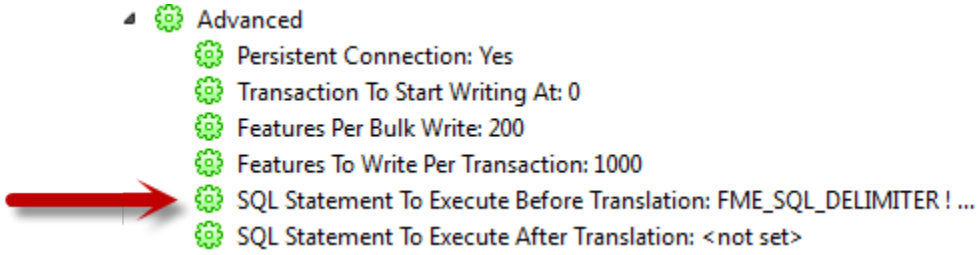
Parent/Child Tables
This scenario is where two tables are connected by a foreign key value. The table being referenced is called the Parent Table, whereas the table referencing it is called the Child Table.

This scenario is complicated because there must be referential integrity between the two tables; i.e. each foreign key in a child table must be a match to a primary key in the parent table. Sometimes this integrity is enforced by a Dependency (or Constraint) that is defined in the database.

So, when it comes to writing data with FME, it's important to ensure that the parent table is loaded first. Child records that refer to a non-existent primary key will be rejected by the Writer.

There are a couple of solutions that can be used to avoid violating such a dependency.

Remove the Constraint
The simplest method is to remove the Constraint from the database before writing any data. This can be done using a SQL statement in the parameter "SQL Statement to Execute Before Translation":



The statement would be something along the lines of:

```
ALTER TABLE MyChildTable DROP CONSTRAINT MyConstraint
```

Then, in the parameter "SQL Statement to Execute After Translation" a SQL statement could be used to recreate the constraint:

```
ALTER TABLE MyChildTable ADD CONSTRAINT MyConstraint FOREIGN KEY
(MyChildKey) REFERENCES MyParentTable(MyParentKey)
```

This way data can be entered into the Parent and Child in any order, without violating a constraint.

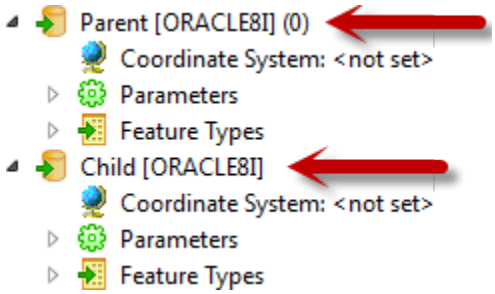
Multiple Writers and Parent/Child Tables

The more complex method of handling referential integrity is to ensure that features are loaded into the parent table before the child table. This way the constraint is not violated because all child records will already have matches in the parent table.

This can be done in FME either by using two Oracle Writers or by using one Writer and carefully controlling the order data is written.

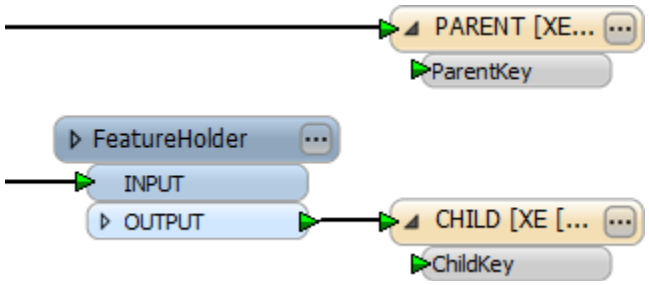
With two Writers, Writer A writes to the Parent table and is set to write first. Writer B writes to the child table and is set to write second. The trick is to make sure that all of Writer A's features are committed before any of Writer B's are.

By dragging the Parent Writer to the top of the list (in the Navigator) the user makes sure that features are sent to the parent table first of all…



There is no equivalent way to control the order of two tables within a single writer. FME would still try to write to the child table simultaneously.

Therefore, this user has added a FeatureHolder transformer. This will force child records to be cached until all parent records have been processed.



Session Review

This session was all about spatial databases and FME.

What You Should Have Learned from this Session
The following are key points to be learned from this session:

Theory

- Connecting to a database requires a set of ***connection parameters*** that may vary from database to database
- Updating features is done with two FME ***Format Attributes***
- ***Reader Parameters*** can be used to improve data reading performance
- FME can read and write ***multiple geometries***, but cannot create a table with multiple geometry columns
- ***Transactions*** help deal with performance and write failures
- ***Referential Integrity*** is important in Child/Parent tables

FME Skills

- The ability to connect to a database, write data, and update individual features
- The ability to use reader parameters, both alone and with concatenated parameters
- The ability to read and write multiple geometries
- The ability to use the *SQLExecutor* and *SQLCreator* transformers
- The ability to use transaction parameters
- The ability to use multiple writers for referential integrity