Pivotal

PRODUCT DOCUMENTATION

Pivotal GemFire®-Greenplum Connector 3.4®

Table of Contents

Pivotal GemFire-Greenplum Connector Documentation

This documentation describes the GemFire-Greenplum Connector.

Published December 13, 2018

Pivotal GemFire-Greenplum Connector 3.4

The GemFire-Greenplum Connector facilitates the acquisition of a copy of the contents of a Pivotal GemFire® region, updating an equivalent representation of the region in a Pivotal Greenplum® (GPDB) table. Data can also flow in the opposite direction: the connector facilitates the acquisition of a copy of the contents of a GPDB table, updating an equivalent representation of the table in a GemFire region.

- Release Notes
- Overview of the GemFire-Greenplum Connector
- Using the Connector
- Importing Data from Greenplum to GemFire
  An import copies all rows from a GPDB table to a GemFire region.
- Exporting Data from GemFire to Greenplum
  An export copies region entries from GemFire to a GPDB table.
- Datatype Mapping
  How the types correspond between a GemFire region and a GPDB table are described in a `cache.xml` file.
- GemFire Event Handlers
  An implementation of an event listener interface defines callbacks that will be invoked at various points within the import or export operation.
- Security
  GemFire's role-based authorization verifies that an authenticated user has the correct permissions for requested operations.
- Statistics Available for Monitoring
  Statistics about import and export operations that are currently in progress are placed into GemFire regions.
- gfsh Command Reference Pages
- Troubleshooting

API Reference Documentation

- GemFire-Greenplum Connector version 3.4 API

GemFire-Greenplum Connector 3.4 Release Notes

What's New in GemFire-Greenplum Connector 3.4.0

This version 3.4.0 of the connector ships with Pivotal GemFire® 9.7.x. This release works with Pivotal Greenplum® versions 4.3.x and 5.x.

See Using the Connector for specification of the Greenplum schema and requirements for using the connector.

Resolved Issues in GemFire-Greenplum Connector 3.4.0

- The `gfsh create gpdb-mapping` option `--id` is now optional. If the `--id` option is not specified, GGC will look for a primary key in the GPDB table. If the `--id` option is not specified and there is no primary key specified in GPDB, then the mapping will fail with the error

  ```
  Table <table-name> lacks a primary key. Please use the --id option.
  ```

  This also applies if GGC is configured in a `cache.xml` file.
- The GemFire `gfsh create jndi-binding` command options have been changed in the following ways:
  - The `--jdbc-driver-class` option is now optional.
  - `--url` is an alias for the `--connnection-url` option.
  - The `--type` option defaults to `SIMPLE`.
  - A specification of `--type=POOLED` defaults to creating a Hikari pool. Or, implement `org.apache.geode.datasource.PooledDataSourceFactory` to customize the class that implements the pool.

Known Issues in GemFire-Greenplum Connector 3.4.0

- Import truncates any time or timestamp from GPDB to a decimal millisecond precision. All represented microsecond precision is lost. For example, the GPDB time of 2017-09-21 12:31:03.636847 becomes 2017-09-21 12:31:03.636 within GemFire.

Overview of the GemFire-Greenplum Connector

The GemFire-Greenplum Connector facilitates the mirroring of the entries of an entire Pivotal GemFire® region into a Pivotal Greenplum Database® (GPDB) table. The connector also facilitates the mirroring of the entries of an entire GPDB table into a GemFire region.

All transfers are initiated from and specified within GemFire. Data copied from GemFire to GPDB makes use of the GemFire export functionality. Data copied from GPDB to GemFire makes use of the GemFire import functionality. The specification of mappings of tables within GPDB to regions in GemFire may be set up with `gfsh` commands or specified within a GemFire `cache.xml` file. Further mappings identify which GemFire fields are to be imported from or exported to GPDB table columns. A subset of the fields and columns may be specified, and then only that subset are imported or exported.

Using The Connector

The connector is included with Pivotal GemFire. The connector's JAR file will automatically be included on the classpath.

To use the connector, specify configuration details in `gfsh` commands or within a `cache.xml` file. Do not mix the use of `gfsh` for configuration with the use of a `cache.xml` file.

To do an explicit mapping of fields, or to map only a subset of the fields, specify all configuration in a `cache.xml` file.

Specification with gfsh

`gfsh` may be used to configure all aspects of transfer and the the mapping, as follows:

- If domain objects are not on the classpath, configure PDX serialization with the GemFire `configure pdx` command after starting locators, but before starting servers. For example:

```
gfsh>configure pdx --read-serialized=true \
  --auto-serializable-classes=io.pivotal.gemfire.demo.entity.*
```

- After starting servers, use the GemFire `create jndi-binding` command to specify all aspects of the data source. For example,

```
gfsh>create jndi-binding --name=datasource --type=SIMPLE \
  --jdbc-driver-class="org.postgresql.Driver" \
  --username="g2c_user" --password="changeme" \
  --connection-url="jdbc:postgresql://localhost:5432/gemfire_db"
```

- After creating regions, set up the gpfdist protocol by using `configure gpfdist-protocol`. For example,

```
gfsh>configure gpfdist-protocol --port=8000
```

- Specify the mapping of the GPDB table to the GemFire region with the `create gpdb-mapping` command. For example,

```
gfsh>create gpdb-mapping --region=/Child --data-source=datasource \
  --pdx-name="io.pivotal.gemfire.demo.entity.Child" --table=child --id=id,parent_id
```

Specification with a cache.xml File

To provide configuration details within a `cache.xml` file, specify the correct `xsi:schemaLocation` attribute within the `cache.xml` file.

For the 3.4.0 connector, use

```
http://schema.pivotal.io/gemfire/gpdb/gpdb-3.4.xsd
```

Connector Requirements and Caveats

- Export is supported from partitioned GemFire regions only. Data cannot be exported from replicated regions. Data can be imported to replicated regions.
- The number of Pivotal Greenplum® Database (GPDB) segments must be greater than or equal to the number of Pivotal GemFire servers. If there is a high ratio of GPDB segments to GemFire servers, the GPDB configuration parameter `gp_external_max_segs` may be used to limit GPDB concurrency. See gp_external_max_segs for details on this parameter. An approach to finding the best setting begins with identifying a representative import operation.
  - Measure the performance of the representative import operation with the default setting.
  - Measure again with `gp_external_max_segs` set to half the total number of GPDB segments. If there is no gain in performance, then the parameter does not need to be adjusted.
  - Iterate with values of `gp_external_max_segs` that are half as much at each iteration, until there is no performance improvement or the value of `gp_external_max_segs` is the same as the number of GemFire servers.

Pivotal

Upgrading Java Applications from Version 2.4 to Version 3.x

API changes implemented for version 3.0.0 that are also in this connector version require code revisions in all applications that use import or export functionality.

For this sample version 2.4 export operation, an upsert type of operation was implied:

```
// Version 2.4 API
long numberExported = GpdbService.createOperation(region).exportRegion();
```

Here is the equivalent version 3.x code to implement the upsert type of operation:

```
// Version 3.x API
ExportConfiguration exportConfig = ExportConfiguration.builder(region)
   .setType(ExportType.UPSERT)
   .build();
ExportResult result = GpdbService.exportRegion(exportConfig);
int numberExported = result.getExportedCount();
```

For this sample version 2.4 import operation,

```
// Version 2.4 API
long numberImported = GpdbService.createOperation(region).importRegion();
```

here is the version 3.x code to implement the import operation:

```
// Version 3.x API
ImportConfiguration importConfig = ImportConfiguration.builder(region)
   .build();
ImportResult result = GpdbService.importRegion(importConfig);
int numberImported = result.getImportedCount();
```

Please note that the new `result` objects' counts are of type `int` instead of type `long`. This is for consistency, as the connector internally uses JDBC's `executeQuery()`, which supports `int`.

Importing Data from Greenplum to GemFire

An import copies all rows from a Pivotal Greenplum® Database (GPDB) table to a Pivotal GemFire® region.

The import implements an upsert functionality: if a GPDB row imported is already present in a GemFire entry, the entry value will be updated if it has changed. If the GPDB row does not already exist as a GemFire entry, a new entry is created.

The mapping of GPDB table rows to GemFire region entries is within the `region` element definition of the GemFire `cache.xml` file.

An import operation may be invoked using API calls or with the `gfsh import` command.

Import Using the API

Import region functionality is exposed through the `GpdbService` class.

Example:

```
ImportConfiguration configuration = ImportConfiguration.builder(region)
   .build();
ImportResult importResult = GpdbService.importRegion(configuration);

// Get the total number of GPDB rows imported into the GemFire region.
int importCount =  importResult.getImportedCount();
```

The `OperationEventListener` interface provides further flexibility by allowing a set of callbacks to be defined. The callbacks are invoked at various points during an import of a GPDB table.

Import Using gfsh

See the command reference page for details on this use of the `gfsh import` command.

Requirements and Caveats

- Multiple regions may import from the same GPDB table concurrently.
- The import operation requires a definition of the GemFire key for an entry. A missing `cache.xml` entry for the `gpdb:id` element or an empty field as `<gpdb:id \>` will throw an error when an import operation is attempted.
- An incomplete `cache.xml` entry for the `gpdb:id` element, which omits fields that should have been part of a composite key, will not result in an error for an import operation. It will, however, result in leaving the GemFire region in an indeterminate state.
- If the GPDB table to be imported is described within a GPDB schema, but that schema is not specified in the GPDB configuration parameter `search_path`, the schema name must be specified by a `schema` attribute within the `gpdb:pdx` element of the GemFire `cache.xml` file.

Exporting Data from GemFire to Greenplum

An export copies entries from a Pivotal GemFire® region to a Pivotal Greenplum® (GPDB) table.

The export operation implements the functionality of one of these:

- The **UPSERT** functionality updates a GPDB row if the GemFire entry to be exported is already present as a GPDB row. If the GPDB row does not already exist, a new row is inserted.
- The **INSERT_ALL** functionality does a GPDB insert operation for each region entry in the GemFire region. Since it does not check for the existence of GPDB rows prior to each insert, the export operation will fail and throw an error if a duplicate primary key already exists in the GPDB table.
- The **INSERT_MISSING** functionality inserts rows into the GPDB table for GemFire region entries that are not already present in the table. It does*not* update any existing rows in the GPDB table.
- The **UPDATE** functionality updates a GPDB row if the GemFire entry to be exported is already present as a GPDB row.

The mapping of GPDB table rows to GemFire region entries is within the `region` element definition of the GemFire `cache.xml` file.

An export operation may be invoked using API calls or with the `gfsh export` command.

Export Using the API

Export region functionality is exposed through the `GpdbService` class. If not explicitly set in the `ExportConfiguration`, an export will implement `UPSERT` functionality.

This example does the default `UPSERT` type of export:

```
ExportConfiguration exportConfig = ExportConfiguration.builder(region)
   .setType(ExportType.UPSERT)
   .build();

ExportResult exportResult = GpdbService.exportRegion(exportConfig);

// Get the total number of GemFire entries exported to GPDB.
int exportedCount =  exportResult.getExportedCount();

// Get the total number of GPDB rows updated.
int updatedCount = exportResult.getUpdatedCount();

// Get the total number of GemFire entries inserted into the GPDB table.
int insertedCount = exportResult.getInsertedCount();
```

For any of the export types,*all* GemFire region entries present as an export operation is initiated may be removed from the region after a successful commit of the GPDB table. Specify this option in the export configuration. Note that *all* exported entries are removed, independent of which rows in the GPDB table have been updated or inserted. GemFire region entries added during the export operation are *not* exported and will not be removed from the GemFire region at the end of the export operation.

This example does an `INSERT_ALL` type of export, removing all entries from the GemFire region after a successful commit of the GPDB table:

```
ExportConfiguration exportConfig = ExportConfiguration.builder(region)
   .setType(ExportType.INSERT_ALL)
   .removeGemFireEntries(true)
   .build();

ExportResult exportResult = GpdbService.exportRegion(exportConfig);

// Get the total number of GemFire entries removed.
int removedCount = exportResult.getRemovedCount();
```

The `OperationEventListener` interface provides further flexibility by allowing a set of callbacks to be defined. The callbacks are invoked at various points during an export to a GPDB table.

Export Using gfsh

See the command reference page for details on this use of the `gfsh export` command.

Pivotal

Requirements and Caveats

- Export is supported from partitioned GemFire regions only. Data cannot be exported from replicated regions.
- At least one GemFire region field must be mapped to a GPDB table column for the export to work. Fields are mapped with the `<gpdb:field>` element in the `cache.xml` file. With no mapped fields, an exception is thrown for an export operation.
- The idempotent behavior of the export operation breaks for the case in which a GPDB key is not defined. This broken case would appear in the `cache.xml` file by defining an empty key: `<gpdb:id />`. The mapping has no GPDB key to use, so a new GPDB row must be created for each GemFire element exported. And, each subsequent export must repeat the creation of the new GPDB rows, causing duplication within the GPDB table.
- An export operation fails if the mapping specifies the `<gpdb:id />` field such that multiple GPDB table rows would be updated for a single GemFire region entry. The error implies an incorrect composite key specification.
- If multiple GemFire regions are mapped to the same GPDB table, the possibility of concurrent export operations exists. Further, if the mapped columns of GPDB table for the GemFire region fields intersect, then multiple updates to the same GPDB table row can be in progress at the same time. The connector does not provide any sort of transactional protocol for this situation. The last update wins.
- If the GPDB table is described within a GPDB schema, but that schema is not specified in the GPDB configuration parameter `search_path`, the schema name must be specified by a `schema` attribute within the `gpdb:pdx` element of the GemFire `cache.xml` file.
- An export operation creates temporary tables in GPDB. Repetitive export operations create and remove temporary tables, likely causing system catalog bloat. The GPDB administrator should be made aware of this, such that appropriate vacuuming will be scheduled.

SQL Implementation Steps

The following sequence of SQL statements implements the export operation, with `UPSERT` functionality, into the GPDB table. Within the SQL statements, the `< >` sequences indicate items that will be replaced by expansions of table-specific definitions. The ellipses (`...`) indicate the location of further clauses within the SQL statement.

1. A new temporary GPDB table is created:

```
DROP TABLE IF EXISTS "tmpTable";
CREATE TEMPORARY TABLE "tmpTable" (<column definitions>);
```

2. A new external GPDB table is created that refers to the external GemFire data source using the `gpfdist` protocol:

```
DROP EXTERNAL TABLE IF EXISTS "extTable";
CREATE EXTERNAL TABLE "extTable" (LIKE "tmpTable")
  LOCATION (<URIs>)
  FORMAT 'TEXT' (DELIMITER E'\u001F' NEWLINE 'LF') ENCODING 'UTF8';
```

3. The data from the external table is copied to the temporary table:

```
INSERT INTO "tmpTable" (<fields>) SELECT <fields> FROM "extTable";
```

4. The data from the temporary table is updated or inserted at the destination table:

```
UPDATE "dstTable" AS d
  SET (<fields>) = (<fields>)
  FROM "tmpTable" AS s
  WHERE d."<id-field>" = s."<id-field>" AND ... ;
INSERT INTO "dstTable" (<fields>)
  SELECT <fields>
  FROM "tmpTable"
  WHERE ROW(<id-fields>) NOT IN (SELECT <id-fields> FROM "dstTable");
```

The following sequence of SQL statements implements the export operation, with `INSERT_ALL` functionality, into the GPDB table. Within the SQL statements, the `< >` sequences indicate items that will be replaced by expansions of table-specific definitions. The ellipses (`...`) indicate the location of further clauses within the SQL statement.

1. A new external GPDB table is created that refers to the external GemFire data source using the `gpfdist` protocol:

```
DROP EXTERNAL TABLE IF EXISTS "extTable";
CREATE EXTERNAL TABLE "extTable" (LIKE "dstTable")
  LOCATION (<URIs>)
  FORMAT 'TEXT' (DELIMITER E'\u001F' NEWLINE 'LF') ENCODING 'UTF8';
```

2. The data from the external table is inserted at the destination table:

```
INSERT INTO "dstTable" (<fields>)
 SELECT <fields> FROM "extTable";
```

The following sequence of SQL statements implements the export operation, with `INSERT_MISSING` functionality, into the GPDB table. Within the SQL statements, the `< >` sequences indicate items that will be replaced by expansions of table-specific definitions. The ellipses (`...`) indicate the location of further clauses within the SQL statement.

1. A new temporary GPDB table is created:

```
DROP TABLE IF EXISTS "tmpTable";
CREATE TEMPORARY TABLE "tmpTable" (<column definitions>);
```

2. A new external GPDB table is created that refers to the external GemFire data source using the `gpfdist` protocol:

```
DROP EXTERNAL TABLE IF EXISTS "extTable";
CREATE EXTERNAL TABLE "extTable" (LIKE "tmpTable")
 LOCATION (<URIs>)
 FORMAT 'TEXT' (DELIMITER E'\u001F' NEWLINE 'LF') ENCODING 'UTF8';
```

3. The data from the external table is copied to the temporary table:

```
INSERT INTO "tmpTable" (<fields>) SELECT <fields> FROM "extTable";
```

4. The data from the temporary table is inserted at the destination table:

```
INSERT INTO "dstTable" (<fields>)
 SELECT <fields>
 FROM "tmpTable"
 WHERE ROW(<id-fields>) NOT IN (SELECT <id-fields> FROM "dstTable");
```

The following sequence of SQL statements implements the export operation, with `UPDATE` functionality, into the GPDB table. Within the SQL statements, the `< >` sequences indicate items that will be replaced by expansions of table-specific definitions. The ellipses (`...`) indicate the location of further clauses within the SQL statement.

1. A new temporary GPDB table is created:

```
DROP TABLE IF EXISTS "tmpTable";
CREATE TEMPORARY TABLE "tmpTable" (<column definitions>);
```

2. A new external GPDB table is created that refers to the external GemFire data source using the `gpfdist` protocol:

```
DROP EXTERNAL TABLE IF EXISTS "extTable";
CREATE EXTERNAL TABLE "extTable" (LIKE "tmpTable")
 LOCATION (<URIs>)
 FORMAT 'TEXT' (DELIMITER E'\u001F' NEWLINE 'LF') ENCODING 'UTF8';
```

3. The data from the external table is copied to the temporary table:

```
INSERT INTO "tmpTable" (<fields>) SELECT <fields> FROM "extTable";
```

4. The data from the temporary table updates the destination table:

```
UPDATE "dstTable" AS d
 SET (<fields>) = (<fields>)
 FROM "extTable" AS s
 WHERE d."<id-field>" = s."<id-field>" AND ...  ;
```

Datatype Mapping

How the types correspond between a Pivotal GemFire® region and a Pivotal Greenplum® (GPDB) table are described in a `cache.xml` file.

GemFire XML Description of a GPDB Table

In order to use the connector, a mapping between a GemFire region and a GPDB table has to be described. This mapping may go in the `cache.xml` file, which has an extended syntax for the mapping information.

The connector makes assumptions about the intended mapping if some parts of the mapping are incomplete. *We advise you to explicitly define all keys and fields, such that the connector has no assumptions to make.*

This `cache.xml` file example provides the XML description of the mappings for the region. The XML with the `gpdb` tag is the mapping between GemFire region fields and the GPDB table columns. It describes how to connect to GPDB, as well as gpfdist protocol to be used to transfer the data between the two systems.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cache xmlns="http://geode.apache.org/schema/cache"
  xmlns:gpdb="http://schema.pivotal.io/gemfire/gpdb"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://geode.apache.org/schema/cache
  http://geode.apache.org/schema/cache/cache-1.0.xsd
  http://schema.pivotal.io/gemfire/gpdb
  http://schema.pivotal.io/gemfire/gpdb/gpdb-3.4.xsd"
  version="1.0">

  <pdx read-serialized="true" persistent="false">
    <pdx-serializer>
      <class-name>org.apache.geode.pdx.ReflectionBasedAutoSerializer</class-name>
      <parameter name="classes">
        <string>io.pivotal.gemfire.demo.entity.*</string>
      </parameter>
    </pdx-serializer>
  </pdx>
  <jndi-bindings>
    <jndi-binding jndi-name="DemoDatasource" type="SimpleDataSource"
      jdbc-driver-class="org.postgresql.Driver" user-name="gpadmin"
      password="changeme" connection-url="jdbc:postgresql://localhost:5432/gemfire_db">
    </jndi-binding>
  </jndi-bindings>
  <region name="Parent">
    <region-attributes refid="PARTITION">
      <partition-attributes redundant-copies="1" />
    </region-attributes>
    <gpdb:store datasource="DemoDatasource">
      <gpdb:types>
        <gpdb:pdx name="io.pivotal.gemfire.demo.entity.Parent"
          schema="public"
          table="parent">
          <gpdb:id field="id" />
          <gpdb:fields>
            <gpdb:field name="id" column="id" />
            <gpdb:field name="name" />
            <gpdb:field name="income" class="java.math.BigDecimal" />
          </gpdb:fields>
        </gpdb:pdx>
      </gpdb:types>
    </gpdb:store>
  </region>
  <gpdb:gpfdist port="8000" />
</cache>
```

jndi-binding

In the example, there is a single JNDI binding to specify the details of the connection to the example's GPDB database. This element defines a `datasource` that is referenced later in `gpdb:store` element using `datasource` attribute.

The connection to the GPDB instance will be to `localhost:5432`, with a default database of `gemfire_db`. The GPDB user name is `gpadmin`, and its password is `changeme`.

For more details on JNDI bindings, see JNDI.

gpdb:store

The `gpdb:store` element specifies the mapping between GPDB table contents and the GemFire region.

There should be one store element for each GPDB data source.

In this example, the `gpdb:store` element identifies the JNDI binding to specify the GPDB instance at the other end of the connector. It is the data source.

gpdb:types

The `gpdb:types` element contains a list of `gpdb:pdx` instances. There is only one `gpdb:pdx` instance in this example.

gpdb:pdx

Each `gpdb:pdx` instance identifies the mapping of a GemFire region's entries to its corresponding GPDB table. This is needed for the data serialization and transfer implemented by the connector. The optional `schema` attribute specifies the GPDB schema for the table. See Creating and Managing Schemas within the GPDB manual for a description of GPDB schemas. The optional `schema` attribute is set to the default value of `public` when not specified. Make the `schema` attribute value the same across all GemFire servers that host the region.

gpdb:id

The `gpdb:id` element specifies the fields to be used as the key in the identification of a GemFire object.

The example uses a singleton key named `id`, also mapped to table column `id`:

```
<gpdb:id field="id" />
```

Here is an example of a composite key with the two fields `parentId` and `id` mapped to table columns `parentId` and `id`.

```
<gpdb:id>
  <gpdb:field ref="parentId" />
  <gpdb:field ref="id" />
</gpdb:id>
```

gpdb:fields

The `gpdb:fields` element lists the fields mapped to values of the GemFire `PdxInstance`. A subset of the available GemFire fields or GPDB columns may be mapped. Omission of this element causes the connector to automap the GemFire region fields to GPDB table columns, using the assumption that the exactly matching field and column names are to be mapped.

gpdb:field

Each `gpdb:field` instance specifies the mapping from a GemFire `PdxIntance` field to a GPDB column. If no GPDB `column` attribute is specified, the GPDB column is assumed to have exactly the same name (case sensitive) as the GemFire field.

Here is an example in which a GemFire `parentId` field is mapped to a GPDB `parent_id` column, and the GemFire `age` field is of the Java class (data type) `java.lang.Integer`.

```
<gpdb:field name="parentId" column="parent_id" />
<gpdb:field name="age" class="java.lang.Integer" />
```

Special cases exist when the `gpdb:field` attributes are missing from the `cache.xml` file. With no `gpdb:field` attributes defined, the connector assumes that the GemFire region fields have the same names as the GPDB table columns. This list describes the current behavior for this type of special case:

- If there exists a GPDB table column for which there is no GemFire region field, then an export operation does not update that GPDB table column. and an insert of a new row places either null or the defined default into the columns with no associated GemFire region field. An import operates on only those GemFire region fields for which the column with the matching name exists.

- If there is a GemFire region field for which there is no matching GPDB table column, then an export operation only exports the GemFire region fields for which there is a GPDB table column of the same name. An import has side effects on the GemFire region. The import correctly acquires those GemFire region fields for which the column with the matching name exists. But, the import sets to null those GemFire region fields for which there is no GPDB table column of the same name; this occurs for both an update and an insert of region entries.

If one or more `gpdb:field` attributes are specified, there can exist a mismatch of fields to columns. This list describes the current behavior for this type of special case:

- If there is exactly one mapping of a field to a column, and the naming of that GemFire region field or a GPDB table column within a `gpdb:field` attribute is incorrect such that it does not match a configured and defined name, an exception is thrown upon export, as this case appears as if there are no mappings.
- With more than one `gpdb:field` attributes specified, if the naming of one GemFire region field or a GPDB table column is incorrect such that it does not match a configured and defined name, upon export an update to a GPDB table row will not change the value of the incorrectly named column. An insert of a new GPDB table row under the same assumptions acquires the correctly mapped GemFire region fields. The incorrectly mapped GPDB column will be set to null or its defined default value. Upon import, either an update of an existing entry or an insert of a new entry acquires the GPDB table column values for any correctly mapped fields, and uses null for any incorrectly named GemFire field.

gpdb:gpfdist

The `gpdb:gpfdist` element is not part of the region specification, but is part of the cache specification. It provides details of the `gpfdist` protocol used in the connector. More details may be found within the section on Working with File-Based External Tables in the GPDB manual.

To increase the security of the data during transfer between GemFire and GPDB, TLS/SSL transfers can be enabled, as in this example:

```
<gpdb:gpfdist hostname="gpfdist.gemfire.test">
  <gpdb:ssl
     key-store="target/test-classes/io/pivotal/gemfire/gpdb/gpfdist/keystore.jks"
     key-store-password="123456"
     trust-store="target/test-classes/io/pivotal/gemfire/gpdb/gpfdist/truststore.jks"
     trust-store-password="123456" />
</gpdb:gpfdist>
```

All four `gpdb:ssl` attributes must be present to configure the transfers to use SSL. The two password fields remain in plain text within the `cache.xml` file, so file system protections should be considered.

The specification of optional attributes overrides default values. These are the optional attributes:

- A `port` specification overrides the listening port of `gpfdist` to be other than its default of the first available port within the inclusive range of 8080-8280. The value can be a single port number or a range. If specified with a single port number and that port number is already in use, an exception will be thrown. If specified with a range, the syntax uses a colon to separate the two endpoints of the range, as in `<lowport>:<highport>`. A value of 0 causes automatic allocation of a port number from an ephemeral port range. An example with a specific port number:

  ```
  <gpdb:gpfdist port="8000" />
  ```

  An example that specifies a port range:

  ```
  <gpdb:gpfdist port="8000:8100" />
  ```

- A `hostname` specifies the host name used in URI generation. If SSL is enabled, this name must match the common name (CN) of installed key store certificate. The default value is an IP address of the local host.

  ```
  <gpdb:gpfdist hostname="host12.example.com" />
  ```

- An `address` specification identifies the IP address for listening to be other than the default value of 0.0.0.0. For example:

  ```
  <gpdb:gpfdist address="192.0.2.0" />
  ```

- `idle-timeout` specifies a timeout in milliseconds. If no data is transmitted within this timeout period, the connection may be closed. The default value is 60000 milliseconds. There will be no timeout if set to 0. For example

  ```
  <gpdb:gpfdist idle-timeout="120000" />
  ```

Requirements and Caveats in the Mapping

- There must be only one GPDB table specified per GemFire region. If more than one GPDB table is specified for a single region by defining distinct GPDB tables across multiple `cache.xml` files, then an exception is thrown. If more than one GPDB table is specified within a single `cache.xml` file, no error will be thrown or issued, and the import or export operation will proceed, resulting in incorrect results.

Supported Datatypes

The following datatypes are supported. The left side shows the GPDB/PostgreSQL datatype, and the right side shows the associated GemFire/Java datatype which it is mapped to.

GPDB	GemFire
Bigint	`Long` (Object wrapper)
Bigint not null	`long` (primitive)
Bigserial	`long` (primitive)
Boolean not null	`boolean` (primitive)
Boolean	`Boolean` (wrapper)
Bit	`BitSet`
Varbit	`BitSet`
Char	`String`
Varchar	`String`
Decimal	`BigDecimal`
Float	`Float` (Object wrapper)
Float not null	`float` (primitive)
Int	`Integer` (Object wrapper)
Int not null	`int` (primitive)
Real	`Float` (Object wrapper)
Real not null	`float` (primitive)
Serial	`int` (primitive)
Smallint	`Short` (Object wrapper)
Smallint not null	`short` (primitive)
Text	`String`
Date	`java.sql.Date`
Time	`Time`
Timetz	`Time`
Timestamp	`Timestamp`
Timestamptz	`Timestamp`

Unsupported GPDB Datatypes

- inet
- cidr
- bytea
- box
- circle
- interval
- seg
- macaddr
- path
- point

- polygon
- xml
- money
- array

Unsupported GemFire Datatypes

- Classes with data members that are objects. Not supported, as this would require a representation of nested objects in GPDB. Here is an example of an unsupported type:

  ```
  class HasAnObjectMember {
      private UserObject theObject;
  }
  ```

- Any data type other than `BitSet`, `Boolean`, `Date`, `Number`, and `String`. Array types are an example of an unsupported data type.

GemFire Event Handlers

An implementation of an event listener interface defines callbacks that will be invoked at various points within the import or export operation.

Interface OperationEventListener

Callbacks are invoked during import or export operations, so defining one or more will impact performance. Each of these methods will be applied for either an import or an export operation.

- `void onBeforeType(TypeOperationEvent event)` Invoked before Pivotal Greenplum® (GPDB) SQL types are resolved into Pivotal GemFire® region fields.
- `void onAfterType(TypeOperationEvent event)` Invoked after GPDB SQL types are resolved into GemFire region fields.
- `void onBeforeExternalTable(ExternalTableOperationEvent event)` Invoked before an External GPDB table is created.
- `void onAfterExternalTable(ExternalTableOperationEvent event)` Invoked after an External GPDB table is created.
- `void onBeforeCommit(CommitOperationEvent event)` Invoked after the import or export of data, but before any GPDB commit.
- `void onAfterCommit(CommitOperationEvent event)` Invoked after the import or export of data and after any GPDB commit.

Code Fragment Examples

```
//CustomOperationEventListener class
public class CustomOperationEventListener
       extends AbstractOperationEventListener{

  @Override
    public void onAfterExternalTable(ExternalTableOperationEvent event) {
      // Do something against an external table after it gets created
    }

  ...
}
```

```
// Example - import operation with an event listener
ImportConfiguration importConfig = ImportConfiguration.builder(region)
    .addOperationEventListener(new CustomOperationEventListener())
    .build();
GpdbService.importRegion(importConfig);

// UPSERT export operation with the same event listener
ExportConfiguration exportConfig = ExportConfiguration.builder(region)
    .setType(ExportType.UPSERT)
    .addOperationEventListener(new CustomOperationEventListener())
    .build();
GpdbService.exportRegion(exportConfig);
```

Security

GemFire's role-based authorization verifies that an authenticated user has the correct permissions for requested operations.

Resource Permissions

See the GemFire section on Implementing Security for definitions of permission levels within GemFire.

In addition to the existing permission levels, there is a finer-grained target of `GPDB` defined for `CLUSTER` operations.

This table specifies the permissions required to execute these connector-specific `gfsh` commands.

`gfsh` Command	Assigned `ResourcePermission`
cancel gpdb operations	CLUSTER:MANAGE:GPDB
configure gpfdist-protocol	CLUSTER:MANAGE:GPDB
create gpdb-mapping	CLUSTER:MANAGE:GPDB
describe gpdb-mapping	CLUSTER:READ:GPDB
describe gpfdist-protocol	CLUSTER:READ:GPDB
destroy gpdb-mapping	CLUSTER:MANAGE:GPDB
export gpdb	DATA:READ:regionName
import gpdb	DATA:WRITE:regionName
list gpdb operations	CLUSTER:READ:GPDB
list gpdb-mapping	CLUSTER:READ:GPDB

This table specifies permissions required to execute these methods.

method	Assigned `ResourcePermission`
GpdbService.cancelOperation()	CLUSTER:MANAGE:GPDB
GpdbService.exportRegion()	DATA:READ:regionName
GpdbService.importRegion()	DATA:WRITE:regionName

Statistics Available for Monitoring

Two Pivotal GemFire® regions are populated with statistics about import and export operations that are currently in progress.

GemFire Regions with Statistics

The `GGC_OPERATIONS_MONITOR` region is a replicated region that contains one entry for each operation. An entry is added when an import or export is initiated, and the entry is removed when the import or export completes. The key identifies the operation; it is a Java UUID. The value is comprised of

- the UID of the user that invokes the import or export command
- either “import” or “export”, to distinguish the operation
- the GemFire region name that is the source or destination of the operation
- the GPDB table name that is the source or destination of the operation
- the start time of the operation, the `long` that results from a call to `System.currentTimeMillis()`

The `GGC_STATISTICS_MONITOR` region is a partitioned region that contains statistics entries for each import or export operation. Each server maintains the statistics for itself. An entry is added when an import or export is initiated, and the entry is removed when the import or export completes. A composite key contains an operation ID, the server name, and a thread ID. The value is comprised of

- the number of GPDB rows or GemFire entries processed
- the number of bytes processed

These values within an entry are updated on a per-server basis after each 10,000 rows/entries are processed.

Acquiring the Statistics

The `gfsh list gpdb operations` command aggregates the number of rows and the number of bytes processed across all servers and then outputs a table with one row for each import or export operation. Each row contains all fields from the `GGC_OPERATIONS_MONITOR` region and the aggregate values calculated from the partitioned `GGC_STATISTICS_MONITOR` region.

See the command reference page for details on the use of the `gfsh list gpdb operations` command.

gfsh Command Reference Pages

Extensions to `gfsh` commands implement connector operations, as given in these reference pages:

- cancel gpdb operation
- configure gpfdist-protocol
- create gpdb-mapping
- describe gpdb-mapping
- describe gpfdist-protocol
- destroy gpdb-mapping
- export gpdb
- import gpdb
- list gpdb operations
- list gpdb-mapping

cancel gpdb operation

Halt an in-progress import or export operation. When the cancel operation finishes:

- For an import operation, the GemFire region will be partially updated with newly imported values.
- For an export operation, the GPDB table will be in the same state it was before the export operation began.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
cancel gpdb operation --operationId=OPERATIONID
```

Name	Description
--operationId	*Required.* The operation ID, which can be found by running `list gpdb operations`.

Sample Output:

```
gfsh>cancel gpdb operation --operationId=f5d241a0-876f-4c64-980a-ed313f3488ca
Found operation to cancel.
```

configure

configure gpfdist-protocol

Specifies configuration details for the gpfdist protocol across the cache.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
configure gpfdist-protocol [--port={value|p1{-|:}pn}]
  [--idle-timeout=value] [--reset-to-defaults]
```

Parameters, create gpfdist-protocol:

Name	Description
‑‑port	A port number or an inclusive range of port numbers that gpfdist should listen on. A value of 0 finds and acquires an unallocated port number. Do not specify in combination with `--reset-to-defaults`.
‑‑idle-timeout	Defines the quantity of milliseconds after which the connection may be closed due to inactivity. A value of 0 disables the timeout. Do not specify in combination with `--reset-to-defaults`.
‑‑reset-to-defaults	Sets a port range of 8080 to 8280 (inclusive). Sets an idle timeout of 60000 milliseconds (1 minute). This option takes precedence when used in combination with `--port` and/or `--idle-timeout`.

Sample Output:

```
gfsh>configure gpfdist-protocol --port=8000
Member  | Status | Message
------- | ------ | -----------------------------------
server1 | OK     | Configuration updated successfully.
server2 | OK     | Configuration updated successfully.
```

create

create gpdb-mapping

Specify the mapping between a GPDB table and a GemFire region.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
create gpdb-mapping --data-source=value --pdx-name=value --region=value
    --table=value [--id=value]
```

Parameters, create gpdb-mapping:

Name	Description
−−region	*Required.* Name of the GemFire region to associate with the GPDB table.
−−data-source	*Required.* The name of the JNDI data source to use for the GPDB JDBC connection.
−−pdx-name	*Required.* Path and class name of the PDX serializer implementation, or a name that will be used for the default implementation.
−−table	*Required.* The GPDB table name to associate with the GemFire region. May be prefixed with the schema name.
−−id	Name of the PDX field to use as the key in the identification of a GemFire entry. For composite keys, specify multiple names within a comma-separated list. If not specified, the primary key from the GPDB table is used. If there is no primary key, an error message is issued.

Sample Output:

```
gfsh>create gpdb-mapping --data-source=datasource --pdx-name=Child \
--region=/Child --table=Child --id=id

Member  | Status | Message
------- | ------ | -------
server1 | OK     | SUCCESS
Created mapping for region Child to GPDB table Child
Changes to configuration for group 'cluster' is persisted.
```

```
gfsh>create gpdb-mapping --region=/Child --data-source=datasource \
--pdx-name="io.pivotal.gemfire.demo.entity.Child" --table=child --id=id,parent_id
Member  | Status | Message
------- | ------ | -------
server1 | OK     | SUCCESS
server2 | OK     | SUCCESS

Changes to configuration for group 'cluster' are persisted.
```

This example shows the error message that results from not specifying the `--id` option for the case in which the GPDB table does not have a primary key:

```
gfsh>create gpdb-mapping --region=/Parent --data-source=datasource \
--pdx-name="io.pivotal.gemfire.demo.entity.Parent" --table=Parent
Member  | Status | Message
------- | ------ | ------------------------------------------------------------
server1 | ERROR  | Table Parent lacks a primary key. Please use the --id option.
server2 | ERROR  | Table Parent lacks a primary key. Please use the --id option.
```

describe

describe gpdb-mapping

Display information about a mapping between a GemFire region and a Pivotal Greenplum Database (GPDB) store.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe gpdb-mapping --region=value
```

Parameters, describe gpdb-mapping:

Name	Description
--region	*Required*. Name of the region that has the mapping to be described.

Example Commands:

```
describe gpdb-mapping --region=customers
```

Sample Output:

```
gfsh>describe gpdb-mapping --region=/Child
Field Name:  | GPDB Column Name:  | Field Type | Field Class
------------ | ------------------ | ---------- | ----------------------
id           | id                 | int8       | class java.math.BigInteger
parent_id    | parent_id          | int8       | class java.math.BigInteger
name         | name               | text       | class java.lang.String
age          | age                | int4       | class java.lang.Integer
```

describe gpfdist-protocol

Display information about a gpfdist configuration. If a server is not specified, displays information about the configuration of all the servers.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe gpfdist-protocol [--name=value]
```

Parameters, describe gpfdist-protocol:

Name	Description
--name	Name of the server to describe.

Example Commands:

```
describe gpfdist-protocol
describe gpfdist-protocol --name=server1
```

Sample Output:

```
gfsh>describe gpfdist-protocol --name=server1
isSSL: false
KeyStore: null
KeyStoreType: JKS
TrustStore: null
TrustStoreType: JKS
Address: 0.0.0.0
Hostname: 192.0.2.0
Port: 8000
IdleTimeout: 60000
```

destroy

destroy gpdb-mapping

Destroy an existing GPDB mapping.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
destroy gpdb-mapping --region=value
```

Parameters, destroy gpdb-mapping:

Name	Description
--region	*Required.* Name of the region for which the GPDB mapping is to be destroyed.

Example Commands:

```
gfsh>destroy gpdb-mapping --region=Customers
```

export gpdb

Export a region to GPDB. Export is supported from partitioned GemFire regions only. Data cannot be exported from replicated regions.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
export gpdb --region=regionpath --type=value [--remove-all-entries(=value)]
```

Name	Description	Default Value
−−region	*Required.* Region from which data is to be exported. Prefix the region name with a slash character.	
−−type	*Required.* Specification of the functionality implemented for the export operation. • `UPSERT` updates rows already present in the GPDB table, and it inserts rows where not already present. • `INSERT_ALL` does a GPDB insert operation for each entry in the GemFire region. • `INSERT_MISSING` does a GPDB insert operation for each GemFire region entry for which there is no corresponding GPDB row; no updates are done for existing GPDB rows. • `UPDATE` updates rows already present in the GPDB table.	
−−remove-all-entries	Optional boolean value that, when true, removes *all* GemFire entries present in the specified region when the export operation is initiated, once changes have been successfully committed to the GPDB table. *All* exported region entries are removed, independent of which rows are updated or inserted into the GPDB table.	false

Example Commands:

```
gfsh>export gpdb --region=/customers --type=UPSERT
GemFire entries exported : 5
Greenplum rows updated   : 5
Greenplum rows inserted  : 0
Duration                 : 0.30s
```

```
gfsh>export gpdb --region=/customers --type=INSERT_ALL --remove-all-entries=true
GemFire entries exported : 5
GemFire entries removed  : 5
Greenplum rows inserted  : 5
Duration                 : 0.25s
```

import gpdb

Import to a GemFire region from a GPDB table.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
import gpdb --region=regionpath
```

Name	Description
−−region	*Required.* Region into which data will be imported. Prefix the region name with a slash character.

Example Commands:

```
import gpdb --region=/customers
GemFire entries imported : 10
Duration                 : 0.18s
```

list gpdb operations

list gpdb operations

Output statistics in a tabular form about all import or export operations that are currently in progress.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list gpdb operations
```

Sample Output:

```
gfsh>list gpdb operations
      Operation Id       | Operation Name | Region Name | Table Name | User Name |         Start Time          | Rows Processed | Bytes Processed
------------------------- | -------------- | ----------- | ---------- | --------- | ---------------------------- | -------------- | ---------------
cd4050ef-13c0-4538-a749-.. | import         | Child       | child      | root      | Mon Oct 17 15:21:35 PDT 2016 | 60000          | 2880000
```

list gpdb-mapping

List the mappings of GPDB tables to GemFire regions for all regions.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list gpdb-mapping
```

Sample Output:

```
gfsh>list gpdb-mapping
Region Name:  | PDX Name:  | Table Name:
------------- | ---------- | ------------
Child         | mystring   | child
```

Troubleshooting

Here are problems and fixes related to using the GemFire-Greenplum Connector.

- **Problem:** When a second server is started, the second server will fail to start under each of the following situations:
    - If two `cache.xml` files have different GPDB table names mapped to the same GemFire region.
    - If two `cache.xml` files have different `schema` attribute values in the specification of the `gpdb:pdx` element for a single GemFire region.
    - If two `cache.xml` files map a different set of ID fields. This implies that the `gpdb:id` element must be fully specified; it cannot be empty.
    - If two `cache.xml` files have different content for their `gpdb:field` elements. Note that if both have an empty `gpdb:fields` element, the second server can start. For differing content in the `cache.xml` files, the situations that cause the second server to fail include:
        - no `gpdb:field` elements are specified for the first server, and all the `gpdb:field` elements are specified for the second server
        - no `gpdb:field` elements are specified for the second server, and all the `gpdb:field` elements are specified for the first server
        - no `gpdb:field` elements are specified for the first server, and some `gpdb:field` elements are specified for the second server
        - no `gpdb:field` elements are specified for the second server, and some `gpdb:field` elements are specified for the first server

  Solution: Correct the `cache.xml` contents and start or restart the servers.
- **Problem:** Error message when attempting an import or export. An error message similar to

```
Could not process command due to GemFire error.
Error while processing command <import gpdb --region=/Child>
```

  Cause of the Problem: There is a communication error between GemFire and the Pivotal Greenplum® Database (GPDB) segment. The GPDB communication to GemFire is unsuccessful.
  Diagnosis and Solution: Use network tools to identify and fix the problem. Consider firewall issues, and make sure that the proper ports are open. Also check that the server's gpfdist is configured correctly. Use the `gfsh describe gpfdist-protocol` command to observe the configuration. If the host name or IP address are incorrect, correct them in the `cache.xml` file's `gpdb:gpfdist` attribute specification or with the `gfsh configure gpfdist-protocol` command.
- **Problem:** Error message when attempting an import or export. An error message similar to

```
No longer connected to 192.0.2.0[1099]. Could not process command due
to GemFire error. Error while processing command
<export gpdb --region=/LargeRegion --type=UPSERT>
Reason : This connection to a distributed system has been disconnected.
```

  Cause of the Problem: Timeouts within the GemFire cluster have resulted in the disconnection of one or more cluster members.
  Diagnosis and Solution: Follow advice on Diagnosing System Problems within the GemFire manual on topics dealing with timeouts, as well as the specific error of Member logs ForcedDisconnectException, Cache and DistributedSystem forcibly closed to adjust the GemFire cluster.
- **Problem:** Error message when attempting an import or export. An error message similar to

```
Could not process command due to GemFire error.
Error while processing command
<import gpdb --region=/Child>
Reason : io.pivotal.gemfire.gpdb.operations.OperationException:
org.postgresql.util.PSQLException: FATAL: database "gemfire_db" does not exist
```

  Cause of the Problem: The GPDB does not exist as specified. Doing an import or export operation presumes that the GPDB exists and is up and running; these operations do not create or instantiate a GPDB.
  Diagnosis and Solution: Find and fix any spelling or address error(s) in the specification of the GPDB.
- **Problem:** A `NullPointerException` causes this start of a stack trace starts within the server's log:

```
[error 2017/04/19 09:21:18.112 KST server1-1 <Function Execution Processor2> tid=0x72]
null 1 execute: Failed operation.
java.lang.NullPointerException
at io.pivotal.gemfire.gpdb.operations.OperationBase$Transaction
     .begin(OperationBase.java:564)
at io.pivotal.gemfire.gpdb.operations.OperationBase
     .execute(OperationBase.java:114)
at io.pivotal.gemfire.gpdb.functions.OperationFunction
     .execute(OperationFunction.java:68)
at org.apache.geode.internal.cache.execute.AbstractExecution
     .executeFunctionLocally(AbstractExecution.java:333)
```

  Cause of the Problem:
  The `connection-url` attribute is invalid, so the JNDI binding for JDBC cannot work.
  Diagnosis and Solution: Find and fix the URL. If created with `gfsh create jndi-binding`, then first destroy it using `gfsh destroy jndi-binding`, and then re-create the JNDI binding with a corrected URL. If specified within the `jndi-binding` element of the `cache.xml` file, fix the spelling within the `cache.xml` file. The

JDBC connection to the GPDB may be tested by using the procedure outlined in the Pivotal Greenplum Knowledge Base article [How to test JDBC and Greenplum Datadirect JDBC](#).