Pivotal

Table of Contents

Pivotal

Pivotal GPText 2.0 Documentation

Pivotal GPText Documentation

PDF

- Pivotal GPText 2.0 Release Notes
- Installing Pivotal GPText
- Using Pivotal GPText
- GPText Function Reference
- GPText Management Utility Reference

Additional Resources

- Pivotal Greenplum Database
- Apache Solr Web Site

Pivotal® GPText 2.0 Release Notes

This document contains release information for Pivotal GPText 2.0.

About Pivotal GPText 2.0

Pivotal GPText joins the Greenplum Database massively parallel-processing database server with Apache SolrCloud enterprise search and the Apache MADlib (incubating) Analytics Library to provide large-scale analytics processing and business decision support. GPText includes free text search as well as support for text analysis.

GPText includes the following features:

- The GPText database schema provides in-database access to Apache Solr indexing and searching
- Custom tokenizers for international text and social media text
- A Universal Query Processor that accepts queries with mixed syntax from supported Solr query processors
- Faceted search results
- Term highlighting in results
- Greater emphasis on high availability

The GPText management utility suite includes command-line utilities to perform the following tasks:

- Start, stop, and monitor ZooKeeper and GPText nodes
- Configure GPText nodes and indexes
- Add and delete replicas for index shards
- Back up and restore GPText indexes
- Recover a GPText node
- Expand the GPText cluster by adding GPText nodes

Prerequisites

The GPText installation includes the installation of Apache Solr Cloud.

Before you install GPText:

- Install and configure your Greenplum Database system, version 4.3.5 or higher. See the *Greenplum Database Installation Guide* at http://gpdb.docs.pivotal.io.
- When you configure Greenplum Database, first reserve memory on each Greenplum segment host for GPText use. To determine the memory to set aside for GPText, multiply the number of GPText nodes to create on each Greenplum segment host by the JVM maximum size. Subtract this memory from the physical RAM when calculating the value for the Greenplum Database `gp_vmem_protect_limit` server configuration parameter. See the Greenplum Database server configuration parameter `gp_vmem_protect_limit` in the *Greenplum Database Reference Guide* for recommended memory calculation formulas or visit the GPDB Virtual Memory Calculator web site.
- GPText runs on Red Hat Enterprise Linux 5.x or 6.x.
- Install Oracle `JRE 1.8.x` and place it in `PATH` on the master and all segment servers.
- GPText cannot be installed onto a shared NFS mount.
- Ensure that `nc` (netcat) is installed on all Greenplum cluster hosts (`yum install nc`).
- Installing `lsof` on the Greenplum master and all hosts is recommended (`sudo yum install lsof`).
- Apache Solr requires a ZooKeeper cluster with at minimum three nodes. You can install a “binding” ZooKeeper cluster with GPText on the Greenplum cluster hosts, or you can use an existing ZooKeeper cluster. When deployed alongside Greenplum Database segments, ZooKeeper performance can be affected under heavy database load. For best performance, install a ZooKeeper cluster with at least three nodes (five nodes recommended) on separate hosts with network connectivity to the Greenplum network.

Known Issues

- Solr does not return all fields when the `fl` Solr search option contains a wildcard that matches field names. For example, given a table with columns

`contenta` and `contentb`, specifying `fl=contenta,contentb,(sum,1,1)` correctly returns three fields. Specifying `fl=cont*,sum(1,1)` correctly returns `contenta` and `contentb`, but omits the pseudo-field `sum(1,1)`. Specifying a wildcard to match all fields (`fl=*,sum(1,1)`) also omits the pseudo-field.

- If Solr fails to load an index because of a configuration file error, and then the index is dropped without first correcting the configuration file error, the index cannot be recreated until GPText is restarted. This can happen if you edit `managed-schema` or `solrconfig.xml` and introduce an XML syntax error or a typo in configuration values. Workaround:
  1. When an index fails to load, check the Solr log to find the cause.
  2. If the cause is a configuration file error, such as invalid XML, use the `gptext-config` utility to edit the file and fix the error. Dropping the index without first correcting the error is not recommended.
  3. If you have dropped an index that failed to load without first correcting the cause of the failure, you must restart GPText before you can recreate the index. Run `gptext-start -r` to restart GPText.

Installing GPText

Prerequisites

The GPText installation includes the installation of Apache Solr Cloud.

Before you install GPText:

- Install and configure your Greenplum Database system, version 4.3.5 or higher. See the *Greenplum Database Installation Guide* at http://gpdb.docs.pivotal.io.
- When you configure Greenplum Database, first reserve memory on each Greenplum segment host for GPText use. To determine the memory to set aside for GPText, multiply the number of GPText nodes to create on each Greenplum segment host by the JVM maximum size. Subtract this memory from the physical RAM when calculating the value for the Greenplum Database `gp_vmem_protect_limit` server configuration parameter. See the Greenplum Database server configuration parameter `gp_vmem_protect_limit` in the *Greenplum Database Reference Guide* for recommended memory calculation formulas or visit the GPDB Virtual Memory Calculator web site.
- GPText requires Red Hat Enterprise Linux 5.x or 6.x.
- Install Oracle JRE 1.8.x and place it in `PATH` on the master and all segment servers.
- GPText cannot be installed onto a shared NFS mount.
- Ensure that `nc` (netcat) is installed on all Greenplum cluster hosts (`sudo yum install nc`).
- Installing `lsof` on the Greenplum master and all hosts is recommended (`sudo yum install lsof`).

Note:
GPText can use an existing Apache ZooKeeper cluster or you can install a "binding" ZooKeeper cluster on the Greenplum cluster during GPText installation. A separate ZooKeeper cluster with at least five nodes is recommended for best performance with heavy database loads. To use a separate ZooKeeper cluster, the cluster must be up and have network connectivity with the Greenplum cluster hosts before you begin installing GPText.

Install the GPText Binaries

1. On the Greenplum master host, extract the GPText distribution file, a compressed tar archive. For example:

   ```
   cd /home/gpadmin
   tar xvfz greenplum-text-release-rhel5_x86_64.tar.gz
   ```

   The release directory contains an installation configuration file, `gptext_install_config`, and the GPText installation binary, which has a name similar to `greenplum-text-version-OS.bin`, for example, `greenplum-text-2.0.0-rhel5_x86_64.bin`.

2. If necessary, grant execute permission to the GPText binary. For example:

   ```
   chmod +x /home/gpadmin/greenplum-text-2.0.0-rhel5_x86_64.bin
   ```

3. If you are installing GPText in a directory that is only accessible to root, for example `/usr/local`, perform these steps:

   a. Create the installation directory as root and change the ownership to the GPText installer, gpadmin.
   b. To install to a directory where the user may or may not have write permissions:
      - Use gpssh to create a directory with the same file path on all hosts (`mdw`, `smdw`, and the segment hosts `sdw1`, `sdw2`, and so on). For example:

        ```
        /usr/local/greenplum-text-<version>
        ```

      - As root, set the file permissions and owner. For example:

        ```
        # chmod 775 /usr/local/greenplum-text-<version>
        # chown gpadmin:gpadmin /usr/local/greenplum-text-<version>
        ```

4. Edit the `gptext_install_config` file to set parameters for the installation. See Set Installation Parameters for details.

5. Run the GPText installation binary as `gpadmin` on the master server:

```
./greenplum-text-<version>.bin -c gptext_install_config
```

6. Accept the Pivotal license agreement.

Set Installation Parameters

A GPText configuration file named `gptext_install_config` contains parameters to configure the GPText installation. Edit the file and set the parameters as described in the following table.

Table 1. GPText installation parameters

Parameter	Description	Default value
`DATA_DIRECTORY`	An array of directory paths where GPText data directories are to be created. The number of directories in the array determines the number of GPText nodes that will be created on each physical host. If `GPTEXT_HOSTS` lists multiple interfaces per host, the GPText nodes are spread evenly across the interface addresses.	`declare -a DATA_DIRECTORY= (/data/primary /data/primary)`
`JAVA_OPTS`	Sets the minimum and maximum memory each SolrCloud JVM can use.	`JAVA_OPTS="-Xms1024M -Xmx2048M"`
`GPTEXT_PORT_BASE` `GP_MAX_PORT_LIMIT`	Set a range of port numbers available to GPText nodes. GPText finds unused ports in the specified range.	18983 - 28983
`ZOO_CLUSTER`	Whether to deploy a GPText binding ZooKeeper cluster or use an existing ZooKeeper cluster. If set to `"BINDING"` the installation deploys a ZooKeeper cluster. To use an existing ZooKeeper cluster, set this parameter to a list of the ZooKeeper nodes in the format `"host1:port,host2:port,host3:port"`. See the note under Prerequisites concerning ZooKeeper performance.	`"BINDING"`
`ZOO_HOSTS`	If `ZOO_HOSTS=(sdw1 sdw1 swd1 sdw1 sdw1)`.	`declare -a ZOO_HOSTS= (localhost localhost localhost localhost localhost)`
`ZOO_DATA_DIR`	The ZooKeeper data directory, required when `"BINDING"`.	`/data/master`
`ZOO_GPTXTNODE`	The node path in ZooKeeper for GPText. This parameter is required whether `"BINDING"` or a list of hosts.	`"gptext"`
`ZOO_PORT_BASE` `ZOO_MAX_PORT_LIMIT`	A range of port numbers to use for the ZooKeeper cluster. Unused ports are allocated from within this range. The range must contain at least 4000 port numbers, that is: `ZOO_MAX_PORT_LIMIT - ZOO_PORT_BASE >= 4000`	`2188 - 12188`

Starting GPText

First, make sure the GPText command-line utilities are in your path by sourcing the following file located in the GPText installation directory:

```
source install_dir/greenplum-text_path.sh
```

To use GPText in a database, you must first use the `gptext-installsql` management utility to install the GPText user-defined functions and other objects in the database:

```
gptext-installsql database [database2 ... ]
```

The GPText objects are created in the `gptext` schema.

Start GPText by running the `gptext-start` management utility at the command line:

```
$ gptext-start
```

Uninstalling GPText

To uninstall GPText, run the `gptext-uninstall` utility. You must have superuser permissions on all databases with GPText schemas to run `gptext-uninstall`.

`gptext-uninstall` runs only if there is at least one database with a GPText schema.

Execute:

```
gptext-uninstall
```

Using Pivotal GPText

Introduction to Pivotal GPText

Pivotal GPText enables processing mass quantities of raw text data (such as social media feeds or e-mail databases) into mission-critical information that guides business and project decisions. GPText joins the Greenplum Database massively parallel-processing database server with Apache SolrCloud enterprise search and the MADlib Analytics Library to provide large-scale analytics processing and business decision support. GPText includes free text search as well as support for text analysis. GPText supports business decision making by offering:

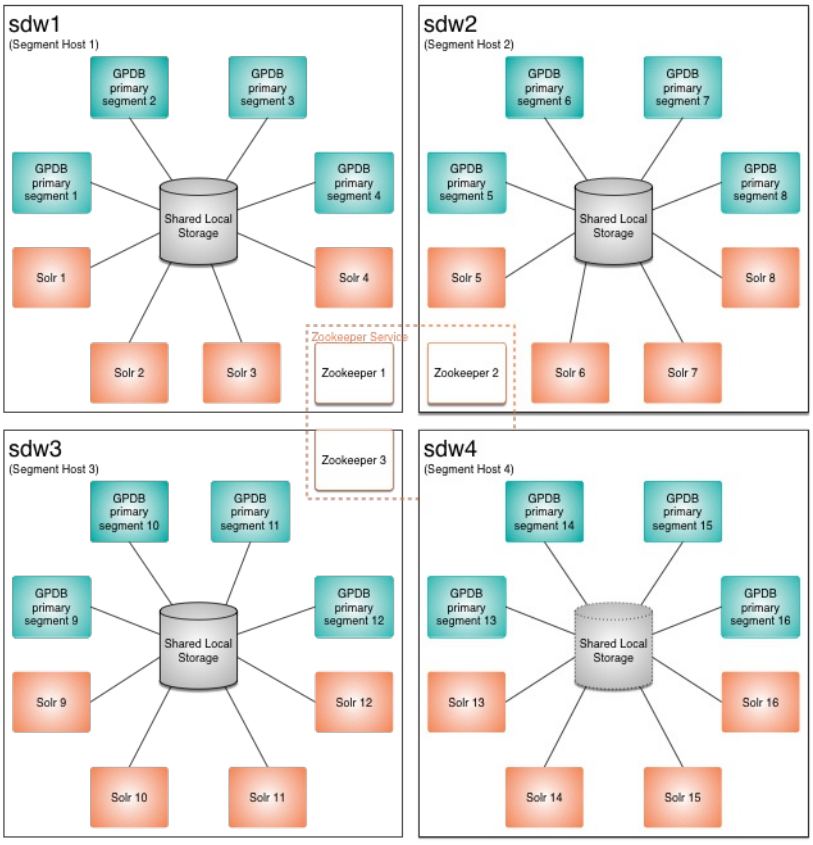
- **Multiple kinds of data:** GPText supports both semi-structured and unstructured data searches, which exponentially increases the kinds of information you can find.
- **Less schema dependence:** GPText does not require static schemas to successfully locate information; schemas can change or be quite simple and still return targeted results.
- **Text analytics:** GPText supports analysis of text data with machine learning algorithms. The MADlib analytics library is integrated with Greenplum Database and is available for use with GPText.

This chapter contains the following topics:

GPText System Architecture

GPText combines a Greenplum Database cluster with an Apache SolrCloud cluster residing on the same set of hosts.

The following figure shows the process architecture of the combined Greenplum Database and ApacheSolr clusters. The figure shows four cluster nodes with four Greenplum segments and four Solr instances deployed on each. An Apache ZooKeeper service manages the SolrCloud cluster. Because ZooKeeper requires an odd number of servers, ZooKeeper nodes are deployed on three of the four hosts. Greenplum Database users access SolrCloud services via GPText user-defined functions installed in Greenplum databases and command-line utilities.



The figure omits the Greenplum master host, secondary master, and mirror segments for the Greenplum primary segments.

The following sections describe how GPText integrates SolrCloud with Greenplum Database and how the two clusters work together to provide parallel text search capabilities in Greenplum Database and maintain high availability.

Greenplum Database Cluster

A Greenplum Database cluster is comprised of the following components:

- A master database instance, executing on a dedicated host, conventionally named `mdw`. (*Not illustrated*)
- A secondary master instance, on a host conventionally named `smdw`, acting as a warm standby for the master instance. (*Not illustrated)*
- An array of database primary segment instances and mirrors (*not illustrated)* deployed on segment hosts, by convention `sdw1` through `sdwn`. A segment instance is an independent Postgres database process managing a portion of the distributed data. Each segment has a mirror on another host in the cluster to provide uninterrupted service in case of a segment or segment host failure. The number of primary segments per host is determined by the hardware configuration—the number and type of processor cores, the amount of physical RAM, local storage capacity, and network capacity—as well as availability and performance requirements.

The Greenplum master instance coordinates the work of the segment instances. Optimal performance of a Greenplum Database cluster requires that all segment hosts be configured identically with the same number of primary and mirror segments on each, and with the database data distributed evenly among the segment instances. The full capacity of the database cluster is utilized when every segment host performs an equal amount of work.

Apache SolrCloud

Apache Solr is a server providing access to Apache Lucene full-text indexes. Apache SolrCloud is a highly available, fault tolerant cluster of Apache Solr servers. The term *GPText cluster* is another way to refer to a SolrCloud cluster deployed by GPText for use with a Greenplum Database system.

A SolrCloud cluster is comprised of the following components:

- An Apache ZooKeeper cluster to manage the SolrCloud cluster. SolrCloud uses ZooKeeper to manage server configuration and to coordinate the cluster's activities. A ZooKeeper cluster always has an odd number of servers. GPText can install a ZooKeeper cluster that is bound to the GPText cluster, or it can share an existing ZooKeeper cluster. If GPText installs the ZooKeeper cluster, it can be managed using GPText functions and utilities.
- Multiple SolrCloud server instances deployed on the Greenplum segment hosts. Each instance is a JVM process running Solr server. SolrCloud instances use local storage, usually the same local storage volumes that store Greenplum Database data. The number of SolrCloud instances per host can be the same as the number of Greenplum primary segments per host, *but this is not a requirement*. The number of instances to execute per host is specified during GPText installation.

GPText provides document indexing and search capabilities for Greenplum Database by adding user-defined functions (UDFs) that access Solr APIs from within database queries.

GPText UDFs perform the following tasks:

- create and manage GPText indexes
- insert documents into indexes
- search indexes

There are also GPText UDFs and command-line utilities to configure, monitor, and manage the SolrCloud cluster and to manage replicas, SolrCloud's high-availability mechanism. (More on replicas in the next section.)

Parallelism in GPText Indexing and Searching

SolrCloud distributes document indexes in slices called *shards*. With GPText, the number of shards for an index is the same as the number of Greenplum segments, so each Greenplum segment operates on an equal portion of the index. Each shard is managed by a SolrCloud instance and the shards are distributed evenly among the SolrCloud instances. The SolrCloud instance and Greenplum segment are not required to be on the same host.

High Availability for GPText Indexes

SolrCloud provides high availability by maintaining replicas of shards and providing automatic failover if a shard fails or becomes unavailable. One replica of each shard is the *lead* replica and any changes to it are applied to the other replicas. The replication factor, which determines the number of replicas to maintain for each shard, is set when the index is created. Replicas may also be added or dropped later using GPText UDFs or command-line utilities.

ZooKeeper determines the locations of shard replicas among the Solr nodes and hosts. When adding a replica using a GPText UDF or command-line utility, a new shard can be explicitly placed on a SolrCloud instance.

GPText Sample Use Case

Forensic financial analysts need to locate communications among corporate executives that point to financial malfeasance in their firm. The analysts use the following workflow:

1. Load the email records into a Greenplum database.
2. Create a Solr index of the email records.
3. Run queries that look for text strings and their authors.
4. Refine the queries until they pair a dummy company name with top three or four executives corresponding about suspect offshore financial transactions. With this data, the analysts can focus the investigation on specific individuals rather than the thousands of authors in the initial data sample.

GPText Workflow

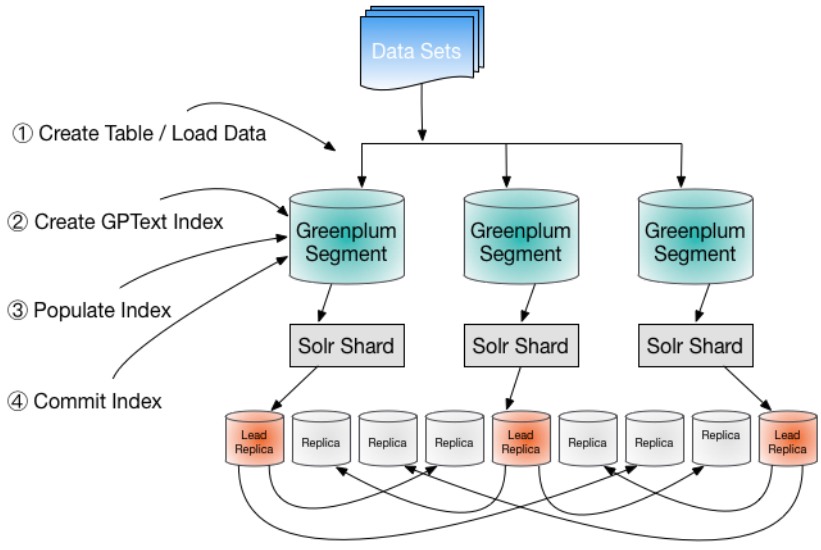
GPText works with Greenplum Database and Apache SolrCloud to store and index big data for information retrieval (query) purposes. High-level workflows include data loading and indexing, and data querying.

This topic describes the following information:

- Data Loading and Indexing Workflow
- Querying Data Workflow

Data Loading and Indexing Workflow

The following diagram shows the GPText workflow for loading and indexing data.



All client interaction with the system is through the Greenplum master instance.

1. Load data into your Greenplum Database system.
   Create a database table to hold data and then add the data to the table. Greenplum provides parallel data loading utilities and protocols that help to transform and load external data in various formats and from various sources. For details, see the *Greenplum Database Administrator Guide*, at http://gpdb.docs.pivotal.io.
2. Create an empty GPText index.
   Use the `gptext.create_index()` user-defined function (UDF) to create an empty GPText index for the table. Each Greenplum segment will manage a slice of the index, called a *shard*. SolrCloud creates multiple replicas for each shard, distributed among the Solr instances, and chooses a lead replica for

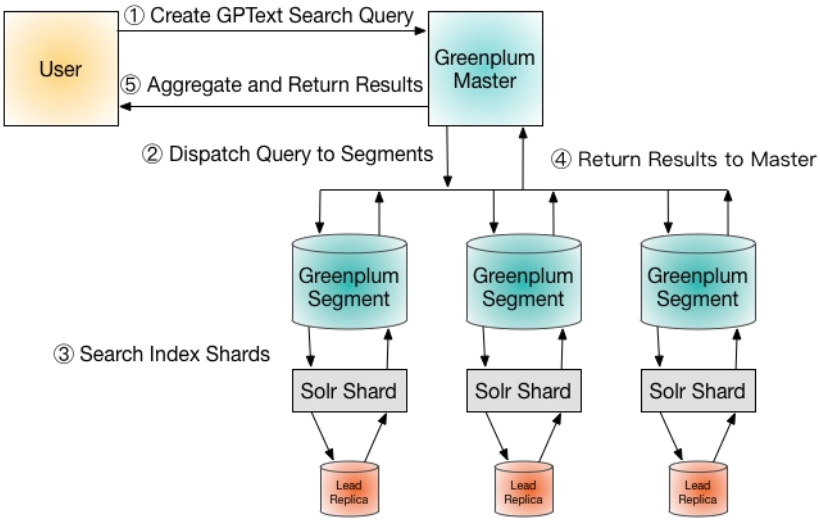
the Greenplum segment to operate upon. Solr manages replication between the replicas.

3. Populate the index with data from the database table.
   Use the `gptext.index()` UDF to add data to the index. This UDF works by dispatching a SQL query to execute on each Greenplum segment. The segments execute the query and add the results to their shards using Solr APIs.

4. Commit changes to the index.
   Commit changes to the GPText index by calling the `gptext.commit_index()` UDF. Until the changes are committed, queries executed on the index cannot access any data added to the index with `gptext.index()`. If needed, uncommitted changes can be rolled back. SolrCloud replicates changes committed to the lead replica to the shards' non-lead replicas.

Querying Data Workflow

The following diagram shows the high-level GPText query process workflow:



1. A user submits a SQL query designed to search the indexed data.
   A GPText search query is a SQL `SELECT` statement on a GPText search UDF that contains full-text search expressions.

2. The Greenplum master dispatches the query to the Greenplum segments.

3. Each segment executes the query, using the Solr API to search its index shard.
   SolrCloud executes the search query on the lead replica for the shard.

4. The Greenplum segments return the results of the search query to the Greenplum master.

5. The Greenplum master aggregates the results from all segments and returns them to the client.

Text Analysis

GPText enables analysis of Solr indexes with Apache MADlib (incubating), an open source library for scalable in-database analytics. MADlib began as a collaboration between the University of California, Berkeley and EMC/Greenplum. MADlib provides data-parallel implementations of mathematical, statistical, and machine learning methods for structured and unstructured data. You can use GPText to perform a variety of MADlib analyses.

Learn more about Apache MADlib (incubating) at http://madlib.incubator.apache.org. A `gppkg` package for MADlib is available on the Pivotal network at http://network.pivotal.io.

Working With Indexes

Indexing is key to preparing documents for text analysis and to achieving the best possible query performance. How you set up and configure indexes can affect the success of your project.

Creating GPText Indexes and Indexing Data

The general steps for creating and using a GPText index are:

1. Add the GPText schema to a database
2. Create an empty Solr index
3. Map Greenplum Database data types to Solr data types
4. Populate the index
5. Commit the index

After you complete these steps, you can create and execute a search query or implement machine learning algorithms.

The examples in this section use a table called `articles` in a database called `wikipedia` that was created with a default `public` schema.

The `articles` table has five columns: `id`, `date_time`, `title`, `content`, and `references`. The type of the `id` column must be `bigint` or `int8`.

The `content` column is the default search column—the column that will be searched if we do not name a different one in the search query.

See the *GPText Function Reference* for details about GPText functions.

Add the GPText schema to a database

Before you can use GPText with a Greenplum Database, the `gptext` schema must be installed. Use the `gptext-installsql` utility to perform this one-time task:

```
$ gptext-installsql wikipedia
```

To open an interactive shell for executing queries on the `wikipedia` database, execute:

```
$ psql wikipedia
```

Create an empty Solr index

To create a new index, use the function `gptext.create_index(schema_name, table_name, id_col_name, default_search_col_name, if_check_id_uniqueness)`.

There can be only one index per database table.

Example:

```
=# SELECT * FROM gptext.create_index('public', 'articles', 'id', 'content', true);
INFO:  Created index wikipedia.public.articles
 create_index
--------------
 t
(1 row)
```

This creates an index named `wikipedia.public.articles`.

To verify the index is created, enter the following query:

```
=# SELECT * FROM gptext.index_status('wikipedia.public.articles');
 content_id |       index_name        | shard_name | shard_state | replica_name | replica_state |                core                |       node_name
    |           base_url            | is_leader
------------+-------------------------+------------+-------------+--------------+---------------+------------------------------------+----------------------
----+-------------------------------+-----------
          0 | wikipedia.public.articles | shard0     | active      | core_node2   | active        | wikipedia.public.articles_shard0_replica2 | 192.0.2.217:18984_s
olr | http://192.0.2.217:18984/solr | f
          0 | wikipedia.public.articles | shard0     | active      | core_node4   | active        | wikipedia.public.articles_shard0_replica3 | 192.0.2.217:18983_s
olr | http://192.0.2.217:18983/solr | f
          0 | wikipedia.public.articles | shard0     | active      | core_node6   | active        | wikipedia.public.articles_shard0_replica1 | 192.0.2.217:18983_s
olr | http://192.0.2.217:18983/solr | t
          1 | wikipedia.public.articles | shard1     | active      | core_node1   | active        | wikipedia.public.articles_shard1_replica1 | 192.0.2.217:18984_s
olr | http://192.0.2.217:18984/solr | f
          1 | wikipedia.public.articles | shard1     | active      | core_node3   | active        | wikipedia.public.articles_shard1_replica3 | 192.0.2.217:18984_s
olr | http://192.0.2.217:18984/solr | f
          1 | wikipedia.public.articles | shard1     | active      | core_node5   | active        | wikipedia.public.articles_shard1_replica2 | 192.0.2.217:18983_s
olr | http://192.0.2.217:18983/solr | t
(6 rows)
```

This example executed on a Greenplum Database cluster with two primary segments. Two shards were created, one for each segment, and each shard has three replicas. The replicas are named core_node1 through core_node6.

You can also run the `gptext-state` command line utility at the command line to verify the index was created.

Map Greenplum Database data types to Solr data types

If a Greenplum Database data type is an array it is mapped to a multi-value type in Solr. For example, `INT[]` maps to a multi-value `int` field.

GPDB Type	Solr Type
`bool`	`boolean`
`bytea`	`binary`
`char`	`string`
`name`	`string`
`int8`	`long`
`int4`	`int`
`int2`	`int`
`int`	`int`
`text`	`text`
`point`	`point`
`float4`	`float`
`float8`	`double`
`money`	`string`
`bpchar`	`string`
`varchar`	`text`
`interval`	`string`
`date`	`tdate`
`time`	`string`
`timestamp`	`tdate`
`timetz`	`string`
`timestamptz`	`tdate`
`bit`	`string`
`varbit`	`string`
`numeric`	`double`

If a Greenplum Database data type is not listed, it will be `text` type in Solr.

Populate the index

To populate the index, use the table function `gptext.index()`, which has the following syntax:

```
SELECT * FROM gptext.index(TABLE(SELECT * FROM table_name), index_name);
```

For example:

```
=# SELECT * FROM gptext.index(TABLE(SELECT * FROM articles), 'wikipedia.public.articles');
 dbid | num_docs
------+----------
    3 |        3
    2 |        3
(2 rows)
```

The results of this command show that three documents from each segment were added to the index.

Considerations and best practices when indexing documents

The arguments to the `gptext.index()` function must be expressions.

`TABLE(SELECT * FROM articles)` creates a "table-valued expression" from the articles table, using the table function `TABLE`.

Choose the data to index or update by changing the inner select list in the query to select the columns and rows you want to index. When adding new documents to an existing index, for example, specify a `WHERE` clause in the `gptext.index()` call to choose only the new documents to index.

Be careful about distribution policies.

The first parameter of `gptext.index()` is `TABLE(SELECT * FROM articles)`. The query in this parameter should have the same distribution policy as the table you are indexing. However, there are two cases where the query will not have the same distribution policy:

1. Your query is a join of two tables
2. You are indexing an intermediate table (staging table) that is distributed differently than the final table.

When the distribution policies differ, you must specify "SCATTER BY" for the query:

```
TABLE(SELECT * FROM articles SCATTER BY distrib_id)
```

where `distrib_id` is the distribution id used when you created your primary/final table.

Consider how GPText search results will be used.

You could, for example, index just the ID column and text columns that are targets of search queries, and then use the ID column to join with other tables in Greenplum Database. An alternative for some applications is to allow GPText to index and store all of the columns, even those that will not be searched, so that results from search queries are complete and do not have to be joined in Greenplum Database.

Commit the index

After you create and populate an index, you must commit the index using `gptext.commit_index(index_name)`.

For example:

```
=# SELECT * FROM gptext.commit_index('wikipedia.public.articles');
 commit_index
--------------
 t
(1 row)
```

Note: The index picks up any new data added since your last index commit when you call this function.

Working With indexes

This topic describes how you can work with indexes. In particular, it covers the following tasks:

Configuring an index

You can modify your indexing behavior globally by using the `gptext-config` utility to edit a set of index configuration files. The files you can edit with `gptext-config` are:

- `solrconfig.xml` – Contains most of the parameters for configuring Solr itself (see http://wiki.apache.org/solr/SolrConfigXml).
- `managed-schema` – Defines the analysis chains that Solr uses for various different types of search fields (see Text Analysis Chains).
- `stopwords.txt` – Lists words you want to eliminate from the final index.
- `protwords.txt` – Lists protected words that you do not want to be modified by the analysis chain. For example, *iPhone*.
- `synonyms.txt` – Lists words that you want replaced by synonyms in the analysis chain.
- `elevate.xml` – Moves specific words to the top of your final index.
- `emoticons.txt` – Defines emoticons for the `text_sm` social media analysis chain. (see The emoticons.txt file).

You can also use `gptext-config` to move files.

Optimizing an index

The function `gptext.optimize_index(index_name, max_segments)` merges all segments into a small number of segments (max_segments) for increased efficiency.

Example:

```
=# SELECT * FROM gptext.optimize_index('wikipedia.public.articles', 10);
 optimize_index
----------------
 t
(1 row)
```

Deleting from an index

You can delete from an index using a query with the function `gptext.delete(index_name, query)`. This will delete all documents that match the search query. To delete all documents, use the query `'*:*'`.

After a successful deletion, you must issue a `gptext.commit_index(index_name)`.

Example that deletes all documents containing `"sports"` in the default search field:

```
=# SELECT * FROM gptext.delete('wikipedia.public.articles', 'sports');
 delete
--------
 t
(1 row)
```

```
SELECT * FROM gptext.commit_index('wikipedia.public.articles');
```

Example that deletes all documents from the index:

```
SELECT * FROM gptext.delete('wikipedia.public.articles', '*:*');
```

```
SELECT * FROM gptext.commit_index('wikipedia.public.articles');
```

Dropping an index

You can completely remove an index with the `gptext.drop_index(schema_name, table_name)` function.

Example:

```
SELECT * FROM gptext.drop_index('public', 'articles');
```

Listing all indexes

You can list all indexes in the GPText cluster using the `gptext-state` command-line utility. For example:

```
$ gptext-state
20160601:15:32:13:011986 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Check GPText cluster status...
20160601:15:32:13:011986 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Current GPText Version: 2.0.0
20160601:15:32:15:011986 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-All nodes are up and running.
20160601:15:32:15:011986 gptext-state:gpdb-sandbox:gpadmin-[INFO]:--------------------------------------------------
20160601:15:32:15:011986 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Index state.
20160601:15:32:15:011986 gptext-state:gpdb-sandbox:gpadmin-[INFO]:--------------------------------------------------
20160601:15:32:15:011986 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-   database    index_name                state
20160601:15:32:15:011986 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-   wikipedia   wikipedia.public.articles   Green
```

About GPText Field Types and Analyzer Chains

When you create a GPText index, Solr creates a set of configuration files for the index. The `managed-schema` file contains definitions for fields, field types, and analyzers. This section is an overview of the contents of the `managed-schema` file and the relationships between the XML elements in the file. For more detailed documentation, refer to the comments in the file or the Apache SolrCloud documentation.

You can edit configuration files using the `gptext-config` utility. For example, the following command loads the `managed-schema` file into an editor:

```
$ gptext-config -i <index-name> -f managed-schema
```

GPText adds `field` elements to the `managed-schema` file for columns included when the index was created with the `gptext.create_index()` function. For example, this is the definition for a text field named `description`:

```
<field name="description" stored="false" type="text_intl" indexed="true"/>
```

- The `name` attribute is the name of the database column. If the column name is not a valid Solr field name, it is altered to conform.
- The `stored` attribute determines if the content of the field will be stored in the index. If the field is stored in the index, GPText search results can return the content of the field. If the attribute is not stored, retrieving the field content requires a SQL join.
- The `type` attribute maps the Greenplum Database type to a Solr type, defined in the same file with a `<fieldType>` element.
- The `indexed` attribute determines whether the field content will be indexed.

The `<field>` element can have additional attributes used with some types. See the comment after the `<fields>` element for a complete list of attributes.

Field Types

The `type` attribute of the `<field>` element determines how Solr stores the field in indexes. The `class` attribute maps the field type to a Solr Java class that recognizes and processes the data type. Solr includes many base field types. See Map Greenplum Database data types to Solr data types for a mapping of Solr types to Greenplum Database types. If you have a custom type, you can add new field types by implementing Solr Java type interfaces, or you can specify an existing base type and customize it with an analyzer chain, as described in the next section.

Analyzer Chains

Text analysis chains determine how Solr indexes text fields in a document. An analyzer chain is a set of Java classes that tokenize and filter the content before it is added to the index. Different analysis chains can be defined for indexing and querying operations.

Field analysis begins with a tokenizer that divides the contents of a field into tokens. In Latin-based text documents, the tokens are words (also called *terms*). In Chinese, Japanese, and Korean (CJK) documents, the tokens are characters.

The tokenizer can be followed by one or more filters executed in succession. Filters restrict the query results, for example, by removing unnecessary terms (“a”, “an”, “the”), converting term formats, or by performing other actions to ensure that only important, relevant terms appear in the result set. Each filter operates on the output of the tokenizer or filter that precedes it. Solr includes many tokenizers and filters that allow analyzer chains to process different character sets, languages, and transformations. See Analyzers, Tokenizers and Filters - The full list for a comprehensive list.

Field types are assigned analyzers in an index’s `managed-schema` file. The following example shows the Solr `text` field type specification:

```
<fieldType name="text" class="solr.TextField" positionIncrementGap="100" autoGeneratePhraseQueries="true">
  <analyzer type="index">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <!-- in this example, we will only use synonyms at query time
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="false"/>
    -->
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
    <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1" generateNumberParts="1" catenateWords="1" catenateNumbers="1" catenateAll="0" splitOnCaseChange="1"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
    <filter class="solr.PorterStemFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
    <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1" generateNumberParts="1" catenateWords="0" catenateNumbers="0" catenateAll="0" splitOnCaseChange="1"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
    <filter class="solr.PorterStemFilterFactory"/>
  </analyzer>
</fieldType>
```

An analyzer has only one tokenizer, `solr.WhitespaceTokenizerFactory` in this example. The tokenizer can be followed by one or more filters executed in succession.

Filters restrict the query results. Each filter operates on the output of the tokenizer or filter that precedes it. For example, the `solr.StopFilterFactory` filter removes unnecessary terms (“a”, “an”, “the”) from the stream of tokens. The words to filter out of the stream are listed in the `stopwords.txt` configuration file. You can edit the `stopwords.txt` file with the `gptext-config` utility to change the list of words excluded from the index.

There are separate analyzer types for index and query operations. The query analyzer chain in this example includes a `solr.SynonymFilterFactory` that looks up each token in a file `synonyms.txt` and, if found, returns the synonym in place of the token.

The analysis chain can include a “stemmer”, `solr.PorterStemFilterFactory` in this example. The stemmer employs an algorithm to change words to their “stems”. For example, “confidential”, “confidentiality”, and “confidentis” are all stemmed to “confidenti”. Using a stemmer can dramatically reduce the size of the index, but users executing searches should be aware that some search expressions will not work as expected because of stemming. For example, searching with a wildcard such as `"confidential*"` will return no matches because the words were stemmed to “confidenti” during indexing. Without a wildcard, the word in the search expression is also stemmed and therefore the search succeeds.

GPText Text Analysis Chains

In addition to the text analysis chains Solr provides, GPText provides the following text analysis chains:

- text_intl, the International Text Analyzer
- text_sm, the Social Media Text Analyzer

text_intl, the International Text Analyzer

`text_intl` is the default GPText analyzer. It is a multiple language text analyzer for `text` fields. It handles Latin-based words and Chinese, Japanese, and Korean (CJK) characters.

`text_intl` processes documents as follows.

1. Separates CJK characters from other language text.
2. Identifies currency tokens or symbols that were ignored in the first pass.

3. For any CJK characters, generates a bigram for the CJK character and, for Korean characters only, preserves the original word.

Note that CJK and non-CJK text are treated as separate tokens. Preserving the original Korean word increases the number of tokens in a document.

Following is the definition from the Solr `managed-schema` template.

```
<fieldType autoGeneratePhraseQueries="true" class="solr.TextField"
        name="text_intl" positionIncrementGap="100">

 <analyzer type="index">
  <tokenizer class="com.emc.solr.analysis.worldlexer.WorldLexerTokenizerFactory"/>
  <filter class="solr.CJKWidthFilterFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="com.emc.solr.analysis.worldlexer.WorldLexerBigramFilterFactory" han="true"
        hiragana="true" katakana="true" hangul="true" />
  <filter class="solr.StopFilterFactory" enablePositionIncrements="true"
        ignoreCase="true" words="stopwords.txt"/>
  <filter class="solr.KeywordMarkerFilterFactory"
        protected="protwords.txt"/>
  <filter class="solr.PorterStemFilterFactory"/> </analyzer>
 <analyzer type="query">
  <tokenizer class="com.emc.solr.analysis.worldlexer.WorldLexerTokenizerFactory"/>
  <filter class="solr.CJKWidthFilterFactory"/>
  <filter class="com.emc.solr.analysis.worldlexer.WorldLexerBigramFilterFactory" han="true"
        hiragana="true" katakana="true" hangul="true" />
  <filter class="solr.StopFilterFactory" enablePositionIncrements="true" ignoreCase="true"
        words="stopwords.txt"/>
  <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>
  <filter class="solr.PorterStemFilterFactory"/>
 </analyzer>
</fieldType>
```

Following are the analysis steps for `text_intl`.

1. The analyzer chain for indexing begins with a tokenizer called `WorldLexerTokenizerFactory`. This tokenizer handles most modern languages. It separates CJK characters from other language text and identifies any currency tokens or symbols.
2. The `solr.CJKWidthFilterFactory` filter normalizes the CJK characters based on character width.
3. The `solr.LowerCaseFilterFactory` filter changes all letters to lower case.
4. The `WorldLexerBigramFilterFactory` filter generates a bigram for any CJK characters, leaves any non-CJK characters intact, and preserves original Korean-language words. Set the `han`, `hiragana`, `katakana`, and `hangul` attributes to `"true"` to generate bigrams for all supported CJK languages.
5. The `solr.StopFilterFactory` removes common words, such as “a”, “an”, and “the”, which are listed in the `stopwords.txt` configuration file (see To configure an index). If there are no words in the `stopwords.txt` file, no words are removed.
6. The `solr.KeywordMarkerFilterFactory` marks the English words to protect from stemming, using the words listed in the `protwords.txt` configuration file (see To configure an index). If `protwords.txt` does not contain a list of words, all words in the document are stemmed.
7. The final filter is the stemmer, in this case `solr.PorterStemFilterFactory`, a fast stemmer for the English language.

Note: The `text_intl` analyzer chain for querying is the same as the `text` analyzer chain for indexing.

An analysis chain, `text`, is included in GPText’s Solr `managed-schema` and is based on Solr’s default analysis chain. Because its tokenizer splits on white space, `text` cannot process CJK languages: white space is meaningless for CJK languages. Best practice is to use the `text_intl` analyzer.

For information about using an analyzer chain other than the default, see Using the text_sm Social Media Analyzer.

GPText Language Processing

The root-level tokenizer, `WorldLexerTokenizerFactory`, tokenizes international languages, including CJK languages. `WorldLexerTokenizerFactory` tokenizes languages based on their Unicode points and, for Latin-based languages, white space.

Note: Unicode is the encoding for all text in the Greenplum Database.

The following are sample input to, and output from, GPText. Each line in the output corresponds to a term.

English and CJK input:

- ₩10 대부분 english자선 단체는.

English and CJK output:

- ₩10
- 대부분
- 대부
- 부분
- english
- 자선
- 단체는
- 단체
- 체는

Bulgarian input:

- Състав на парламента: вж. протоколи

Bulgarian output:

- състав
- на
- парламента
- вж
- протоколиа

Danish input:

- Genoptagelse af sessionen

Danish output:

- genoptagelse
- af
- sessionen

text_intl Filters

The text_intl analyzer uses the following filters:

- The `CJKWidthFilterFactory` normalizes width differences in CJK characters. This filter normalizes all character widths to fullwidth.
- The `WorldLexerBigramFilterFactory` filter forms bigrams (pairs) of CJK terms that are generated from `WorldLexerTokenizerFactory`. This filter does not modify non-CJK text.
  `WorldLexerBigramFilterFactory` accepts attributes that guide the creation of bigrams for CJK scripts. For example, if the input contains HANGUL script but the `hangul` attribute is set to `false`, this filter will not create bigrams for that script. To ensure that `WorldLexerBigramFilterFactory` creates bigrams as required, set the CJK attributes `han`, `hiragana`, `katakana`, and `hangul` to `true`.

text_sm, the Social Media Text Analyzer

The GPText `text_sm` text analyzer analyzes text from sources such as social media feeds. `text_sm` consists of a tokenizer and two filters. To configure the `text_sm` text analyzer, use the `gptext-config` utility to edit the `managed-schema` file. See To use the text_sm Social Media Analyzer for details.

`text_sm` normalizes emoticons: it replaces emoticons with text using the `emoticons.txt` configuration file. For example, it replaces a happy face emoticon, `:-)`, with the text “happy”.

The following is the definition from the Solr `managed-schema` template.

```
<fieldType autoGeneratePhraseQueries="true"
          class="solr.TextField" name="text_sm"
          positionIncrementGap="100" termVectors="true"
          termPositions="true" termOffsets="true">
 <analyzer type="index">
 <tokenizer class =
        "com.emc.solr.analysis.text_sm.twitter.TwitterTokenizerFactory"
        delimiter="\t"
        emoticons="emoticons.txt"/>
<!-- Case insensitive stop word removal.
  Add enablePositionIncrements=true in both the index and query
 analyzers to leave a 'gap' for more accurate phrase queries. -->
   <filter class="solr.StopFilterFactory"
       enablePositionIncrements="true" ignoreCase="true"
       words="stopwords.txt"/>
 <filter class="solr.LowerCaseFilterFactory"/>
 <filter class="solr.KeywordMarkerFilterFactory"
       protected="protwords.txt"/>
 <filter class =
      "com.emc.solr.analysis.text_sm.twitter.EmoticonsClassifierFilterFactory"
      delimiter="\t" emoticons="emoticons.txt"/>
 <filter class =
      "com.emc.solr.analysis.text_sm.twitter.TwitterStemFilterFactory"/>
 <analyzer type="query">
 <tokenizer class =
         "com.emc.solr.analysis.text_sm.twitter.TwitterTokenizerFactory"
         delimiter="\t"
         emoticons="emoticons.txt"
         />
 <filter class="solr.StopFilterFactory"
       enablePositionIncrements="true" ignoreCase="true"
       words="stopwords.txt"/>
 <filter class="solr.LowerCaseFilterFactory"/>
 <filter class="solr.KeywordMarkerFilterFactory"
       protected="protwords.txt"/>
 <filter class =
      "com.emc.solr.analysis.text_sm.twitter.EmoticonsClassifierFilterFactory"
      delimiter="\t"
      emoticons="emoticons.txt"/>
 <filter class =
      "com.emc.solr.analysis.text_sm.twitter.TwitterStemFilterFactory"/>
 </analyzer>
</fieldType>
```

The TwitterTokenizer

The Twitter tokenizer extends the English language tokenizer, `solr.WhitespaceTokenizerFactory,` to recognize the following elements as terms.

- Emoticons
- Hyperlinks
- Hashtag keywords (for example, #keyword)
- User references (for example, @username)
- Numbers
- Floating point numbers
- Numbers including commas (for example 10,000)
- time expressions (for example, 9:30)

The text_sm filters

`com.emc.solr.analysis.socialmedia.twitter.EmoticonsClassifierFilterFactory` classifies emoticons as `happy` , `sad` , or `wink` . It is based on the `emoticons.txt` file (one of the files you can edit with `gptext-config`), and is intended for future use, such as in sentiment analysis.

The TwitterStemFilterFactory

`com.emc.solr.analysis.socialmedia.twitter.TwitterStemFilterFactory` extends the `solr.PorterStemFilterFactory` class to bypass stemming of the social media patterns recognized by the `twitter.TwitterTokenizerFactory` .

The emoticons.txt file

This file contains lists of emoticons for “happy,” “sad,” and “wink.” They are separated by a tab by default. You can change the separation to any character or string by changing the value of `delimiter` in the social media analysis chain. The following is a sample line from the `text_sm` analyzer chain:

```
<filter class =
      "com.emc.solr.analysis.text_sm.twitter.EmoticonsClassifierFilterFactory"
      delimiter="\t" emoticons="emoticons.txt"/>
```

Using the text_sm Social Media Analyzer

The Solr `managed-schema` file specifies the analyzer to use to index a field. The default analyzer is `text_intl`. To specify the `text_sm` social media analyzer, you use the `gptext-config` utility to modify the Solr `managed-schema` for your index.

The steps are:

1. Create an index using `gptext.create_index()`.
   The index contains a line similar to the following near the end of the file:

   ```
   <field name="text_search_col" indexed="true" stored="false"
   type="text_intl"/>
   ```

   The `type` field specifies the analyzer to use. `text_intl` is the default.

2. Use `gptext-config` to edit the `managed-schema` file:

   ```
   gptext-config -f managed-schema -i <index_name>
   ```

3. Modify the line as follows:

   ```
   <field name="text_search_col" indexed="true" stored="false" type="text_sm"/>
   ```

gptext-config fetches the `managed-schema` file from the configuration files directory for your index and opens it in the `vi` editor. After you edit the file, save it, and quit `vi`, gptext-config returns the file to its original configuration files directory.

Storing Additional Fields in an Index

Solr can store additional fields in the index when it indexes a document and then return the fields in search results. You can avoid an expensive join in Greenplum Database if you store all of the required columns as fields in the Solr index.

GPText packs the additional fields in a buffer added to the search results. Individual fields can be retrieved from this buffer using the `gptext.gptext_retrieve_field()`, `gptext.gptext_retrieve_field_int()`, and `gptext.gptext_)retrieve_field_float()` functions.

Follow these steps to configure the index to store the the additional fields.

1. Use `gptext-config` to edit the managed-schema to determine which columns you would like stored.
   `gptext-config -i <index_name> -f managed-schema`

2. Change the `<stored=BOOL>` value of the additional fields you want to store from `FALSE` to `TRUE`.

3. Reindex and commit the index.

See Retrieving Additional Stored Fields for information about adding additional stored fields to the results of a GPText query.

Using Multiple Analyzer Chains

If you want to index a field using two different analyzer chains simultaneously, you can do this:

Create a new empty index. Then use the `gptext-config` utility to add a new field to the index that is a copy of the field you are interested in, but with a different name and analyzer chain.

Let us assume that your index, as initially created, includes a field to index named `mytext`. Also assume that this field will be indexed using the default international analyzer (`text_intl`).

You want to add a new field to the index's `managed-schema` that is a copy of `mytext` and that will be indexed with a different analyzer (say the `text_sm` analyzer). To do so, follow these steps:

1. Create an empty index with `gptext.create_index()`.
2. Open the index's `managed-schema` file for editing with `gptext-config`.
3. Add a `<field>` in the `managed-schema` for a new field that will use a different analyzer chain. For example:
   `<field indexed="true" name="mytext2" stored="false" type="text_sm"/>`
   By defining the type of this new field to be `text_sm`, it will be indexed using the social media analyzer rather than the default `text_intl`.
4. Add a `<copyField>` in `managed-schema` to copy the original field to the new field. For example:
   `<copyField dest="mytext2" source="mytext"/>`
5. Index and commit as you normally would.

The database column `mytext` is now in the index twice with two different analyzer chains. One column is `mytext`, which uses the default international analyzer chain, and the other is the newly created `mytext2,` which uses the social media analyzer chain.

Using Different Analyzer Chains for Individual Fields

You can use different analyzers for individual fields by editing the managed-schema configuration file. For example, if one field contains English text and another contains Chinese language text, you can specify different analyzers for the two fields.

Example

You have a table named `email_tbl` with the following definition:

```
create table email_tbl (
  id bigint,
  english_content text,
  chinese_content text,
  timestamp date,
  username text,
  age int,
  ... ) # additional columns that are not indexed
```

- You want to index the six columns shown— `id`, `english_content`, `chinese_content`, `timestamp`, `username`, and `age`.
- For the column `english_content`, you want to use the English language analyzer called "text_en" for the text segmentation.
- For the column `chinese_content`, you want to use the international language analyzer named "text_intl".

Here are steps to implement this example:

1. Create the GPText index for the table.

   ```
   SELECT * FROM gptext.create_index('public', 'email_tbl', 'id', 'english_content');
   ```

2. Modify the analyzer for each column in `managed-schema`.

   ```
   $ gptext-config -i db.public.email_tbl -f managed-schema
   ```

3. Find the element for the `english_content` field.

   ```
   <field name="english_content" type="*" indexed="true" stored="true" />
   ```

   Change the `type` attribute to `text_en`.

   ```
   <field name="english_content" type="text_en" indexed="true" stored="true" />
   ```

4. Find the element for the `chinese_content` field.

```
<field name="chinese_content" type="*" indexed="true" stored="true" />
```

   Change the `type` attribute to `text_intl`.

```
<field name="chinese_content" type="text_intl" indexed="true" stored="true" />
```

5. Index the table.

```
SELECT * FROM gptext.index(TABLE(SELECT id, english_content, chinese_content, timestamp, username, age FROM email_tbl),
'db.public.email_tbl');
```

6. Commit the index.

```
SELECT * FROM gptext.commit_index('db.public.email_tbl');
```

The field types `text_en` and `text_intl` are defined in `<fieldType>` entries in the managed-schema file and then referenced in the `type` attribute of the `<field>` element.

You can define a custom field type by adding a `<fieldType>` entry with custom analyzers and then setting the field's `type` attribute to the name of the custom field type. For example, the following "text_customize" field type is a copy of the "text_en" field type entry with the synonym filter commented out in the index analyzer. This custom field type will apply the synonym filter to queries, but not to the index.

```
<fieldType name="text_customize" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <!-- in this example, we will only use synonyms at query time
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="false"/>
    -->
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

A field type can also be customized by adding analyzers as child elements of the `<field>` element:

```
<field name="english_content" type="text" indexed="true" stored="false">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <!-- in this example, we will only use synonyms at query time
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="false"/>
    -->
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</field>
```

Queries

To retrieve data, you submit a query that performs a search based on criteria that you specify. Simple queries return straight-forward results. You can use the default query parser, or specify a different query parser at query time.

Creating a Simple Search Query

After a Solr index is committed, you can create simple queries with the `gptext.search()` function:

```
gptext.search(src_table, index_name, search_query, filter_queries[, options])
```

Example - Top 10 results, no filter query

```
SELECT w.id, w.date_time, w.title, q.score
FROM articles w,
gptext.search(TABLE(SELECT 1 SCATTER BY 1),
'wikipedia.public.articles' , 'Libya AND  (Gadaffi OR Kadafi OR
Qad*I)', null, 'rows=10') q
WHERE q.id = w.id;
```

Example - Top 100 results, no filter query

```
SELECT w.title, q.score FROM wikipedia w,
gptext.search(TABLE(SELECT 1 SCATTER BY 1),
'wikipedia.public.articles', 'solr search query', null, 'rows=100') q
WHERE w.id = q.id;
```

Example - All results, no filter query

```
SELECT w.title, q.score FROM wikipedia w,
gptext.search(TABLE(SELECT 1 SCATTER BY 1),
'wikipedia.public.articles', 'solr search query', null) q
WHERE w.id = q.id;
```

Example - Top 100 results, with filter query

```
SELECT w.title, q.score FROM wikipedia w,
gptext.search(TABLE(SELECT 1 SCATTER BY 1),
'wikipedia.public.articles', 'solr search query',
'solr filter query', 'rows=100') q
WHERE w.id = q.id;
```

Creating a Faceted Search Query

Faceting breaks query results into multiple categories and shows counts for each category.

With the `faceted_field_search()` function, the categories are field names. Here is the syntax for the `faceted_field_search()` function:

```
faceted_field_search(index_name, query, filter_queries, facet_fields, facet_limit, minimum)
```

In this example, the query runs on a set of social media feeds and eliminates spam and truncated fields:

```
SELECT * FROM
gptext.faceted_field_search('twitter.public.message', '*:*', null, '{spam, truncated}', 0, 0);
```

In this example, the query searches all tweets in the data set by users (`author_id` s) who have created at least five tweets:

```
SELECT * FROM
gptext.faceted_field_search('twitter.public.message', '*:*', null, '{author_id}', -1, 5);
```

Highlighting Search Terms in Query Results

The `gptext.highlight()` function highlights search terms by inserting markup tags before and after each occurrence of the term in results returned from a `gptext.search()` query. For example, if the search term is “iphone”, each occurrence of “iphone” in the data is marked up:

```
<em>iphone</em>
```

You can change the default markup strings from `<em>` and `</em>` by setting the `gptext.hl_pre_tag` and `gptext.hl_post_tag` server configuration parameters.

The index must have been created with terms enabled. Use `gptext.enable_terms()` to enable term vectors and the reindex the data if it was already indexed. See `gptext.enable_terms()` in the *Pivotal GPText User’s Guide.*

Important: Highlighting adds overhead to the query, including index space, indexing time, and search time.

The `gptext.highlight()` syntax is:

```
gptext.highlight(column_data, column_name, offsets)
```

The `column_data` parameter contains the text data that will be marked up with highlighting tags.

The `column_name` parameter is the name of the corresponding table column.

The `offsets` parameter is a `gptext hstore` type that contains key-value pairs that specify the locations of the search term in the text data. This value is constructed by the `gptext.search()` function when highlighting is enabled. The key contains the column name and the value is a comma-separated list of offsets where the data appears.

To enable highlighing in a gptext.search() query, add the `hl` and `hl.fl` options:

```
hl=true&hl.fl=field1,field2
```

Setting the `hl=true` option enables highlighting for the search. The `hl.fl` option specifies a list of the field names to highlight.

Example - Top 10 results, with search terms highlighted

```
SELECT t.id, gptext.highlight(t.message, 'message', s.hs)
FROM twitter.message t,
    gptext.search(TABLE(SELECT 1 SCATTER BY 1),
        'demo.twitter.message',
        '{!gptextqp}iphone', null,
        'rows=10&hl=true&hl.fl=message' ) s
WHERE t.id = s.id;
```

Retrieving Additional Stored Fields

A GPText index can be configured to store additional fields when documents are indexed with the `gptext.index()` function. The additional stored fields can be returned with search results so that it is unnecessary to join GPText search results with the original table in the Greenplum Database.

See Storing Additional Fields in an Index for instructions to configure the index to store the additional fields.

Retrieve the additional field values in a GPText search by specifying the list of fields in the `gptext.search()` options argument. For example:

```
SELECT *
FROM gptext.search(TABLE(SELECT 1 SCATTER by 1),
   'wikipedia.public.articles',
   '*:*', NULL, 'fl=revision,author');
```

This query returns all results and the `revision` and `author` fields for each result.

To retrieve all stored fields, use the `*` wildcard for the field list: `'fl=*'`.

In the search results, the requested fields are packed into an extra field named `rf` with the following format:

```
column_value { name: "_version" value: "1544714234398507008" } column_value { name: "revision" value: "9.70" } column_value { name: "author" value: "jdough" }
```

Use the `gptext.gptext_retrieve_field(rf,<column_name>)` function to retrieve a single field from this structure as a text value. For example:

```
gptext.gptext_retrieve_field(rf, 'author')
```

If the specified field name does not exist in the `rf` structure, the function returns `NULL`.

Use `gptext.gptext_retrieve_field_float()` to retrieve the column value converted to a float value or `gptext.gptext_retrieve_field_int()` to retrieve the column value converted to an integer.

Using Advanced Querying Options

When you submit a query, Solr processes the query using a query parser. There are several Solr query parsers with different capabilities. For example, the `ComplexPhraseQueryParser` can parse wildcards, and the `SurroundQueryParser` supports span queries: finding words in the vicinity of a search word in a document.

You can use the most appropriate parser for your query (see Changing the Query Parser at Query Time).

GPText supports these query parsers:

1. `QParserPlugin`, the default GPText query parser. `QParserPlugin` is a superset of the `LuceneQParserPlugin`, Solr's native Lucene query parser. `QParserPlugin` is a general purpose query parser with broad capabilities. `QParserPlugin` does not support span queries and handles operator precedence in an unintuitive manner. The support for field selection is also rather weak. See http://wiki.apache.org/solr/SolrQuerySyntax.
2. The `ComplexPhraseQueryParser`, which supports wildcards, ORs, ranges, and fuzzies inside phrase queries. See https://issues.apache.org/jira/browse/SOLR-1604.
3. The `SurroundQueryParser`, which supports the family of span queries. See http://wiki.apache.org/solr/SurroundQueryParser.
4. The `DisMax` (or `eDisMax`) Query Parser, which handles operator precedence in an intuitive manner and is best suited for user queries. See http://wiki.apache.org/solr/DisMaxQParserPlugin.
5. The Unified Query Parser can use all the query parsers in combination. See Using the Universal Query Parser for more information.

A good general reference for query parsers can be found at: http://www.lucidimagination.com/blog/2009/02/22/exploring-query-parsers.

Changing the Query Parser at Query Time

You can change the query parser at query time with the `defType` Solr option in the `gptext.search()` function that supports Solr options. For example, this query uses the `dismax` query parser to return the top 100 results for "olympics", with no filter query, and using the default search field:

Top 100 results, no filter query, default search field used:

```
SELECT w.title, q.score FROM wikipedia w,
gptext.search(TABLE(SELECT 1 SCATTER BY 1),
'wikipedia.public.articles', 'olympics', null,
'rows=100&defType=dismax') q
WHERE w.id = q.id;
```

The `options` parameter includes 'defType=dismax'.

Invoke the Universal Query Parser by inserting `{!gptextqp} <search query>` in the query parameter of the `gptext.search()` function.

Using the Solr localParams syntax

You can use the Solr `localParams` syntax with all GPText search functions to change the query parser at query time by replacing the `<query>` term with `'{!type=dismax}<query>'`.

Note: The default query parser is specified in the `requestHandler` definitions in `solrconfig.xml`. You can edit `solrconfig.xml` with the management utility `gptext-config`.

Using the Universal Query Parser

With the Universal Query Parser, you can perform searches using features of any of the query parsers in one search string. Invoke the Universal Query Parser with the gptext.search() function's `search_query` parameter in this format:

```
'{!gptextqp} search_query'
```

The following search illustrates using the Universal Query Parser:

```
SELECT w.title, q.score
FROM wikipedia w,
  gptext.search(TABLE(SELECT 1 SCATTER BY 1),'wikipedia.public.articles',
     '{!gptextqp} mobil* 2W (Obama OR Clinton)', null) q
WHERE w.id = q.id;
```

The search query in this example includes syntax for three query parsers:

- `mobil*` – Complex Query with wildcard
- `2W` – Proximity Query, requesting a maximum of two words proximity between the phrases
- `Obama OR Clinton` – Lucene Query

Administering GPText

GPText administration includes security considerations, monitoring Solr index statistics, and troubleshooting.

Changing GPText Server Configuration Parameters

Configuration parameters used with GPText are built-in to GPText with default values. You can change the values for these parameters by setting the new values in a Greenplum Database session. The new values are stored in ZooKeeper. GPText indexes use the values of configuration parameters when they are created. Changing configuration parameters affects new indexes, but does not affect existing indexes.

See GPText Configuration Parameters for a complete list of configuration parameters.

A one-time Greenplum Database configuration change is needed for Greenplum Database to allow setting and displaying GPText configuration variables. As the `gpadmin` user, enter the following commands in a shell:

```
$ gpconfig -c custom_variable_classes -v 'gptext'
$ gpstop -u
```

Then connect to a database that contains the GPText schema and execute the `gptext.version()` function to expose the GPText configuration variables:

```
twitter=# select * from gptext.version();
```

Change the values of GPText configuration variables using the `SET` command in a session with a database that contains the GPText schema. The following example sets values for three configuration parameters in a `psql` session:

```
twitter=# set gptext.idx_buffer_size=10485760;
SET
twitter=# set gptext.idx_delim='|';
SET
twitter=# set gptext.extension_factor=5;
SET
```

You can view the current value of a configuration parameter that you have set using the `SHOW` command:

```
twitter=# show gptext.idx_delim;
 gptext.idx_delim
------------------
 |
(1 row)
```

Security and GPText Indexes

GPText security is based on Greenplum Database security. Your privileges to execute GPText functions depend on your privileges for the database table that is the source for the index. For example, if you have SELECT privileges for a table in the Greenplum database, then you have SELECT privileges for an index generated from that table.

Executing GPText functions requires one of OWNER, SELECT, INSERT, UPDATE, or DELETE privileges, depending on the function. The OWNER is the person who created the table and has all privileges. See the *Greenplum Database Administrator Guide* for information about setting privileges.

Checking ZooKeeper Status

Use the zkManager utility from the command line to check the ZooKeeper cluster status. If the Zookeepr cluster is bound to GPText, you can start and stop the cluster using zkManager.

To check the ZooKeeper cluster status, run the following command:

```
zkManager state
```

The utility lists the hosts, ports, latency, and follower/leader mode for each ZooKeeper instance. If a node is down, it's mode is listed as Down.

If the ZooKeeper cluster was installed by the GPText installer, the zkManager utility can be used to start or stop the ZooKeeper cluster. To start the cluster, run the following command:

```
zkManager start
```

To stop ZooKeeper, run this command:

```
zkManager stop
```

Checking SolrCloud Status

You can check the status of the SolrCloud cluster and indexes by running the `gptext-state` utility from the command line.

To check the state of the GPText nodes and indexes, run the `gptext-state` utility with no options:

```
gptext-state
```

This command reports the status of the GPText nodes and status of each GPText index.

Run `gptext-state list` to view just the indexes.

The `gptext-state healthcheck` command checks the GPText configuration files, the index status, required disk space, user privileges, and index and database consistency. By default, the required disk space check passes if there is at least 20% disk free. You can set a different disk free threshold using the `--disk_free` option. For example:

```
[gpadmin@gpdb-sandbox ~]$ gptext-state healthcheck --disk_free=25
20160629:15:45:24:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Execute healthcheck on GPText cluster!
20160629:15:45:24:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Check GPText config files ...
20160629:15:45:24:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-GOOD
20160629:15:45:24:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Check GPText index status ...
20160629:15:45:25:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-GOOD
20160629:15:45:25:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Checking for required disk space...
20160629:15:45:25:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-GOOD
20160629:15:45:25:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Checking for required user privileges...
20160629:15:45:25:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-GOOD
20160629:15:45:25:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Checking for indexes and database consistency...
20160629:15:45:27:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-GOOD
20160629:15:45:27:669652 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Done.
```

See the `gptext-state` utility reference for additional options.

Recovering GPText Nodes

Use the `gptext-recover` utility to recover down GPText nodes, for example after a failed Greenplum segment host is recovered.

With no arguments, the `gptext-recover` utility discovers down GPText nodes and restarts them.

With the `-f` (or `--force`) option, if a GPText node cannot be restarted and no shards are down, the node is deleted and created again on the same host. Missing replicas are added and the failed node and failed replicas are removed.

The `-H` option allows recreating down GPText nodes on a new host that replaces a failed host. It forces down nodes to be deleted and recreated on the new host. If shards are down, it advises reindexing. If only some replicas are down, it recreates the replicas on the new host and updates `gptext.conf`.

Gathering Solr Index Statistics

You can gather Solr index statistics using the SQL function gptext.index_statistics(), or by running the `gptext-state` utility from the command line.

To list all GPText indexes, enter the following command at the command line:

```
gptext-state list
```

To obtain statistics for an index:

```
SELECT * FROM gptext.index_statistics('wikipedia.public.articles');
```

To obtain statistics for all indexes:

```
SELECT * FROM gptext.index_statistics(null);
```

A command line sample that retrieves all statistics for an index:

```
gptext-state --index wikipedia.public.articles
```

A command line sample that retrieves the number of documents in an index:

```
gptext-state --index wikipedia.public.articles --stats-columns num_docs
```

A command line sample that retrieves '`num_docs`' and the index `size`:

```
gptext-state --index wikipedia.public.articles
--stats-columns num_docs,size
```

A command line sample that retrieves all statistics for all indexes:

```
gptext-state
```

Backing Up and Restoring GPText Indexes

With the `gptext-backup` management utility, you can back up a GPText index to a shared directory so that, if needed, you can quickly recover from a failure. The backup can be restored to the same GPText system or to another system with the same number of Greenplum segments.

The `gptext-backup` management utility backs up an index and its configuration files to a shared file system, which must be accessible and writable by each host in the Greenplum cluster. The `--path` command-line option specifies the location of a directory on the mounted file system. The `--name` option provides a name for the backup.

The `gptext-backup` utility first checks that:

- the GPText cluster is up
- the shared file system is valid
- the directory specified with the `--name` option does not already exist at the location specified by the `--path` option

The utility creates the new directory and saves one copy of each index shard to that directory, along with the index's configuration files.

To restore an index, use the `gptext-restore` management utility. The GPText system you restore to must be on a Greenplum cluster with the same number of segments. The database, schema, and base table for the index must be present.

The `--index` option specifies the name of the GPText index that will be restored. If the index exists, you must first drop it with the `gptext.delete()` user-defined function.

The `--path` option specifies the location of the directory containing the backup files—the directory that `gptext-backup` created on the shared file system.

See gptext-backup for syntax and details for running `gptext-backup`. See gptext-restore for syntax and details for running `gptext-backup`.

Expanding the GPText Cluster

The `gptext-expand` management utility adds GPText nodes to the cluster. There are two ways to add nodes: - Add GPText nodes to existing hosts in the cluster. This option increases the number of GPText nodes on each host. - Add GPText nodes to new hosts added when using the Greenplum `gpexpand`

management utility to expand the Greenplum Database system.

Adding GPText Nodes to Existing Segment Hosts

To add nodes to existing segment hosts, run the `gptext-expand` utility with a command like the following:

```
gptext-expand -e -p /data1/nodes, /data2/nodes
```

This example adds two GPText nodes to each host.

The `-e` (`--existing`) option specifies that nodes are to be added to existing hosts.

The `-p` (`--expand_paths`) option provides a list of directories where the new nodes' data directories are to be created. These should be the same directories that contain the Greenplum segment data directories and existing GPText data directories. The number of directories in the list is the number of new nodes that are added.

A directory can be repeated in the directory list multiple times to increase the number of new GPText nodes to create. For example, if there is currently one GPText node per host in the `/data1/nodes` directory, you could add three nodes with a command like the following:

```
gptext-expand -e -p /data1/nodes, /data2/nodes, /data2/nodes
```

This adds one node to the `/data1/nodes` directory and two nodes to the `/data2/nodes` directory so there are two GPText nodes in each directory.

Adding GPText nodes affects new indexes, but not existing indexes. Replicas for new indexes will be distributed across all of the nodes, including both old nodes and the newly created nodes. Replicas for indexes that existed before running `gptext-expand` are not automatically moved. Rebalancing existing replicas requires reindexing.

Adding GPText Nodes to New Hosts

To add GPText nodes to hosts after expanding the Greenplum cluster with the `gpexpand` management utility, call `gptext-expand` with the name of database containing the gpexpand schema, for example, if the gpexpand schema was created in the postgres database:

```
gptext-expand -d postgres
```

The `gptext-expand` utility installs GPText binaries on new hosts and then creates new GPText nodes on the new hosts.

Expanding a Greenplum cluster increases the number of segments, so the number of GPText index shards for existing indexes must be increased to equal the new number of segments. This requires reindexing documents for all existing documents. Newly created indexes will automatically be distributed among the new shards.

Troubleshooting

GPText errors are of the following types:

- Solr errors
- `gptext` errors

Most of the Solr errors are self-explanatory.

`gptext` errors are caused by misuse of a function or utility. They provide a message that tells you when you have used an incorrect function or argument.

Monitoring Logs

You can examine the Greenplum Database and Solr logs for more information if errors occur. Greenplum Database logs reside in:

```
segment-directory/pg-log
```

Solr logs reside in:

```
<GPDB path>/solr/logs
```

Determining Segment Status with gptext-state

Use the `gptext-state` utility to determine if any primary or mirror segments are down. See `gptext-state` in the *GPText Function Reference*.

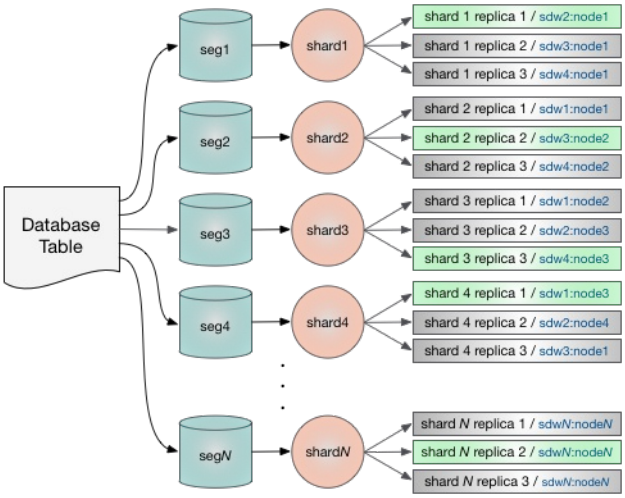
GPText High Availability

The GPText high availability feature ensures that you can continue working with GPText indexes as long as each shard in the index has at least one working replica.

A GPText index has one shard for each Greenplum segment, so there is a one-to-one coorespondence between Greenplum segments and GPText index shards. The shard managed by a Greenplum segment is an index of the documents that are managed by that segment.

The GPText high availability mechanism is to maintain multiple copies, or replicas, of the shard. The ZooKeeper service that manages SolrCloud chooses a GPText instance (SolrCloud node) for each replica to ensure even distribution and high availability. For each shard, one replica is elected *leader* and the Greenplum segment associated with the shard operates on this leader replica. The GPText instance managing the lead replica may or may not be on another Greenplum host, so indexing and searching operations are passed over the Greenplum cluster's interconnect network. SolrCloud replicates changes made to the leader replica to the remaining replicas.

The following figure illustrates the relationships between Greenplum segments and GPText index shards and replicas. The leader replica (green)



The number of replicas to create for each shard, the *replication factor*, is a SolrCloud property. By default, GPText starts SolrCloud with a replication factor of three. The replication factor for each individual index is the value of the SolrCloud replication factor when the index is created. Changing the replication factor does not alter the replication factor for existing indexes.

Greenplum Segment or Host Failure

If a Greenplum primary segment fails and its mirror is activated, GPText functions and utilities continue to access the leader replica. No intervention is needed.

If a host in the cluster fails, both Greenplum and GPText are affected. Mirrors for the Greenplum primary segments located on the failed host are activated on other hosts. SolrCloud elects a new leader replica for affected shards. Because Greenplum segment mirrors and GPText shard replicas are distributed throughout the cluster, a single host failure should not prevent the cluster from continuing to operate. The performance of database queries and indexing operations will be affected until the failed host is recovered and the cluster is brought back into balance.

ZooKeeper Cluster Availability

SolrCloud is dependent on a working, available ZooKeeper cluster. For ZooKeeper to be active, a majority of the ZooKeeper cluster nodes must be up and able to communicate with each other. A ZooKeeper cluster with three nodes can continue to operate if one of the nodes fails, since two is a majority of three. To tolerate two failed nodes, the cluster must have at least five nodes so that the number of working nodes remaining after the failure are a majority. To tolerate n node failures, then, a ZooKeeper cluster must have $2n+1$ nodes. This is why ZooKeeper clusters usually have an odd number of nodes.

The best practice for a high-availability GPText cluster is a ZooKeeper cluster with five or seven nodes so that the cluster can tolerate two or three failed nodes.

Managing GPText Cluster Health

GPText document indexing and searching services remain available as long as each shard of an index has at least one working replica. To ensure availability in the event of a failure, it is important to monitor the status of the cluster and ensure that all of the index shard replicas are healthy. You can monitor the SolrCloud cluster and indexes using the SolrCloud Dashboard or using GPText functions and management utilities. Access the SolrCloud Dashboard with a web browser on any GPText instance with a URL such as `http://sdw3:18983/solr`. (The port numbers for GPText instances are set with the `GPTEXT_PORT_BASE` parameter in the installation parameters file at installation time.)

Refer to the Apache SolrCloud documentation for help using the SolrCloud Dashboard.

Monitoring the Cluster with GPText

The GPText `gptext-state` management utility allows you to query the state of the GPText cluster and indexes. You can also use `gptext.index_status()` to view the status of all indexes or a specified index.

To see the GPText cluster state run the `gptext-state` command-line utility with the `-d` option to specify a database that has the GPText schema installed.

```
gptext-state -d mydb
```

The utility reports any GPText nodes that are down and lists the status of every GPText index. For each index, the database name, index name, and status are reported. The status column contains “Green”, “Yellow”, or “Red”: - Green – all replicas for all shards are healthy - Yellow – all shards have at least one healthy replica but at least one replica is down - Red – no replicas are available for at least one index shard

To see the distribution of index shards and replicas in the GPText cluster, execute this SQL statement.

```
SELECT index_name, shard_name, replica_name, node_name
FROM gptext.index_summary()
ORDER BY node_name;
```

To list all GPText indexes, run the `gptext-state list` command.

```
gptext-state list -d mydb
```

The `gptext-state healthcheck` command checks the health of the cluster. The `-f` flag specifies the percentage of available disk space required to report a healthy cluster. The default is 10.

```
gptext-state healthcheck -f 20 -d mydb
```

See `gptext-state` in the Management Utilities reference for help with additional `gptext-state` options.

The gptext.index_status() user-defined function reports the status of all GPText indexes or a specified index.

```
SELECT * FROM gptext.index_status();
```

Specify an index name to report only the status of that index.

```
SELECT * FROM gptext.index_status('mydb.public.messages');
```

Adding and Dropping Replicas

The `gptext-replica` utility adds or drops a replica of a single index shard. Use the `gptext.add_replica()` and `gptext.delete_replica()` user-defined functions to perform the same tasks from within the database.

If a replica of a shard fails, use `gptext-replica` to add a new replica and then drop the failed replica to bring the index back to “Green” status.

```
gptext-replica add -i mydb.public.messages -s shard3
```

Here is the equivalent, using the `gptext.add_replica()` function:

```
SELECT * FROM gptext.add_replica('mydb.public.messages', shard3);
```

ZooKeeper determines where the replica will be located, but you can also specify the node where the replica is created:

```
gptext-replica add -i mydb.public.messages -s shard3 -n sdw3
```

In the `gptext.add_replica()` function, add the node name as a third argument.

To drop a replica, call `gptext.delete_replica()` with the name of the index, the name of the shard, and the name of the replica. You can find the name of the replica by calling `gptext.index_status(index_name)`. The name is in the format `core_noden`. An optional `-o` flag specifies that the replica is to be deleted only if it is down.

```
gptext-replica drop -i mydb.public.messages -s shard3 -r core_node4 -o
```

Here is the equivalent of the above command using the `gptext.delete_replica()` user-defined function.

```
SELECT * FROM
gptext.delete_replica('mydb.public.messages', 'shard3', 'cord_node4', true);
```

GPText Best Practices

Each GPText/Apache Solr node is a Java Virtual Machine (JVM) process and is allocated memory at startup. The maximum amount of memory the JVM will use is set with the `-Xmx` parameter on the Java command line. Performance problems and out of memory failures can occur when the nodes have insufficient memory.

This topic discusses three GPText use cases that stress Solr JVM memory in different ways and the best practices for preventing or alleviating performance problems.

- Indexing large numbers of documents in an index
- Indexing large documents
- Creating a large number of GPText indexes

Indexing Large Numbers of Documents

Indexing documents consumes data in Solr JVM memory. When the index is committed, parts of the memory are released, but some data remains in memory to support fast search. By default, Solr performs an automatic soft commit when 1,000,000 documents are indexed or 20 minutes (1,200,000 milliseconds) have passed. A soft commit pushes documents from memory to the index, freeing JVM memory. A soft commit also makes the documents visible in searches. A soft commit does not, however, make the index updates durable; it is still necessary to commit the index with the `gptext.commit()` user-defined function.

You can configure an index to perform a more frequent automatic soft commit by editing the `solrconfig.xml` file for the index:

```
$ gptext-config -f solrconfig.xml -i <db>.<schema>.<index-name>
```

The `<autoSoftCommit>` element is a child of the `<updateHandler>` element. Edit the `<maxDocs>` and `<maxTime>` values to reduce the time between automatic commits. For example, the following settings perform an autocommit every 100,000 documents or 10 minutes.

```
<autoSoftCommit>
  <maxDocs>100000</maxDocs>
  <maxTime>600000</maxTime>
</autoSoftCommit>
```

Indexing Very Large Documents

Indexing very large documents can use a large amount of JVM memory. To manage this, you can set the `gptext.idx_buffer_size` configuration parameter to reduce the size of the indexing buffer.

See Changing GPText Server Configuration Parameters for instructions to change configuration parameter values.

Configure Maximum JVM Heap Size

Each Solr core file consumes JVM heap memory. Adding more indexes increases JVM swapping and garbage collection frequency so that it takes longer to create indexes and to load the core files when GPText is started. If you continue to create indexes without increasing the JVM heap, an out of memory error will eventually occur.

Monitor performance at startup and during index creation and increase the JVM size when you begin to see degraded performance. You can also use tools such as jconsole, included with the Java Developer Kit, to monitor Java heap usage. If garbage collections are occurring too frequently and freeing too little memory, JVM heap should be increased.

Use the `-Xmx` JVM command line option to increase the JVM heap size. For example, this `gptext-config` command sets the JVM maximum heap to 4GB:

```
$ gptext-config -o "-Xmx=4096M"
```

Manage Indexing and Search Loads

With high indexing or search load, JVM garbage collection pauses can cause the Solr overseer queue to back up. For a heavily loaded GPText system, you can prevent some performance problems by scheduling document indexing for times when search activity is low.

Configure File System Caching for ZooKeeper

Good Solr performance is dependent on fast response for ZooKeeper requests. ZooKeeper performs best when its database is cached so it does not have to go to disk for lookups. If you find that ZooKeeper JVMs have frequent disk accesses, look for ways to improve file caching or move ZooKeeper disks to faster storage.

The ZooKeeper `zkClientTimeout` is the time a client is allowed to not talk to ZooKeeper before having it's session expired.

Glossary

A

analyzer
: Defines the set of terms for a Solr index field. An analyzer consists of a tokenizer and a set of optional filters to be applied to the input text. For example, an analyzer can consist of a WhiteSpaceTokenizerFactory followed by a LowerCaseFilterFactory as a filter. See also: *taxonomy*.

B

bigram
: A sequence of two adjacent elements in a token string. A sequence of three consecutive tokens is a trigram and a sequence of n consecutive tokens is an n-gram.

binary classification
: The process of sorting data into one of two categories, for example, classifying a given text according to whether the text is associated with a positive or negative sentiment.

  Classification problems with more than two classes into which the data is to be categorized are called multiclass classification problems.

C

centroid
: In clustering problems, a centroid represents the approximate center of a cluster.

  A centroid does not have to map directly to a data point in the cluster. For example, in k-means clustering the coordinates of a centroid are the mean of the coordinates of data points (documents) pertaining to that cluster and are constantly updated as new data point assignments are made.

cluster
: A set of identical data points. For example, if some documents are to be grouped into three clusters, the result of a machine learning algorithm is three clusters: all the documents within a particular cluster are similar to each other, but different from the documents in other clusters. The results and the quality of the clusters depend on various factors, including the algorithm used, the parameters that were configured, and set of features used.

collection
: A complete, logical index in a GPText or SolrCloud system. A logical index is composed of multiple shards. A GPText collection has one shard per Greenplum segment.

config set
: The Solr configuration files that define the structure and configuration of a GPText index, such as `managed-schema` and `solrconfig.xml`. These files are stored in ZooKeeper.

core
: A replica of a shard, implemented in GPText and SolrCloud as a Lucene index. Also called *Solr Core*.

corpus
: A collection of documents. Plural: corpora.

D

dictionary
: A list of unique words or terms from the documents that comprise the vocabulary of the document collection.

dimension
: A generalized term for a feature of data, such as word counts in a document. Dimensions are typically large in number.

An n-dimensional vector is a vector according to which a document can be expressed in an n-dimensional feature space. For example, if your dictionary contains n unique terms, a document could be expressed in an n–dimensional vector where each position contains the count with which a particular term from the dictionary appears in that document. A feature space could be the entire dictionary or could be another dictionary (or set of features) extracted by using feature selection.

dimensionality reduction
: The process of reducing the dimensions (or features) according to which the data (or documents) is expressed in a feature space. For example, selecting terms (or features) from the dictionary that appear in more than k documents in the entire corpus gives one set of reduced dimensions.

L

lead replica
: A replica that has been elected leader for a shard. When documents are indexed, SolrCloud sends them to the lead replica for the shard and the lead distributes them to all of the shard's remaining replicas. Also called *leader*.

M

machine learning
: A branch of artificial intelligence that focuses on the construction and study of systems that can learn from data.

N

natural language processing
: A field of study that combines computer science, artificial intelligence, and liguistics to study interactions between computers and human languages.

O

ontology
: The formal representation of knowledge as a set of concepts (ideas, entities, events) and their properties and relations according to a system of categories within a domain. Ontologies provide the structural frameworks for organizing information for fields such as artificial intelligence. Ontology is not a synonym for taxonomy.

P

proximity, term proximity
: A search that looks for documents in which two or more separately matching term occurences are within a specified distance (a number of intermediate words or characters).

Q

query parser
: A component that parses the input queries provided for search.

R

replica
: A single copy of a shard, managed as a Solr core. One replica for each shard is elected as the*lead* replica, or *leader*. All updates and searches are directed to the lead replica; changes are replicated from the lead to the remaining replicas.

S

sentiment analysis
: Classifies opinions expressed in text documents into categories such as “positive” and “negative”.

shard
: A logical piece or slice of a collection. In GPText, there is one shard for each Greenplum Database segment. A shard is made up of one or more replicas. One replica is elected the lead shard, or *leader*, and updates to the leader are replicated to the other replicas.

silhouette coefficient (SC)
: A quantitive measure of data clustering performance. SC measures how tightly grouped all the data in the cluster are. Its values range between –1 and 1. Values near 1 indicate that clustering was good, and values near -1 indicate that clustering was not good and the data point must have been assigned to another cluster. Values near 0 indicate that the cluster assignment was ambiguous, and the data point is somewhere on the boundary of the cluster.

sparse vector
: A vector whose elements are mostly zeros or are unpopulated. See the Getting Started with GPText Guide for more information.

stem
: The part of a word that is common to all its inflected variants (how you modify a word to express its different grammatical categories, for example, by conjugating a verb). For example, receives, receiving, and received all derive from the stem “receiv”.

stemming
: The process for reducing an inflected or derived word to its stem, base, or root form. The stem is not necessarily the same as the root form. For example, receives, receiving, and received all derive from the stem “receiv”; the root form is receive.

support vector machine (SVM)
: A supervised learning model that classifies data by analyzing the data, recognizing patterns in the data, and placing the data in specific classes. Applications include sentiment analysis, separating spam email from legitimate email, and, if the Sorting Hat were an SVM, determining the House to which new Hogwarts students are assigned.

T

taxonomy
: A hierarchical system of classification; a method for dividing terms, concepts, or other entities into ordered groups or categories. Taxonomies differ from ontologies in that they are generally focused, simple tree relationships, and ontologies have wider, broader scopes.

term
: A distinct word or value within a field.

TF-IDF score
: Term frequency-inverse document frequency. A numeric statistic that reflects how important a word is to a set of documents. The tf-idf score increases proportionate to the number of times a word appears in a document.

TF-IDF vector
: A vector containing tf-idf scores.

token
: Units into which an input string is broken. For example, a token can be individual terms in a bigram (multiple terms) or trigram that appear in the input text.

tokenizer
: Breaks a stream of text into tokens based on delimiters, the separators that specify the characters to consider as the token boundaries, or some regular expressions. For example, a delimiter could be a white space.

  Tokenizers are not aware of fields in a document.

token filter
: Takes a stream of tokens produced by a tokenizer, examines each token, and either passes the token along or discards it. For example, a token filter may remove white space, unnecessary words such as “a”, “an”, or “the”, or remove dots from acronyms. Token filters produce another stream of tokens that can be input to other token filters.

U

Universal Query Parser
: The GPText Universal Query Parser parses queries containing expressions supported by any supported query parser.

Z

ZooKeeper
: Apache ZooKeeper is a server product that provides cluster management services such as centralized configuration management, load balancing, and failover. It is a SolrCloud requirement. ZooKeeper handles leader elections for SolrCloud replicas. For high availability, ZooKeeper is deployed as a cluster of 3, 5, or 7 nodes. The ZooKeeper cluster can be deployed with GPText on the same cluster hosts or an existing ZooKeeper cluster can be used.

Pivotal GPText Reference Guide

This guide provides reference information about Pivotal GPText functions and command-line utilities.

GPText Function Reference

The following functions are available in Pivotal GPText.

Indexing

gptext.create_index() – creates an empty index.

gptext.index() – populates an index.

gptext.commit_index() – finalizes index operations.

gptext.enable_terms() – enables term vectors and positions to allow extracting terms and their positions from `text` fields.

Modifying or Deleting an Index

gptext.add_field() – adds a field to an index.

gptext.delete() – deletes documents matching a search query.

gptext.drop_field() – deletes a field from an index.

gptext.drop_index()– deletes an index.

Search

gptext.search() – searches an index.

gptext.search_count() – returns number of documents that match search.

gptext.gptext*retrieve*field – extracts a single field from the `rf` search result column as text.

gptext.gptext*retrieve*field_int – extracts a single field from the `rf` search result column and converts to an integer.

gptext.gptext*retrieve*field_float – extracts a single field from the `rf` search result column and converts to a float.

gptext.highlight() – returns search result with search term highlighted.

Faceted Search

gptext.faceted_field_search() – search, faceted by fields.

gptext.faceted_query_search() – search, faceted by queries.

gptext.faceted_range_search() – search, faceted by defined ranges.

Working With Terms

gptext.enable_terms() – enables term vectors and positions.

gptext.terms() – gets the term vectors for the indexed documents in a Solr index for the specified field.

Configuration and Monitoring

gptext.index_status() – shows status for an index or all indexes.

gptext.reload_index() – reloads Solr configuration files.

gptext.version() – returns version of GPText installation.

High Availability

gptext.add_replica() – Adds a replica of an index shard.

gptext.delete_replica() – Deletes a replica of an index shard.

General Purpose Functions

gptext.count_t() – counts number of rows in a table.

Privileges

Your privileges to execute the GPText functions depend on your Greenplum Database privileges for the table from which the index is generated. For example, if you have SELECT privileges for a table in the Greenplum database, you have SELECT privileges for an index generated from that table.

Executing index functions requires one of OWNER, SELECT, INSERT, UPDATE, or DELETE privileges, depending on the function. The OWNER is the person who created the table and has all privileges. See the *Security* section of the *GPText User's Guide* for information about setting privileges.

The *Privileges required* section for each of the GPText functions specifies the privileges required to execute that function.

Usage

The `gptext` functions in this section must be executed as SQL queries in the form:

```
SELECT * FROM gptext.function();
```

You must run these queries from the Greenplum Database master.

The examples in this section use one of the following:

- A Greenplum database named `wikipedia` set up as follows:
  - A `public` schema.
  - A table named `articles`.
    The `articles` table has the following columns:
    `id`, `date_time`, `title`, `content`, `refs`.
    The default search column is `content`.
    The name of the index on the `articles` table is `wikipedia.public.articles`.

Indexing

Indexing functions create, set up, populate, and finalize (commit) Solr indexes.

gptext.create_index()

Creates an empty Solr index.

Syntax

```
gptext.create_index(schema_name, table_name, id_col_name,
  def_search_col_name [, if_check_id_uniqueness])
```

or

```
gptext.create_index(schema_name, table_name, p_columns, p_types,
  id_col_name, def_search_col_name [, if_check_id_uniqueness])
```

Parameters

`schema_name`
: The name of the schema in the Greenplum database.

`table_name`
: The name of the table in the Greenplum database.

`p_columns`
: A text array containing the names of the table columns to index. If `p_columns` and `p_types` are omitted, all table columns are indexed.

  The columns must be valid columns in the table. The columns identified by the `id_col_name` and `def_search_col_name` must be included in the array.

  If the `p_columns` parameter is supplied, the `p_types` parameter must also be supplied. The sizes of the `p_columns` and `p_types` arrays must be the same.

`p_types`
: A text array containing the Solr data types of the columns in the `p_columns` array.

  Text types can be mapped to the name of an analyzer chain, for example `text_intl`, `text_sm`, or any type defined in the `managed_schema`. See Mapping Greenplum Database Data Types to Solr Data Types for equivalent Solr data types for other Greenplum types.

  The `p_types` parameter must be supplied if the `p_columns` parameter is supplied.

`id_col_name`
: The name of the `id` column in `table_name`. The column must be of type bigint or int8.

`def_search_col_name`
: The name of the default column to search in `table_name`, if no other column is named in a query.

`if_check_id_uniqueness`
: Optional. A Boolean value. The default is true. Set to false to index a table with a non-unique ID field.

Return type

`boolean`

Privileges required

Only the OWNER can execute this function.

Remarks

A GPText index is a Solr collection.

The contents of the `id_col_name` column should, in most cases, be a unique ID for each row. It must be of type `bigint` or `int8`. If the `if_check_id_uniqueness` argument is true, the default, a document with an ID matching an existing ID cannot be added to the index.

If the `if_check_id_uniquess` argument is false, documents with duplicate IDs are allowed to be added to the index. The content of other fields may or may not be the same as existing documents with the same ID. When a query returns multiple documents with the same ID, it is the user's responsibility to anticipate and handle the multiple documents. For example, a table could have a `revision` column that is incremented when a new version of a document is added to the index, allowing queries that omit all but the most recent version from search results.

The name of the index created has the format:

```
<database_name>.<schema_name>.<table_name>
```

Populate the new index with gptext.index().

The number of replicas for each shard is determined when the index is created. It is the value of the `gptext.replication_factor` server configuration parameter, 3 by default.

If the `gptext.failover_factor` server configuration parameter is set, `gptext.create_index()` fails if the ratio of the number of GPText nodes that are up to the total number of GPText nodes is less than the `gptext.failover_factor` value (from 0.0 to 1.0). Index shards can only be created on active GPText nodes, so the `gptext.failover_factor` parameter prevents overloading the active GPText nodes when too many nodes are down.

Example

- Create an index, `wikipedia.public.articles`, with `content` as the default search field. `sql => SELECT * FROM gptext.create_index('public', 'articles', 'id', 'content');`
- Create an index, `wikipedia.public.articles`, with `content` as the default search field. Index the `id`, `content`, and `title` fields. `sql => SELECT * FROM gptext.create_index('public', 'articles', '{"id", "content", "title"}', '{"long", "text", "text"}', 'id', 'content')`

gptext.enable_terms()

Enables term vectors and positions to allow extracting terms and their positions from fields of data type `text`.

Syntax

```
gptext.enable_terms(index_name, field_name)
```

Parameters

`index_name`
: The name of the index for which you want to enable terms.

`field_name`
: The name of the field for which you want to enable terms.

Return type

`boolean`

Privileges required

Only the OWNER can execute this function.

Remarks

Solr can mark terms and their positions in documents when indexing. This capability is disabled by default. Use `gptext.enable_terms()` to enable the capability.

Call `gptext.enable_terms()` for each field where you want to enable terms.

After calling this function, you must index or re-index with gptext.index().

Example

```
=# SELECT * FROM gptext.enable_terms('wikipedia.public.articles', 'content');
WARNING:  Enable terms for field: content of index: wikipedia.public.articles successfully. Reindex data needed.
 enable_terms
--------------
 t
(1 row)

SELECT * FROM gptext.index(TABLE(SELECT * FROM articles), 'wikipedia.public.articles');
SELECT * FROM gptext.commit_index('wikipedia.public.articles');
```

gptext.index()

Populates an index by indexing data in a table.

Syntax

```
gptext.index(TABLE(SELECT * FROM table_name), index_name)
```

Parameters

`TABLE(SELECT * FROM table_name)`
: The table to be indexed, with data type `anytable`.

`index_name`
: Name of the index that was created with `gptext.create_index()` and is to be populated.

Return type

```
SETOF dbid INT, num_docs
BIGINT
```

where `dbid` is the `dbid` of the segment that the documents were sent to, and `num_docs` is the number of documents that were indexed.

Privileges required

You must have the INSERT or UPDATE privilege to execute this function.

Remarks

`index_name` must have been created with `gptext.create_index()`.

The arguments to the `gptext.index()` function must be expressions. For example, TABLE(SELECT * FROM articles) creates a “table-valued expression” from the articles table, using the table function `TABLE`.

You can selectively index/update by changing the inner select list in the query.

After successfully indexing, you must commit the index with `gptext.commit_index(index_name)`.

The output includes a two-column table with `dbid` (the Greenplum segment ID) and `num_docs` (the number of documents added to the index for that segment) as the columns.

Note:

Be careful about distribution policies:

The first parameter of `gptext.index()` is `TABLE(SELECT * FROM messages)`. The query in this parameter should have the same distribution policy as the table you are indexing. There are two cases where the query will not have the same distribution policy:

1. Your query is a join of two tables

2. You are indexing an intermediate table (staging table) that is distributed differently than the final table.

When the distribution policies differ, you must specify `"SCATTER BY"` for the query like this:

`TABLE(SELECT * FROM messages SCATTER BY distrib_id)` ,

where `distrib_id` is the distribution id used when you created your primary/final table.

Example

```
=# SELECT * FROM gptext.index(TABLE(select * FROM articles), 'wikipedia.public.articles');
 dbid | num_docs
------+----------
    3 |        6
    2 |        5
(2 rows)
```

gptext.commit_index()

Finishes an index operation. The results of an indexing operation are not available until this function is called for the index.

Syntax

```
gptext.commit_index(index_name)
```

Parameters

`index_name`
The name of the index to commit.

Return type

`boolean`

Privileges required

You must have the INSERT, UPDATE, or DELETE privilege to execute this function.

Remarks

Must be called after gptext.index() and gptext.delete().

Example

```
=# SELECT * FROM gptext.commit_index('wikipedia.public.articles');
 commit_index
--------------
 t
(1 row)
```

Modifying or Deleting an Index

You can change an index by adding or dropping fields, reverting an index to its previous state, or deleting the index.

gptext.add_field()

Adds a field to your schema if the field was added to the database after the index was created.

Syntax

```
gptext.add_field (index_name, field_name[, is_default_search_col [, if_enable_terms]])
```

Parameters

`index_name`
: The name of the index to which you want to add the field.

`field_name`
: The name of the field to be indexed.

`is_default_search_col`
: Optional. Boolean value. Is this to become the default search column (field)?

`if_enable_terms`
: Optional. Boolean value. Enable terms support on this field when added to the GPText index?

Return type

`SETOF boolean`

Privileges required

Only the OWNER can execute this function.

Remarks

Call this function for each field you add.

Before and after you add one or more fields, reload the Solr configuration files with gptext.reload_index(). The initial `reload_index()` call is required because of Solr 4.0 behavior and may not be required in subsequent versions.

After you add one or more fields, you must also create and populate a new index with gptext.create_index() `and` gptext.index(), then commit with index with gptext.commit_index().

Example

Adds the field `external_links` to the index, then recreates, repopulates, and commits the index.

```
=# SELECT * FROM gptext.reload_index('wikipedia.public.articles');
=# SELECT * FROM gptext.add_field('wikipedia.public.articles', 'external_links', false, false);
INFO:  Add field: external_links for index: wikipedia.public.articles
 add_field
-----------
 t
(1 row)

=# SELECT * FROM gptext.reload_index('wikipedia.public.articles');
 reload_index
--------------
 t
(1 row)

SELECT * FROM gptext.commit_index('wikipedia.public.articles');
```

gptext.delete()

Deletes all documents that match the search query.

Syntax

```
gptext.delete(index_name, query)
```

Parameters

`index_name`
: The name of the index.

`query`
: Documents matching this query will be deleted. To delete all documents use the query `'*'` or `'*:*'`.

Return type

`boolean`

Privileges required

You must have the DELETE privilege to execute this function.

Remarks

After a successful delete, commit the index as follows.

```
gptext.commit_index(index_name)
```

Examples

- Delete all documents containing the word “unverified” in the default search field:

```
=# SELECT * FROM gptext.delete('wikipedia.public.articles', 'unverified');
 delete
--------
 t
(1 row)=# SELECT * FROM gptext.commit_index('wikipedia.public.articles');
 commit_index
--------------
 t
(1 row)
```

- Delete all documents from the index:

```
wikipedia=# SELECT * FROM gptext.delete('wikipedia.public.articles', '*:*');
 delete
--------
 t
(1 row)

wikipedia=# SELECT * FROM gptext.commit_index('wikipedia.public.articles');
 commit_index
--------------
 t
(1 row)
```

gptext.drop_field()

Removes a field from your schema.

Syntax

```
gptext.drop_field(index_name, field_name)
```

Parameters

`index_name`
: The name of the index from which to drop the field.

`field_name`
: The name of the field to drop.

Return type

`boolean`

Privileges required

Only the OWNER can execute this function.

Remarks

Call this function for each field you drop.

Before and after dropping one or more fields, you must reload the Solr configuration files with gptext.reload_index(), then commit the index with gptext.commit_index().

The initial `reload_index()` is required by Solr 4.0 behavior and may not be necessary in subsequent versions.

Example

Drops the field `external_links` from the index.

```
=# SELECT * FROM gptext.reload_index('wikipedia.public.articles');
 reload_index
--------------
 t
(1 row)

=# SELECT * FROM gptext.drop_field('wikipedia.public.articles', 'external_links');
INFO:  Drop field: external_links for index: wikipedia.public.articles
 drop_field
------------
 t
(1 row)

=# SELECT * FROM gptext.reload_index('wikipedia.public.articles');
 reload_index
--------------
 t
(1 row)

=# SELECT * FROM gptext.commit_index('wikipedia.public.articles');
 commit_index
--------------
 t
(1 row)
```

gptext.drop_index()

Removes an index.

Syntax

```
gptext.drop_index(index_name)
```

Parameters

`index_name`
: The name of the index to drop.

Return type

`boolean`

Privileges required

Only the OWNER can execute this function.

Remarks

A dropped index cannot be recovered.

Example

```
=# SELECT * FROM gptext.drop_index('wikipedia.public.articles');
 drop_index
------------
 t
(1 row)
```

Search

Search functions enable querying an index.

Changing the query parser at query time

When using the search functions, you can change the query parser used by Solr at query time. A different query parser may be required, depending on the nature of the query. See the Using Advanced Querying Options for a list of the query parsers GPText supports.

To change the query parser at query time, use the `defType` Solr option with the `gptext.search()` function.

To change the query parser for any search function at query time, use the Solr `localParams` syntax, replacing the `<query>` term with `'{!type=edismax}<query>'`.

With the GPText Universal Query Parser, you can use features of any of the supported query parsers in one query. To use the Universal Query Parser, replace the `<query>` term with `'{!gptextqp}<query>'`. See Using the Universal Query Parser for information and examples.

gptext.search()

Searches an index.

Syntax

```
gptext.search(src_tbl, index_name, search_query, filter_queries)
```

or:

```
gptext.search(src_tbl, index_name, search_query, filter_queries[, options])
```

Parameters

`src_table`
: Specifies a `SELECT` statement on an existing, indexed table on which to perform the search. The `src_table` parameter is an *anytable* data type, specified in the following format

```
TABLE(SELECT * FROM <src_table>)
```

`index_name`
: The name of the index to search.

`search_query`
: Text value containing a Solr text search query.

`filter_queries`
: A text array of filter queries, if any. If none, set this parameter to `null`.

`options`
: An optional ampersand-delimited list of Solr query parameters. See Solr options.

Return type

`SETOF gptext.search_scored_result`

This is a composite type where:

Column	Type
id	bigint
score	double precision
offsets (conditional)	gptext hstore
rf (conditional)	text

If the `options` parameter is included in `gptext.search()`, the result includes the `offsets` column. This column contains key-value pairs, where the key is the column name and the value is a comma-separated list of offsets to locations where the search term occurs. This data is used by the gptext.highlight() function to add highlighting tags to the column data. If highlighting is not enabled with the `'hl=true'` option, the `offsets` column is `NULL`.

If the `fl` option is included in the `options` parameter to specify additional fields to add to the result, the `rf` column contains the additional fields in a formatted text value. The gptext.gptext_retrieve_field() function can be used to extract a single field value from the `rf` column. There are variants of the gptext.gptext_retrieve_field() function to retrieve integer and float values from the `rf` column value.

Privileges required

You must have the SELECT privilege to execute this function.

Solr options

Solr queries allow the following optional refinements, specified as an ampersand-delimited list in the `options` parameter.

`defType`
: The name of the query parser to use for this query.

  Example: `defType=edismax`

`rows`
: The maximum number of rows to return per segment. If omitted, all rows are returned.

  Example: `rows=100` returns 100 rows per segment or all rows if there are fewer than 100.

`sort`
: Sorts on a field or score in ascending or descending order.

  Examples:

  - `score desc` (default if no sort defined)
  - `date_time asc`
  - `date_time asc score desc` sorts on `date_time` ascending, then on `score` descending

`start`
: The number of the first record to return.

  Examples:

  - `start=0` default: returned records starts with record 0
  - `start=25` returned records starts with record 25

`hl`
: Enable highlighting.

  Example: `hl=true`

`hl.fl`
: Comma-separated list of field names to consider when highlighting.

  Examples:

  - `hl.fl=field1`
  - `hl.fl=field1,field2`

`fl`
: Comma-separated list of fields to include in search results. The fields must be set to `stored=true` in the managed-schema for the index.

  Example: `fl=version,author`

Remarks

- The output includes a table with columns `id` (the ID named in gptext.create_index()) and `score` (the `tf-idf` score). A column named `offsets` is included if highlighting is specified in the `options` parameter. An column named `rf` is included when a list of additional fields to include is specified in the `options` parameter.
- To change the query parser at query time, specify the `defType` option in the options parameter list. For example, setting the options parameter to ' `rows=100&defType=edismax` ' limits the output to 100 rows per segment and will change the query parser to `edismax`.
- The `TABLE` query is planned and affects the estimate for `gptext.search()`, but does not execute. For example, if your query includes

  ```
  gptext.search(TABLE(SELECT * FROM t), ...)
  ```

  the query planner estimates the number of results as the number of rows in `t`. This can cause the query planner to ignore the use of an index scan. Use a query like `TABLE(SELECT 1 SCATTER BY 1)` to avoid this issue.
- If you do not specify options, `gptext.search()` returns all rows.
- The options separator has changed from comma to ampersand (&) in order to support highlighting. If you do not use highlighting, you can revert to using the comma separator by setting the `gptext.search_param_separator` to `','`.
- The Solr option `rows` specifies the maximum number of rows to return per segment. For example, if you have four segments `'rows=100'` returns at most a total of 400 rows. To limit the number of rows returned for an entire query, set a `LIMIT` in the SQL query. For example, the following query returns at most a total of 20 rows for the query: `select t.id, q.score, t.message\_text from twitter.message t, gptext.search(...) q where t.id=q.id LIMIT '20';`

- The `gptext hstore` type is a limited form of the Postgres `hstore` type. It only has the `hstore` input and output functions implemented, as `gptext_hstore_in` and `gptext_hstore_out`.

Examples

- Runs a GPText query that looks for Wikipedia articles that contain the term “optimization”, and joins the results to the original Greenplum Database `articles` table:

```
=# SELECT a.id, a.title, q.score
FROM articles a,
gptext.search(TABLE(SELECT * FROM articles), 'wikipedia.public.articles',
'optimization', null) q
WHERE a.id = q.id ORDER BY score DESC;
   id   |     title     |    score
--------+---------------+-------------
 43835553 | Cubesort       | 0.054204933
 29050607 | Cycle sort     | 0.044258144
  6508027 | Tree sort      | 0.036136623
    29352 | Selection sort | 0.034337204
    20039 | Merge sort     | 0.033537593
  3268249 | Quicksort      | 0.030907018
 25977485 | Bubble sort    | 0.028614337
 23954341 | Timsort        | 0.023363508
    77355 | Shellsort      | 0.023128692
    15205 | Insertion sort |  0.02289147
    13995 | Heapsort       | 0.011179198
(11 rows)
```

- Returns 3 results from each shard beginning at record 0:

```
=# SELECT a.id, a.title, q.score
FROM articles a,
gptext.search(TABLE(SELECT * FROM articles), 'wikipedia.public.articles',
'optimization', null, 'rows=3&start=0') q
WHERE a.id = q.id;
   id   |     title     |    score
--------+---------------+-------------
    13995 | Heapsort       | 0.011179198
    20039 | Merge sort     | 0.033537593
  6508027 | Tree sort      | 0.036136623
    15205 | Insertion sort |  0.02289147
    77355 | Shellsort      | 0.023128692
 23954341 | Timsort        | 0.023363508
(6 rows)
```

- Returns 10 rows with the text “iphone” highlighted in the `message` column.

```
SELECT t.id,
gptext.highlight(t.message, 'message', s.hs)
FROM twitter.message t,
gptext.search(TABLE(SELECT 1 SCATTER BY 1),
'demo.twitter.message','{!gptextqp}iphone', null,
'rows=10&hl=true&hl.fl=message'
) s
WHERE t.id = s.id;
```

gptext.search_count()

Returns the number of documents that match the search query.

Syntax

```
gptext.search_count(index_name, search_query, filter_queries)
```

Parameters

`index_name`
: The name of the index. dd>

`search_query`
: The search query.

`filter_queries`
: A comma-delimited array of filter queries, if any. If none, set this parameter to `null`.

Return type

`bigint`

Privileges required

You must have the SELECT privilege to execute this function.

Remarks

None.

Example

```
=# SELECT * FROM gptext.search_count('wikipedia.public.articles', 'bubble', null);                                    count
-------
   4
(1 row)
```

gptext.gptext_retrieve_field()

Retrieves a single field from the `rf` search result column as a text value.

Syntax

```
gptext.gptext_retrieve_field(rf, field_name)
```

Parameters

`rf`
: The name of the `rf` column.

`field_name`
: The name of the field to retrieve.

Remarks

The `fl=<field_list>` Solr search option is added to the `options` parameter of the `gptext.search()` function to request additional stored fields. The additional fields are returned in the results in a column named `rf`. The `rf` column value has a format like the following:

```
column_value { name: "_version" value: "1544714234398507008" } column_value { name: "revision" value: "9.70" } column_value { name: "author" value: "jdough" }
```

The `gptext.gptext_retrieve_field()` function extracts the value for a single specified field and returns it as a text value. If there is no field with the specified name in the `rf` column, it returns NULL.

Storing additional fields in an index requires editing `managed-schema` to specify the fields that should be stored. See Storing Additional Fields in an Index

for instructions.

gptext.gptext_retrieve_field_int()

Retrieves a single field from the `rf` search result column as an integer value.

Syntax

```
gptext.gptext_retrieve_field_int(rf, field_name)
```

Parameters

`rf`
: The name of the `rf` column.

`field_name`
: The name of the integer field to retrieve.

Remarks

The `gptext.gptext_retrieve_field_int()` function is the same as the gptext.gptext_retrieve_field function, except that the extracted field value is converted to an integer value.

gptext.gptext_retrieve_field_float()

Retrieves a single field from the `rf` search result column as a float value.

Syntax

```
gptext.gptext_retrieve_field_float(rf, field_name)
```

Parameters

`rf`
: The name of the `rf` column.

`field_name`
: The name of the float field to retrieve.

Remarks

The `gptext.gptext_retrieve_field_float()` function is the same as the gptext.gptext_retrieve_field function, except that the extracted field value is converted to a float value.

gptext.highlight()

Highlights terms by inserting markup tags into data.

Syntax

```
gptext.highlight(column_data, column_name, offsets)
```

Parameters

`column_data`
: The text data from the table which is to be tagged with highlighting tags.

`column_name`
: The name of the corresponding column from the table.

`offsets`
: A `gptext hstore` value that contains key-value pairs that indicate the locations of the text to highlight within the column data. See Remarks for information about the `gptext hstore` data type.

Prequisite

- To use highlighting, term vectors must be enabled before creating the index. To enable term vectors, call gptext.enable_terms() for each field where you want to enable terms, then index or re-index with gptext.index().

Remarks

- The `offsets` parameter is a `gptext hstore`, where each key is a column name and the value is a comma-separated list of offsets into the column data. This `hstore` is constructed by gptext.search() with highlighting enabled in the `offsets` parameter.
  Following is an example of the `offsets` `hstore` content:

  ```
  "field1"=>"0:5,9:14","field2"=>"13:20"
  ```

  `gptext.highlight()` will insert two sets of tags into the `field1` data and one set into the `field2` data at the indicated offsets.
- The `gptext hstore` type is a limited form of the Postgres `hstore` type. It has only the `hstore` input and output functions implemented, as `gptext_hstore_in` and `gptext_hstore_out`.
- The highlight tags are defined by the `gptext.hl_pre_tag` and `gptext.hl_post_tag` server configuration parameters. Their default values are `<em>` and `</em>`, respectively.

Example

- Returns 10 rows with the text “iphone” highlighted in the `message` column.

```
SELECT t.id,
gptext.highlight(t.message 'message', s.hs)
FROM twitter.message t,
gptext.search(TABLE(SELECT 1 SCATTER BY 1),
'demo.twitter.message','{!gptextqp}iphone', null,
'rows=10&hl=true&hl.fl=message'
) s
WHERE t.id = s.id;
```

Faceted Search

Faceting breaks up a search result into multiple categories, showing counts for each.

gptext.faceted_field_search()

The `faceted_field_search()` function breaks search results into field name categories.

Syntax

```
gptext.faceted_field_search (index_name, query, filter_queries, facet_fields, facet_limit, minimum)
```

Parameters

`index_name`
: The name of the index.

`query`
: Query statement. Use `*:*` to query for all results.

`filter_queries`
: A text array of filter queries, if any. If none, set this parameter to `null`.

`facet_fields`
: An array of field names to facet. Use PostgreSQL array notation.

`facet_limit`
: Maximum number of results to be returned for each aggregation (facet).

`minimum`
: Minimum number of results required before an aggregation (facet) will be returned. Enter 0 to return all facets.

Return type

`SETOF gptext.facet_field_result`

Privileges required

You must have the SELECT privilege to execute this function.

Remarks

None.

Examples

- Facet on `spam` and `truncated` fields, no limit, no minimum, all tweets:

```
SELECT * FROM
gptext.faceted_field_search('twitter.public.message', '*:*', null, '{spam, truncated}', 0, 0);
```

- Facet on `author_id`, no limit, with a minimum of five tweets, all tweets:

```
SELECT * FROM
gptext.faceted_field_search('twitter.public.message', '*:*', null, '{author_id}', 0, 5);
```

gptext.faceted_query_search()

The `faceted_query_search()` function breaks search results into categories defined by queries that you provide.

Syntax

```
gptext.faceted_query_search(index_name, query, filter_queries, facet_queries)
```

Parameters

`index_name`
: The name of the index.

`query`
: Query statement. Use `*:*` to query for all results.

`filter_queries`

A text array of filter queries, if any. If none, set this parameter to `null`.

facet_queries

Type: text[]. Required. An array of facet queries.

Return type

```
SETOF
gptext.facet_query_result
```

Privileges required

You must have the SELECT privilege to execute this function.

Remarks

None.

Example

Facet on product price ranges (0-100, 101-200, 201-300, 300+) of cameras:

```
SELECT * FROM
  gptext.faceted_query_search('store.public.catalog',
  'product_type:camera', null,
  '{price:[* TO 100], price:[101 TO 200],
  price:[201 TO 300], price:[301 TO *]}');
```

gptext.faceted_range_search()

The `faceted_range_search()` function breaks search results into range categories, with ranges defined by the `range_start`, `range_end`, and `range_gap` parameters.

Syntax

```
gptext.faceted_range_search(index_name, query, filter_queries,
  field_name, range_start, range_end, range_gap)
```

Parameters

`index_name`

The name of the index.

`query`

Query statement. Use `*:*` to query for all results.

`filter_queries`

An text array of filter queries, if any. If none, set this parameter to `null`.

`field_name`

The name of the field on which to facet.

`range_start`

Beginning of the range.

`range_end`

End of the range.

`range_gap`

Size of range increment, a text value.

Return type

`SETOF gptext.facet_range_result`

Privileges required

You must have the SELECT privilege to execute this function.

Remarks

None.

Example

Facet on date range, starting 1 year ago, ending now, every day:

```
SELECT * FROM
  gptext.faceted_range_search('twitter.public.message', '*:*',
  null, 'created_at', 'NOW/YEAR-1YEAR', 'NOW/YEAR', '+1DAY');
```

Working with Terms

gptext.terms()

Gets the term vectors for the indexed documents in a Solr index for the specified field. You can use `gptext.terms()` to create tables.

Syntax

```
gptext.terms(src_table, index_name, field_name,
        search_query, filter_queries[, options])
```

Parameters

`src_table`
: An `anytable` value that specifies a `SELECT` statement on an existing, indexed table on which to perform the search. Specify in the format:

```
TABLE(SELECT * FROM <src_table>;)
```

`index_name`
: The name of the index to query for terms.

`field_name`
: The name of the field to query for term.

`search_query`
: A query that the field must match.

`filter_queries`
: A comma-delimited array of filter queries, if any. If none, set this parameter to `null`.

`options`
: An optional, comma-delimited list of Solr query parameters.

Return type

`SETOF gptext.term_info`

This is a composite type where:

Column	Type
id	bigint
term	text
positions	integer[]

Privileges required

You must have the SELECT privilege to execute this function.

Remarks

- To enable using `gptext.terms()`, execute the GPText function gptext.enable_terms(), then reindex with gptext.index().
- The `TABLE` query is planned and affects the estimate of of `gptext.terms()`, but does not execute. For example, if your query includes:

  ```
  gptext.terms(TABLE(SELECT * FROM t), ...)
  ```

  The query planner estimates the number of results as the number of rows in `t`. This can cause the query planner to ignore the use of an index scan. Use a query like `TABLE(SELECT 1 SCATTER BY 1)` to avoid this issue.

Examples - Create a Terms Table

```
CREATE TABLE twitter.terms AS
  SELECT * FROM gptext.terms(TABLE(SELECT * FROM
  twitter.message), 'demo.twitter.message', 'message_text',
  'iphone', null);
```

Configuration and Monitoring

Index configuration and monitoring functions enable managing indexes, tracking index statistics, checking status of index segments, and ensuring that index contents are current.

gptext.index_status()

Shows status for a specified index or for all indexes.

Syntax

```
gptext.index_status([index_name])
```

Parameters

`index_name`
: The name of the index. Optional. Returns status for all indexes if no index is specified.

Return Type

`SETOF gptext.index_status_result`

gptext.reload_index()

Reloads Solr configuration files if they have been modified.

Syntax

```
gptext.reload_index([index_name])
```

Parameters

`index_name`
Optional. The name of the index for which to reload the configuration files.

Return type

`boolean`

Privileges required

Only the OWNER can execute this function.

Remarks

None.

Example

```
=# SELECT * FROM gptext.reload_index('wikipedia.public.articles');
 reload_index
--------------
 t
(1 row)
```

gptext.version()

Returns the version of your GPText installation, including build number.

Syntax

```
SELECT * FROM gptext.version()
```

Parameters

None.

Return type

`text`

Privileges required

You do not need any privileges to execute this function.

Remarks

None.

Example

```
=# SELECT * FROM gptext.version();
              version
----------------------------------------
 Greenplum Text Analytics 2.0.0 (main)
(1 row)
```

High Availability

gptext.add_replica()

Adds a replica of an index shard.

Syntax

```
gptext.add_replica(index_name, shard_name[, node_name])
```

Parameters

`index_name`
: Name of the index.

`shard_name`
: Name of the shard to replicate.

`node_name`
: Name of the node where the replica is to be added. Optional. If omitted, SolrCloud chooses the node.

Return type

`boolean`

Remarks

This function is used by the GPText management utility `gptext-replica add`.

The value of the `gptext.replication_factor` configuration parameter when an index is created determines how many replicas are created for each shard. In a Greenplum system, there are the same number of shards as there are Greenplum segments. The number of replicas created for a new index is the number of segments times the value of the `gptext.replication_factor` configuration parameter, 3 by default. The replicas are distributed evenly among the live GPText nodes.

Replicas consume space on the host where they are created, so they are usually only created to replace a replica that has failed or become unavailable or to relocate a replica to another GPText instance. When adding replicas, you should maintain equal distribution of replicas among the GPText nodes and avoid placing multiple replicas for the same shard on the same host.

The total number of replicas for an index that can be placed on each GPText node is set when the index is created. (In Solr, this is the MaxShardsPerNode

parameter.) GPText sets this limit by calculating the number of replicas to create per node and adding an additional factor, specified in the `gptext.extension_factor` server configuration parameter. This parameter can be set between 0 and 10; the default value is 2. Since the limit is set when the index is created, it is recommended to set the `gptext.extension_factor` parameter to a higher number to allow new replicas to be created when necessary.

Example

```
=# SELECT * FROM gptext.add_replica('wikipedia.public.articles', 'shard1');
 success |                 core_name
---------+-------------------------------------------
 t       | wikipedia.public.articles_shard1_replica4
(1 row)
```

gptext.delete_replica()

Deletes a named replica from the specified index and shard.

Syntax

```
gptext.delete_replica(index_name, shard_name, replica_name[, only_if_down])
```

Parameters

`index_name`
: Name of the index.

`shard_name`
: Name of the shard that contains the replica to delete.

`replica_name`
: Name of the replica to remove.

`only_if_down`
: Optional. When true, no action is taken if the replica is active. Default is false.

Return type

`boolean`

Remarks

Use the gptext.index_status() function to find the name of the replica to drop. Names are in the format `core_nodeX`, where `X` is a number.

This function is called from the `gptext-replica drop` management utility.

Examples

1. Delete the `core_node5` replica if it is down.

```
=# SELECT * FROM gptext.delete_replica('wikipedia.public.articles', 'shard1', 'core_node5', true);
ERROR:  Delete replica failed: Attempted to remove replica : wikipedia.public.articles/shard1/core_node5 with onlyIfDown='true', but state is 'active'.
CONTEXT:  SQL function "delete_replica" statement 1
```

2. Delete the `core_node5` replica even if it is active.

```
=# SELECT * FROM gptext.delete_replica('wikipedia.public.articles', 'shard1', 'core_node5');
 success
---------
 t
(1 row)
```

General Purpose Functions

gptext.count_t()

Counts the number of records in a table.

Syntax

```
gptext.count_t(table_name)
```

Parameters

`table_name`
: Name of the table for which to count records.

Return type

`integer`

Privileges required

You need SELECT privileges on `table_name` to execute this function.

Example

```
=# SELECT * FROM gptext.count_t('wikipedia.public.articles');
 count_t
---------
      11
(1 row)
```

GPText Management Utilities

Management utilities are GPText command-line utilities that are used to manage the GPText cluster. The utilities must be run on the Greenplum master as the gpadmin user.

To ensure the utilities are in your path, source the GPText environment script at `install_dir/greenplum-text_path.sh`. For example:

```
source /usr/local/gptext-version/greenplum-text_path.sh
```

Help

To get help for a utility, specify the flag `-h` or `--help`. A short help message displays with a list of parameters.

Debugging

To get verbose output for debugging for all functions except `gptext-state`, specify the flags `-v` or `--verbose`.

GPText Utilities

- gptext-start – starts or restarts the GPText cluster.
- gptext-stop – shuts down the GPText cluster.
- gptext-state – display the state of the GPText cluster.
- gptext-recover – restarts GPText nodes that are down.
- gptext-installsql – installs or removes the gptext schema and user-defined functions in Greenplum databases.
- gptext-replica – adds or drops a replica of an index shard.
- gptext-config – performs GPText configuration options.
- gptext-backup – backs up a GPText index to a shared file system.
- gptext-restore – restores a GPText index from a backup on a shared file system.
- gptext-expand – adds new GPText nodes to existing hosts in the cluster.
- gptext-uninstall – uninstalls GPText, including data and installed files, and ZooKeeper nodes if they were installed with the GPText installer.
- zkManager – checks the ZooKeeper cluster state. If ZooKeeper was installed with GPText, `zkManager` can start or stop the ZooKeeper cluster.

gptext-start

Starts or restarts the GPText cluster.

Syntax

```
gptext-start -h

gptext-start [-r] [-v]
```

Parameters

-h --help	Displays a usage message and exits.
-r	

--restart	Restarts the GPText cluster.
-v --verbose	Displays debug output.

Notes

The `gptext-start -r` command calls the `solr restart` command to stop and restart all of the Solr instances in the cluster. The GPText utility determines if the processes are running before it completes, but it cannot verify that all of the Solr processes were stopped. If it is important to be certain that Solr processes were stopped, for example if you have changed the JVM options, use `gptext-stop` followed by `gptext-start` instead of `gptext-start -r`.

Examples

1. Start the GPText cluster.

```
gptext-start
```

2. Restart the GPText cluster.

```
gptext-start -r
```

gptext-stop

Stop the GPText cluster nodes.

Syntax

```
gptext-stop -h

gptext-stop [-v] [-f]
```

Parameters

-h --help	Displays a usage message and exits.
-v --verbose	Displays debug output.
-f --force	Forcefully stops all Solr processes.

Examples

1. Stop the GPText cluster.

```
gptext-stop
```

2. Force stop the GPText cluster.

```
gptext-stop -f
```

gptext-state

Displays the state of the cluster.

Syntax

```
gptext-state -h

gptext-state [-d db-name] [-i index-name] [-c]

gptext-state list [-d db-name]

gptext-state healthcheck [--stats] [--index=<index_name>]
        [--healthcheck [--disk_free=<percent>]]
        [--stats-columns=<col1,...>] [--database=<database_name>]
```

Parameters

-h --help	Displays a usage message and exits.
-d *db-name* --database=*db-name*	The name of a database containing the GPText schema. `gptext-state` searches all databases for the functions it needs to run. If the user does not have access permission to the database it begins with, it fails. In this case, use the `--database=` parameter to specify an accessible database to search.
-i *index-name* --index=*index-name*	The name of an index. This option cannot be used with the `list` or `healthcheck` subcommands.
-c *stats-list* --stats_columns=*stats-list*	Used with the `-i` or `--index`, option specifies a comma-separated list of statistics to display. The list may contain `replication_factor`, `max_shards_per_node`, `num_docs`, and `size_in_bytes`. If no `-c` or `--stats_columns` option is supplied, all four statistics are displayed.
-f *diskfree* --disk_free=*diskfree*	Used with the `healthcheck` command, specifies the percentage disk free required per host to report a healthy GPText cluster. The default is 10.

Notes

All parameters are optional, except that `--index` is required when you specify `--stats_columns`.

When executed with no arguments, `gptext-state` displays a list of GPText indexes with the columns `database`, `index_name`, and `state`. The `state` column displays the status of the index as `Green`, `Yellow`, or `Red`. - A Green state means that all shards and replicas are healthy. - A Yellow state means that all shards are available, but one or more replicas is down. - A Red state means that one more more shards is down.

Examples

1. Show the GPText cluster state, specifying `wikipedia` as a database containing the GPText schema.

```
$ gptext-state -d wikipedia
20160603:13:54:27:302662 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Check GPText cluster status...
20160603:13:54:27:302662 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Current GPText Version: 2.0.0
20160603:13:54:28:302662 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-All nodes are up and running.
20160603:13:54:28:302662 gptext-state:gpdb-sandbox:gpadmin-[INFO]:--------------------------------------------------
20160603:13:54:28:302662 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Index state.
20160603:13:54:28:302662 gptext-state:gpdb-sandbox:gpadmin-[INFO]:--------------------------------------------------
20160603:13:54:28:302662 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-   database    index_name                state
20160603:13:54:28:302662 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-   wikipedia   wikipedia.public.articles   Green
20160603:13:54:28:302662 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-   gptextdoc   gptextdoc.public.docs       Green
```

2. Show `replication_factor` and `num_docs` statistics for the GPText index `wikipedia.public.articles`. Specify `wikipedia` as the database with the GPText schema.

```
$ gptext-state -i wikipedia.public.articles -c replication_factor,num_docs -d wikipedia
20160603:13:57:16:303262 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Check GPText cluster statistics...
20160603:13:57:18:303262 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-   Replicas Up:   6
20160603:13:57:18:303262 gptext-state:gpdb-sandbox:gpadmin-[INFO]:--------------------------------------------------
20160603:13:57:18:303262 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Index wikipedia.public.articles statistics.
20160603:13:57:18:303262 gptext-state:gpdb-sandbox:gpadmin-[INFO]:--------------------------------------------------
20160603:13:57:18:303262 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-   replication_factor   num_docs
20160603:13:57:18:303262 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-   3                    11
```

3. List all indexes, specifying `wikipedia` as a database containing the GPText schema.

```
$ gptext-state list -d wikipedia
20160603:13:58:10:303550 gptext-state:gpdb-sandbox:gpadmin-[INFO]:------------------------------------------------------------
20160603:13:58:10:303550 gptext-state:gpdb-sandbox:gpadmin-[INFO]:- Index list
20160603:13:58:10:303550 gptext-state:gpdb-sandbox:gpadmin-[INFO]:------------------------------------------------------------
20160603:13:58:10:303550 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-  gptextdoc.public.docs
20160603:13:58:10:303550 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-  wikipedia.public.articles
```

4. Perform a health check with a 20% free disk requirement.

```
$ gptext-state healthcheck -f 20 -d wikipedia
20160603:13:58:56:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Execute healthcheck on GPText cluster!
20160603:13:58:56:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Check GPText config files ...
20160603:13:58:56:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-GOOD
20160603:13:58:56:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Check GPText index status ...
20160603:13:58:56:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-GOOD
20160603:13:58:56:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Checking for required disk space...
20160603:13:58:57:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-GOOD
20160603:13:58:57:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Checking for required user privileges...
20160603:13:58:57:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-GOOD
20160603:13:58:57:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Checking for indexes and database consistency...
20160603:13:58:58:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-GOOD
20160603:13:58:58:303655 gptext-state:gpdb-sandbox:gpadmin-[INFO]:-Done.
```

gptext-recover

Recovers GPText nodes.

Syntax

```
gptext-recover -h

gptext-recover [-f] [-v]

gptext-recover [-H] <new-host1>,<new-host2>,... [-v]
```

Parameters

-h --help	Displays a usage message and exits.
-f --force	Forces recovery for any GPText nodes that are down. If the node is unrecoverable, deletes the node, creates a new node, and recreates replicas.
-H --new-hosts	Recover down nodes on new hosts. For example “host1,host2”.
-v --verbose	Displays debug output.

Notes

The `-f` and `-H` options cannot be used at the same time.

If shards are down, `gptext-recover` advises you to reindex.

If no shards are down, `gptext-recover` restores any replicas that are down.

gptext-installsql

Installs or removes the gptext schema and user-defined functions in databases.

Syntax

```
gptext-installsql -h

gptext-installsql [-c] [-v] db_name [db2_name...]
```

Parameters

-c --clean	Removes the gptext schema and UDFs from the specified databases.
-h --help	Displays a usage message and exits.
-v --verbose	Displays debug output.

Notes

None.

Examples

1. Install GPText UDFs in databases `wikipedia` and `twitter`.

```
$ gptext-installsql wikipedia twitter
20160603:14:03:53:305130 gptext-installsql:gpdb-sandbox:gpadmin-[INFO]:-Creating 'gptext' schema and UDFs in database wikipedia...
20160603:14:03:53:305130 gptext-installsql:gpdb-sandbox:gpadmin-[INFO]:-Creating 'gptext' schema and UDFs in database twitter...
20160603:14:03:54:305130 gptext-installsql:gpdb-sandbox:gpadmin-[INFO]:-Validating gptext installation
20160603:14:03:59:305130 gptext-installsql:gpdb-sandbox:gpadmin-[INFO]:-Done.
```

2. Delete GPText UDFs in database `wikipedia`.

```
$ gptext-installsql -c wikipedia
20160603:14:03:04:304847 gptext-installsql:gpdb-sandbox:gpadmin-[INFO]:-Connecting to database wikipedia
20160603:14:03:05:304847 gptext-installsql:gpdb-sandbox:gpadmin-[INFO]:-Dropping 'gptext' schema and UDFs...
20160603:14:03:05:304847 gptext-installsql:gpdb-sandbox:gpadmin-[INFO]:-Validating clean operation
20160603:14:03:09:304847 gptext-installsql:gpdb-sandbox:gpadmin-[INFO]:-Done.
```

gptext-replica

Add or delete a replica for an index shard.

Syntax

```
gptext-replica -h

gptext-replica add -i index-name -s shard [-n node]

gptext-replica drop -i index-name -s shard -r replica [-o]
```

Parameters

-h --help	Displays a usage message and exits.
-i *index* --index= *index*	Required. The name of the index.
-s *shard* --shard=*shard*	Required. The name of the shard to add a replica to.
-n *node* --node=*node*	Optional. The node where the replica is to be added.
-r *replica* --replica=*replica*	Required for the drop command only. The name of the replica to drop.
-o --onlyifdown	Optional. Used only with the drop command. Only drop the replica if it's down.

Notes

To find the name of a replica to drop, check `gptext.index_status()`. The name is `core_nodeX` where *X* is a number.

Examples

1. Add a replica for index `wikipedia.public.articles` in shard `shard0`, on node `node1`.

```
gptext-replica add -i wikipedia.public.articles -s shard0 -n node1
```

2. Drop the replica named `core_node1` for index `wikipedia.public.articles` in shard `shard0` if the replica is down.

```
gptext-replica drop -i wikipedia.public.articles -s shard0 -r core_node3 -o
```

gptext-config

Performs GPText configuration tasks:

- Edit, add, or upload configuration files in ZooKeeper
- Revert configured files in ZooKeeper
- Edit JVM configuration options
- Upload jar files to the GPText home directory

Syntax

```
gptext-config -h | --help

gptext-config -f file_name -i index_name [-r] [-e] [-b]

gptext-config -a append_file -f file_name -i index_name

gptext-config -u path/local_file_name -f path/zookeeper_file_name -i index_name

gptext-config -j path/jar_file

gptext-config -o jvm_options

gptext-config -f file_name -i index_name -a file_to_append

gptext-config -k solr_key -s <solr_value>
```

Parameters

-i *index-name* --index=*index-name*	Name of the index.
-f *filename* --file=*filename*	The name of a file to edit, append, or upload. The `-i` option must be included to specify the index. The following files are supported: • `solrconfig.xml` – Contains most of the parameters for configuring Solr itself (see [https://cwiki.apache.org/confluence/display/solr/Configuring+solrconfig.xml](http://wiki.apache.org/solr/SolrConfigXml)). • `schema.xml` – Defines the analysis chains that Solr uses for various different types of search fields (see [Setting up Text Analysis Chains](indexes.html#topic5)). • `stopwords.txt` – Lists words you want to eliminate from the final index. You can also edit language specific stopwords by specifying a filename in the format `stopwords_language_code.txt`, where `language_code` is a two-character code such as `en`, `fr`, or `es`. • `protwords.txt` – Lists protected words that you do not want to be modified by the analysis chain. For example, *iPhone*. • `synonyms.txt` – Lists words that you want replaced by synonyms in the analysis chain. • `emoticons.txt` – Defines emoticons for the `text_sm` social media analysis chain (see [gptext-start](#topic13)).

	• `currency.txt` – Defines exchange rates between one currency and another (see https://cwiki.apache.org/confluence/display/solr/Working+with+Currencies+and+Exchange+Rates). • jar_file – the name of a jar file to upload to *GPText_Install_Directory*/lib/.
-e *command* --editor=*command*	Editor to use. Choices are any editor that takes a filename on the command line as a parameter. For example, vi, vim, emacs, nano, etc. If absent, vi is used.
-a *filename* --append= *filename*	Appends a named file to a configuration file and distributes the resulting files. Requires the `-f` and `-i` parameters. `-f` names the configuration file to which you want to append the file named (including local path) with the `-a` parameter.
-r --revert *filename*	Revert named file to previous version.
-b *n* --batch_size=*n*	How many Solr instances to configure concurrently. The default (64) is generally more than enough. A larger number may increase speed.
-u *local_file_path* --upload *local_file_path*	Upload a configuration file at *local_file_path* to ZooKeeper. Specify the destination Zookeeper file name with the `-f` option and specify the index name with the `-i` option.
-j *jarfile* --jar=*jarfile*	Uploads a jar file to *GPText_Install_Directory*/lib/.
-o *"JVM_Options"*	Modifies JVM options. To ensure that the JVMs are restarted after changing JVM options, restart the GPText cluster using the `gptext-stop` and `gptext-start` utilities.
-k --solr_prop= -s ---solr_val=	Sets a custom property in the `solr.xml` configuration file. The -k option specifies the name of the property. The `-s` option specifies the value. Currently, the only custom GPText property is the `trackCommit` property, which enables or disables commit tracking.

Notes

Use the `gptext-config` utility to edit the configuration files for a specified index.

Warning: Never edit the template configuration files. If you do, every index you create after editing the templates will be created with your modified versions. Use the `gptext-config` utility to ensure that you are editing the configuration files for your index, rather than the template configuration files.

`gptext-config` automatically reindexes after editing files if the configuration changes made require it.

If you use the `-f` (`--file`) parameter to edit one of the index configuration files, GPText automatically places the edited file in its proper directory.

To move an index configuration file from the local file system to the index configuration directory in all of the segments, specify the local file with the `-u` (`--upload`) option and the destination file with the `-f` (–file`) option.

The `-k` (`--solr_prop`) and `-s` (`--solr_val`) parameters must always be used together. They set a custom GPText property in the `solr.xml` configuration file. Currently, they are only used to enable or disable commit tracking. Changes to `solr.xml` do not take effect until the GPText instance is restarted.

Examples

1. Edit the `managed-schema` file in index `wikipedia.public.articles`, using the vi editor:

```
gptext-config -f managed-schema -i wikipedia.public.articles -e vi
```

2. Append the file `stopwords.add` to `stopwords.txt` in index `wikipedia.public.articles`:

```
gptext-config -a stopwords.add -f stopwords.txt -i wikipedia.public.articles
```

3. Revert file `managed-schema` in index `wikipedia.public.articles` after editing it.

```
gptext-config -f managed-schema -i wikipedia.public.articles -r
```

4. Upload the local file `custom.txt` to the ZooKeeper file `custom.conf` in index `wikipedia.public.articles`:

```
gptext-config -u custom.txt -f custom.conf -i wikipedia.public.articles
```

5. Upload jar file `text.jar` to the `lib` directory in the GPText home directory:

```
gptext-config -j text.jar
```

6. Set JVM options:

```
gptext-config -o "-Xms256M -Xmx400M"
```

gptext-backup

Backs up a GPText index to a shared file system.

Syntax

```
gptext-backup -h

gptext-backup -p <path> -i <index> -n <name> [-v]
```

Parameters

-h --help	Displays a usage message and exits.
-p *path* --path *path*	The path where the shared file system is mounted on each host. The file system must be accessible from all hosts in the cluster and readable and writable by the gpadmin user.
-i *index_name* --index *index_name*	The name of the GPText index to back up.
-n *backup_name* --name *backup_name*	A name for the backup.

Example

```
gptext-backup -i myindex -p /mnt/backupfs/gptext-backups -n mybackup
```

Notes

You can back up an index so that you can restore it to a different GPText system or avoid having to reindex if the existing index becomes corrupted.

The shared file system must be mounted on all hosts with GPText nodes and must be writable by the gpadmin user. The file system could be, for example, an NFS mount or a SSH server with sshfs support. The file system must be configured and accessible before you execute the `gptext-backup` utility and able to accept connections from each host in the cluster.

The `gptext-backup` utility creates a new subdirectory at the specified path with the backup name specified. The command fails if the directory already exists.

When the backup is complete, the backup directory contains the following:

backup.info
: A text file containing three comma-separated strings: the database name, schema name, and index name for the index that was backed up.

backup.properties
: A text file with properties that describe the backup, such as the date and time the backup started, the name of the backup, and the names of the Solr collection and collection configuration.

zk_backup
: A directory containing the following:

- `collection_state.json` – a JSON file describing the status of the Solr collection.
- `configs/<collection-name>/` – a directory containing copies of the Solr configuration files stored in ZooKeeper for the index, for example `managed-schema`, `solrconfig.xml`, `protwords.txt`, `stopwords.txt`.

snapshot.shard0 … snapshot.shard*N*
: A directory for each shard, with the files containing content of the shard.

If the backup fails—for example if there is insufficient disk space—an error message is displayed, but the backup directory is not removed. Be sure to remove the backup directory before restarting the backup.

gptext-restore

Restore a GPText index from a backup created on a shared file system with the `gptext-backup` utility.

Syntax

```
gptext-restore -h

gptext-restore -i <index_name> -p <path> [-v]
```

Parameters

-h --help	Displays a usage message and exits.
-p *path* --path *path*	The path to the backup directory on each host.
-i *index_name* --index *index_name*	The name of the GPText index to restore. The index must not already exist in the target GPText system.

Example

```
gptext-restore -i myindex -p /mnt/backupfs/gptext-backups/mybackup
```

Notes

Use the `gptext-restore` utility to restore a GPText index backup from a shared file system. You can restore the backup to a new GPText system or restore a backup to recover a corrupted GPText index.

The index you are restoring must not exist. If you are restoring an index to recover a corrupted index, you must first delete the existing index with the `gptext.delete()` UDF. The `gptext-restore` utility creates a new index and will output an error and quit if the index you are restoring exists.

gptext-expand

Expands a GPText cluster by adding new GPText nodes to existing hosts in a GPText cluster or to hosts added by the Greenplum Database `gpexpand` management utility. Replicas for indexes created after the new GPText nodes are added will be distributed across the new and existing nodes. Documents must be reindexed to rebalance replicas on existing hosts or, after expanding the Greenplum cluster, to redistribute the index to new shards.

Synax

```
gptext-expand -h

gptext-expand -e -p <paths> [-v]

gptext-expand -d <database> [-v]
```

Parameters

-h --help	Displays a usage message and exits.
-e --existing	Adds GPText nodes to existing hosts in the GPText cluster. Either the `-e` or the `-d` option must be specified.
-p --expand_paths	Specifies paths to directories where the new GPText nodes' data directories are to be created. These directories should be parallel to the Greenplum Database segment data directories. If there is more than one directory, place them in a comma-delimited list, for example `-p /data1/nodes, /data1/nodes, /data2/nodes`. Required when expanding on existing hosts.
-d --database	Specifies the name of the database containing the gpexpand schema used to expand the Greenplum Database cluster. Either the `-e` or the `-d` option must be specified.
-v --verbose	Displays debug output.

Notes

- The `-p` and `-d` options cannot be used together.
- Existing replicas are not automatically redistributed. To rebalance replicas among the expanded GPText cluster, you must reindex.
- When expanding to new hosts, you must reindex to redistribute the index among existing and new shards.

gptext-uninstall

Uninstalls GPText, including data and installed files. Uninstalls ZooKeeper nodes if they were installed with the GPText installer.

- Stops any running GPText instances.
- Deletes all Solr directories in segment directories.
- Deletes the installation directory.
- Removes all GPText schemas and indexes from all databases.
- Uninstalls ZooKeeper if it was installed with the GPText installer.

Syntax

```
gptext-uninstall -h | --help

gptext-uninstall [-v | --verbose]
```

Parameters

-h --help	Displays a usage message and exits.
-r --restart	Restarts the GPText cluster.
-v --verbose	Displays debug output.

Notes

- To use `gptext-uninstall`, you must have superuser permissions on all databases with GPText schemas.
- `gptext-uninstall` runs only if there is at least one database with a GPText schema.

Examples

1. Uninstall GPText.

   ```
   gptext-uninstall
   ```

zkManager

Checks the ZooKeeper cluster state. If ZooKeeper was installed with GPText, `zkManager` can start or stop the ZooKeeper cluster.

Syntax

```
zkManager [-h | --help]

zkManager state [-v | --verbose]

zkManager start [-v | --verbose]

zkManager stop [-v | --verbose] [-f | --force]
```

Parameters

-h --help	Display a usage message and quit.
-f --force	When used with the `stop` command, performs a forced stop.
-v --verbose	Displays debug output when executing the command.

Notes

- The `zkManager start` and `zkManager stop` commands are only available if the ZooKeeper cluster was installed by the GPText installer.
- By default, all `gptext-*` utilities check the ZooKeeper cluster state. If the cluster is not healthy, the ZooKeeper state information is displayed to warn the user.
- The `nc` (netcat) command must be installed on the master host. Run `nc` in a terminal to ensure the command is installed.

Examples

1. Start the ZooKeeper cluster, if ZooKeeper was installed by the GPText binary:

   ```
   zkManager start
   ```

2. Stop the ZooKeeper cluster, if ZooKeeper was installed by the GPText binary:

   ```
   zkManager stop
   ```

3. Force stop the ZooKeeper cluster, if ZooKeeper was installed by the GPText binary:

   ```
   zkManager stop -f
   ```

4. Check the state of the ZooKeeper cluster:

   ```
   $ zkManager state
   20160603:14:17:01:307386 zkManager:gpdb-sandbox:gpadmin-[INFO]:-Execute zookeeper state process.
   20160603:14:17:01:307386 zkManager:gpdb-sandbox:gpadmin-[INFO]:-   Host                  port   Latency min/avg/max   Mode
   20160603:14:17:01:307386 zkManager:gpdb-sandbox:gpadmin-[INFO]:-   gpdb-sandbox.localdomain   2188   0/0/17            follower
   20160603:14:17:01:307386 zkManager:gpdb-sandbox:gpadmin-[INFO]:-   gpdb-sandbox.localdomain   2189   0/0/17            leader
   20160603:14:17:01:307386 zkManager:gpdb-sandbox:gpadmin-[INFO]:-   gpdb-sandbox.localdomain   2190   0/0/70            follower
   20160603:14:17:06:307386 zkManager:gpdb-sandbox:gpadmin-[INFO]:-Done.
   ```

GPText Configuration Parameters

GPText configuration parameters can be overridden by setting a new value in a Greenplum Database session. Changes made to configuration parameters only affect future GPText operations; existing indexes use the parameter values that were set when they were created.

See Changing GPText Server Configuration Parameters for information about changing configuration parameters and examples.

The following table lists the GPText configuration parameters with their defaults and value constraints.

`admin_timeout`	Timeout, in seconds, for admin requests (create_index, etc.).	`30`	`INT_MAX`	`3600`
`commit_timeout`	Timeout, in seconds, for prepare commit and commit operations.	`30`	`INT_MAX`	`3600`
`delete_timeout`	Timeout, in seconds, for delete requests.	`30`	`INT_MAX`	`3600`
`extension_factor`	Maximum number of replicas that can be added for an index per GPText node after the index is created.	`0`	`10`	`2`
`facet_timeout`	Timeout, in seconds, for faceting queries.	`30`	`INT_MAX`	`3600`
`failover_factor`	Minimum ratio of Solr nodes that must be up in order to create a new index. (`SolrNodesUp/TotalSolrNodes`)	`0.0`	`1.0`	`0.8`
`hl_post_tag`	Markup that `gptext-highlight()` inserts after terms in search results.			`'</em>'`
`hl_pre_tag`	Markup that `gptext-highlight()` inserts before terms in search results.			`'<em>'`
`idx_buffer_size`	Size of indexing buffer in bytes.	`4096`	`67108864`	`4194304`
`idx_delim`	Delimiter to use during indexing.			comma ’`,`’
`idx_encapsulator`	The character optionally used to surround values to preserve characters such as the CSV separator or whitespace.			quote ’`"`’
`idx_escape`	Escape character to use for indexing.			backslash ’\’
`index_timeout`	Timeout, in seconds, for receiving response to indexing operation.	`30`	`INT_MAX`	`3600`
`optimize_timeout`	Timeout, in seconds, for optimize operations.	`30`	`INT_MAX`	`3600`
`ping_timeout`	Timeout, in seconds, for ping requests.	`30`	`INT_MAX`	`120`
`replication_factor`	The number of replicas per shard for a newly created index.	`0`	`10`	`3`
`replication_timeout`	Timeout, in seconds, for replication operations (backup, restore).	`30`	`INT_MAX`	`43200`
`rollback_timeout`	Timeout, in seconds, for rollback operations.	`30`	`INT_MAX`	`3600`
`search_batch_size`	Batch size for search requests.	`1`	`INT_MAX`	`2500000`
`search_buffer_size`	Buffer size for search results, in bytes.	`4096`	`67108864`	`4194304`
`search_param_separator`	Delimiter to use in the `options` parameter of the `gptext.search()` UDF.			`'&'`
`search_post_buffer_size`	Post buffer size for search, in bytes.	`512`	`4194304`	`4096`
`search_timeout`	Timeout, in seconds, for searches.	30	INT_MAX	600
`stats_timeout`	Timeout, in seconds, for obtaining statistics.	`30`	`INT_MAX`	`600`
`terms_batch_size`	Batch size for terms operations.	`1`	`INT_MAX`	`50000`