Greenplum

October 3, 2008

Welcome to Greenplum Database 3.2.0.0

Greenplum Database is a massively parallel processing (MPP) database server that leverages PostgreSQL open-source technology. Greenplum Database 3.2 is a major release which introduces several new features, performance and stability enhancements, and internal changes to the system catalogs. Please refer to the following sections for more information about this release:

New Features in Greenplum Database 3.2

Greenplum MapReduce

MapReduce is a programming model originally developed by Google for processing and analyzing large data sets on an array of commodity servers. Greenplum MapReduce allows programmers who are familiar with the MapReduce paradigm to write map and reduce functions and submit them to the Greenplum Database parallel data flow engine for processing. The Greenplum Database system takes care of the details of distributing the input data, executing the program across a set of machines, handling machine failures, and managing the required inter-machine communication.

In order for Greenplum to be able to process the Map and Reduce functions written by a user, the functions need to be defined in a specially formatted Greenplum MapReduce document, which is then passed to the Greenplum MapReduce client program, `gpmapreduce`, for execution by the Greenplum Database parallel engine.

Supporting Documentation for Greenplum MapReduce

For more information on Greenplum MapReduce, see the following sections of the *Greenplum Database 3.2 Administrator Guide* (`GPAdminGuide.pdf`):

- Chapter 27, Using Greenplum MapReduce
- Appendix B, Management Utility Reference, "gpmapreduce"
- Appendix F, Greenplum MapReduce Specification

In-Database Compression

Greenplum Database 3.2 introduces in-database compression using `gzip` compression technology. This allows customers to increase performance and reduce the space required to store data. Customers can expect to see a 3-10x space reduction with increased effective I/O performance to match (provided that your segment systems have the available CPU power to compress and uncompress the data).

In-database compression is available on tables that utilize the append-only storage model. See Append-Only Tables.

Programmable Parallel Analytics

Greenplum Database 3.2 introduces a new level of parallel analysis capabilities for mathematicians and statisticians. For the first time customers can use the R statistical language (see PL/R Support), or build custom functions using our linear algebra and machine learning primitives (see Advanced Analytic Functions), and run them in parallel directly against data in the database.

PL/R Support

PL/R is a PostgreSQL language extension that allows you to write user-defined functions and aggregates in the R statistical computing language. More information on R is available on the R Project web site: http://www.r-project.org. More information on PL/R is available at: http://www.joeconway.com/plr.

Because R and PL/R are available under the terms of the GNU General Public License, they are packaged separately from the 3.2 Greenplum Database installation. Greenplum has compiled and packaged platform-specific packages of R and PL/R (available for download on Greenplum Network). In order to use PL/R functions, R and PL/R must be installed on every host in your Greenplum Database array.

This release includes built-in support for PL/R in the `CREATE LANGUAGE` command. Once you have installed the required R and PL/R libraries on you Greenplum hosts, you can use `CREATE LANGUAGE` to add PL/R support to your database. For example:

```
CREATE LANGUAGE plr;
```

Advanced Analytic Functions

Greenplum Database 3.2 introduces the following new built-in functions for advanced statistics and analysis. These functions are executed in parallel within Greenplum Database:

Table A.1 New Advanced Analytic Functions in 3.2

Function Name	**Description**
`matrix_add(`*`array`*`[], `*`array`*`[])`	Adds two two-dimensional matrices. The matrices must be conformable.
`matrix_multiply(`*`array`*`[], `*`array`*`[])`	Multiplies two two-dimensional arrays. The matrices must be conformable.
`matrix_multiply(`*`array`*`[], `*`expr`*`)`	Multiplies a two-dimensional array and a scalar numeric value.
`matrix_transpose(`*`array`*`[])`	Transposes a two-dimensional array.
`pinv(`*`array`*`[])`	Calculates the Moore-Penrose pseudoinverse of a matrix.
`unnest(`*`int`*`[])`	Transforms a one dimensional array into rows. Returns a set of `anyelement`, a polymorphic psuedotype in PostgreSQL.
`sum(`*`array`*`[])`	Performs matrix summation. Can take as input a two-dimensional array that is treated as a matrix.
`pivot_sum(`*`label`*`[], `*`label`*`, `*`expr`*`)`	A pivot aggregation using sum to resolve duplicate entries.
`mregr_coef(`*`expr`*`, `*`array`*`[])`	The functions `mregr_coef` and `mregr_r2` are both multiple linear regression aggregates. `mregr_coef` calculates the regression coefficients.
`mregr_r2(`*`expr`*`, `*`array`*`[])`	The functions `mregr_coef` and `mregr_r2` are both multiple linear regression aggregates. `mregr_r2` calculates the r-squared error value.
`nb_classify(`*`text`*`[], `*`bigint`*`, `*`bigint`*`[], `*`bigint`*`[])`	Classify rows using a naive Bayes classifier. This function is used with a baseline of training data to predict the classification of new rows.

Supporting Documentation for Advanced Analytic Functions

For more information on these functions, see the following section of the *Greenplum Database 3.2 Administrator Guide* (`GPAdminGuide.pdf`):

- Chapter 16, Querying Data, "Advanced Analytic Functions"

Enhanced Database Monitoring

Greenplum Database 3.2 introduces a new infrastructure for monitoring database performance and usage. This new infrastructure enables the gathering and storage of comprehensive metrics about system utilization and query performance.

This functionality is currently disabled, and will be enabled in an upcoming 3.2 maintenance release. This maintenance release will fully activate the Greenplum Performance Monitor functionality and provide the data collection, SQL storage, and graphical user interface components of the infrastructure.

Enhanced Table Partitioning

This underlying partitioning infrastructure is the same in 3.2 as in previous Greenplum Database releases. Partitioned tables use a combination of table inheritance and `CHECK` constraints. Table inheritance creates a persistent relationship between a parent table and its child table partitions, so that all of the schema information from the parent table propagates down to the child table partitions. `CHECK` constraints both limit the data a child table partition can contain, and are used to eliminate partitions at query runtime that are not relevant to a given query predicate.

What has changed about table partitioning in 3.2, is the way partitioned tables are created and managed. In previous releases, tables were partitioned by running the `gpcreatepart` management utility or via manual SQL operations. Also, it was necessary to use rewrite rules to route data inserted through the parent table to the correct child table. Rewrite rules did not apply to `COPY` commands, only `INSERT`, and significantly slowed load operations when loading through the parent table.

In 3.2, partitioned tables are now created with one simple `CREATE TABLE` command. New syntax has been added to `CREATE TABLE` to support the creation of multi-level range and list partition designs. Similarly, `ALTER TABLE` now supports new syntax for managing existing partition designs. With the new 3.2 partitioning, rewrite rules are no longer needed to route incoming data to the correct partition. The partition design and rules are now stored in system catalog tables, which significantly improves runtime rule evaluation and load performance.

The old partitioning utilities have been deprecated in 3.2. `gpcreatepart` is no longer distributed with the Greenplum Database 3.2 installation. `gpaddpart` and `gpdeletepart` are available in 3.2, but can only be used to maintain old 3.1 style partitions. For users who have existing 3.1 style partitioned tables and wish to upgrade their partition designs to 3.2, Greenplum will provide an upgrade utility to assist with the migration. Contact *Greenplum Customer Support* if you are interested in this partition upgrade utility.

Supporting Documentation for Enahanced Table Partitioning

For more information on enhanced table partitioning in 3.2, see the following sections of the *Greenplum Database 3.2 Administrator Guide* (`GPAdminGuide.pdf`):

- Chapter 14, Defining Database Objects, “Partitioning Large Tables”
- Appendix A, SQL Command Reference, “CREATE TABLE”
- Appendix A, SQL Command Reference, “ALTER TABLE”

Append-Only Tables

Append-only tables are a new table storage model for large tables in 3.2. Traditional heap table storage tables favor OLTP-type workloads where data is often modified after it is initially loaded. `UPDATE` and `DELETE` operations require table-level locking and row-level versioning information to be stored in order to ensure that database transactions are processed reliably.

In most data warehouse workloads, however, most of the data resides in denormalized fact tables, which are typically the largest tables in the system. Fact tables are typically appended to in batches and then accessed by read-only queries. Data is seldom changed after it is written. These append-only characteristics allow a range of storage and performance optimizations to be applied.

Moving large fact tables to an append-only storage model eliminates the storage overhead of the per-row update visibility information (about 20 bytes per row is saved). This allows for a leaner and easier-to-optimize page structure. This page structure also allows for in-database compression (using `gzip`).

Append-only tables offer a more efficient storage format than traditional heap storage tables, but lack certain features available with heap tables:

- Cannot `UPDATE` rows in an append-only table
- Cannot `DELETE` rows from an append-only table
- Cannot `ALTER TABLE...ADD COLUMN` to an append-only table
- Cannot add indexes to an append-only table

The storage model of a table is declared at `CREATE TABLE` time. The storage model of a table cannot be changed on an existing table. The `WITH` clause of the `CREATE TABLE` command is used to declare the storage options of the table. If not declared, the table will be created as a regular heap-storage table. For example, to create an append-only table with compression level of 5:

```
CREATE TABLE foo (a int, b text) WITH (appendonly=true,
compresslevel=5);
```

Supporting Documentation for Append-Only Tables

For more information on append-only tables, see the following sections of the *Greenplum Database 3.2 Administrator Guide* (`GPAdminGuide.pdf`):

- Chapter 14, Defining Database Objects, “Choosing the Table Storage Type”
- Appendix A, SQL Command Reference, “CREATE TABLE”

PostgreSQL 8.3 Regular Expression Functions

Greenplum Database is based on PostgreSQL 8.2.9, but the following regular expression functions from PostgreSQL 8.3 have been added to Greenplum Database 3.2:

```
regexp_matches(<string>, <pattern> [, <flags> ])
regexp_split_to_table(<string>, <pattern> [, <flags> ])
regexp_split_to_array(<string>, <pattern> [, <flags> ])
```

For more information on these functions, see the PostgreSQL 8.3 documentation at: http://www.postgresql.org/docs/8.3/static/functions-matching.html

Memory Overcommit Protection

On some OS platforms, a segment host can become unresponsive when it runs out of memory. To protect against this condition occurring, Greenplum 3.2 introduces a new server configuration parameter:

```
gp_vmem_protect_limit
```

This parameter sets the amount of memory (in MBs) that all `postgres` processes of a segment instance can consume. To prevent over allocation of memory, set to:

```
(physical_memory/segments) + (swap_space/segments/2)
```

For example, on a segment host with 16GB physical memory, 16GB swap space, and 4 segment instances the calculation would be:

```
(16/4) + (16/4/2) = 6GB
6 * 1024 = 6144MB
```

If a query causes this limit to be exceeded, memory will not be allocated and the query will fail. Note that this is a local parameter and must be set for every segment instance.

This parameter replaces the now deprecated `gp_process_memory_cutoff` parameter which was harder to calculate. It set a per-process memory limit rather than a per-segment limit.

Performance Enhancement for Single Row INSERT Operations

In Greenplum Database 3.2, `INSERT` rows are now distributed to segments in the same fashion as a `COPY` command. Inserted rows are dispatched directly to their target segment instance without requiring a motion operation. This significantly improves performance of single-row-insert statements anywhere from 100-300 percent.

Performance Enhancement for DISTINCT Aggregate Operations

In previous releases, it was often beneficial to rewrite queries involving `DISTINCT`-qualified aggregate functions, for example `COUNT(DISTINCT x)`, in such a way as to eliminate duplicate values of the distinct attributes before an aggregation.

In 3.2, optimizer improvements render such rewrites unnecessary. In fact, the query optimizer's internal transformation will now usually produce a plan that is superior to what can be achieved by rewriting the SQL.

Performance Enhancement for Hash Aggregate Operations

This release includes enhancements to the query executor for improved performance of hash aggregation (HashAgg) operations. In cases where the query planner under estimates the cost of a HashAgg operation, the executor will now attempt to reduce the size of data that spills to disk during execution, thereby improving performance. Aggregation of numeric data has also been improved. Users should see a 10-20% performance enhancement on applicable queries involving hash aggregation.

Enhanced Interconnect Scalability

The *interconnect* is the networking layer of Greenplum Database. When a user connects to a database and issues a query, processes are created on each of the segments to handle the work of that query. The interconnect refers to the inter-process communication between the segments, as well as the network infrastructure on which this communication relies.

In previous releases, the interconnect used TCP (Transmission Control Protocol) to send messages over the network. With TCP, Greenplum had a scalability limit of 1000 segment instances. To remove this limit, Greenplum has introduced a reliable UDP-based transmission protocol that is optimized for data flow processing.

Changed Features in Greenplum Database 3.2

- Management Utility Changes
- SQL Command Changes
- Server Configuration Parameter Changes
- System Catalog Changes

Management Utility Changes

The following management utilities have new or changed command-line options in 3.2:

Table A.2 Management Utility Changes in 3.2

Utility	New / Changed Options	Deprecated Options
`gpaddmirrors`	**-i** *mirror_config_file* (file format changed) **-m** *datadir_config_file* **-o** *output_sample_mirror_config_file* **--version**	**-c** `ssh	rsync` (rsync always used in 3.2) **-v** (changed to `--version`)
`gpaddpart`		Old partitioning utilities are deprecated in 3.2. This utility can still be used to maintain 3.1.x.x style partitioned tables, but is not compatible with 3.2 partitioned tables.
`gpchecknet`	New utility for testing network performance	
`gpcheckos`		**-u** *ssh_user* **-p** *ssh_port* **-c** *custom_config_file*
`gpcheckperf`		**-r** `{n|N}` (network performance test removed - see `gpchecknet`)
`gpcreatepart`		Deprecated and removed from 3.2. To create and maintain a new partitioned table in 3.2, use the `CREATE TABLE` and `ALTER TABLE` commands.
`gpcrondump`	**--column-inserts** **-E** *encoding* **--inserts** **--no-owner** **--no-privileges** **--oids** **-u** *backup_directory* **--use-set-session-authorization** **-w** *dbid* **-y** *report_file*	
`gpdeletepart`		Old partitioning utilities are deprecated in 3.2. This utility can still be used to maintain 3.1.x.x style partitioned tables, but is not compatible with 3.2 partitioned tables.
`gpdetective`	New diagnostic and log capture utility	
`gpfdist`	Now supports `bzip2` compressed files in addition to `gzip`	**-q** *quote_character* **-h** *header* **-x** *escape_character* (Data formatting options removed from gpfdist command-line. gpfdist now looks to the external table definition `FORMAT` clause for instructions on how to parse the data for both TEXT and CSV formatted files)
`gpinitstandby`		**-i** (do not start standby sync agent)

Table A.2 Management Utility Changes in 3.2

Utility	New / Changed Options	Deprecated Options
`gpmapreduce`	New client utility for submitting Greenplum MapReduce jobs	
`gpmigrator`	New 3.1.x.x to 3.2 upgrade utility	
`gprecoverseg`	**-i** *input_config* **-o** *output_config* **-S** *seg_dbid* **-s** *primary_datadir,mirror_datadir*	**-c** `ssh	rsync` (new smart sync added in 3.2) **-v** (changed to `--version`)
`gpscp`		**-u** *ssh_user* **-p** *ssh_port*
`gpssh`		**-u** *ssh_user* **-p** *ssh_port*
`gpssh-exkeys`		**-u** *ssh_user* **-p** *ssh_port*
`gpstart`		**-z** (force start)
`gpstate`		-**Q** (quick status)

SQL Command Changes

The following SQL commands have new or changed syntax in 3.2.

Table A.3 Changed SQL Commands in 3.2

Command	Description of Change
`ALTER TABLE`	New clauses added for altering partitioned tables (`ALTER | DROP | RENAME | TRUNCATE | ADD | SPLIT | EXCHANGE PARTITION`)
`CREATE TABLE`	New `PARTITION BY` clause for creating partitioned tables New append-only storage options added to the `WITH` clause
`SELECT`	Aggregate expressions now allow a `FILTER` clause: *aggregate_name* (*expression* [, ...]) [**FILTER** (WHERE *condition*)]

Server Configuration Parameter Changes

- New Parameters
- Parameters with Changed Defaults
- Deprecated Parameters

New Parameters

Table A.4 New Server Configuration Parameters in 3.2

Parameter	Description
gp_enable_agg_distinct	Enables or disables two-phase aggregation to compute a single distinct-qualified aggregate. This applies only to subqueries that include a single distinct-qualified aggregate function.
gp_enable_agg_distinct_pruning	Enables or disables three-phase aggregation and join to compute distinct-qualified aggregates. This applies only to subqueries that include one or more distinct-qualified aggregate functions.
gp_enable_fast_sri	When set to `on`, the query planner plans single row inserts so that they are sent directly to the correct segment instance (no motion operation required). This significantly improves performance of single-row-insert statements.
gp_enable_multiphase_agg	Enables or disables the query planner's use of two or three-stage parallel aggregation plans. This approach applies to any subquery with aggregation. If `gp_enable_multiphase_agg` is off, then `gp_enable_agg_distinct` and `gp_enable_agg_distinct_pruning` are disabled.
gp_external_grant_privileges	(Added in 3.1.1.4) Enables or disables non-superusers to issue a `CREATE EXTERNAL [WEB] TABLE` command in cases where the `LOCATION` clause specifies `http` or `gpfdist`.
gp_hashagg_compress_spill_files	When a hash aggregation operation spills to disk during query processing, specifies the compression algorithm to use on the spill files. If using `zlib`, it must be in your $PATH on all segments.
gp_max_local_distributed_cache	(Added in 3.1.1.5) Sets the number of local to distributed transactions to cache. Higher settings may improve performance.
gp_safefswritesize	Specifies a minimum size for safe write operations to append-only tables in a non-mature file system. When a number of bytes greater than zero is specified, the append-only writer adds padding data up to that number in order to prevent data corruption due to file system errors. Each non-mature file system has a known safe write size that must be specified here when using Greenplum Database with that type of file system. This is commonly set to a multiple of the extent size of the file system; for example, Linux ext3 is 4096 bytes, so a value of 32768 is commonly used.
gp_snapshotadd_timeout	(hidden parameter introduced in 3.1.1.9) Under very heavy workload, concurrent sessions can sometimes collide waiting for a snapshot of the database at the segment level. When this occurs, the segment instance may fail with a message in its log such as: '`SharedSnapshotAdd: found existing entry for our session-id`' As a work around to this issue, increase this timeout (in seconds) at the segment. To increase the timeout, you must set this parameter in the postgresql.conf of all segment instances.
gp_vmem_protect_limit	Sets the amount of memory (in number of MBs) that all postgres processes of a segment instance can consume. To prevent over allocation of memory, set to: `(physical_memory/segments) + (swap_space/segments/2)`
log_dispatch_stats	When set to `on`, logs information about the dispatch of each statement.
max_appendonly_tables	Sets the maximum number of append-only relations that can be written to or loaded concurrently. Append-only table partitions and subpartitions are considered as unique tables against this limit. Increasing the limit will allocate more shared memory at server start.

Parameters with Changed Defaults

The following parameters have changed default values from 3.1.1.x:

- `enable_mergejoin=off` (was on in 3.1.1.x)
- `enable_nestloop=off` (was on in 3.1.1.x)
- `log_line_prefix=|%m|%u|%d|%p|%I|%X|:-` (`%X` now logs distributed transaction ID, local transaction ID)
- `gp_enable_agg_distinct=on` (was off in 3.1.1.x)
- `gp_selectivity_damping_for_scans=on` (hidden parameter - was off in 3.1.1.x)
- `gp_selectivity_damping_for_joins=off` (hidden parameter - was on in 3.1.1.x)
- `wal_send_client_timeout=30s` (hidden parameter - was a half second in 3.1.1.x)

Deprecated Parameters

The following parameters are removed in 3.2. If these are referenced in your `postgresql.conf` files, they must be removed.

- `gp_propagate_broadcast_cost`
- `gp_process_memory_cutoff` (replaced by `gp_vmem_protect_limit`)

System Catalog Changes

The following new system catalog tables and views have been added or modified in 3.2.

Table A.5 New System Catalogs in 3.2

Catalog	Description
`gp_distributed_log`	New system view for distributed transaction management
`gp_master_mirroring`	New system table to track the state of synchronization between the primary master and the standby master
`gp_transaction_log`	New system view for distributed transaction management
`pg_class`	Changes made to support append-only tables: `relkind`, `relstorage`, `relaosegxid`, `relaosegrelid` columns
`pg_partition`	New system table for table partitioning
`pg_partition_columns`	New system view for table partitioning
`pg_partition_rule`	New system table for table partitioning
`pg_partition_templates`	New system view for table partitioning
`pg_partitions`	New system view for table partitioning

Resolved Issues in Greenplum Database 3.2

This section lists the customer reported issues that are now resolved in Greenplum Database 3.2. The resolved issues are divided into the following categories:

Resolved Query Dispatch and Execution Issues

Table A.6 Resolved Issues in 3.2 - Query Dispatch and Execution

Issue Number	Description

2875	**Queries of Temporary Tables Fail After Automatic Segment Failover**
	In 3.1.1.3 and later, certain queries involving temporary tables would fail when a segment failure occurred mid-stream. After automatic failover to the mirror, users would sometimes encounter the following errors:
	`ERROR:  Interconnect error QD lost contact with QE...`
	`ERROR:  freeGangsForPortal: Bad primary writer gang...`

2963	**Error Using the pg_size_pretty Function** Calculating the database size using the PostgreSQL function `pg_size_pretty` resulted in catalog-related errors.
3022	**Queries Using Date Functions Return Unexpected Results** In 3.1.1.4 and later, certain queries involving table joins on the results of the `alldates()` or `current_date()` functions were not planned correctly. This could result in unexpected result sets or the following error: `QD unable to connect...`
3295	**Failed Query with Window Function and ORDER BY clause** In 3.1.1.x releases, certain queries using window functions (such as rank) and an ORDER BY clause where the ordering attribute was the output column of the window function would fail with the following error: `Unexpected internal error: QD process received signal SIGSEGV`
3511	**Failed Query with Redundant DISTINCT-qualified Aggregates** In 3.1.1.x releases, certain queries with redundant `DISTINCT`-qualified aggregate clauses would fail with the following error: `Unexpected internal error: QD process received signal SIGSEGV`

Resolved Data Loading Issues

Table A.7 Resolved Issues in 3.2 - Data Loading

Issue Number	Description
2462	**External WEB Tables do not Return an Error if EXECUTE Command Fails** In 3.1.x.x releases, external web tables that were defined with the `EXECUTE` clause did not return an error when the command ran by the external table failed at runtime.

Table A.7 Resolved Issues in 3.2 - Data Loading

Issue Number	Description
2984	**COPY Loads Errant Characters When Client and Server Encodings are Different** In 3.1.1.3 and later, COPY would insert errant characters at the beginning of strings when converting between certain client and server encodings (such as `UTF8` to `EUC_JP`).

3281	**Failed gpfdist Process Under Heavy Load**
	In releases prior to 3.1.1.6, multiple concurrent accesses to an external table that used the `gpfdist` protocol would sometimes cause the `gpfdist` file server process to fail with the following errors:
	`ERROR: connection failed while gpfdist://...`
	`WARN: port_getn: Invalid argument...`

Resolved Management Utility Issues

Table A.8 Resolved Issues in 3.2 - Management Utilities

Issue Number	Description
2651	**gpstop -i Results in Core Dump on the Master** Using the `gpstop` utility with the `-i` option for immediate shutdown resulted in a core dump on the master and SEGV errors. Error handling in the utility has been changed to resolve this issue. Note that `gpstop -i` is not the recommended way to cleanly shutdown a Greenplum system.
2793	**Not Able to Add Partitions in Middle of Range** The 3.1.x.x utilities `gpcreatepart` and `gpaddpart` were unable to add child partitions except at the end of a range of tables. These utilities are deprecated in 3.2. Enhanced table partitioning in this release resolves this issue.

Resolved Backup and Restore Issues

Table A.9 Resolved Issues in 3.2 - Backup and Restore

Issue Number	Description
3645	**gp_dump Locks Schemas not Involved in Dump Operation** In 3.1.1.x releases, using gp_dump to dump a single schema would lock all other schemas in the database as well. gp_dump has been modified in 3.2 to obtain exclusive locks on just the specific tables or schemas selected for backup.

Resolved Transaction and Concurrency Issues

Table A.10 Resolved Issues in 3.2 - Transactions and Concurrency

Issue Number	Description
2593	**Error dropping partitioned tables** In 3.1.x x releases, dropping a table with a large number of partitions would sometimes result in the following error: `two-phase state file for transaction is corrupt...` This was due to a limit on the size of the file that saves the prepare state for a transaction. The limit has been modified in this release to resolve this issue.
3164	**Too many updates/deletes within transaction error in serializable transactions** In 3.1.1.4 and later, certain serializable transactions containing function calls or `TRUNCATE` commands would fail with the error: `ERROR: Too many updates/deletes within transaction...`

Table A.10 Resolved Issues in 3.2 - Transactions and Concurrency

Issue Number	Description
3266	**SEGV during rollback of transaction** In prior 3.1.x.x releases, when rolling back a query with `pg_cancel_backend`, the system encountered a SEGV error and the postmaster process failed. This issue was caused by certain server configuration parameters not being propagated correctly to the segments and has been fixed in this release.
3726	**Snapshot Collision Timeout Causes Failed Segment** In prior 3.1.1.x releases, under very heavy workload concurrent sessions had the potential to collide waiting for a snapshot of the database at the segment level. Should this occur, the segment instance would fail with a message in its log such as: `SharedSnapshotAdd: found existing entry for our session-id` As a work around to this issue, a new server configuration parameter has been added, `gp_snapshotadd_timeout` which allows you to increase the timeout (in seconds) at the segment. To increase the timeout, you must set this parameter in the `postgresql.conf` of all segment instances and restart your system.

Known Issues in Greenplum Database 3.2

This section lists the customer reported issues that remain unresolved in Greenplum Database 3.2. Work arounds are provided where applicable.

Table A.11 Known Issues in 3.2

Issue Number	Description
1589	**PostgreSQL Statistics Views and Functions do not Work as Expected in Greenplum Database** PostgreSQL has a number of views (pg_stat_* , pg_statio_*) for showing usage statistics. All of these views only report on the usage of the master (system catalogs), not the usage of user data on the segments. Many of the PostgreSQL statistics functions have the same problem. For example, pg_stat_get_tuples_inserted() shows only those inserts into the master (usually 0), not the number inserted into the table in question.
2115	**Query Planner not Choosing Bitmap Indexes** The Greenplum query planner can overestimate the cost of using bitmap index operations. To favor the use of bitmap indexes, users can temporarily experiment with the following server configuration parameters before a query run: `gp_segments_for_planner=4` (or less) `enable_hashjoin=off`
2516, 796	**ORDER BY Clause Ignored in Views** If you create a view with an `ORDER BY` clause, the `ORDER BY` clause is ignored when you do a `SELECT` from the view.
2535	**ALTER TABLE...ALTER COLUMN Disabled for Indexed Columns** Changing the datatype for indexed columns is not allowed. If you need to alter the data type of an indexed column, Greenplum recommends dropping the index, altering the column datatype, and then recreating the index.
2553	**First Query of a Session Always Requires a Reconnect in readonly Mode** When running in readonly fault tolerance mode and a segment is down, the first query of a session will always attempt to reconnect when a segment is down. This is because failed segments are not marked as invalid in the *gp_configuration* table when running in readonly mode.

Table A.11 Known Issues in 3.2

Issue Number	Description
2980	**Primary Keys not Enforced on Partitioned Tables** To be able to enforce a primary key or unique constraint, the primary or unique key columns must start with the partitioning key column.

3125, 3213	**Transaction Within a Function Not Recognized as a Sub-Transaction**
	When a function containing multiple transaction blocks is run and an error occurs in one transaction block, the entire function exits with the errors:
	`ERROR: The distributed transaction 'Prepare' broadcast failed to one or more segments`
	`ERROR:  current transaction is aborted, commands ignored until end of transaction block`

3530	**Concurrent Vacuum Operations Cause Bitmap Index Errors** In 3.1.1.x releases, multiple concurrent VACUUM operations on tables with a bitmap index can sometimes cause the following error: `ERROR: could not open relation...` Users who encounter this error must run a `REINDEX` command on tables with bitmap indexes after any inserts, updates or loads in order to update the bitmap index.
3592	**Confusing Error Message When in Read-only Fault Operational Mode** When a segment failure is detected, the system always issues the following message when it fails over to the mirror: `WARNING: Greenplum Database detected segment failure(s), system is reconnected in read-write mode` This error message is confusing because it says the system is in read-write mode even when it is actually in read-only mode.
3824	**Using gp_dump From a 3.2 Installation Fails When Run on 3.1.x.x Databases** Using the 3.2 version of the `gp_dump` utility on a 3.1.x.x installation fails with the following error: `ERROR: relation "pg_partition_rule" does not exist` As a work around you can use your current 3.1 version of `gp_dump` against your 3.1 system and use the 3.2 version of `gp_restore` to restore your data into a 3.2 system.
3837	**gpinitsystem -p Option Does Propagate Parameter Changes to Segments** The `gpinitsystem` utility has a `-p` option, which allows you to specify a file containing server configuration parameter settings to include in the `postgresql.conf` files of all segments upon system initialization. This option is currently not passing settings to the segments as expected. As a work around, you can edit `$GPHOME/share/postgresql/postgresql.conf.sample` prior to initialization, copy it to all segment host installations using `gpscp`, and then run `gpinitsystem`.

Upgrading to Greenplum Database 3.2 from 3.1

For users currently running Greenplum Database 3.1, there are two available upgrade paths:

- **In-Place Upgrade to 3.2** — Greenplum provides a utility to upgrade an existing Greenplum Database 3.1.x.x installation to 3.2. This utility makes the required changes to the system catalogs in place without requiring users to dump and restore their databases. The upgrade utility is packaged separately from the Greenplum Database 3.2 distribution and will be generally available within 30 days of the release of 3.2.

- **Upgrade via Backup/Restore** — For systems that have sufficient disk capacity to accommodate two simultaneous versions, an upgrade using dump and restore is another possible upgrade path. This involves taking a backup of your 3.1.x.x databases and global objects (using `gp_dump` or `gpcrondump`), initializing a new 3.2 Greenplum Database system (using `gpinitsystem`), and then restoring your databases and global objects into your new 3.2 Greenplum Database system (using `gp_restore` or `gpdbrestore`). See the *Greenplum Database Administrator Guide* for more information on the initialization and backup/restore procedures.

Installing Greenplum Database 3.2 (New Users)

These are the high-level steps to install and initialize a new Greenplum Database system. For detailed instructions, please see the installation chapter of the *Greenplum Database 3.2 Administrator Guide*.

1. Run the installer on the Greenplum Database master host.
2. As root, set the OS tuning parameters for your platform on all Greenplum hosts.
3. Allocate a `gpadmin` user to own and run your installation. This user must exist on all Greenplum hosts.
4. Source the `greenplum_path.sh` file in your `gpadmin` user profile (`.bashrc`). This sets the environment variables needed by Greenplum Database.
5. Create your data directory locations on all Greenplum hosts.
6. Use the `gpssh-exkeys` utility to exchange SSH keys between all hosts in your Greenplum array. Note that for a single host demo configuration you still must exchange ssh keys between the current host and itself.
7. (multi-host configuration only) Use the `gpscp` and `gpssh` utilities to copy and install the Greenplum Database software on all segment hosts.
8. Use the `gpinitsystem` utility to initialize and start your Greenplum Database system. This utility requires a configuration file. For example:

   ```
   gpinitsystem -c gp_init_config
   ```

   A sample `gp_init_config` configuration file can be found in `$GPHOME/docs/cli_help/gp_init_config_example`. Edit this file to reflect your desired Greenplum Database array configuration.

Greenplum Database 3.2 Documentation

The following Greenplum Database documentation is available in the `$GPHOME/docs` directory of your Greenplum installation, or you can go to http://gpn.greenplum.com to download the latest documentation:

GPAdminGuide.pdf - *Greenplum Database 3.2 Administrator Guide*

In 3.2, the previous User, Administrator, Performance and Overview Guides are now combined into one book, the *Greenplum Database 3.2 Administrator Guide.*