Backend flowchart

From PostgreSQL wiki

This material is referenced by a flowchart (http://www.postgresql.org/developer/backend/) .

Contents

Initialization

bootstrap (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/bootstrap)

Creates initial template database via initdb Because PostgreSQL requires access to system tables for almost every operation, getting those system tables in place is a problem. You can't just create the tables and insert data into them in the normal way, because table creation and insertion requires the tables to already exist. This code *jams* the data directly into tables using a special syntax used only by the bootstrap procedure.

main (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/main)

Passes control to postmaster or postgres This checks the process name(argv[0]) and various flags, and passes control to the postmaster or postgres backend code.

postmaster (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/postmaster)

Controls postgres server startup/termination This creates shared memory, and then goes into a loop waiting for connection requests. When a connection request arrives, a *postgres* backend is started, and the connection is passed to it.

libpq (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/libpq)

Backend libpq library routines This handles communication to the client processes.

Main Query Flow

tcop (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/tcop)

Traffic cop, dispatches request to proper module This contains the *postgres* backend main handler, as well as the code that makes calls to the parser, optimizer, executor,

and *commands* functions.

parser (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/parser)

Converts SQL query to query tree This converts SQL queries coming from *libpq* into command-specific structures to be used by the optimizer/executor, or *commands* routines. The SQL is lexically analyzed into keywords, identifiers, and constants, and passed to the parser. The parser creates command-specific structures to hold the elements of the query. The command-specific structures are then broken apart, checked, and passed to *commands* processing routines, or converted into *Lists*of *Nodes* to be handled by the optimizer and executor.

rewrite (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/rewrite)

Rule and view support

optimizer (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/optimizer)

Creates path and plan

This uses the parser output to generate an optimal plan for the executor.

optimizer_path (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/optimizer/path)

Creates path from parser output This takes the parser query output, and generates all possible methods of executing the request. It examines table join order, *where* clause restrictions, and optimizer table statistics to evaluate each possible execution method, and assigns a cost to each.

optimizer_geqo (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/optimizer/geqo)

Genetic query optimizer optimizer/path evaluates all possible ways to join the requested tables. When the number of tables becomes great, the number of tests made becomes great too. The Genetic Query Optimizer considers each table separately, then figures the most optimal order to perform the join. For a few tables, this method takes longer, but for a large number of tables, it is faster. There is an option to control when this feature is used.

optimizer_plan (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/optimizer/plan)

Optimizes path output This takes the *optimizer/path* output, chooses the path with the least cost, and creates a plan for the executor.

optimizer_prep (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/optimizer/prep)

Handle special plan cases This does special plan processing.

optimizer_util (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/optimizer/util)

Optimizer support routines This contains support routines used by other parts of the optimizer.

executor (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/executor)

Executes complex node plans from optimizer This handles *select, insert, update,* and *delete* statements. The operations required to handle these statement types include heap scans, index scans, sorting, joining tables, grouping, aggregates, and uniqueness.

Command Support

commands (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/commands)

Commands that do not require the executor These process SQL commands that do not require complex handling. It includes *vacuum, copy,* alter, create table, create type, *and many others. The code is called with the structures generated by the parser. Most of the routines* do some processing, then call lower-level functions in the catalog directory to do the actual work.

catalog (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/catalog)

System catalog manipulation This contains functions that manipulate the system tables or catalogs. Table, index, procedure, operator, type, and aggregate creation and manipulation routines are here. These are low-level routines, and are usually called by upper routines that pre-format user requests into a predefined format.

access (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/access)

Various data access methods These control the way data is accessed in heap, indexes, and transactions.

access_common (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/access/common)

Common access routines

access_gin (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/access/gin)

Generalized inverted index access method

access_gist (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/access/gist)

generalized search tree access method

access_hash (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/access/hash)

hash access method

access_heap (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/access/heap)

heap is use to store data rows

access_index (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/access/index)

used by all index types

access_nbtree (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/access/nbtree)

Lehman and Yao's btree management algorithm

access_spgist (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/access/spgist)

Space-Partitioned GiST access method

access_transam (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/access/transam)

transaction manager (BEGIN/ABORT/COMMIT)

nodes (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/nodes)

creation/manipulation of nodes and lists PostgreSQL stores information about SQL queries in structures called nodes. *Nodes* are generic containers that have a *type* field and then a type-specific data section. Nodes are usually placed in *Lists.* A *List* is container with an *elem* element, and a *next* field that points to the next *List.* These *List* structures are chained together in a forward linked list. In this way, a chain of *List* s can contain an unlimited number of *Node* elements, and each *Node* can contain any data type. These are used extensively in the parser, optimizer, and executor to store requests and data.

storage (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/storage)

Manages various storage systems These allow uniform resource access by the backend.

storage_buffer (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/storage/buffer)

Shared buffer pool manager

storage_file (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/storage/file)

File manager

storage_freespace (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/storage/freespace)

Free space map

storage_ipc (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/storage/ipc)

Semaphores and shared memory

storage_large_object (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/storage/large_object)

Large objects

storage_lmgr (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/storage/lmgr)

Lock manager

storage_page (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/storage/page)

Page manager

storage_smgr (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/storage/smgr)

Storage/disk manager

utils (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils)

support routines

utils_adt (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/adt)

built-in data type routines This contains all the PostgreSQL builtin data types.

utils_cache (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/cache)

system/relation/function cache routines PostgreSQL supports arbitrary data types, so no data types are hard-coded into the core backend routines. When the backend needs to find out about a type, is does a lookup of a system table. Because these system tables are referred to often, a cache is maintained that speeds lookups. There is a system relation cache, a function/operator cache, and a relation information cache. This last cache maintains information about all recently-accessed tables, not just system ones.

error (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/error)

error reporting routines Reports backend errors to the front end.

utils_fmgr (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/fmgr)

function manager This handles the calling of dynamically-loaded functions, and the calling of functions defined in the system tables.

utils_hash (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/hash)

hash routines for internal algorithms These hash routines are used by the cache and memory-manager routines to do quick lookups of dynamic data storage structures maintained by the backend.

utils_init (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/init)

various initialization stuff

utils_mb (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/mb)

single and multibyte encoding

utils_misc (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/misc)

miscellaneous stuff

utils_mmgr (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/mmgr;)

memory manager(process-local memory) When PostgreSQL allocates memory, it does so in an explicit context. Contexts can be statement-specific, transaction-specific, or persistent/global. By doing this, the backend can easily free memory once a statement or transaction completes.

utils_resowner (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/resowner)

resource owner tracking

utils_sort (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/sort)

sort routines for internal algorithms When statement output must be sorted as part of a backend operation, this code sorts the tuples, either in memory or using disk files.

utils_time (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/utils/time)

transaction time qualification routines These routines do checking of tuple internal columns to determine if the current row is still valid, or is part of a non-committed transaction or superseded by a new row.

Support Facilities

include (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/include)

include files There are include directories for each subsystem.

lib (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/lib)

support library This houses several generic routines.

port (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/port)

compatibility routines

regex (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/regex)

regular expression library This is used for regular expression handling in the backend, i.e. '~'.

snowball (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/snowball)

Snowball support This is used to support full text Snowball stemming library.

replication (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/replication)

streaming replication Supports streaming replication via log shipping.

tsearch (http://git.postgresql.org/gitweb/?p=postgresql.git;a=tree;f=src/backend/tsearch)

text search library This is used to support full text searching.
