21st May 2020: PostgreSQL 13 Beta 1 Released!

27.2. The Statistics Collector

27.2.1. Statistics Collection Configuration
27.2.2. Viewing Statistics
27.2.3. Statistics Functions

PostgreSQL's *statistics collector* is a subsystem that supports collection and reporting of information about server activity. Presently, the collector can count accesses to tables and indexes in both disk-block and individual-row terms. It also tracks the total number of rows in each table, and information about vacuum and analyze actions for each table. It can also count calls to user-defined functions and the total time spent in each one.

PostgreSQL also supports reporting dynamic information about exactly what is going on in the system right now, such as the exact command currently being executed by other server processes, and which other connections exist in the system. This facility is independent of the collector process.

27.2.1. Statistics Collection Configuration

Since collection of statistics adds some overhead to query execution, the system can be configured to collect or not collect information. This is controlled by configuration parameters that are normally set in `postgresql.conf`. (See Chapter 19 for details about setting configuration parameters.)

The parameter track_activities enables monitoring of the current command being executed by any server process.

The parameter track_counts controls whether statistics are collected about table and index accesses.

The parameter track_functions enables tracking of usage of user-defined functions.

The parameter track_io_timing enables monitoring of block read and write times.

Normally these parameters are set in `postgresql.conf` so that they apply to all server processes, but it is possible to turn them on or off in individual sessions using the SET command. (To prevent ordinary users from hiding their activity from the administrator, only superusers are allowed to change these parameters with `SET`.)

The statistics collector transmits the collected information to other PostgreSQL processes through temporary files. These files are stored in the directory named by the stats_temp_directory parameter, `pg_stat_tmp` by default. For better performance, `stats_temp_directory` can be pointed at a RAM-based file system, decreasing physical I/O requirements. When the server shuts down cleanly, a permanent copy of the statistics data is stored in the `pg_stat` subdirectory, so that statistics can be retained across server restarts. When recovery is performed at server start (e.g. after immediate shutdown, server crash, and point-in-time recovery), all statistics counters are reset.

27.2.2. Viewing Statistics

Several predefined views, listed in Table 27.1, are available to show the current state of the system. There are also several other views, listed in Table 27.2, available to show the results of statistics collection. Alternatively, one can build custom views using the underlying statistics functions, as discussed in Section 27.2.3.

When using the statistics to monitor collected data, it is important to realize that the information does not update instantaneously. Each individual server process transmits new statistical counts to the collector just before going idle; so a query or transaction still in progress does not affect the displayed totals. Also, the collector itself emits a new report at most once per `PGSTAT_STAT_INTERVAL` milliseconds (500 ms unless altered while building the server). So the displayed information lags behind actual activity. However, current-query information collected by `track_activities` is always up-to-date.

Another important point is that when a server process is asked to display any of these statistics, it first fetches the most recent report emitted by the collector process and then continues to use this snapshot for all statistical views and functions until the end of its current transaction. So the statistics will show static information as long as you continue the current transaction. Similarly, information about the current queries of all sessions is collected when any such information is first requested within a transaction, and the same information will be displayed throughout the transaction. This is a feature, not a bug, because it allows you to perform several queries on the statistics and correlate the results without worrying that the numbers are changing underneath you. But if you want to see new results with each query, be sure to do the queries outside any transaction block. Alternatively, you can invoke `pg_stat_clear_snapshot`(), which will discard the current transaction's statistics snapshot (if any). The next use of statistical information will cause a new snapshot to be fetched.

A transaction can also see its own statistics (as yet untransmitted to the collector) in the views `pg_stat_xact_all_tables`, `pg_stat_xact_sys_tables`, `pg_stat_xact_user_tables`, and `pg_stat_xact_user_functions`. These numbers do not act as stated above; instead they update continuously throughout the transaction.

Some of the information in the dynamic statistics views shown in Table 27.1 is security restricted. Ordinary users can only see all the information about their own sessions (sessions belonging to a role that they are a member of). In rows about other sessions, many columns will be null. Note, however, that the existence of a session and its general properties such as its sessions user and database are visible to all users. Superusers and members of the built-in role `pg_read_all_stats` (see also Section 21.5) can see all the information about all sessions.

Table 27.1. Dynamic Statistics Views

View Name	Description
`pg_stat_activity`	One row per server process, showing information related to the current activity of that process, such as state and current query. See pg_stat_activity for details.
`pg_stat_replication`	One row per WAL sender process, showing statistics about replication to that sender's connected standby server. See pg_stat_replication for details.
`pg_stat_wal_receiver`	Only one row, showing statistics about the WAL receiver from that receiver's connected server. See pg_stat_wal_receiver for details.
`pg_stat_subscription`	At least one row per subscription, showing information about the subscription workers. See pg_stat_subscription for details.
`pg_stat_ssl`	One row per connection (regular and replication), showing information about SSL used on this connection. See pg_stat_ssl for details.
`pg_stat_gssapi`	One row per connection (regular and replication), showing information about GSSAPI authentication and encryption used on this connection. See pg_stat_gssapi for details.
`pg_stat_progress_create_index`	One row for each backend running `CREATE INDEX` or `REINDEX`, showing current progress. See Section 27.4.1.
`pg_stat_progress_vacuum`	One row for each backend (including autovacuum worker processes) running `VACUUM`, showing current progress. See Section 27.4.2.
`pg_stat_progress_cluster`	One row for each backend running `CLUSTER` or `VACUUM FULL`, showing current progress. See Section 27.4.3.

Table 27.2. Collected Statistics Views

View Name	Description
`pg_stat_archiver`	One row only, showing statistics about the WAL archiver process's activity. See pg_stat_archiver for details.
`pg_stat_bgwriter`	One row only, showing statistics about the background writer process's activity. See pg_stat_bgwriter for details.
`pg_stat_database`	One row per database, showing database-wide statistics. See pg_stat_database for details.
`pg_stat_database_conflicts`	One row per database, showing database-wide statistics about query cancels due to conflict with recovery on standby servers. See pg_stat_database_conflicts for details.
`pg_stat_all_tables`	One row for each table in the current database, showing statistics about accesses to that specific table. See pg_stat_all_tables for details.
`pg_stat_sys_tables`	Same as `pg_stat_all_tables`, except that only system tables are shown.
`pg_stat_user_tables`	Same as `pg_stat_all_tables`, except that only user tables are shown.
`pg_stat_xact_all_tables`	Similar to `pg_stat_all_tables`, but counts actions taken so far within the current transaction (which are *not* yet included in `pg_stat_all_tables` and related views). The columns for numbers of live and dead rows and vacuum and analyze actions are not present in this view.
`pg_stat_xact_sys_tables`	Same as `pg_stat_xact_all_tables`, except that only system tables are shown.
`pg_stat_xact_user_tables`	Same as `pg_stat_xact_all_tables`, except that only user tables are shown.
`pg_stat_all_indexes`	One row for each index in the current database, showing statistics about accesses to that specific index. See pg_stat_all_indexes for details.
`pg_stat_sys_indexes`	Same as `pg_stat_all_indexes`, except that only indexes on system tables are shown.
`pg_stat_user_indexes`	Same as `pg_stat_all_indexes`, except that only indexes on user tables are shown.
`pg_statio_all_tables`	One row for each table in the current database, showing statistics about I/O on that specific table. See pg_statio_all_tables for details.
`pg_statio_sys_tables`	Same as `pg_statio_all_tables`, except that only system tables are shown.
`pg_statio_user_tables`	Same as `pg_statio_all_tables`, except that only user tables are shown.
`pg_statio_all_indexes`	One row for each index in the current database, showing statistics about I/O on that specific index. See pg_statio_all_indexes for details.
`pg_statio_sys_indexes`	Same as `pg_statio_all_indexes`, except that only indexes on system tables are shown.
`pg_statio_user_indexes`	Same as `pg_statio_all_indexes`, except that only indexes on user tables are shown.
`pg_statio_all_sequences`	One row for each sequence in the current database, showing statistics about I/O on that specific sequence. See pg_statio_all_sequences for details.
`pg_statio_sys_sequences`	Same as `pg_statio_all_sequences`, except that only system sequences are shown. (Presently, no system sequences are defined, so this view is always empty.)
`pg_statio_user_sequences`	Same as `pg_statio_all_sequences`, except that only user sequences are shown.
`pg_stat_user_functions`	One row for each tracked function, showing statistics about executions of that function. See pg_stat_user_functions for details.
`pg_stat_xact_user_functions`	Similar to `pg_stat_user_functions`, but counts only calls during the current transaction (which are *not* yet included in `pg_stat_user_functions`).

The per-index statistics are particularly useful to determine which indexes are being used and how effective they are.

The pg_statio_ views are primarily useful to determine the effectiveness of the buffer cache. When the number of actual disk reads is much smaller than the number of buffer hits, then the cache is satisfying most read requests without invoking a kernel call. However, these statistics do not give the entire story: due to the way in which PostgreSQL handles disk I/O, data that is not in the PostgreSQL buffer cache might still reside in the kernel's I/O cache, and might therefore still be fetched without requiring a physical read. Users interested in obtaining more detailed information on PostgreSQL I/O behavior are advised to use the PostgreSQL statistics collector in combination with operating system utilities that allow insight into the kernel's handling of I/O.

Table 27.3. `pg_stat_activity` View

Column	Type	Description
`datid`	`oid`	OID of the database this backend is connected to
`datname`	`name`	Name of the database this backend is connected to
`pid`	`integer`	Process ID of this backend
`usesysid`	`oid`	OID of the user logged into this backend
`usename`	`name`	Name of the user logged into this backend
`application_name`	`text`	Name of the application that is connected to this backend
`client_addr`	`inet`	IP address of the client connected to this backend. If this field is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum.
`client_hostname`	`text`	Host name of the connected client, as reported by a reverse DNS lookup of `client_addr`. This field will only be non-null for IP connections, and only when log_hostname is enabled.
`client_port`	`integer`	TCP port number that the client is using for communication with this backend, or `-1` if a Unix socket is used
`backend_start`	`timestamp with time zone`	Time when this process was started. For client backends, this is the time the client connected to the server.
`xact_start`	`timestamp with time zone`	Time when this process' current transaction was started, or null if no transaction is active. If the current query is the first of its transaction, this column is equal to the `query_start` column.
`query_start`	`timestamp with time zone`	Time when the currently active query was started, or if `state` is not `active`, when the last query was started
`state_change`	`timestamp with time zone`	Time when the `state` was last changed
`wait_event_type`	`text`	The type of event for which the backend is waiting, if any; otherwise NULL. Possible values are: • `LWLock`: The backend is waiting for a lightweight lock. Each such lock protects a particular data structure in shared memory. `wait_event` will contain a name identifying the purpose of the lightweight lock. (Some locks have specific names; others are part of a group of locks each with a similar purpose.) • `Lock`: The backend is waiting for a heavyweight lock. Heavyweight locks, also known as lock manager locks or simply locks, primarily protect SQL-visible objects such as tables. However, they are also used to ensure mutual exclusion for certain internal operations such as relation extension. `wait_event` will identify the type of lock awaited. • `BufferPin`: The server process is waiting to access to a data buffer during a period when no other process can be examining that buffer. Buffer pin waits can be protracted if another process holds an open cursor which last read data from the buffer in question. • `Activity`: The server process is idle. This is used by system processes waiting for activity in their main processing loop. `wait_event` will identify the specific wait point. • `Extension`: The server process is waiting for activity in an extension module. This category is useful for modules to track custom waiting points. • `Client`: The server process is waiting for some activity on a socket from user applications, and that the server expects something to happen that is independent from its internal processes. `wait_event` will identify the specific wait point. • `IPC`: The server process is waiting for some activity from another process in the server. `wait_event` will identify the specific wait point. • `Timeout`: The server process is waiting for a timeout to expire. `wait_event` will identify the specific wait point. • `IO`: The server process is waiting for a IO to complete. `wait_event` will identify the specific wait point.
`wait_event`	`text`	Wait event name if backend is currently waiting, otherwise NULL. See Table 27.4 for details.

Column	Type	Description
state	text	Current overall state of this backend. Possible values are: • active: The backend is executing a query. • idle: The backend is waiting for a new client command. • idle in transaction: The backend is in a transaction, but is not currently executing a query. • idle in transaction (aborted): This state is similar to idle in transaction, except one of the statements in the transaction caused an error. • fastpath function call: The backend is executing a fast-path function. • disabled: This state is reported if track_activities is disabled in this backend.
backend_xid	xid	Top-level transaction identifier of this backend, if any.
backend_xmin	xid	The current backend's xmin horizon.
query	text	Text of this backend's most recent query. If state is active this field shows the currently executing query. In all other states, it shows the last query that was executed. By default the query text is truncated at 1024 characters; this value can be changed via the parameter track_activity_query_size.
backend_type	text	Type of current backend. Possible types are autovacuum launcher, autovacuum worker, logical replication launcher, logical replication worker, parallel worker, background writer, client backend, checkpointer, startup, walreceiver, walsender and walwriter. In addition, background workers registered by extensions may have additional types.

The pg_stat_activity view will have one row per server process, showing information related to the current activity of that process.

Note

The wait_event and state columns are independent. If a backend is in the active state, it may or may not be waiting on some event. If the state is active and wait_event is non-null, it means that a query is being executed, but is being blocked somewhere in the system.

Table 27.4. wait_event Description

Wait Event Type	Wait Event Name	Description
LWLock	ShmemIndexLock	Waiting to find or allocate space in shared memory.
	OidGenLock	Waiting to allocate or assign an OID.
	XidGenLock	Waiting to allocate or assign a transaction id.
	ProcArrayLock	Waiting to get a snapshot or clearing a transaction id at transaction end.
	SInvalReadLock	Waiting to retrieve or remove messages from shared invalidation queue.
	SInvalWriteLock	Waiting to add a message in shared invalidation queue.
	WALBufMappingLock	Waiting to replace a page in WAL buffers.
	WALWriteLock	Waiting for WAL buffers to be written to disk.
	ControlFileLock	Waiting to read or update the control file or creation of a new WAL file.
	CheckpointLock	Waiting to perform checkpoint.
	CLogControlLock	Waiting to read or update transaction status.
	SubtransControlLock	Waiting to read or update sub-transaction information.
	MultiXactGenLock	Waiting to read or update shared multixact state.
	MultiXactOffsetControlLock	Waiting to read or update multixact offset mappings.
	MultiXactMemberControlLock	Waiting to read or update multixact member mappings.
	RelCacheInitLock	Waiting to read or write relation cache initialization file.
	CheckpointerCommLock	Waiting to manage fsync requests.
	TwoPhaseStateLock	Waiting to read or update the state of prepared transactions.
	TablespaceCreateLock	Waiting to create or drop the tablespace.
	BtreeVacuumLock	Waiting to read or update vacuum-related information for a B-tree index.
	AddinShmemInitLock	Waiting to manage space allocation in shared memory.
	AutovacuumLock	Autovacuum worker or launcher waiting to update or read the current state of autovacuum workers.
	AutovacuumScheduleLock	Waiting to ensure that the table it has selected for a vacuum still needs vacuuming.
	SyncScanLock	Waiting to get the start location of a scan on a table for synchronized scans.
	RelationMappingLock	Waiting to update the relation map file used to store catalog to filenode mapping.
	AsyncCtlLock	Waiting to read or update shared notification state.
	AsyncQueueLock	Waiting to read or update notification messages.
	SerializableXactHashLock	Waiting to retrieve or store information about serializable transactions.
	SerializableFinishedListLock	Waiting to access the list of finished serializable transactions.

Wait Event Type	Wait Event Name	Description
	`SerializablePredicateLockListLock`	Waiting to perform an operation on a list of locks held by serializable transactions.
	`OldSerXidLock`	Waiting to read or record conflicting serializable transactions.
	`SyncRepLock`	Waiting to read or update information about synchronous replicas.
	`BackgroundWorkerLock`	Waiting to read or update background worker state.
	`DynamicSharedMemoryControlLock`	Waiting to read or update dynamic shared memory state.
	`AutoFileLock`	Waiting to update the `postgresql.auto.conf` file.
	`ReplicationSlotAllocationLock`	Waiting to allocate or free a replication slot.
	`ReplicationSlotControlLock`	Waiting to read or update replication slot state.
	`CommitTsControlLock`	Waiting to read or update transaction commit timestamps.
	`CommitTsLock`	Waiting to read or update the last value set for the transaction timestamp.
	`ReplicationOriginLock`	Waiting to setup, drop or use replication origin.
	`MultiXactTruncationLock`	Waiting to read or truncate multixact information.
	`OldSnapshotTimeMapLock`	Waiting to read or update old snapshot control information.
	`LogicalRepWorkerLock`	Waiting for action on logical replication worker to finish.
	`CLogTruncationLock`	Waiting to execute `txid_status` or update the oldest transaction id available to it.
	`clog`	Waiting for I/O on a clog (transaction status) buffer.
	`commit_timestamp`	Waiting for I/O on commit timestamp buffer.
	`subtrans`	Waiting for I/O a subtransaction buffer.
	`multixact_offset`	Waiting for I/O on a multixact offset buffer.
	`multixact_member`	Waiting for I/O on a multixact_member buffer.
	`async`	Waiting for I/O on an async (notify) buffer.
	`oldserxid`	Waiting for I/O on an oldserxid buffer.
	`wal_insert`	Waiting to insert WAL into a memory buffer.
	`buffer_content`	Waiting to read or write a data page in memory.
	`buffer_io`	Waiting for I/O on a data page.
	`replication_origin`	Waiting to read or update the replication progress.
	`replication_slot_io`	Waiting for I/O on a replication slot.
	`proc`	Waiting to read or update the fast-path lock information.
	`buffer_mapping`	Waiting to associate a data block with a buffer in the buffer pool.
	`lock_manager`	Waiting to add or examine locks for backends, or waiting to join or exit a locking group (used by parallel query).
	`predicate_lock_manager`	Waiting to add or examine predicate lock information.
	`serializable_xact`	Waiting to perform an operation on a serializable transaction in a parallel query.
	`parallel_query_dsa`	Waiting for parallel query dynamic shared memory allocation lock.
	`tbm`	Waiting for TBM shared iterator lock.
	`parallel_append`	Waiting to choose the next subplan during Parallel Append plan execution.
	`parallel_hash_join`	Waiting to allocate or exchange a chunk of memory or update counters during Parallel Hash plan execution.
`Lock`	`relation`	Waiting to acquire a lock on a relation.
	`extend`	Waiting to extend a relation.
	`page`	Waiting to acquire a lock on page of a relation.
	`tuple`	Waiting to acquire a lock on a tuple.
	`transactionid`	Waiting for a transaction to finish.
	`virtualxid`	Waiting to acquire a virtual xid lock.
	`speculative token`	Waiting to acquire a speculative insertion lock.
	`object`	Waiting to acquire a lock on a non-relation database object.
	`userlock`	Waiting to acquire a user lock.
	`advisory`	Waiting to acquire an advisory user lock.
`BufferPin`	`BufferPin`	Waiting to acquire a pin on a buffer.
`Activity`	`ArchiverMain`	Waiting in main loop of the archiver process.
	`AutoVacuumMain`	Waiting in main loop of autovacuum launcher process.
	`BgWriterHibernate`	Waiting in background writer process, hibernating.
	`BgWriterMain`	Waiting in main loop of background writer process background worker.
	`CheckpointerMain`	Waiting in main loop of checkpointer process.
	`LogicalApplyMain`	Waiting in main loop of logical apply process.
	`LogicalLauncherMain`	Waiting in main loop of logical launcher process.
	`PgStatMain`	Waiting in main loop of the statistics collector process.
	`RecoveryWalAll`	Waiting for WAL from a stream at recovery.
	`RecoveryWalStream`	Waiting when WAL data is not available from any kind of sources (local, archive or stream) before trying again to retrieve WAL data, at recovery.

Wait Event Type	Wait Event Name	Description
	`SysLoggerMain`	Waiting in main loop of syslogger process.
	`WalReceiverMain`	Waiting in main loop of WAL receiver process.
	`WalSenderMain`	Waiting in main loop of WAL sender process.
	`WalWriterMain`	Waiting in main loop of WAL writer process.
`Client`	`ClientRead`	Waiting to read data from the client.
	`ClientWrite`	Waiting to write data to the client.
	`GSSOpenServer`	Waiting to read data from the client while establishing the GSSAPI session.
	`LibPQWalReceiverConnect`	Waiting in WAL receiver to establish connection to remote server.
	`LibPQWalReceiverReceive`	Waiting in WAL receiver to receive data from remote server.
	`SSLOpenServer`	Waiting for SSL while attempting connection.
	`WalReceiverWaitStart`	Waiting for startup process to send initial data for streaming replication.
	`WalSenderWaitForWAL`	Waiting for WAL to be flushed in WAL sender process.
	`WalSenderWriteData`	Waiting for any activity when processing replies from WAL receiver in WAL sender process.
`Extension`	`Extension`	Waiting in an extension.
`IPC`	`BgWorkerShutdown`	Waiting for background worker to shut down.
	`BgWorkerStartup`	Waiting for background worker to start up.
	`BtreePage`	Waiting for the page number needed to continue a parallel B-tree scan to become available.
	`CheckpointDone`	Waiting for a checkpoint to complete.
	`CheckpointStart`	Waiting for a checkpoint to start.
	`ClogGroupUpdate`	Waiting for group leader to update transaction status at transaction end.
	`ExecuteGather`	Waiting for activity from child process when executing `Gather` node.
	`Hash/Batch/Allocating`	Waiting for an elected Parallel Hash participant to allocate a hash table.
	`Hash/Batch/Electing`	Electing a Parallel Hash participant to allocate a hash table.
	`Hash/Batch/Loading`	Waiting for other Parallel Hash participants to finish loading a hash table.
	`Hash/Build/Allocating`	Waiting for an elected Parallel Hash participant to allocate the initial hash table.
	`Hash/Build/Electing`	Electing a Parallel Hash participant to allocate the initial hash table.
	`Hash/Build/HashingInner`	Waiting for other Parallel Hash participants to finish hashing the inner relation.
	`Hash/Build/HashingOuter`	Waiting for other Parallel Hash participants to finish partitioning the outer relation.
	`Hash/GrowBatches/Allocating`	Waiting for an elected Parallel Hash participant to allocate more batches.
	`Hash/GrowBatches/Deciding`	Electing a Parallel Hash participant to decide on future batch growth.
	`Hash/GrowBatches/Electing`	Electing a Parallel Hash participant to allocate more batches.
	`Hash/GrowBatches/Finishing`	Waiting for an elected Parallel Hash participant to decide on future batch growth.
	`Hash/GrowBatches/Repartitioning`	Waiting for other Parallel Hash participants to finishing repartitioning.
	`Hash/GrowBuckets/Allocating`	Waiting for an elected Parallel Hash participant to finish allocating more buckets.
	`Hash/GrowBuckets/Electing`	Electing a Parallel Hash participant to allocate more buckets.
	`Hash/GrowBuckets/Reinserting`	Waiting for other Parallel Hash participants to finish inserting tuples into new buckets.
	`LogicalSyncData`	Waiting for logical replication remote server to send data for initial table synchronization.
	`LogicalSyncStateChange`	Waiting for logical replication remote server to change state.
	`MessageQueueInternal`	Waiting for other process to be attached in shared message queue.
	`MessageQueuePutMessage`	Waiting to write a protocol message to a shared message queue.
	`MessageQueueReceive`	Waiting to receive bytes from a shared message queue.
	`MessageQueueSend`	Waiting to send bytes to a shared message queue.
	`ParallelBitmapScan`	Waiting for parallel bitmap scan to become initialized.
	`ParallelCreateIndexScan`	Waiting for parallel `CREATE INDEX` workers to finish heap scan.
	`ParallelFinish`	Waiting for parallel workers to finish computing.
	`ProcArrayGroupUpdate`	Waiting for group leader to clear transaction id at transaction end.
	`Promote`	Waiting for standby promotion.
	`ReplicationOriginDrop`	Waiting for a replication origin to become inactive to be dropped.
	`ReplicationSlotDrop`	Waiting for a replication slot to become inactive to be dropped.
	`SafeSnapshot`	Waiting for a snapshot for a `READ ONLY DEFERRABLE` transaction.
	`SyncRep`	Waiting for confirmation from remote server during synchronous replication.
`Timeout`	`BaseBackupThrottle`	Waiting during base backup when throttling activity.
	`PgSleep`	Waiting in process that called `pg_sleep`.
	`RecoveryApplyDelay`	Waiting to apply WAL at recovery because it is delayed.
`IO`	`BufFileRead`	Waiting for a read from a buffered file.
	`BufFileWrite`	Waiting for a write to a buffered file.
	`ControlFileRead`	Waiting for a read from the control file.
	`ControlFileSync`	Waiting for the control file to reach stable storage.

Wait Event Type	Wait Event Name	Description
	`ControlFileSyncUpdate`	Waiting for an update to the control file to reach stable storage.
	`ControlFileWrite`	Waiting for a write to the control file.
	`ControlFileWriteUpdate`	Waiting for a write to update the control file.
	`CopyFileRead`	Waiting for a read during a file copy operation.
	`CopyFileWrite`	Waiting for a write during a file copy operation.
	`DataFileExtend`	Waiting for a relation data file to be extended.
	`DataFileFlush`	Waiting for a relation data file to reach stable storage.
	`DataFileImmediateSync`	Waiting for an immediate synchronization of a relation data file to stable storage.
	`DataFilePrefetch`	Waiting for an asynchronous prefetch from a relation data file.
	`DataFileRead`	Waiting for a read from a relation data file.
	`DataFileSync`	Waiting for changes to a relation data file to reach stable storage.
	`DataFileTruncate`	Waiting for a relation data file to be truncated.
	`DataFileWrite`	Waiting for a write to a relation data file.
	`DSMFillZeroWrite`	Waiting to write zero bytes to a dynamic shared memory backing file.
	`LockFileAddToDataDirRead`	Waiting for a read while adding a line to the data directory lock file.
	`LockFileAddToDataDirSync`	Waiting for data to reach stable storage while adding a line to the data directory lock file.
	`LockFileAddToDataDirWrite`	Waiting for a write while adding a line to the data directory lock file.
	`LockFileCreateRead`	Waiting to read while creating the data directory lock file.
	`LockFileCreateSync`	Waiting for data to reach stable storage while creating the data directory lock file.
	`LockFileCreateWrite`	Waiting for a write while creating the data directory lock file.
	`LockFileReCheckDataDirRead`	Waiting for a read during recheck of the data directory lock file.
	`LogicalRewriteCheckpointSync`	Waiting for logical rewrite mappings to reach stable storage during a checkpoint.
	`LogicalRewriteMappingSync`	Waiting for mapping data to reach stable storage during a logical rewrite.
	`LogicalRewriteMappingWrite`	Waiting for a write of mapping data during a logical rewrite.
	`LogicalRewriteSync`	Waiting for logical rewrite mappings to reach stable storage.
	`LogicalRewriteTruncate`	Waiting for truncate of mapping data during a logical rewrite.
	`LogicalRewriteWrite`	Waiting for a write of logical rewrite mappings.
	`RelationMapRead`	Waiting for a read of the relation map file.
	`RelationMapSync`	Waiting for the relation map file to reach stable storage.
	`RelationMapWrite`	Waiting for a write to the relation map file.
	`ReorderBufferRead`	Waiting for a read during reorder buffer management.
	`ReorderBufferWrite`	Waiting for a write during reorder buffer management.
	`ReorderLogicalMappingRead`	Waiting for a read of a logical mapping during reorder buffer management.
	`ReplicationSlotRead`	Waiting for a read from a replication slot control file.
	`ReplicationSlotRestoreSync`	Waiting for a replication slot control file to reach stable storage while restoring it to memory.
	`ReplicationSlotSync`	Waiting for a replication slot control file to reach stable storage.
	`ReplicationSlotWrite`	Waiting for a write to a replication slot control file.
	`SLRUFlushSync`	Waiting for SLRU data to reach stable storage during a checkpoint or database shutdown.
	`SLRURead`	Waiting for a read of an SLRU page.
	`SLRUSync`	Waiting for SLRU data to reach stable storage following a page write.
	`SLRUWrite`	Waiting for a write of an SLRU page.
	`SnapbuildRead`	Waiting for a read of a serialized historical catalog snapshot.
	`SnapbuildSync`	Waiting for a serialized historical catalog snapshot to reach stable storage.
	`SnapbuildWrite`	Waiting for a write of a serialized historical catalog snapshot.
	`TimelineHistoryFileSync`	Waiting for a timeline history file received via streaming replication to reach stable storage.
	`TimelineHistoryFileWrite`	Waiting for a write of a timeline history file received via streaming replication.
	`TimelineHistoryRead`	Waiting for a read of a timeline history file.
	`TimelineHistorySync`	Waiting for a newly created timeline history file to reach stable storage.
	`TimelineHistoryWrite`	Waiting for a write of a newly created timeline history file.
	`TwophaseFileRead`	Waiting for a read of a two phase state file.
	`TwophaseFileSync`	Waiting for a two phase state file to reach stable storage.
	`TwophaseFileWrite`	Waiting for a write of a two phase state file.
	`WALBootstrapSync`	Waiting for WAL to reach stable storage during bootstrapping.
	`WALBootstrapWrite`	Waiting for a write of a WAL page during bootstrapping.
	`WALCopyRead`	Waiting for a read when creating a new WAL segment by copying an existing one.
	`WALCopySync`	Waiting a new WAL segment created by copying an existing one to reach stable storage.
	`WALCopyWrite`	Waiting for a write when creating a new WAL segment by copying an existing one.
	`WALInitSync`	Waiting for a newly initialized WAL file to reach stable storage.

Wait Event Type	Wait Event Name	Description
	`WALInitWrite`	Waiting for a write while initializing a new WAL file.
	`WALRead`	Waiting for a read from a WAL file.
	`WALSenderTimelineHistoryRead`	Waiting for a read from a timeline history file during walsender timeline command.
	`WALSync`	Waiting for a WAL file to reach stable storage.
	`WALSyncMethodAssign`	Waiting for data to reach stable storage while assigning WAL sync method.
	`WALWrite`	Waiting for a write to a WAL file.

Note

For tranches registered by extensions, the name is specified by extension and this will be displayed as `wait_event`. It is quite possible that user has registered the tranche in one of the backends (by having allocation in dynamic shared memory) in which case other backends won't have that information, so we display `extension` for such cases.

Here is an example of how wait events can be viewed

```
SELECT pid, wait_event_type, wait_event FROM pg_stat_activity WHERE wait_event is NOT NULL;
 pid  | wait_event_type |  wait_event
------+-----------------+---------------
 2540 | Lock            | relation
 6644 | LWLock          | ProcArrayLock
(2 rows)
```

Table 27.5. `pg_stat_replication` View

Column	Type	Description
`pid`	`integer`	Process ID of a WAL sender process
`usesysid`	`oid`	OID of the user logged into this WAL sender process
`usename`	`name`	Name of the user logged into this WAL sender process
`application_name`	`text`	Name of the application that is connected to this WAL sender
`client_addr`	`inet`	IP address of the client connected to this WAL sender. If this field is null, it indicates that the client is connected via a Unix socket on the server machine.
`client_hostname`	`text`	Host name of the connected client, as reported by a reverse DNS lookup of `client_addr`. This field will only be non-null for IP connections, and only when log_hostname is enabled.
`client_port`	`integer`	TCP port number that the client is using for communication with this WAL sender, or `-1` if a Unix socket is used
`backend_start`	`timestamp with time zone`	Time when this process was started, i.e., when the client connected to this WAL sender
`backend_xmin`	`xid`	This standby's `xmin` horizon reported by hot_standby_feedback.
`state`	`text`	Current WAL sender state. Possible values are: • `startup`: This WAL sender is starting up. • `catchup`: This WAL sender's connected standby is catching up with the primary. • `streaming`: This WAL sender is streaming changes after its connected standby server has caught up with the primary. • `backup`: This WAL sender is sending a backup. • `stopping`: This WAL sender is stopping.
`sent_lsn`	`pg_lsn`	Last write-ahead log location sent on this connection
`write_lsn`	`pg_lsn`	Last write-ahead log location written to disk by this standby server
`flush_lsn`	`pg_lsn`	Last write-ahead log location flushed to disk by this standby server
`replay_lsn`	`pg_lsn`	Last write-ahead log location replayed into the database on this standby server
`write_lag`	`interval`	Time elapsed between flushing recent WAL locally and receiving notification that this standby server has written it (but not yet flushed it or applied it). This can be used to gauge the delay that `synchronous_commit` level `remote_write` incurred while committing if this server was configured as a synchronous standby.
`flush_lag`	`interval`	Time elapsed between flushing recent WAL locally and receiving notification that this standby server has written and flushed it (but not yet applied it). This can be used to gauge the delay that `synchronous_commit` level `on` incurred while committing if this server was configured as a synchronous standby.
`replay_lag`	`interval`	Time elapsed between flushing recent WAL locally and receiving notification that this standby server has written, flushed and applied it. This can be used to gauge the delay that `synchronous_commit` level `remote_apply` incurred while committing if this server was configured as a synchronous standby.

Column	Type	Description
`sync_priority`	`integer`	Priority of this standby server for being chosen as the synchronous standby in a priority-based synchronous replication. This has no effect in a quorum-based synchronous replication.
`sync_state`	`text`	Synchronous state of this standby server. Possible values are: • `async`: This standby server is asynchronous. • `potential`: This standby server is now asynchronous, but can potentially become synchronous if one of current synchronous ones fails. • `sync`: This standby server is synchronous. • `quorum`: This standby server is considered as a candidate for quorum standbys.
`reply_time`	`timestamp with time zone`	Send time of last reply message received from standby server

The `pg_stat_replication` view will contain one row per WAL sender process, showing statistics about replication to that sender's connected standby server. Only directly connected standbys are listed; no information is available about downstream standby servers.

The lag times reported in the `pg_stat_replication` view are measurements of the time taken for recent WAL to be written, flushed and replayed and for the sender to know about it. These times represent the commit delay that was (or would have been) introduced by each synchronous commit level, if the remote server was configured as a synchronous standby. For an asynchronous standby, the `replay_lag` column approximates the delay before recent transactions became visible to queries. If the standby server has entirely caught up with the sending server and there is no more WAL activity, the most recently measured lag times will continue to be displayed for a short time and then show NULL.

Lag times work automatically for physical replication. Logical decoding plugins may optionally emit tracking messages; if they do not, the tracking mechanism will simply display NULL lag.

Note

The reported lag times are not predictions of how long it will take for the standby to catch up with the sending server assuming the current rate of replay. Such a system would show similar times while new WAL is being generated, but would differ when the sender becomes idle. In particular, when the standby has caught up completely, `pg_stat_replication` shows the time taken to write, flush and replay the most recent reported WAL location rather than zero as some users might expect. This is consistent with the goal of measuring synchronous commit and transaction visibility delays for recent write transactions. To reduce confusion for users expecting a different model of lag, the lag columns revert to NULL after a short time on a fully replayed idle system. Monitoring systems should choose whether to represent this as missing data, zero or continue to display the last known value.

Table 27.6. `pg_stat_wal_receiver` View

Column	Type	Description
`pid`	`integer`	Process ID of the WAL receiver process
`status`	`text`	Activity status of the WAL receiver process
`receive_start_lsn`	`pg_lsn`	First write-ahead log location used when WAL receiver is started
`receive_start_tli`	`integer`	First timeline number used when WAL receiver is started
`received_lsn`	`pg_lsn`	Last write-ahead log location already received and flushed to disk, the initial value of this field being the first log location used when WAL receiver is started
`received_tli`	`integer`	Timeline number of last write-ahead log location received and flushed to disk, the initial value of this field being the timeline number of the first log location used when WAL receiver is started
`last_msg_send_time`	`timestamp with time zone`	Send time of last message received from origin WAL sender
`last_msg_receipt_time`	`timestamp with time zone`	Receipt time of last message received from origin WAL sender
`latest_end_lsn`	`pg_lsn`	Last write-ahead log location reported to origin WAL sender
`latest_end_time`	`timestamp with time zone`	Time of last write-ahead log location reported to origin WAL sender
`slot_name`	`text`	Replication slot name used by this WAL receiver
`sender_host`	`text`	Host of the PostgreSQL instance this WAL receiver is connected to. This can be a host name, an IP address, or a directory path if the connection is via Unix socket. (The path case can be distinguished because it will always be an absolute path, beginning with `/`.)

Column	Type	Description
sender_port	integer	Port number of the PostgreSQL instance this WAL receiver is connected to.
conninfo	text	Connection string used by this WAL receiver, with security-sensitive fields obfuscated.

The pg_stat_wal_receiver view will contain only one row, showing statistics about the WAL receiver from that receiver's connected server.

Table 27.7. pg_stat_subscription View

Column	Type	Description
subid	oid	OID of the subscription
subname	text	Name of the subscription
pid	integer	Process ID of the subscription worker process
relid	Oid	OID of the relation that the worker is synchronizing; null for the main apply worker
received_lsn	pg_lsn	Last write-ahead log location received, the initial value of this field being 0
last_msg_send_time	timestamp with time zone	Send time of last message received from origin WAL sender
last_msg_receipt_time	timestamp with time zone	Receipt time of last message received from origin WAL sender
latest_end_lsn	pg_lsn	Last write-ahead log location reported to origin WAL sender
latest_end_time	timestamp with time zone	Time of last write-ahead log location reported to origin WAL sender

The pg_stat_subscription view will contain one row per subscription for main worker (with null PID if the worker is not running), and additional rows for workers handling the initial data copy of the subscribed tables.

Table 27.8. pg_stat_ssl View

Column	Type	Description
pid	integer	Process ID of a backend or WAL sender process
ssl	boolean	True if SSL is used on this connection
version	text	Version of SSL in use, or NULL if SSL is not in use on this connection
cipher	text	Name of SSL cipher in use, or NULL if SSL is not in use on this connection
bits	integer	Number of bits in the encryption algorithm used, or NULL if SSL is not used on this connection
compression	boolean	True if SSL compression is in use, false if not, or NULL if SSL is not in use on this connection
client_dn	text	Distinguished Name (DN) field from the client certificate used, or NULL if no client certificate was supplied or if SSL is not in use on this connection. This field is truncated if the DN field is longer than NAMEDATALEN (64 characters in a standard build).
client_serial	numeric	Serial number of the client certificate, or NULL if no client certificate was supplied or if SSL is not in use on this connection. The combination of certificate serial number and certificate issuer uniquely identifies a certificate (unless the issuer erroneously reuses serial numbers).
issuer_dn	text	DN of the issuer of the client certificate, or NULL if no client certificate was supplied or if SSL is not in use on this connection. This field is truncated like client_dn.

The pg_stat_ssl view will contain one row per backend or WAL sender process, showing statistics about SSL usage on this connection. It can be joined to pg_stat_activity or pg_stat_replication on the pid column to get more details about the connection.

Table 27.9. pg_stat_gssapi View

Column	Type	Description
pid	integer	Process ID of a backend
gss_authenticated	boolean	True if GSSAPI authentication was used for this connection
principal	text	Principal used to authenticate this connection, or NULL if GSSAPI was not used to authenticate this connection. This field is truncated if the principal is longer than NAMEDATALEN (64 characters in a standard build).
encrypted	boolean	True if GSSAPI encryption is in use on this connection

The pg_stat_gssapi view will contain one row per backend, showing information about GSSAPI usage on this connection. It can be joined to pg_stat_activity or pg_stat_replication on the pid column to get more details about the connection.

Table 27.10. pg_stat_archiver View

Column	Type	Description
archived_count	bigint	Number of WAL files that have been successfully archived
last_archived_wal	text	Name of the last WAL file successfully archived
last_archived_time	timestamp with time zone	Time of the last successful archive operation
failed_count	bigint	Number of failed attempts for archiving WAL files
last_failed_wal	text	Name of the WAL file of the last failed archival operation
last_failed_time	timestamp with time zone	Time of the last failed archival operation
stats_reset	timestamp with time zone	Time at which these statistics were last reset

The pg_stat_archiver view will always have a single row, containing data about the archiver process of the cluster.

Table 27.11. `pg_stat_bgwriter` View

Column	Type	Description
`checkpoints_timed`	`bigint`	Number of scheduled checkpoints that have been performed
`checkpoints_req`	`bigint`	Number of requested checkpoints that have been performed
`checkpoint_write_time`	`double precision`	Total amount of time that has been spent in the portion of checkpoint processing where files are written to disk, in milliseconds
`checkpoint_sync_time`	`double precision`	Total amount of time that has been spent in the portion of checkpoint processing where files are synchronized to disk, in milliseconds
`buffers_checkpoint`	`bigint`	Number of buffers written during checkpoints
`buffers_clean`	`bigint`	Number of buffers written by the background writer
`maxwritten_clean`	`bigint`	Number of times the background writer stopped a cleaning scan because it had written too many buffers
`buffers_backend`	`bigint`	Number of buffers written directly by a backend
`buffers_backend_fsync`	`bigint`	Number of times a backend had to execute its own `fsync` call (normally the background writer handles those even when the backend does its own write)
`buffers_alloc`	`bigint`	Number of buffers allocated
`stats_reset`	`timestamp with time zone`	Time at which these statistics were last reset

The `pg_stat_bgwriter` view will always have a single row, containing global data for the cluster.

Table 27.12. `pg_stat_database` View

Column	Type	Description
`datid`	`oid`	OID of this database, or 0 for objects belonging to a shared relation
`datname`	`name`	Name of this database, or `NULL` for the shared objects.
`numbackends`	`integer`	Number of backends currently connected to this database, or `NULL` for the shared objects. This is the only column in this view that returns a value reflecting current state; all other columns return the accumulated values since the last reset.
`xact_commit`	`bigint`	Number of transactions in this database that have been committed
`xact_rollback`	`bigint`	Number of transactions in this database that have been rolled back
`blks_read`	`bigint`	Number of disk blocks read in this database
`blks_hit`	`bigint`	Number of times disk blocks were found already in the buffer cache, so that a read was not necessary (this only includes hits in the PostgreSQL buffer cache, not the operating system's file system cache)
`tup_returned`	`bigint`	Number of rows returned by queries in this database
`tup_fetched`	`bigint`	Number of rows fetched by queries in this database
`tup_inserted`	`bigint`	Number of rows inserted by queries in this database
`tup_updated`	`bigint`	Number of rows updated by queries in this database
`tup_deleted`	`bigint`	Number of rows deleted by queries in this database
`conflicts`	`bigint`	Number of queries canceled due to conflicts with recovery in this database. (Conflicts occur only on standby servers; see pg_stat_database_conflicts for details.)
`temp_files`	`bigint`	Number of temporary files created by queries in this database. All temporary files are counted, regardless of why the temporary file was created (e.g., sorting or hashing), and regardless of the log_temp_files setting.
`temp_bytes`	`bigint`	Total amount of data written to temporary files by queries in this database. All temporary files are counted, regardless of why the temporary file was created, and regardless of the log_temp_files setting.
`deadlocks`	`bigint`	Number of deadlocks detected in this database
`checksum_failures`	`bigint`	Number of data page checksum failures detected in this database (or on a shared object), or NULL if data checksums are not enabled.
`checksum_last_failure`	`timestamp with time zone`	Time at which the last data page checksum failure was detected in this database (or on a shared object), or NULL if data checksums are not enabled.
`blk_read_time`	`double precision`	Time spent reading data file blocks by backends in this database, in milliseconds
`blk_write_time`	`double precision`	Time spent writing data file blocks by backends in this database, in milliseconds
`stats_reset`	`timestamp with time zone`	Time at which these statistics were last reset

The `pg_stat_database` view will contain one row for each database in the cluster, plus one for the shared objects, showing database-wide statistics.

Table 27.13. `pg_stat_database_conflicts` View

Column	Type	Description
`datid`	`oid`	OID of a database
`datname`	`name`	Name of this database
`confl_tablespace`	`bigint`	Number of queries in this database that have been canceled due to dropped tablespaces

Column	Type	Description
`confl_lock`	`bigint`	Number of queries in this database that have been canceled due to lock timeouts
`confl_snapshot`	`bigint`	Number of queries in this database that have been canceled due to old snapshots
`confl_bufferpin`	`bigint`	Number of queries in this database that have been canceled due to pinned buffers
`confl_deadlock`	`bigint`	Number of queries in this database that have been canceled due to deadlocks

The `pg_stat_database_conflicts` view will contain one row per database, showing database-wide statistics about query cancels occurring due to conflicts with recovery on standby servers. This view will only contain information on standby servers, since conflicts do not occur on master servers.

Table 27.14. `pg_stat_all_tables` View

Column	Type	Description
`relid`	`oid`	OID of a table
`schemaname`	`name`	Name of the schema that this table is in
`relname`	`name`	Name of this table
`seq_scan`	`bigint`	Number of sequential scans initiated on this table
`seq_tup_read`	`bigint`	Number of live rows fetched by sequential scans
`idx_scan`	`bigint`	Number of index scans initiated on this table
`idx_tup_fetch`	`bigint`	Number of live rows fetched by index scans
`n_tup_ins`	`bigint`	Number of rows inserted
`n_tup_upd`	`bigint`	Number of rows updated (includes HOT updated rows)
`n_tup_del`	`bigint`	Number of rows deleted
`n_tup_hot_upd`	`bigint`	Number of rows HOT updated (i.e., with no separate index update required)
`n_live_tup`	`bigint`	Estimated number of live rows
`n_dead_tup`	`bigint`	Estimated number of dead rows
`n_mod_since_analyze`	`bigint`	Estimated number of rows modified since this table was last analyzed
`last_vacuum`	`timestamp with time zone`	Last time at which this table was manually vacuumed (not counting `VACUUM FULL`)
`last_autovacuum`	`timestamp with time zone`	Last time at which this table was vacuumed by the autovacuum daemon
`last_analyze`	`timestamp with time zone`	Last time at which this table was manually analyzed
`last_autoanalyze`	`timestamp with time zone`	Last time at which this table was analyzed by the autovacuum daemon
`vacuum_count`	`bigint`	Number of times this table has been manually vacuumed (not counting `VACUUM FULL`)
`autovacuum_count`	`bigint`	Number of times this table has been vacuumed by the autovacuum daemon
`analyze_count`	`bigint`	Number of times this table has been manually analyzed
`autoanalyze_count`	`bigint`	Number of times this table has been analyzed by the autovacuum daemon

The `pg_stat_all_tables` view will contain one row for each table in the current database (including TOAST tables), showing statistics about accesses to that specific table. The `pg_stat_user_tables` and `pg_stat_sys_tables` views contain the same information, but filtered to only show user and system tables respectively.

Table 27.15. `pg_stat_all_indexes` View

Column	Type	Description
`relid`	`oid`	OID of the table for this index
`indexrelid`	`oid`	OID of this index
`schemaname`	`name`	Name of the schema this index is in
`relname`	`name`	Name of the table for this index
`indexrelname`	`name`	Name of this index
`idx_scan`	`bigint`	Number of index scans initiated on this index
`idx_tup_read`	`bigint`	Number of index entries returned by scans on this index
`idx_tup_fetch`	`bigint`	Number of live table rows fetched by simple index scans using this index

The `pg_stat_all_indexes` view will contain one row for each index in the current database, showing statistics about accesses to that specific index. The `pg_stat_user_indexes` and `pg_stat_sys_indexes` views contain the same information, but filtered to only show user and system indexes respectively.

Indexes can be used by simple index scans, “bitmap” index scans, and the optimizer. In a bitmap scan the output of several indexes can be combined via AND or OR rules, so it is difficult to associate individual heap row fetches with specific indexes when a bitmap scan is used. Therefore, a bitmap scan increments the `pg_stat_all_indexes.idx_tup_read` count(s) for the index(es) it uses, and it increments the `pg_stat_all_tables.idx_tup_fetch` count for the table, but it does not affect `pg_stat_all_indexes.idx_tup_fetch`. The optimizer also accesses indexes to check for supplied constants whose values are outside the recorded range of the optimizer statistics because the optimizer statistics might be stale.

Note

The `idx_tup_read` and `idx_tup_fetch` counts can be different even without any use of bitmap scans, because `idx_tup_read` counts index entries retrieved from the index while `idx_tup_fetch` counts live rows fetched from the table. The latter will be less if any dead or not-yet-committed rows are fetched using the index, or if any heap fetches are avoided by means of an index-only scan.

Table 27.16. `pg_statio_all_tables` View

Column	Type	Description
`relid`	`oid`	OID of a table
`schemaname`	`name`	Name of the schema that this table is in
`relname`	`name`	Name of this table
`heap_blks_read`	`bigint`	Number of disk blocks read from this table
`heap_blks_hit`	`bigint`	Number of buffer hits in this table
`idx_blks_read`	`bigint`	Number of disk blocks read from all indexes on this table
`idx_blks_hit`	`bigint`	Number of buffer hits in all indexes on this table
`toast_blks_read`	`bigint`	Number of disk blocks read from this table's TOAST table (if any)
`toast_blks_hit`	`bigint`	Number of buffer hits in this table's TOAST table (if any)
`tidx_blks_read`	`bigint`	Number of disk blocks read from this table's TOAST table indexes (if any)
`tidx_blks_hit`	`bigint`	Number of buffer hits in this table's TOAST table indexes (if any)

The `pg_statio_all_tables` view will contain one row for each table in the current database (including TOAST tables), showing statistics about I/O on that specific table. The `pg_statio_user_tables` and `pg_statio_sys_tables` views contain the same information, but filtered to only show user and system tables respectively.

Table 27.17. `pg_statio_all_indexes` View

Column	Type	Description
`relid`	`oid`	OID of the table for this index
`indexrelid`	`oid`	OID of this index
`schemaname`	`name`	Name of the schema this index is in
`relname`	`name`	Name of the table for this index
`indexrelname`	`name`	Name of this index
`idx_blks_read`	`bigint`	Number of disk blocks read from this index
`idx_blks_hit`	`bigint`	Number of buffer hits in this index

The `pg_statio_all_indexes` view will contain one row for each index in the current database, showing statistics about I/O on that specific index. The `pg_statio_user_indexes` and `pg_statio_sys_indexes` views contain the same information, but filtered to only show user and system indexes respectively.

Table 27.18. `pg_statio_all_sequences` View

Column	Type	Description
`relid`	`oid`	OID of a sequence
`schemaname`	`name`	Name of the schema this sequence is in
`relname`	`name`	Name of this sequence
`blks_read`	`bigint`	Number of disk blocks read from this sequence
`blks_hit`	`bigint`	Number of buffer hits in this sequence

The `pg_statio_all_sequences` view will contain one row for each sequence in the current database, showing statistics about I/O on that specific sequence.

Table 27.19. `pg_stat_user_functions` View

Column	Type	Description
`funcid`	`oid`	OID of a function
`schemaname`	`name`	Name of the schema this function is in
`funcname`	`name`	Name of this function
`calls`	`bigint`	Number of times this function has been called
`total_time`	`double precision`	Total time spent in this function and all other functions called by it, in milliseconds
`self_time`	`double precision`	Total time spent in this function itself, not including other functions called by it, in milliseconds

The `pg_stat_user_functions` view will contain one row for each tracked function, showing statistics about executions of that function. The track_functions parameter controls exactly which functions are tracked.

27.2.3. Statistics Functions

Other ways of looking at the statistics can be set up by writing queries that use the same underlying statistics access functions used by the standard views shown above. For details such as the functions' names, consult the definitions of the standard views. (For example, in psql you could issue `\d+ pg_stat_activity`.) The access functions for per-database statistics take a database OID as an argument to identify which database to report on. The per-table and per-index functions take a table or index OID. The functions for per-function statistics take a function OID. Note that only tables, indexes, and functions in the current database can be seen with these functions.

Additional functions related to statistics collection are listed in Table 27.20.

Table 27.20. Additional Statistics Functions

Function	Return Type	Description
`pg_backend_pid()`	`integer`	Process ID of the server process handling the current session
`pg_stat_get_activity(integer)`	`setof record`	Returns a record of information about the backend with the specified PID, or one record for each active backend in the system if `NULL` is specified. The fields returned are a subset of those in the `pg_stat_activity` view.
`pg_stat_get_snapshot_timestamp()`	`timestamp with time zone`	Returns the timestamp of the current statistics snapshot
`pg_stat_clear_snapshot()`	`void`	Discard the current statistics snapshot
`pg_stat_reset()`	`void`	Reset all statistics counters for the current database to zero (requires superuser privileges by default, but EXECUTE for this function can be granted to others.)
`pg_stat_reset_shared(text)`	`void`	Reset some cluster-wide statistics counters to zero, depending on the argument (requires superuser privileges by default, but EXECUTE for this function can be granted to others). Calling `pg_stat_reset_shared('bgwriter')` will zero all the counters shown in the `pg_stat_bgwriter` view. Calling `pg_stat_reset_shared('archiver')` will zero all the counters shown in the `pg_stat_archiver` view.
`pg_stat_reset_single_table_counters(oid)`	`void`	Reset statistics for a single table or index in the current database to zero (requires superuser privileges by default, but EXECUTE for this function can be granted to others)
`pg_stat_reset_single_function_counters(oid)`	`void`	Reset statistics for a single function in the current database to zero (requires superuser privileges by default, but EXECUTE for this function can be granted to others)

`pg_stat_get_activity`, the underlying function of the `pg_stat_activity` view, returns a set of records containing all the available information about each backend process. Sometimes it may be more convenient to obtain just a subset of this information. In such cases, an older set of per-backend statistics access functions can be used; these are shown in Table 27.21. These access functions use a backend ID number, which ranges from one to the number of currently active backends. The function `pg_stat_get_backend_idset` provides a convenient way to generate one row for each active backend for invoking these functions. For example, to show the PIDs and current queries of all backends:

```
SELECT pg_stat_get_backend_pid(s.backendid) AS pid,
       pg_stat_get_backend_activity(s.backendid) AS query
    FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS s;
```

Table 27.21. Per-Backend Statistics Functions

Function	Return Type	Description
`pg_stat_get_backend_idset()`	`setof integer`	Set of currently active backend ID numbers (from 1 to the number of active backends)
`pg_stat_get_backend_activity(integer)`	`text`	Text of this backend's most recent query
`pg_stat_get_backend_activity_start(integer)`	`timestamp with time zone`	Time when the most recent query was started
`pg_stat_get_backend_client_addr(integer)`	`inet`	IP address of the client connected to this backend
`pg_stat_get_backend_client_port(integer)`	`integer`	TCP port number that the client is using for communication
`pg_stat_get_backend_dbid(integer)`	`oid`	OID of the database this backend is connected to
`pg_stat_get_backend_pid(integer)`	`integer`	Process ID of this backend
`pg_stat_get_backend_start(integer)`	`timestamp with time zone`	Time when this process was started
`pg_stat_get_backend_userid(integer)`	`oid`	OID of the user logged into this backend
`pg_stat_get_backend_wait_event_type(integer)`	`text`	Wait event type name if backend is currently waiting, otherwise NULL. See Table 27.4 for details.
`pg_stat_get_backend_wait_event(integer)`	`text`	Wait event name if backend is currently waiting, otherwise NULL. See Table 27.4 for details.
`pg_stat_get_backend_xact_start(integer)`	`timestamp with time zone`	Time when the current transaction was started

Submit correction

If you see anything in the documentation that is not correct, does not match your experience with the particular feature or requires further clarification, please use this form to report a documentation issue.

Privacy Policy | Code of Conduct | About PostgreSQL | Contact
Copyright © 1996-2020 The PostgreSQL Global Development Group