Postgres Plus® Advanced Server Performance Features Guide

Postgres Plus Advanced Server 8.3 R2

July 2, 2009

Postgres Plus Advanced Server Performance Features Guide, Version 1.0
by EnterpriseDB Corporation

EnterpriseDB Corporation, 235 Littleton Road, Westford, MA 01866, USA
T +1 978 589 5700 **F** +1 978 589 5701 **E** info@enterprisedb.com **www**.enterprisedb.com

Table of Contents

1 Introduction

This guide describes the performance features found in Postgres Plus Advanced Server. The primary performance features include:

- InfiniteCache lets you utilize memory on other computers connected to your network to increase the amount of memory in the shared buffer cache.

- Asynchronous Pre-Fetch reduces disk-idle time by distributing I/O across the drives in a RAID array.

- Dynatune makes optimal use of the system resources that are available on the host machine.

- The Dynamic Runtime Instrumentation Tools Architecture (DRITA) records *wait events* that affect system performance and offers a set of tools for inspecting those events.

- Optimizer Hints are directives that you embed in comment-like syntax immediately following the `SELECT, UPDATE`, or `DELETE` key words to influence the query optimizer.

1.1 Typographical Conventions Used in this Guide

Certain typographical conventions are used in this manual to clarify the meaning and usage of various commands, statements, programs, examples, etc. This section provides a summary of these conventions.

In the following descriptions a *term* refers to any word or group of words that are language keywords, user-supplied values, literals, etc. A term's exact meaning depends upon the context in which it is used.

- *Italic font* introduces a new term, typically, in the sentence that defines it for the first time.
- `Fixed-width (mono-spaced) font` is used for terms that must be given literally such as SQL commands, specific table and column names used in the examples, programming language keywords, etc. For example, `SELECT * FROM emp;`
- *`Italic fixed-width font`* is used for terms for which the user must substitute values in actual usage. For example, `DELETE FROM` *`table_name`*;
- A vertical pipe | denotes a choice between the terms on either side of the pipe. A vertical pipe is used to separate two or more alternative terms within square brackets (optional choices) or braces (one mandatory choice).
- Square brackets [ ] denote that one or none of the enclosed term(s) may be substituted. For example, `[ a | b ]`, means choose one of "`a`" or "`b`" or neither of the two.
- Braces {} denote that exactly one of the enclosed alternatives must be specified. For example, `{ a | b }`, means exactly one of "`a`" or "`b`" must be specified.
- Ellipses ... denote that the proceeding term may be repeated. For example, `[ a | b ] ...` means that you may have the sequence, "`b a a b a`".

2 Infinite Cache

Database performance is typically governed by two competing factors:

- Memory access is fast; disk access is slow.
- Memory space is scarce; disk space is abundant.

Postgres Plus Advanced Server (Advanced Server) tries very hard to minimize disk I/O by keeping frequently used data in memory. When the first server process starts, it creates an in-memory data structure known as the *buffer cache*. The buffer cache is organized as a collection of 8K (8192 byte) pages: each page in the buffer cache corresponds to a page in some table or index. The buffer cache is shared between all processes servicing a given database.

When you select a row from a table, Advanced Server reads the page that contains the row into the shared buffer cache. If there isn't enough free space in the cache, Advanced Server *evicts* some other page from the cache. If Advanced Server evicts a page that has been modified, that data is written back out to disk; otherwise, it is simply discarded. Index pages are cached in the shared buffer cache as well.

Figure 1.1 demonstrates the flow of data in a typical Advanced Server session:

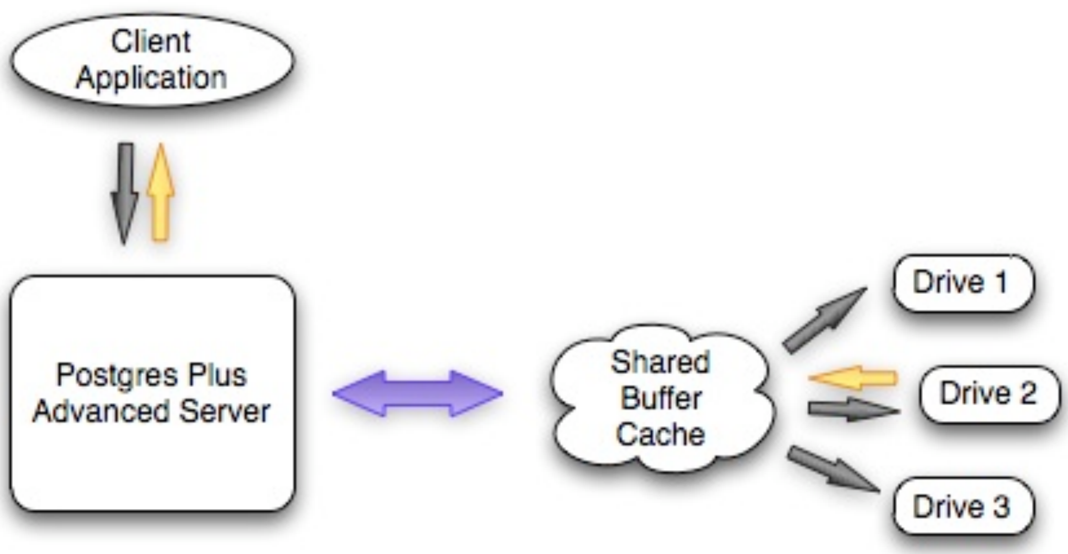


Figure 1.1 – Data Flow

A client application sends a query to the Postgres server and the server searches the shared buffer cache for the required data. If the requested data is found in the cache, the server immediately sends the data back to the client. If not, the server reads the page that holds the data into the shared buffer cache, evicting one or more pages if necessary. If the server decides to evict a page that has been modified, that page is written to disk.

As you can see, a query will execute much faster if the required data is found in the shared buffer cache.

One way to improve performance is to increase the amount of memory that you can devote to the shared buffer cache. However, most computers impose a strict limit on the amount of RAM that you can install. To help circumvent this limit, Infinite Cache lets you utilize memory from other computers connected to your network.

With Infinite Cache properly configured, Advanced Server will dedicate a portion of the memory installed on each *cache server* as a secondary memory cache. When a client application sends a query to the server, the server first searches the shared buffer cache for the required data; if the requested data is not found in the cache, the server searches for the necessary page in one of the cache servers.

Figure 1.2 shows the flow of data in an Advanced Server session with Infinite Cache:

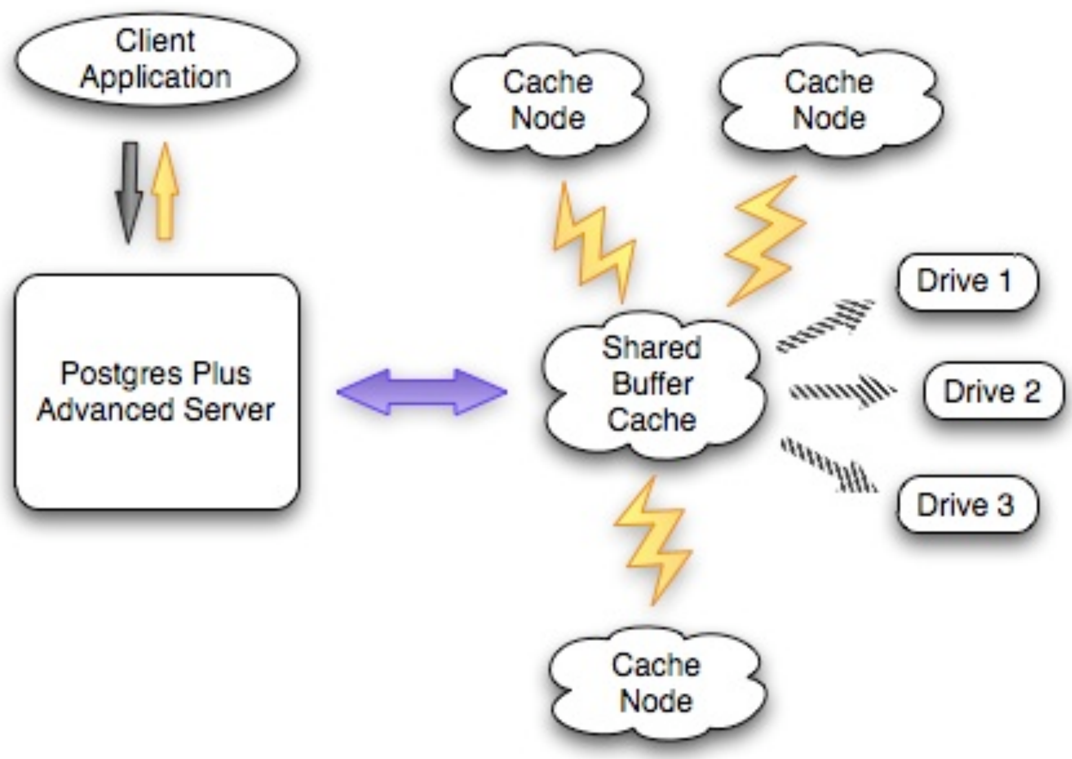


Figure 1.2 – Data flow with Infinite Cache

When a client application sends a query to the server, the server searches the shared buffer cache for the required data. If the requested data is found in the cache, the server immediately sends the data back to the client. If not, the server sends a request for the page to a specific cache server; if the cache server holds a copy of the page it sends the data back to the server and the server copies the page into the shared buffer cache. If the required page is not found in the primary cache (the shared buffer cache) or in the secondary cache (the cloud of cache servers), Advanced Server must read the page from disk.

As you can see, Infinite Cache can improve performance by utilizing RAM from other computers on your network in order to avoid reading frequently accessed data from disk.

Infinite Cache offers a second performance advantage: compression.

Without Infinite Cache, Advanced Server will read each page from disk as an 8K chunk; when a page resides in the shared buffer cache, it consumes 8K of RAM. With Infinite Cache, Postgres can *compress* each page before sending it to a cache server. A compressed page can take significantly less room in the secondary cache, making more space available for other data and effectively increasing the size of the cache. A compressed page consumes less network bandwidth as well, decreasing the amount of time required to retrieve a page from the secondary cache.

The fact that Infinite Cache can compress each page may make it attractive to configure a secondary cache server on the same computer that runs your Postgres server. If, for example, your computer is configured with 6GB of RAM, you may want to allocate a smaller amount (say 1GB) for the primary cache (the shared buffer cache) and a larger amount (4GB) to the secondary cache (Infinite Cache), reserving 1GB for the operating system. Since the secondary cache resides on the same computer, there is very little overhead involved in moving data between the primary and secondary cache. All data stored in the Infinite Cache is compressed so the secondary cache can hold many more pages than would fit into the (uncompressed) shared buffer cache. If you had allocated 5GB to the shared buffer cache, the cache could hold no more than 65000 pages (approximately). By assigning 4GB of memory to Infinite Cache, the cache may be able to hold 130000 pages (at 2x compression), 195000 pages (at 3x compression) or more. The compression factor that you achieve is determined by the amount of redundancy in the data itself and the `edb_icache_compression_level` parameter.

To use Infinite Cache, you must specify a list of one or more cache servers (computers on your network) and start the `edb_icache` daemon on each of those servers.

2.1 Configuring the Infinite Cache Server

The `postgresql.conf` file includes three configuration parameters that control the behavior of the Infinite Cache feature. The configuration file is read each time you start Advanced Server. To set a parameter, open `postgresql.conf` (located in the `$PGDATA` directory) and edit the section of the configuration file shown below:

```
# - Infinite Cache
#edb_enable_icache = off
#edb_icache_servers = '' #'host1:port1,host2,ip3:port3,ip4'
#edb_icache_compression_level = 6
```

Remove the pound sign that precedes each parameter that you want to set at startup, and specify the parameter settings. When you've updated and saved the configuration file, start Advanced Server for the changes to take effect.

The following example shows a typical collection of Infinite Cache settings:

```
edb_enable_icache            = on
edb_icache_servers           = 'localhost, 1.2.3.4:11000, 5.6.7.8'
edb_icache_compression_level = 6
```

2.1.1 edb_enable_icache

The `edb_enable_icache` parameter enables or disables Infinite Cache. If `edb_enable_icache` is set to `on`, Infinite Cache is enabled; if the parameter is set to `off`, Infinite Cache is disabled.

If you set edb_enable_icache to on, you must also specify a list of cache servers by setting the `edb_icache_servers` parameter (described in the next section).

The default value of `edb_enable_icache` is `off`.

2.1.2 edb_icache_servers

The `edb_icache_servers` parameter specifies a list of one or more servers with active edb-icache daemons. `edb_icache_servers` is a string value that takes the form of a comma-separated list of *hostname:port* pairs. You can specify each pair in any of the following forms:

- hostname
- IP-address
- hostname:portnumber
- IP-address:portnumber

If you do not specify a portnumber, Infinite Cache assumes that the cache server is listening at port 11211. This configuration parameter will take effect only if `edb_enable_icache` is set to `on`.

2.1.3 edb_icache_compression_level

The edb_icache_compression_level parameter controls the compression level that is applied to each page before storing it in the distributed Infinite Cache.

When Advanced Server reads data from disk, it typically reads the data in 8K increments. If edb_icache_compression_level is set to 0, each time Advanced Server sends an 8K page to the Infinite Cache server that page is stored (uncompressed) in 8K of cache memory. If the edb_icache_compression_level parameter is set to 9, Advanced Server applies the maximum compression possible before sending it to the Infinite Cache server, so a page that previously took 8K of cached memory might take 2K of cached memory. Exact compression numbers are difficult to predict, as they are dependent on the nature of the data on each page.

This parameter must be an integer in the range 0 to 9.

- A compression level of 0 disables compression; it uses no CPU time for compression, but requires more storage space and network resources to process.

- A compression level of 9 invokes the maximum amount of compression; it increases the load on the CPU, but less data flows across the network, so network demand is reduced. Each page takes less room on the Infinite Cache server, so memory requirements are reduced.

- A compression level of 5 or 6 is a reasonable compromise between the amount of compression received and the amount of CPU time invested.

By default, edb_icache_compression_level is set to 6.

The compression level must be set by the superuser and can be changed for the current session while the server is running. The following command disables the compression mechanism for the currently active session:

```
SET edb_icache_compression_level = 0
```


2.2 Controlling the edb-icache Daemons

`edb-icache` is a high-performance memory caching daemon that distributes and stores data in shared buffers. Advanced Server transparently interacts with `edb-icache` daemons to store and retrieve data.

Before starting Advanced Server, the `edb-icache` daemons must be running on each server node. Log into each server and start the `edb-icache` server (on that host) by issuing the following command:

```
# edb-icache -u enterprisedb -d -m 1024
```

where:

> `-u` specifies the user name
>
> `-m` specifies the amount of memory to be used by edb-icache (default is 64MB)
>
> `-d` designates that the service should run in the background

To kill an `edb-icache` daemon, execute the command:

```
# killall -HUP edb-icache
```

on each cache server.

2.2.1 Command Line Options

To view the command line options for the `edb-icache` daemon, use the following command from the `edb_Infinite Cache` subdirectory, located in the Advanced Server installation directory:

```
# edb-icache -h
```

The command line options are:

Parameter	Description
`-p <port_number>`	The TCP port number the Infinite Cache daemon is listening on. The default is 11211.
`-U <UDP_number>`	The UDP port number the Infinite Cache daemon is listening on. The default is 0 (off).
`-s <pathname>`	The Unix socket pathname the Infinite Cache daemon is listening on. If included, the server limits access to the host on which the Infinite Cache daemon is running, and disables network support for Infinite Cache.
`-a <mask>`	The access mask for the Unix socket, in octal form. The default value is 0700.
`-l <ip_addr>`	Specifies the IP address that the daemon is listening on. If an individual address is not specified, the default value is INDRR_ANY; all IP addresses assigned to the resource are available to the daemon.
`-d`	run as a daemon
`-r`	maximize core file limit
`-u <username>`	Assume the identity of the given user (when run as root).
`-m <numeric>`	max memory to user for items in megabytes, default is 64 MB
`-M`	return error on memory exhausted (rather than removing items)
`-c <numeric>`	Max simultaneous connections, default is 1024
`-k`	lock down all paged memory. Note that there is a limit on how much memory you may lock. Trying to allocate more than that would fail, so be sure you set the limit correctly for the user you started the daemon with (not for -u <username> user; under sh this is done with 'ulimit -S -l NUM_KB').
`-v`	verbose (print errors/warnings while in event loop)
`-vv`	very verbose (include client commands and responses)
`-h`	print the help and exit
`-i`	print memcached and libevent licenses
`-b`	run a managed instance (mnemonic: buckets)
`-P <file>`	save PID in <file>, only used with -d option
`-f <factor>`	chunk size growth factor, default 1.25
`-n <bytes>`	minimum space allocated for key+value+flags, default 48

2.2.2 edb-icache-tool

`edb-icache-tool` provides a command line interface that queries the `edb-icache` daemon to retrieve statistical information about a specific cache node. The syntax is:

```
edb-icache-tool <host[:port]> stats
```

`host` specifies the address of the host that you are querying.

`port` specifies the port that the daemon is listening on.

The following command retrieves statistical information about an Infinite Cache server located at the address, `192.168.23.85` and listening on port `11211`:

```
    # edb-icache-tool 192.168.23.85:11211 stats

Field                   Value
bytes                   1051681421
bytes_read              1410538244
bytes_written           42544414583
cmd_get                 5139685
cmd_set                 126588
connection_structures   104
curr_connections        4
curr_items              126588
evictions               0
get_hits                5139530
get_misses              155
limit_maxbytes          1073741824
pid                     3047
pointer_size            32
rusage_system           109.077417
rusage_user             21.423743
threads                 1
time                    1242367107
total_connections       115
total_items             126588
uptime                  1095
version                 1.2.6
```

2.3 Warming the edb-icache Servers

When Advanced Server starts, the primary and secondary caches are empty. When Advanced Server processes a client request, Advanced Server reads the required data from disk and stores a copy in each cache. You can improve server performance by *warming* (or pre-loading) the data into the memory cache before a client asks for it.

There are two advantages to warming the cache. Advanced Server will find data in the cache the first time it is requested by a client application, instead of waiting for it to be read from disk. Also, manually warming the cache with the data that your applications are most likely to need saves time by avoiding future random disk reads. If you don't warm the cache at startup, Postgres Plus Advanced Server performance may not reach full speed until the client applications happen to load commonly used data into the cache.

There are several ways to load pages to warm the Infinite Cache server nodes. You can use the `edb_icache_warm` binary to warm the caches from the command line, or you can use the `edb_icache_warm()` functions from within edb-psql or via scripts to warm the cache.

2.3.1 The edb_icache_warm() Functions

The `edb_icache_warm()` functions come in two variations; the first variation warms not only the table, but any indexes associated with the table. If you use the second variation, you must make additional calls to warm any associated indexes.

2.3.1.1 edb_icache_warm(table-spec)

This function warms the given table-spec and any associated indexes into the cache. You may specify `table-spec` as a table name, OID, or `regclass` value.

```
# edb-psql edb -c "select edb_icache_warm('accounts')"
```

When you call `edb_icache_warm(table-spec)`, Advanced Server reads every page in the given table, compresses each page (if configured to do so), and then sends the compressed data to an Infinite Cache server. `edb_icache_warm(table-spec)` will also read, compress, and cache every page in each index defined for the given table.

2.3.1.2 edb_icache_warm(table-spec, startbyte, endbyte):

This function warms the pages that contain the specified range of bytes into the cache. You must make subsequent calls to specify indexes separately when using this function.

```
# edb-psql edb -c "select edb_icache_warm('accounts', 1, 10000)"
```

Note that this function is typically called by a utility program (such as `edb_icache_warm)` to spread the warming process among several processes that operate in parallel.

2.3.2 Using the edb_icache_warm Binary

You can use the `edb_icache_warm` command-line utility to load the cache servers with specified tables, allowing fast access to relevant data from the cache.

The syntax for `edb_icache_warm` is:

```
# edb_icache_warm -d database -t tablename
```


The only required parameter is *tablename*. *tablename* can be specified with or without the `-t` option. All other parameters are optional; if omitted, default values are inferred from Advanced Server environment variables.

The options for `edb_icache_warm` are:

Option	Variable	Description
-h	*hostname*	The name of the host running Advanced Server. Include this parameter if you are running Advanced Server on a remote host. The default value is PGHOST.
-p	*portname*	Port in use by Advanced Server. Default value is PGPORT.
-j	*process count*	Number of (parallel) processes used to warm the cache. The default value is 1.
-U	*username*	The Advanced Server username. Unless specified, this defaults to PGUSER.
-d	*database*	The name of database containing the tables to be warmed. Default value is PGDATABASE.
-t	*tablename*	Name of table to be warmed. The index for the table is also warmed. Required.

2.4 Retrieving Statistics from Infinite Cache

You can view Infinite Cache statistics by using the `edb_icache_stats()` function at the `edb-psql` command line (or any other query tool).

2.4.1 edb_icache_stats()

This function returns a result set that reflects the state of an Infinite Cache node or nodes and the related usage statistics. The result set includes:

Statistic	Description
hostname	Host name (or IP address) of server
port	Port number at which edb-icache daemon is listening
state	Health of this server
write_failures	Number of write failures
bytes	Total number of bytes in use
bytes_read	Total number of bytes received by this server (from the network)
bytes_written	Total number of bytes sent by this server (to the network)
cmd_get	Cumulative number of read requests sent to this server
cmd_set	Cumulative number of write requests sent to this server
connection_structures	Number of connection structures allocated by the server
curr_connections	Number of open connections
curr_items	Number of items currently stored by the server
evictions	Number of valid items removed from cache to free memory for new items
get_hits	Number of read requests satisfied by this server
get_misses	Number of read requests not satisfied by this server
limit_maxbytes	Number of bytes allocated on this server for storage
pid	Process ID (on cache server)
pointer_size	Default pointer size on host OS (usually 32 or 64)
rusage_user	Accumulated user time for this process (seconds.microseconds)
rusage_system	Accumulated system time for this process (seconds.microseconds)
threads	Number of worker threads requested
total_time	Number of seconds since this server's base date (usually midnight, January 1, 1970, UTC)
total_connections	Total number of connections opened since the server started running
total_items	Total number of items stored by this server (cumulative)
uptime	Amount of time that server has been active
version	edb-icache version

You can use SQL queries to view Infinite Cache statistics. To view the server status of all Infinite Cache nodes:

```
SELECT hostname, port, state FROM edb_icache_stats()
```

```
 hostname      | port  | state
---------------+-------+--------
 192.168.23.85 | 11211 | UNHEALTHY
 192.168.23.85 | 11212 | ACTIVE
(2 rows)
```

To view complete statistics (shown here using edb-psql's expanded display mode, `\x`) for a specified node:

```
SELECT * FROM edb_icache_stats() WHERE hostname = '192.168.23.85:11211'
```

```
-[RECORD 1]-----------+--------------
hostname              | 192.168.23.85
port                  | 11211
state                 | ACTIVE
write_failures        | 0
bytes                 | 225029460
bytes_read            | 225728252
bytes_written         | 192806774
cmd_get               | 23313
cmd_set               | 27088
connection_structures | 53
curr_connections      | 3
curr_items            | 27088
evictions             | 0
get_hits              | 23266
get_misses            | 47
limit_maxbytes        | 805306368
pid                   | 4240
pointer_size          | 32
rusage_user           | 0.481926
rusage_system         | 1.583759
threads               | 1
total_time            | 1242199782
total_connections     | 66
total_items           | 27088
uptime                | 714
version               | 1.2.6
```

2.4.2 edb_icache_server_list

The `edb_icache_server_list` view exposes information about the status and health of all Infinite Cache servers listed in the `edb_icache_servers` GUC. The `edb_icache_server_list` view is created using the `edb_icache stats()` API; this view exposes the following information for each server:

Statistic	Description
hostname	Host name (or IP address) of server
port	Port number at which edb-icache daemon is listening
state	Health of this server
write_failures	Number of write failures
total_memory	Number of bytes allocated to the cache on this server
memory_used	Number of bytes currently used by the cache
memory_free	Number of unused bytes remaining in the cache
hit_ratio	Percentage of cache hits

The `state` column will contain one of the following four values, reflecting the health of the given server:

Server State	Description
`Active`	The server is known to be up and running.
`Unhealthy`	An error occurred while interacting with the cache server. Postgres will attempt to re-establish the connection with the server.
`Offline`	Postgres can no longer contact the given server.
`Manual Offline`	You have taken the server offline with the edb_icache_server_enable() function.

The following `SELECT` statement returns the health of each node in the Infinite Cache server farm:

```
SELECT hostname, port, state FROM edb_icache_server_list
```

```
   hostname     | port   | state
---------------+-------+-------
 192.168.23.85 | 11211 | ACTIVE
 192.168.23.85 | 11212 | ACTIVE
(2 rows)
```


To view complete details about a specific Infinite Cache node (shown here using edb-psql's `\x` expanded-view option):

```
SELECT * FROM edb_icache_server_list WHERE hostname = '192.168.23.85:11211'
```

```
-[RECORD 1]-----------+--------------
hostname               | 192.168.23.85
port                   | 11211
state                  | ACTIVE
write_failures         | 0
total_memory           | 805306368
memory_used            | 225029460
memory_free            | 580276908
hit_ratio              | 99.79
```

2.4.3 edb_icache_server_enable()

You can use the `edb_icache_server_enable()` function to take the Infinite Cache server offline for maintenance or other planned downtime. The syntax is:

```
    void edb_icache_server_enable(host TEXT, port INTEGER, online BOOL)
```

`host` specifies the host that you want to disable. The host name may be specified by name or numeric address.

`port` specifies the port number that the Infinite Cache server is listening on.

`online` specifies the state of the Infinite Cache server. The value of *`online`* must be `true` or `false`.

To take a server offline, specify the host that you want to disable, the port number that the Infinite Cache server is listening on, and `false`. To bring the Infinite Cache server back online, specify the host name and port number, and pass a value of `true`.

The state of a server taken offline with the `edb_icache_server_enable()` function is `MANUAL OFFLINE`. Postgres Plus Advanced Server will not automatically reconnect to an Infinite Cache server that you have taken offline with `edb_icache_server_enable(..., false)`; you must bring the server back online by calling `edb_icache_server_enable(..., true)`.

2.4.4 Infinite Cache Log Entries

When you start Advanced Server, a message that includes Infinite Cache status, cache node count and cache node size is written to the server log. The following example shows the server log for an active Infinite Cache installation with two 750 MB cache servers:

```
** EnterpriseDB Dynamic Tuning Agent*************************************
*       System Utilization: 66 %                                        *
*       Autovacuum Naptime: 60   Seconds                                *
*       Infinite Cache: on                                               *
*       Infinite Cache Servers: 2                                        *
*       Infinite Cache Size: 1.500  GB                                   *
*************************************************************************
```

2.5 *Allocating Memory to the Cache Servers*

As mentioned earlier in this document, each computer imposes a limit on the amount of *physical* memory that you can install. However, modern operating systems typically simulate a larger *address space* so that programs can transparently access more memory than is actually installed. This "virtual memory" allows a computer to run multiple programs which may simultaneously require more memory than is physically available. For example, you may run an e-mail client, a web browser, and a database server which each require 1GB of memory on a machine that contains only 2GB of physical RAM. When the operating system runs out of physical memory, it starts swapping bits and pieces of the currently running programs to disk to make room to satisfy your current demand for memory.

This can bring your system to a grinding halt.

Since the primary goal of Infinite Cache is to improve performance by limiting disk I/O, you should avoid dedicating so much memory to Infinite Cache that the operating system must start swapping data to disk. If the operating system begins to swap to disk, you lose the benefits offered by Infinite Cache.

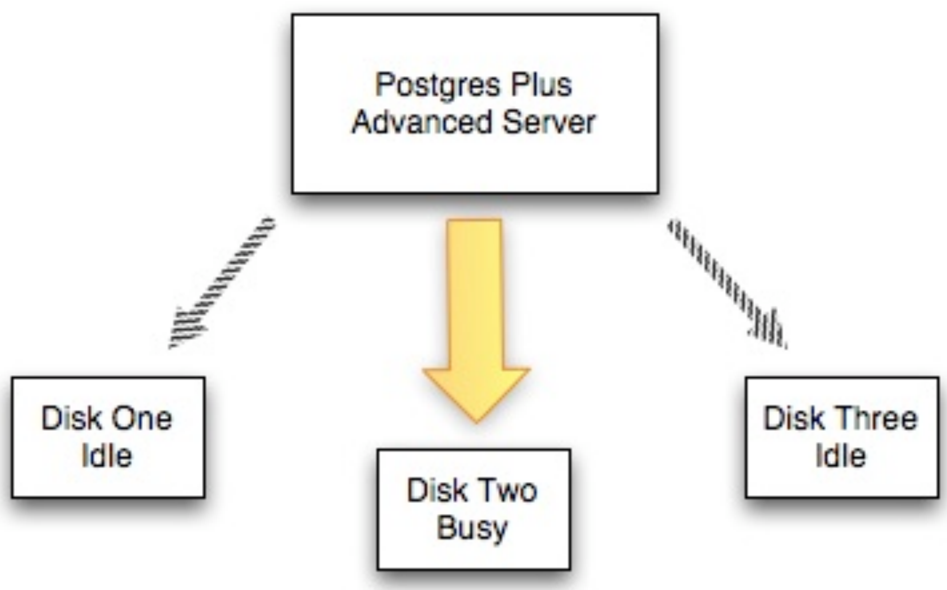
The overall demand for physical memory can vary throughout the day; if the server is frequently idle, you may never encounter swapping. If you have dedicated a large portion of physical memory to the cache, and system usage increases, the operating system may start swapping. To get the best performance and avoid disk swapping, dedicate a server node to Infinite Cache so other applications on that computer will not compete for physical memory.

3 Asynchronous Pre-Fetch

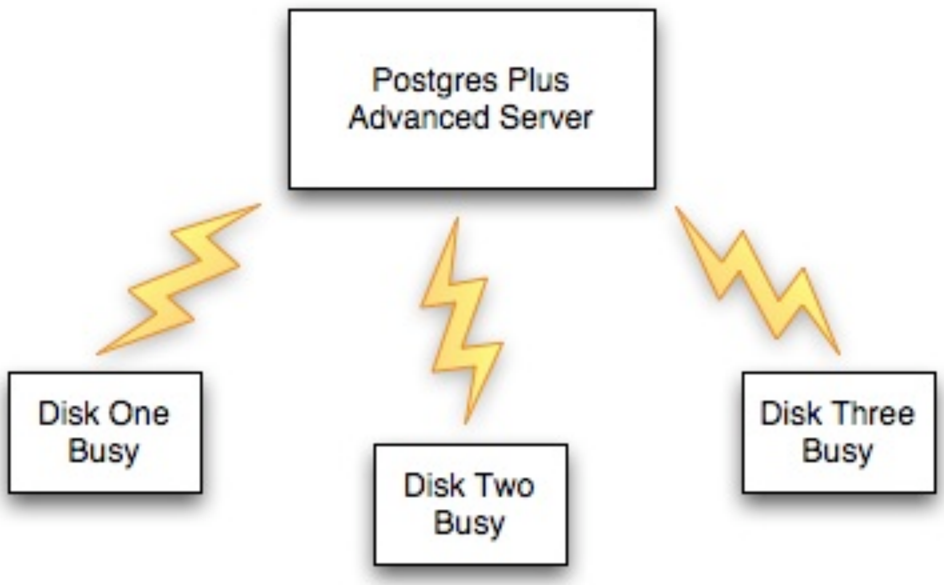
Asynchronous Pre-Fetch is a new feature in Advanced Server, release 8.3 R2 that exploits RAID array I/O systems, eliminating disk-idle time by ensuring I/O is continuously distributed across the drives in the RAID array.

RAID is a collection of techniques that distribute data across multiple disk drives in a manner that is transparent to the application. A RAID array (an array of disk drives managed by a RAID controller) can use striping, mirroring or a combination of the two methods to improve reliability and performance. When you write data to a *striped* RAID array, the controller splits the data into multiple chunks and writes each chunk to a different drive. When you write data to a *mirrored* array, the controller writes an exact copy of the data to each drive in the array.

Without Asynchronous Pre-Fetch, Postgres Plus Advanced Server traverses each table one page at a time, waiting for each disk read to complete before requesting the next page from the disk. Since the server never has more than one outstanding I/O request in progress, the operating system can only keep a single disk busy at any given point in time. In other words, without Asynchronous Pre-Fetch, the Postgres server effectively serializes all disk I/O.



With Asynchronous Pre-Fetch, the server will *prefetch* pages that it is likely to need in the very near future. To understand the performance benefits offered by Asynchronous Pre-Fetch, consider a typical index scan operation. Advanced Server starts by reading the page that contains the root of the index; that page may point to other pages in the index and it may point to the pages that contain the actual table data. As it processes the entries in an index page, the server accesses the related table data by reading the pages pointed to by each entry. If the table data is stored on a mirrored RAID array, any member of the array can satisfy a read request because each member contains an exact copy of all data; the RAID controller can route the read request to any idle array member.



Since an index page typically points to many other pages, the Asynchronous Pre-Fetch feature will schedule many read requests that can be carried out *in parallel*; the RAID controller can distribute the pending requests between all members of the array to improve performance.

Asynchronous Pre-Fetch also offers a performance boost to striped arrays because the controller can read two or more pages (in parallel) without waiting for the first read to complete.

Two parameter values control the behavior of Asynchronous Pre-Fetch; they are `effective_io_concurrency` and `edb_prefetch_indexscans`. You can modify the parameter settings in the `postgresql.conf` file.

3.1 effective_io_concurrency

When effective_io_concurrency is set to 0 all I/O is performed synchronously; Advanced Server issues a single I/O request and waits for that request to complete before proceeding. When it is set to 1, each block is requested asynchronously immediately following the previous operation, allowing the database to perform CPU work while the I/O proceeds simultaneously. In either case only a single drive in the I/O subsystem can be active leaving other drives participating in the same array idle.

To keep all the drives in the array active, Advanced Server issues multiple I/O requests concurrently. To do this Advanced Server must know how many drives participate in the I/O subsystem. Based on that value, Advanced Server estimates how many concurrent requests must be scheduled to keep all drives active. The higher the value of effective_io_concurrency, the more requests Advanced Server will try to keep pending at all times.

The optimal value is usually the number of data drives participating in the I/O system. For example for RAID-0 or RAID-1, it should be the total number of drives, whereas for RAID-5 it should be the number of data drives excluding the parity drives.

The expected speed increase for I/O influenced by this parameter is typically a factor equal to the number of drives; a 5-drive RAID array should see a five-fold increase in speed. However, not all of the I/O produced by a query is accelerated, so the effect on overall query execution time is less. Currently only Bitmap Heap Scans are accelerated.

Some caveats apply:

- In the case of OLTP systems with many concurrent active queries, using a single drive per query may be perfectly acceptable. Each query will make use of only a single drive, but in aggregate, concurrent OLTP queries will make effective use of all the drives. Increasing effective_io_concurrency effectively instructs Advanced Server to monopolize more I/O resources for each individual query; this is advantageous on systems handling few queries with available I/O resources but could have a detrimental effect on other concurrent queries.

- effective_io_concurrency is only used for Bitmap Heap Scans. For normal sequential scans the operating system should handle read-ahead internally (On Linux, see the blockdev command, in particular --setra and --setfra). For normal index scans, use the edb_prefetch_indexscans option.

- Advanced Server issues Asynchronous Pre-Fetch using the posix_fadvise() system routine (with the POSIX_FADV_WILLNEED option). This system call is not available or functional on every operating system. In particular, it has no effect on Solaris and does not exist on Windows.

- While the expected optimal value for `effective_io_concurrency` is the number of data drives, in practice, obtaining the maximum speedup sometimes requires some experimentation and experience shows that overestimating the number of drives can result in extra benefits.

- You should disable the Infinite Cache feature (`set edb_enable_icache = "off"`) if `effective_io_concurrency` is set to a number other than `0`. Asynchronous Pre-Fetch instructs the operating system to pre-fetch pages that are likely to be needed in the near future; Infinite Cache may cache those pages on a remote server farm instead of reading them from a local disk causing Infinite Cache and Asynchronous Pre-Fetch to interfere with each other, resulting in unnecessary disk I/O.

3.2 edb_prefetch_indexscans

By default, Advanced Server does not use Asynchronous Pre-Fetch for regular index scans. Index scans are often used in situations where not all rows will be used. Normally queries that read a large number of rows will use bitmap index scans.

If you execute a query that uses all the rows and performs a regular index scan, Asynchronous Pre-Fetch will offer a large performance boost. You will gain speed because Advanced Server will keep all drives as busy as possible; you will also gain speed because the operating system will typically sort the I/O requests and carry out those requests in sequential (rather than random) order.

In order for `edb_prefetch_indexscans` to have any effect, `effective_io_concurrency` must be set to a value greater than `1`.

The effect of `edb_prefetch_indexscans` will depend on the details of the index being scanned. If you execute a query against highly "de-clustered" indexes (where the index key is not correlated with the physical order of the heap), with narrow index keys, and the index scan is retrieving a large number of records, Asynchronous Pre-Fetch will be particularly effective.

`edb_prefetch_indexscans` is not recommended for use in some cases:

- It is not recommended for queries that do not read all the rows accessed in index scans. Queries with "`NOT IN (select ...)`" clauses will only access the first matching record; queries using "`SELECT min(indexed_col)`" or "`SELECT max(indexed_col)`" can use an index to find only the first or last matching record.

- It is not recommended for cursors where the client doesn't plan to scan the entire result set.

4 Dynatune

Postgres Plus Advanced Server supports dynamic tuning of the database server to make the optimal usage of the system resources available on the host machine on which it is installed. The two parameters that control this functionality are located in the `postgresql.conf` file. These parameters are:

- `edb_dynatune`
- `edb_dynatune_profile`

4.1 edb_dynatune

`edb_dynatune` determines how much of the host system's resources are to be used by the database server based upon the host machine's total available resources and the intended usage of the host machine.

When Postgres Plus Advanced Server is initially installed, the `edb_dynatune` parameter is set in accordance with the selected usage of the host machine on which it was installed - i.e., development machine, mixed use machine, or dedicated server. For most purposes, there is no need for the database administrator to adjust the various configuration parameters in the `postgresql.conf` file in order to improve performance.

You can change the value of the `edb_dynatune` parameter after the initial installation of Postgres Plus Advanced Server by editing the `postgresql.conf` file. The postmaster must be restarted in order for the new configuration to take effect.

The `edb_dynatune` parameter can be set to any integer value between 0 and 100, inclusive. A value of 0, turns off the dynamic tuning feature thereby leaving the database server resource usage totally under the control of the other configuration parameters in the `postgresql.conf` file.

A low non-zero, value (e.g., 1 - 33) dedicates the least amount of the host machine's resources to the database server. This setting would be used for a development machine where many other applications are being used.

A value in the range of 34 - 66 dedicates a moderate amount of resources to the database server. This setting might be used for a dedicated application server that may have a fixed number of other applications running on the same machine as Postgres Plus Advanced Server.

The highest values (e.g., 67 - 100) dedicate most of the server's resources to the database server. This setting would be used for a host machine that is totally dedicated to running Postgres Plus Advanced Server.

Once a value of `edb_dynatune` is selected, database server performance can be further fine-tuned by adjusting the other configuration parameters in the `postgresql.conf` file. Any adjusted setting overrides the corresponding value chosen by `edb_dynatune`. You can change the value of a parameter by un-commenting the configuration parameter, specifying the desired value, and restarting the database server.

4.2 edb_dynatune_profile

The `edb_dynatune_profile` parameter is used to control tuning aspects based upon the expected workload profile on the database server. This parameter takes effect upon startup of the database server.

The possible values for `edb_dynatune_profile` are:

Value	**Usage**
`oltp`	Recommended when the database server is processing heavy online transaction processing workloads.
`reporting`	Recommended for database servers used for heavy data reporting.
`mixed`	Recommended for servers that provide a mix of transaction processing and data reporting.

5 Dynamic Runtime Instrumentation Tools Architecture (DRITA)

Note: *This information is also included in the Oracle Compatibility Developer's Guide.*

The Dynamic Runtime Instrumentation Tools Architecture (DRITA) allows a DBA to query catalog views to determine the *wait events* that affect the performance of individual sessions or the system as a whole. DRITA records the number of times each event occurs as well as the time spent waiting; you can use this information to diagnose performance problems.

DRITA compares *snapshots* to evaluate the performance of a system. A snapshot is a saved set of system performance data at a given point in time. Each snapshot is identified by a unique ID number; you can use snapshot ID numbers with DRITA reporting functions to return system performance statistics.

DRITA consumes minimal system resources.

5.1 Initialization Parameters

DRITA includes a configuration parameter, `timed_statistics`, to control the collection of timing data. This is a dynamic parameter that can be set in the `postgresql.conf` file or while a session is in progress. The valid values are `TRUE` or `FALSE`; the default value is `FALSE`.

5.2 Setting up and Using DRITA

To use DRITA, you must first create a small set of tables and functions. To create the tables and functions that store and report information, run the following scripts:

```
snap_tables.sql
snap_functions.sql
```

After creating the required tables and functions, take a beginning snapshot. The beginning snapshot will be compared to a later snapshot to gauge system performance. To take a beginning snapshot:

```
SELECT * from edbsnap()
```

Then, run the workload that you would like to evaluate; when the workload has completed (or at a strategic point during the workload), take an ending snapshot:

```
SELECT * from edbsnap()
```


5.3 DRITA Functions

5.3.1.1 get_snaps()

The `get_snaps()` function returns a list of snapshot ID's; you can use the snapshot ID's to run one or more reporting functions. To view a list of snapshot ID's and the time they were taken, enter the following command:

```
SELECT * FROM get_snaps();

          get_snaps
------------------------------
 1 15-JUN-09 17:43:50.072733
 5 15-JUN-09 18:18:15.792194
 6 16-JUN-09 09:55:03.969197
 7 16-JUN-09 11:00:01.083305
 8 16-JUN-09 11:07:59.481583
 9 16-JUN-09 11:34:45.338325
 10 16-JUN-09 11:38:05.415392
 11 16-JUN-09 11:42:31.551796
 12 16-JUN-09 11:49:44.698102
 13 16-JUN-09 11:53:11.371272
 14 16-JUN-09 11:53:32.627307
 15 16-JUN-09 12:49:38.718433
 16 16-JUN-09 14:20:00.781601
 17 16-JUN-09 14:35:17.584266
 18 16-JUN-09 14:42:22.257647
 19 16-JUN-09 14:43:07.621677
(16 rows)
```

5.3.1.2 sys_rpt()

The `sys_rpt()` function returns system wait information. The signature is:

> `sys_rpt(`*`beginning_id`*`, `*`ending_id`*`, `*`top_n`*`)`

Parameters

`beginning_id`

> `beginning_id` is an integer value that represents the beginning session identifier.

`ending_id`

> `ending_id` is an integer value that represents the ending session identifier.

`top_n`

> `top_n` represents the number of rows to return

This example demonstrates a call to the `sys_rpt()` function:


```
SELECT * FROM sys_rpt(18, 19, 10);

                                   sys_rpt
-----------------------------------------------------------------------------
 WAIT NAME                                COUNT      WAIT TIME        % WAIT
 ---------------------------------------------------------------------------
 db file read                             31         0.187628         80.75
 query plan                               20         0.027784         11.96
 infinitecache read                       63         0.004523         1.95
 wal flush                                6          0.004067         1.75
 wal write                                1          0.004063         1.75
 wal file sync                            1          0.003664         1.58
 infinitecache write                      5          0.000548         0.24
 db file write                            5          0.000082         0.04
 wal write lock acquire                   0          0.000000         0.00
 bgwriter communication lock acquire      0          0.000000         0.00
(12 rows)
```

5.3.1.3 sess_rpt()

The `sess_rpt()` function returns session wait information. The signature is:

`sess_rpt(`*`beginning_id`*`, `*`ending_id`*`, `*`top_n`*`)`

Parameters

`beginning_id`

`beginning_id` is an integer value that represents the beginning session identifier.

`ending_id`

`ending_id` is an integer value that represents the ending session identifier.

`top_n`

`top_n` represents the number of rows to return

The following example demonstrates a call to the `sess_rpt()` function:

```
SELECT * FROM sess_rpt(18, 19, 10);

                              sess_rpt
-----------------------------------------------------------------------------
ID     USER        WAIT NAME              COUNT TIME(ms)    %WAIT SES   %WAIT ALL
 ----------------------------------------------------------------------------

 17373 enterprise db file read             30    0.175713    85.24       85.24
 17373 enterprise query plan               18    0.014930    7.24        7.24
 17373 enterprise wal flush                6     0.004067    1.97        1.97
 17373 enterprise wal write                1     0.004063    1.97        1.97
 17373 enterprise wal file sync            1     0.003664    1.78        1.78
```



```
 17373 enterprise infinitecache read     38    0.003076   1.49        1.49
 17373 enterprise infinitecache write    5     0.000548   0.27        0.27
 17373 enterprise db file write          5     0.000082   0.04        0.04
 17373 enterprise wal write lock acquire 0     0.000000   0.00        0.00
 17373 enterprise bgwriter comm lock ac  0     0.000000   0.00        0.00
(12 rows)
```

5.3.1.4 sessid_rpt()

The `sessid_rpt()` function returns session ID information for a specified backend. The signature is:

```
sessid_rpt(beginning_id, ending_id, backend_id)
```

Parameters

`beginning_id`

> `beginning_id` is an integer value that represents the beginning session identifier.

`ending_id`

> `ending_id` is an integer value that represents the ending session identifier.

`backend_id`

> `backend_id` is an integer value that represents the backend identifier.

The following code sample demonstrates a call to `sessid_rpt()`:

```
SELECT * FROM sessid_rpt(18, 19, 17373);

                                sessid_rpt
-----------------------------------------------------------------------------
 ID    USER       WAIT NAME              COUNT TIME(ms)  %WAIT SES   %WAIT ALL
 ----------------------------------------------------------------------------
 17373 enterprise db file read            30    0.175713  85.24       85.24
 17373 enterprise query plan              18    0.014930  7.24        7.24
 17373 enterprise wal flush               6     0.004067  1.97        1.97
 17373 enterprise wal write               1     0.004063  1.97        1.97
 17373 enterprise wal file sync           1     0.003664  1.78        1.78
 17373 enterprise infinitecache read      38    0.003076  1.49        1.49
 17373 enterprise infinitecache write     5     0.000548  0.27        0.27
 17373 enterprise db file write           5     0.000082  0.04        0.04
 17373 enterprise wal write lock acquire  0     0.000000  0.00        0.00
 17373 enterprise bgwriter comm lock ac   0     0.000000  0.00        0.00
(12 rows)
```


5.3.1.5 sesshist_rpt()

The `sesshist_rpt()` function returns session wait information for a specified backend. The signature is:

```
sesshist_rpt(beginning_id, ending_id, backend_id)
```

Parameters

`beginning_id`

> `beginning_id` is an integer value that represents the beginning session identifier.

`ending_id`

> `ending_id` is an integer value that represents the ending session identifier.

`backend_id`

> `backend_id` is an integer value that represents the backend identifier.

The following example demonstrates a call to the `sesshist_rpt()` function:

```
SELECT * FROM sesshist_rpt (18, 17373);

                                sesshist_rpt
 ----------------------------------------------------------------------------
 ID    USER       SEQ   WAIT NAME
    ELAPSED(ms)  File   Name                 # of Blk   Sum of Blks
 ----------------------------------------------------------------------------
 17373 enterprise 1     infinitecache read
    84           1249   pg_attribute         44         1
 17373 enterprise 2     query plan
    12           0      N/A                  0          0
 17373 enterprise 3     infinitecache read
    110          1255   pg_proc              64         1
 17373 enterprise 4     db file read
    3326         16421  session_waits_pk     2          1
 17373 enterprise 5     db file read
    4201         16421  session_waits_pk     3          1
 17373 enterprise 6     db file read
    5386         16421  session_waits_pk     0          1
 17373 enterprise 7     db file read
    13414        16416  edb$session_waits    3          1
 17373 enterprise 8     db file read
    4609         1260   pg_authid            0          1
 17373 enterprise 9     query plan
    12842        0      N/A                  0          0
 17373 enterprise 10    infinitecache read
    50           2619   pg_statistic         10         1
 17373 enterprise 11    infinitecache read
```

```
    51            2696   pg_statistic_relid_a 1          1
 17373 enterprise 12     infinitecache read
    51            1249   pg_attribute         8          1
 17373 enterprise 13     infinitecache read
    65            2654   pg_amop_opr_opc_inde 1          1
 17373 enterprise 14     infinitecache read
    77            2654   pg_amop_opr_opc_inde 3          1
 17373 enterprise 15     infinitecache read
    81            2696   pg_statistic_relid_a 4          1
 17373 enterprise 16     db file read
    11915         2696   pg_statistic_relid_a 3          1
 17373 enterprise 17     infinitecache read
    32            2696   pg_statistic_relid_a 3          1
 17373 enterprise 18     query plan
    12            0      N/A                  0          0
 17373 enterprise 19     infinitecache read
    50            1249   pg_attribute         12         1
 17373 enterprise 20     infinitecache read
    52            2659   pg_attribute_relid_a 2          1
 17373 enterprise 21     infinitecache read
    52            1255   pg_proc              2          1
 17373 enterprise 22     infinitecache read
    58            2617   pg_operator          3          1
 17373 enterprise 23     infinitecache read
    52            2690   pg_proc_oid_index    5          1
 17373 enterprise 24     infinitecache read
    58            1255   pg_proc              28         1
 17373 enterprise 25     infinitecache read
    50            2618   pg_rewrite           4          1
(27 rows)
```

5.3.1.6 truncsnap()

Use the `truncsnap()` function to purge all records from the snapshot tables:

```
SELECT * FROM truncsnap();

      truncsnap
----------------------
 Snapshots truncated.
(1 row)
```

A call to the `get_snaps()` function after calling the `truncsnap()` function shows that all records have been purged from the snapshot tables:

```
SELECT * FROM get_snaps
 get_snaps
-----------
(0 rows)
```

5.3.1.7 purgesnap()

The `purgesnap()` function purges a range of snapshots within the snap tables. Pass the snapshot ID's for the start of the range and the end of the range to purge:

```
SELECT * FROM purgesnap(6, 9);
```

```
             purgesnap
------------------------------------
 Snapshots in range 6 to 9 deleted.
(1 row)
```

A call to the `get_snaps()` function after calling the `purgesnap()` function shows that columns `6` through `9` have been purged from the snapshot tables:

```
 SELECT * FROM get_snaps
         get_snaps
------------------------------
 1 15-JUN-09 17:43:50.072733
 5 15-JUN-09 18:18:15.792194
 10 16-JUN-09 11:38:05.415392
 11 16-JUN-09 11:42:31.551796
 12 16-JUN-09 11:49:44.698102
 13 16-JUN-09 11:53:11.371272
 14 16-JUN-09 11:53:32.627307
 15 16-JUN-09 12:49:38.718433
 16 16-JUN-09 14:20:00.781601
 17 16-JUN-09 14:35:17.584266
 18 16-JUN-09 14:42:22.257647
 19 16-JUN-09 14:43:07.621677
(12 rows)
```

5.4 Simulating Statspack AWR Reports

The snapshot tables and functions described in this section return information comparable to the information contained in an Oracle Statspack/AWR (Automatic Workload Repository) report. When taking a snapshot, performance data from system catalog tables is saved into history tables. The reporting functions listed below report on the differences between two given snapshots.

Catalog Table	New DRITA Table	Reporting Function
pg_stat_database	edb$stat_database	stat_db_rpt()
pg_stat_all_tables	edb$stat_all_tables	stat_tables_rpt()
pg_stat_io_tables	edb$statio_all_tables	statio_tables_rpt()
Pgstat_all_indexes	edb$stat_all_indexes	stat_indexes_rpt()
pg_statio_all_indexes	edb$statio_all_indexes	statio_indexes_rpt()

The reporting functions can be executed individually or you can execute all five functions by calling the `edbreport()` function.

5.4.1.1 edbreport()

The `edbreport()` function includes data from the other reporting functions, plus additional system information. The signature is:

```
edb_report(beginning_id, ending_id)
```

Parameters

`beginning_id`

`beginning_id` is an integer value that represents the beginning session identifier.

`ending_id`

`ending_id` is an integer value that represents the ending session identifier.

The following code sample demonstrates a call to the `edbreport()` function:

```
SELECT * FROM edbreport(18, 19);

                                edbreport
---------------------------------------------------------------------------
EnterpriseDB Report for database edb        16-JUN-09
Version: EnterpriseDB 8.3.0.106 on i686-pc-linux-gnu, compiled by GCC gcc
(GCC) 4.1.0

      Begin snapshot: 18 at 16-JUN-09 14:42:22.257647
      End snapshot:   19 at 16-JUN-09 14:43:07.621677
```

```
Size of database edb is 2124 MB
     Tablespace: pg_default Size: 2136 MB Owner: enterprisedb
     Tablespace: pg_global Size: 283 kB Owner: enterprisedb

Schema: public                 Size: 2114 MB         Owner: enterprisedb


               Top 10 Relations by pages

TABLE                                        RELPAGES
---------------------------------------------------------------------------
accounts                                      222231
history                                       513
pg_proc                                       92
edb$statio_all_indexes                        86
edb$stat_all_indexes                          86
pg_depend                                     56
tellers                                       53
edb$stat_all_tables                           51
edb$statio_all_tables                         49
pg_attribute                                  43



               Top 10 Indexes by pages

INDEX                                        RELPAGES
---------------------------------------------------------------------------
accounts_pkey                                 46127
pg_proc_proname_args_nsp_index                81
pg_depend_reference_index                     48
pg_depend_depender_index                      46
edb$stat_idx_pk                               40
edb$statio_idx_pk                             40
pg_attribute_relid_attnam_index               33
pg_operator_oprname_l_r_n_index               20
edb$statio_tab_pk                             19
edb$stat_tab_pk                               19


               Top 10 Relations by DML

SCHEMA          RELATION                       UPDATES    DELETES    INSERTS
---------------------------------------------------------------------------
public          accounts                       7399697    0          7000000
public          tellers                        199699     0          700
public          branches                       199699     0          70
public          history                        0          150000     199699
sys             edb$stat_all_indexes           0          336        2128
sys             edb$statio_all_indexes         0          336        2128
sys             edb$stat_all_tables            0          264        1672
sys             edb$statio_all_tables          0          264        1672
sys             edb$session_wait_history       0          75         525
sys             edb$session_waits              0          9          125


  DATA from pg_stat_database

DATABASE NUMBACKENDS XACT COMMIT XACT ROLLBACK  BLKS READ BLKS HIT HIT RATIO
---------------------------------------------------------------------------
edb      0           5           0              59         2538      97.73
```

```
   DATA from pg_buffercache

RELATION                            BUFFERS
---------------------------------------------------------------------------
accounts                            21884
pg_proc                             34
pg_proc_proname_args_nsp_index      27
edb$statio_all_indexes              24
edb$stat_all_indexes                24
pg_attribute                        23
pg_operator                         19
edb$statio_all_tables               17
edb$stat_all_tables                 17
edb$stat_idx_pk                     14


    DATA from pg_stat_all_tables ordered by seq scan

SCHEMA     RELATION      SEQ       REL      READ    IDX
                         SCAN      TUP      SCAN    TUP READ INS    UPD DEL
----------------------------------------------------------------------------
pg_catalog pg_class       8         2952     78      65        0      0   0
pg_catalog pg_index       4         448      23      28        0      0   0
pg_catalog pg_namespace   4         76       1       1         0      0   0
pg_catalog pg_database    3         6        0       0         0      0   0
pg_catalog pg_authid      2         1        0       0         0      0   0
sys        edb$snap       1         15       0       0         1      0   0
public     accounts       0         0        0       0         0      0   0
public     branches       0         0        0       0         0      0   0
sys        wait_history   0         0        0       0         25     0   0
sys        session_waits  0         0        0       0         0      10  0




     DATA from pg_stat_all_tables ordered by rel tup read

SCHEMA             RELATION                       SEQ SCAN   REL TUP READ
   IDX SCAN   IDX TUP READ   INS    UPD    DEL

---------------------------------------------------------------------------
pg_catalog         pg_class                       8          2952
   78         65             0      0      0
 pg_catalog         pg_index                       4          448
   23         28             0      0      0
 pg_catalog         pg_namespace                   4          76
   1          1              0      0      0
 sys                edb$snap                       1          15
   0          0              1      0      0
 pg_catalog         pg_database                    3          6
   0          0              0      0      0
 pg_catalog         pg_authid                      2          1
   0          0              0      0      0
 public             accounts                       0          0
   0          0              0      0      0
 public             branches                       0          0
   0          0              0      0      0
 sys                edb$session_wait_history       0          0
   0          0              25     0      0
 sys                edb$session_waits              0          0
```



```
   0          0              10     0      0


  DATA from pg_statio_all_tables

SCHEMA       RELATION
                          HEAP  HEAP  IDX  IDX  TOAST  TOAST  TIDX  TIDX
                          READ  HIT   READ HIT  READ   HIT    READ  HIT
----------------------------------------------------------------------------
pg_catalog   pg_class
                          0     137   3    104  0      0      0     0
pg_catalog   pg_attribute
                          1     121   1    264  0      0      0     0
sys          edb$stat_all_indexes
                          5     111   5    225  0      0      0     0
sys          edb$statio_all_indexes
                          5     111   5    225  0      0      0     0
sys          edb$stat_all_tables
                          4     87    4    175  0      0      0     0
sys          edb$statio_all_tables
                          4     87    4    175  0      0      0     0
pg_catalog   pg_opclass
                          0     38    1    5    0      0      0     0
pg_catalog   pg_proc
                          0     37    0    92   0      0      0     0
pg_catalog   pg_index
                          1     30    1    22   0      0      0     0
sys          edb$session_wait_history
                          1     24    0    48   0      0      0     0


  DATA from pg_stat_all_indexes

SCHEMA       RELATION         INDEX
       IDX SCAN   IDX TUP READ   IDX TUP FETCH
----------------------------------------------------------------------------
pg_catalog   pg_cast          pg_cast_source_target_index
       140        21             21
pg_catalog   pg_attribute     pg_attribute_relid_attnum_index
       134        303            303
pg_catalog   pg_class         pg_class_oid_index
       48         48             48
pg_catalog   pg_proc          pg_proc_oid_index
       44         44             44
pg_catalog   pg_class         pg_class_relname_nsp_index
       30         17             17
pg_catalog   pg_statistic     pg_statistic_relid_att_index
       21         10             10
pg_catalog   pg_rewrite       pg_rewrite_rel_rulename_index
       15         15             15
pg_catalog   pg_index         pg_index_indrelid_index
       13         18             18
sys          edb$system_waits system_waits_pk
       12         38             6
pg_catalog   pg_index         pg_index_indexrelid_index
       10         10             10


  DATA from pg_statio_all_indexes

SCHEMA      RELATION                  INDEX
   IDX BLKS READ   IDX BLKS HIT
----------------------------------------------------------------------------
```

```
pg_catalog  pg_attribute             pg_attribute_relid_attnum_index
   1              264
sys         edb$stat_all_indexes     edb$stat_idx_pk
   5              225
sys         edb$statio_all_indexes   edb$statio_idx_pk
   5              225
sys         edb$stat_all_tables      edb$stat_tab_pk
   4              175
sys         edb$statio_all_tables    edb$statio_tab_pk
   4              175
pg_catalog  pg_cast                  pg_cast_source_target_index
   0              139
pg_catalog  pg_proc                  pg_proc_oid_index
   0              82
pg_catalog  pg_class                 pg_class_relname_nsp_index
   3              56
pg_catalog  pg_class                 pg_class_oid_index
   0              48
sys         edb$session_wait_history  session_waits_hist_pk
   0              48


   System Wait Information

WAIT NAME                               COUNT      WAIT TIME       % WAIT
---------------------------------------------------------------------------
db file read                            31         0.187628        80.75
query plan                              20         0.027784        11.96
infinitecache read                      63         0.004523        1.95
wal flush                               6          0.004067        1.75
wal write                               1          0.004063        1.75
wal file sync                           1          0.003664        1.58
infinitecache write                     5          0.000548        0.24
db file write                           5          0.000082        0.04
wal write lock acquire                  0          0.000000        0.00
bgwriter communication lock acquire     0          0.000000        0.00


   Database Parameters from postgresql.conf

PARAMETER                        SETTING      CONTEXT     MINVAL   MAXVAL
----------------------------------------------------------------------------
add_missing_from                 off          user
allow_system_table_mods          off          postmaster
archive_command                               sighup
archive_timeout                  0            sighup      0        2147483647
array_nulls                      on           user
authentication_timeout           10           sighup      1        600
autovacuum                       on           sighup
autovacuum_analyze_scale_factor  0.1          sighup      0        100
autovacuum_analyze_threshold     250          sighup      0        2147483647
autovacuum_freeze_max_age        200000000    postmaster  10000000 2000000000
autovacuum_naptime               60           sighup      1        2147483647
autovacuum_vacuum_cost_delay     -1           sighup      -1       1000
autovacuum_vacuum_cost_limit     -1           sighup      -1       10000
autovacuum_vacuum_scale_factor   0.2          sighup      0        100
autovacuum_vacuum_threshold      1000         sighup      0        2147483647

 ...

(384 rows)
```


5.4.1.2 stat_db_rpt()

The signature is:

stat_db_rpt(*beginning_id*, *ending_id*)

Parameters

beginning_id

beginning_id is an integer value that represents the beginning session identifier.

ending_id

ending_id is an integer value that represents the ending session identifier.

The following example demonstrates the stat_db_rpt() function:

```
SELECT * FROM stat_db_rpt(18, 19);

                                stat_db_rpt
-----------------------------------------------------------------------------
   DATA from pg_stat_database

 DATABASE NUMBACKENDS XACT COMMIT XACT ROLLBACK  BLKS READ BLKS HIT HIT RATIO
 ----------------------------------------------------------------------------
 edb      0           5           0              59        2538     97.73
(5 rows)
```

5.4.1.3 stat_tables_rpt()

The signature is:

function_name(*beginning_id*, *ending_id*, *top_n*, *scope*)

Parameters

beginning_id

beginning_id is an integer value that represents the beginning session identifier.

ending_id

ending_id is an integer value that represents the ending session identifier.

top_n

`top_n` represents the number of rows to return

`scope`

`scope` determines the scope of the statistics returned (`ALL`, `USER` or `SYS`).

The following code sample demonstrates the `stat_tables_rpt()` function:

```
SELECT * FROM stat_tables_rpt(18, 19, 10, 'ALL');

stat_tables_rpt
-----------------------------------------------------------------------------
DATA from pg_stat_all_tables ordered by seq scan

SCHEMA        RELATION
    SEQ SCAN   REL TUP READ IDX SCAN   IDX TUP READ   INS    UPD    DEL
-----------------------------------------------------------------------------
pg_catalog    pg_class
    8          2952         78         65             0      0      0
pg_catalog    pg_index
    4          448          23         28             0      0      0
pg_catalog    pg_namespace
    4          76           1          1              0      0      0
pg_catalog    pg_database
    3          6            0          0              0      0      0
pg_catalog    pg_authid
    2          1            0          0              0      0      0
sys           edb$snap
    1          15           0          0              1      0      0
public        accounts
    0          0            0          0              0      0      0
public        branches
    0          0            0          0              0      0      0
sys           edb$session_wait_history
    0          0            0          0              25     0      0
sys           edb$session_waits
    0          0            0          0              10     0      0

DATA from pg_stat_all_tables ordered by rel tup read

SCHEMA       RELATION
    SEQ SCAN   REL TUP READ IDX SCAN   IDX TUP READ INS    UPD    DEL
-----------------------------------------------------------------------------
pg_catalog   pg_class
    8          2952         78         65           0      0      0
pg_catalog   pg_index
    4          448          23         28           0      0      0
pg_catalog   pg_namespace
    4          76           1          1            0      0      0
sys          edb$snap
    1          15           0          0            1      0      0
pg_catalog   pg_database
    3          6            0          0            0      0      0
pg_catalog   pg_authid
    2          1            0          0            0      0      0
public       accounts
    0          0            0          0            0      0      0
public       branches
    0          0            0          0            0      0      0
sys          edb$session_wait_history
```



```
    0          0            0          0            25      0      0
sys           edb$session_waits
    0          0            0          0            10      0      0
(29 rows)
```

5.4.1.4 statio_tables_rpt()

The signature is:

```
statio_tables_rpt(beginning_id, ending_id, top_n, scope)
```

Parameters

`beginning_id`

> `beginning_id` is an integer value that represents the beginning session identifier.

`ending_id`

> `ending_id` is an integer value that represents the ending session identifier.

`top_n`

> `top_n` represents the number of rows to return

`scope`

> `scope` determines the scope of the statistics returned (`ALL`, `USER` or `SYS`).

The following example demonstrates the `statio_tables_rpt()` function:

```
SELECT * FROM statio_tables_rpt(18, 19, 10, 'ALL');

                             statio_tables_rpt
-----------------------------------------------------------------------------
   DATA from pg_statio_all_tables

SCHEMA           RELATION
    HEAP     HEAP     IDX      IDX      TOAST      TOAST      TIDX      TIDX
    READ     HIT      READ     HIT      READ       HIT        READ      HIT
 ----------------------------------------------------------------------------
 pg_catalog       pg_class
    0        137      3        104      0          0          0         0
 pg_catalog       pg_attribute
    1        121      1        264      0          0          0         0
 sys              edb$stat_all_indexes
    5        111      5        225      0          0          0         0
 sys              edb$statio_all_indexes
    5        111      5        225      0          0          0         0
 sys              edb$stat_all_tables
    4        87       4        175      0          0          0         0
```



```
 sys               edb$statio_all_tables
    4         87       4        175       0           0           0           0
 pg_catalog        pg_opclass
    0         38       1        5         0           0           0           0
 pg_catalog        pg_proc
    0         37       0        92        0           0           0           0
 pg_catalog        pg_index
    1         30       1        22        0           0           0           0
 sys               edb$session_wait_history
    1         24       0        48        0           0           0           0
(15 rows)
```

5.4.1.5 stat_indexes_rpt()

The signature is:

`stat_indexes_rpt(beginning_id, ending_id, top_n, scope)`

Parameters

`beginning_id`

`beginning_id` is an integer value that represents the beginning session identifier.

`ending_id`

`ending_id` is an integer value that represents the ending session identifier.

`top_n`

`top_n` represents the number of rows to return

`scope`

`scope` determines the scope of the statistics returned (`ALL`, `USER` or `SYS`).

The following code sample demonstrates the `stat_indexes_rpt()` function:

```
SELECT * FROM stat_indexes_rpt(18, 19, 10, 'ALL');

                         stat_indexes_rpt
 -----------------------------------------------------------------------------
   DATA from pg_stat_all_indexes

 SCHEMA               RELATION                 INDEX
    IDX SCAN    IDX TUP READ IDX TUP FETCH
 -----------------------------------------------------------------------------
 pg_catalog           pg_cast                  pg_cast_source_target_index
    140         21           21
 pg_catalog           pg_attribute             pg_attribute_relid_attnum_index
    134         303          303
```



```
 pg_catalog          pg_class                pg_class_oid_index
    48          48          48
 pg_catalog          pg_proc                 pg_proc_oid_index
    44          44          44
 pg_catalog          pg_class                pg_class_relname_nsp_index
    30          17          17
 pg_catalog          pg_statistic            pg_statistic_relid_att_index
    21          10          10
 pg_catalog          pg_rewrite              pg_rewrite_rel_rulename_index
    15          15          15
 pg_catalog          pg_index                pg_index_indrelid_index
    13          18          18
 sys                 edb$system_waits        system_waits_pk
    12          38          6
 pg_catalog          pg_index                pg_index_indexrelid_index
    10          10          10
(14 rows)
```

5.4.1.6 statio_indexes_rpt()

The signature is:

`statio_indexes_rpt(`*`beginning_id`*`, `*`ending_id`*`, `*`top_n`*`, `*`scope`*`)`

Parameters

`beginning_id`

`beginning_id` is an integer value that represents the beginning session identifier.

`ending_id`

`ending_id` is an integer value that represents the ending session identifier.

`top_n`

`top_n` represents the number of rows to return

`scope`

`scope` determines the scope of the statistics returned (`ALL`, `USER` or `SYS`).

The following example demonstrates the `statio_indexes_rpt()` function:

```
SELECT * FROM statio_indexes_rpt(18, 19, 10, 'ALL');

                              statio_indexes_rpt
 -----------------------------------------------------------------------------
```



```
  DATA from pg_statio_all_indexes

SCHEMA             RELATION                  INDEX
             IDX BLKS READ   IDX BLKS HIT
-----------------------------------------------------------------------------
pg_catalog         pg_attribute              pg_attribute_relid_attnum_index
             1               264
sys                edb$stat_all_indexes      edb$stat_idx_pk
             5               225
sys                edb$statio_all_indexes    edb$statio_idx_pk
             5               225
sys                edb$stat_all_tables       edb$stat_tab_pk
             4               175
sys                edb$statio_all_tables     edb$statio_tab_pk
             4               175
pg_catalog         pg_cast                   pg_cast_source_target_index
             0               139
pg_catalog         pg_proc                   pg_proc_oid_index
             0               82
pg_catalog         pg_class                  pg_class_relname_nsp_index
             3               56
pg_catalog         pg_class                  pg_class_oid_index
             0               48
sys                edb$session_wait_history  session_waits_hist_pk
             0               48
(14 rows)
```

5.5 Performance Tuning Recommendations

To use DRITA reports for performance tuning, review the top five events in a given report, looking for any event that takes a disproportionately large percentage of resources. In a streamlined system, user I/O will probably make up the largest number of waits. Waits should be evaluated in the context of CPU usage and total time; an event may not be significant if it takes 2 minutes out of a total measurement interval of 2 hours, if the rest of the time is consumed by CPU time. The component of response time (CPU "work" time or other "wait" time) that consumes the highest percentage of overall time should be evaluated.

When evaluating events, watch for:

Event type	Description
Checkpoint waits	Checkpoint waits may indicate that checkpoint parameters need to be adjusted, (`checkpoint_segments` and `checkpoint_timeout`).
WAL-related waits	WAL-related waits may indicate `wal_buffers` are under-sized.
SQL Parse waits	If the number of waits is high, try to use prepared statements.
db file random reads	If high, check that appropriate indexes and statistics exist.
db file random writes	If high, may need to decrease `bgwriter_delay`.
btree random lock acquires	May indicate indexes are being rebuilt. Schedule index builds during less active time.

Performance reviews should also include careful scrutiny of the hardware, the operating system, the network and the application SQL statements.

5.6 Event Descriptions

Event Name	Description
add in shmem lock acquire	Obsolete/unused
bgwriter communication lock acquire	The bgwriter (background writer) process has waited for the short-term lock that synchronizes messages between the bgwriter and a backend process.
btree vacuum lock acquire	The server has waited for the short-term lock that synchronizes access to the next available vacuum cycle ID.
buffer free list lock acquire	The server has waited for the short-term lock that synchronizes access to the list of free buffers (in shared memory).
checkpoint lock acquire:	A server process has waited for the short-term lock that prevents simultaneous checkpoints.
checkpoint start lock acquire	The server has waited for the short-term lock that synchronizes access to the bgwriter checkpoint schedule.
clog control lock acquire	The server has waited for the short-term lock that synchronizes access to the commit log.
control file lock acquire	The server has waited for the short-term lock that synchronizes write access to the control file (this should usually be a low number).
db file extend	A server process has waited for the operating system while adding a new page to the end of a file.
db file read	A server process has waited for the completion of a read (from disk).
db file write	A server process has waited for the completion of a write (to disk).
db file sync	A server process has waited for the operating system to flush all changes to disk.
first buf mapping lock acquire	The server has waited for a short-term lock that synchronizes access to the shared-buffer mapping table.
freespace lock acquire	The server has waited for the short-term lock that synchronizes access to the freespace map.
Infinite Cache read	The server has waited for an Infinite Cache read request.
Infinite Cache write	The server has waited for an Infinite Cache write request.
lwlock acquire	The server has waited for a short-term lock that has not been described elsewhere in this section.
multi xact gen lock acquire	The server has waited for the short-term lock that synchronizes access to the next available multi-transaction ID (when a SELECT...FOR SHARE statement executes).
multi xact member lock acquire	The server has waited for the short-term lock that synchronizes access to the multi-transaction member file (when a SELECT...FOR SHARE statement executes).
multi xact offset lock acquire	The server has waited for the short-term lock that synchronizes access to the multi-transaction offset file (when a SELECT...FOR SHARE statement executes).
oid gen lock acquire	The server has waited for the short-term lock that synchronizes access to the next available OID (object ID).
query plan	The server has computed the execution plan for a SQL statement.
rel cache init lock acquire	The server has waited for the short-term lock that prevents simultaneous relation-cache loads/unloads.
shmem index lock acquire	The server has waited for the short-term lock that synchronizes access to the shared-memory map.
sinval lock acquire	The server has waited for the short-term lock that synchronizes access

	to the cache invalidation state.
`sql parse`	The server has parsed a SQL statement.
`subtrans control lock acquire`	The server has waited for the short-term lock that synchronizes access to the subtransaction log.
`tablespace create lock acquire`	The server has waited for the short-term lock that prevents simultaneous CREATE TABLESPACE or DROP TABLESPACE commands.
`two phase state lock acquire`	The server has waited for the short-term lock that synchronizes access to the list of prepared transactions.
`wal insert lock acquire`	The server has waited for the short-term lock that synchronizes write access to the write-ahead log. A high number may indicate that WAL buffers are sized too small.
`wal write lock acquire`	The server has waited for the short-term lock that synchronizes write-ahead log flushes.
`wal file sync`	The server has waited for the write-ahead log to sync to disk (related to the wal_sync_method parameter which, by default, is 'fsync' - better performance can be gained by changing this parameter to open_sync).
`wal flush`	The server has waited for the write-ahead log to flush to disk.
`wal write`	The server has waited for a write to the write-ahead log buffer (expect this value to be high).
`xid gen lock acquire`	The server has waited for the short-term lock that synchronizes access to the next available transaction ID.

5.7 Catalog Views

The DRITA catalog views provide access to performance information relating to system waits.

5.7.1 edb$system_waits

The `edb$system_waits` view summarizes the number of waits and the total wait time per session for each wait named. It also displays the average and max wait times. `edb$system_waits` summarizes the following information:

```
  Column     |      Type      | Modifiers |     Definition
------------+---------------+-----------+------------------
 edb_id      | numeric        |           |identifier
 dbname      | text           |           |database name
 wait_name   | text           |           |name of the event
 wait_count  | numeric        |           |number of times the event occurs
 avg_wait    | numeric(50,6)  |           |average wait time in microseconds
 max_wait    | numeric        |           |maximum wait time in microseconds
 total_wait  | numeric        |           |total wait time in microseconds
 wait_name   | text           |           |name of the event
```

The following example shows the result of a `SELECT` statement on the `edb$system_waits` view:

```
select * from sys.edb$system_waits;

 edb_id | dbname  |wait_name   | wait_count |avg_wait  | max_wait  | totalwait
--------+--------+------------+------------+---------+----------+----------
      1 | edb     |db fileread|        301 |0.011516 | 0.629986 | 2.742500
      1 | edb     |wal flush   |         26 |0.010364 | 0.085380 | 0.269452
      1 | edb     |wal write   |         26 |0.010355 | 0.085371 | 0.269232
      1 | edb     |query plan |        277 |0.001367 | 0.049425 | 0.192442
      2 | edb     |wal flush   |         28 |0.040443 | 0.095150 | 0.431984
      2 | edb     |wal write   |         28 |0.040434 | 0.095093 | 0.431698
      2 | edb     |query plan |        299 |0.001479 | 0.049425 | 0.262596
```

5.7.2 edb$session_waits

The `edb$session_waits` view summarizes the number of waits and the total wait time per session for each wait named and identified by backend ID. It also displays the average and max wait times. `edb$session_waits` summarizes the following information:

```
    Column        |      Type       | Modifiers |Definition
-----------------+----------------+-----------+----------------
 backend_id       | bigint          |           |session identifier
 wait_count       | bigint          |           |number of times the event
                                                 occurs
 avg_wait_time    | numeric         |           |average wait time in
                                                 microseconds
```

```
  max_wait_time   | numeric(50,6) |            |maximum wait time in
                                                microseconds
  total_wait_time | numeric(50,6) |            |total wait time in
                                                microseconds
  wait_name       | text          |            |name of the event
```

The following code sample shows the result of a `SELECT` statement on the `edb$session_waits` view:

```
SELECT * FROM sys.edb$session_waits;

 edb_id | dbname | backend_id |  wait_name     | wait_count | avg_wait_time |
max_wait_time| total_wait_time |  usename      |     current_query
--------+--------+------------+---------------+------------+---------------+-
-------------+-----------------+-------------+--------------------------
      1 | edb    |      22935 | db file read  |        175 |      0.008399 |
    0.629986 |        1.469887 | enterprisedb | <IDLE>
      1 | edb    |      22988 | db file read  |        116 |      0.009556 |
    0.040627 |        1.108438 | enterprisedb | select * from edbsnap();
      1 | edb    |      22988 | wal flush     |        26  |      0.010364 |
    0.085380 |        0.269452 | enterprisedb | select * from edbsnap();
(3 rows)
```

5.7.3 edb$session_wait_history

The `edb$session_wait_history` view contains the last 25 wait events for each backend ID active during the session. The `edb$session_wait_history` view includes the following information:

```
   Column    |  Type   | Modifiers |   Definition
 ------------+--------+-----------+--------------------------
  edb_id     | numeric|           |identifier
  dbname     | text   |           |database name
  backend_id | bigint |           |session identifier
  seq        | bigint |           |number between 1 and 25
  wait_name  | text   |           |name of the event
  elapsed    | bigint |           |elapsed time in microseconds
  p1         | bigint |           |variable #1- meaning dependent on
                                                event
  p2         | bigint |           |variable #2- meaning dependent on
                                                event
  p3         | bigint |           |variable #3- meaning dependent on
                                                event
```

The following code sample shows the result of a `SELECT` statement on the `edb$session_wait_history` view:

```
SELECT * FROM sys.edb$session_wait_history;

 edb_id | dbname | backend_id | seq |   wait_name    | elapsed | p1  | p2  | p3
--------+--------+------------+-----+----------------+---------+-----+----+----
      1 | edb    |      22935 |   1 | query plan     |      54 |   0 |   0 |   0
      1 | edb    |      22935 |   2 | db file read   |    1116 |2689|   8 |   1
      1 | edb    |      22935 |   3 | db file read   |     983 |1255|  32 |   1
      1 | edb    |      22935 |   4 | db file read   |   13717 |2691|  19 |   1
      1 | edb    |      22935 |   5 | query plan     |      75 |    0|   0 |   0
```



```
      1 | edb     |       22935 |    6 | db file read   |   11053 |1255|  7 |  1
      1 | edb     |       22935 |    7 | db file read   |     404 |2689|  4 |  1
 (7 rows)
```


6 Optimizer Hints

Note: *This information is also included in the Oracle Compatibility Developer's Guide.*

When a `DELETE`, `SELECT`, or `UPDATE` command is issued, the Postgres Plus Advanced Server database server goes through a process to produce the result set of the command which is the final set of rows returned by the database server. How this result set is produced is the job of the *query planner*, also known as the *query optimizer*. Depending upon the specific command, there may be one or more alternatives, called *query plans*; the planner may consider as possible ways to create the result set. The selection of the plan to be used to actually execute the command is dependent upon various factors including:

- Costs assigned to various operations to retrieve the data (see the Planner Cost Constants in the `postgresql.conf` file).
- Settings of various planner method parameters (see the Planner Method Configuration section in the `postgresql.conf` file).
- Column statistics that have been gathered on the table data by the `ANALYZE` command (see the *Postgres Plus* documentation set for information on the `ANALYZE` command and column statistics).

Generally speaking, of the various feasible plans, the query planner chooses the one of least estimated cost for actual execution.

However, it is possible in any given `DELETE`, `SELECT`, or `UPDATE` command to directly influence selection of all or part of the final plan by using optimizer hints. *Optimizer hints* are directives embedded in comment-like syntax immediately following the `DELETE`, `SELECT`, or `UPDATE` key words that tell the planner to utilize or not utilize a certain approach for producing the result set.

Synopsis

```
{ DELETE | SELECT | UPDATE } /*+ { hint [ comment ] } [...] */
  statement_body

{ DELETE | SELECT | UPDATE } --+ { hint [ comment ] } [...]
  statement_body
```

Optimizer hints may be given in two different formats as shown above. Note that in both formats, a plus sign (+) must immediately follow the `/*` or `--` opening comment symbols with no intervening space in order for the following tokens to be interpreted as hints.

In the first format, the hint and optional comment may span multiple lines. In the second format, all hints and comments must be on a single line. The remainder of the statement must start on a new line.

Description

The following points regarding the usage of optimizer hints should be noted:

- The database server will always try to use the specified hints if at all possible.
- If a planner method parameter is set so as to disable a certain plan type, then this plan will not be used even if it is specified in a hint, unless there are no other possible options for the planner. Examples of planner method parameters are `enable_indexscan`, `enable_seqscan`, `enable_hashjoin`, `enable_mergejoin`, and `enable_nestloop`. These are all Boolean parameters.
- Remember that the hint is embedded within a comment. As a consequence, if the hint is misspelled or if any parameter to a hint such as view, table, or column name is misspelled, or non-existent in the SQL command, there will be no indication that any sort of error has occurred. No syntax error will be given and the entire hint is simply ignored.
- If an alias is used for a table or view name in the SQL command, then the alias name, not the original object name, must be used in the hint. For example, in the command, `SELECT /*+ FULL(acct) */ * FROM accounts acct` ..., `acct`, the alias for `accounts`, must be specified in the `FULL` hint, not the table name, `accounts`.
- Use the `EXPLAIN` command to ensure that the hint is correctly formed and the planner is using the hint. See the *Postgres Plus* documentation set for information on the `EXPLAIN` command.
- In general, optimizer hints should not be used in production applications. Typically, the table data changes throughout the life of the application. By ensuring that the more dynamic columns are `ANALYZE`d frequently, the column statistics will be updated to reflect value changes and the planner will use such information to produce the least cost plan for any given command execution. Use of optimizer hints defeats the purpose of this process and will result in the same plan regardless of how the table data changes.

Parameters

`hint`

An optimizer hint directive.

`comment`

A string with additional information. Note that there are restrictions as to what characters may be included in the comment. Generally, *`comment`* may only consist of alphabetic, numeric, the underscore, dollar sign, number sign and space characters. These must also conform to the syntax of an identifier. Any subsequent hint will be ignored if the comment is not in this form.

statement_body

The remainder of the `DELETE`, `SELECT`, or `UPDATE` command.

The following sections describe the various optimizer hint directives in more detail.

6.1 Default Optimization Modes

There are a number of optimization modes that can be chosen as the default setting for a Postgres Plus Advanced Server database cluster. This setting can also be changed on a per session basis by using the `ALTER SESSION` command as well as in individual `DELETE`, `SELECT`, and `UPDATE` commands within an optimizer hint. The configuration parameter that controls these default modes is named `OPTIMIZER_MODE`. The following table shows the possible values.

Table 3-1 Default Optimization Modes

Hint	Description
`ALL_ROWS`	Optimizes for retrieval of all rows of the result set.
`CHOOSE`	Does no default optimization based on assumed number of rows to be retrieved from the result set. This is the default.
`FIRST_ROWS`	Optimizes for retrieval of only the first row of the result set.
`FIRST_ROWS_10`	Optimizes for retrieval of the first 10 rows of the results set.
`FIRST_ROWS_100`	Optimizes for retrieval of the first 100 rows of the result set.
`FIRST_ROWS_1000`	Optimizes for retrieval of the first 1000 rows of the result set.
`FIRST_ROWS(`*n*`)`	Optimizes for retrieval of the first *n* rows of the result set. This form may not be used as the object of the `ALTER SESSION SET OPTIMIZER_MODE` command. It may only be used in the form of a hint in a SQL command.

These optimization modes are based upon the assumption that the client submitting the SQL command is interested in viewing only the first "n" rows of the result set and will then abandon the remainder of the result set. Resources allocated to the query are adjusted as such.

Examples

Alter the current session to optimize for retrieval of the first 10 rows of the result set.

```
ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS_10;
```

The current value of the `OPTIMIZER_MODE` parameter can be shown by using the `SHOW` command. Note that this command is a utility dependent command. In PSQL, the `SHOW` command is used as follows:

```
SHOW OPTIMIZER_MODE;

optimizer_mode
-----------------
 first_rows_10
```



```
(1 row)
```

The Oracle compatible `SHOW` command has the following syntax:

```
SHOW PARAMETER OPTIMIZER_MODE;

NAME
-------------------------------------------------
VALUE
-------------------------------------------------
optimizer_mode
first_rows_10
```

The following example shows an optimization mode used in a `SELECT` command as a hint:

```
SELECT /*+ FIRST_ROWS(7) */ * FROM emp;

empno | ename  |    job    | mgr  |      hiredate      |   sal   |  comm   | deptno
-------+--------+-----------+------+--------------------+---------+---------+--------
  7369 | SMITH  | CLERK     | 7902 | 17-DEC-80 00:00:00 |  800.00 |         |     20
  7499 | ALLEN  | SALESMAN  | 7698 | 20-FEB-81 00:00:00 | 1600.00 |  300.00 |     30
  7521 | WARD   | SALESMAN  | 7698 | 22-FEB-81 00:00:00 | 1250.00 |  500.00 |     30
  7566 | JONES  | MANAGER   | 7839 | 02-APR-81 00:00:00 | 2975.00 |         |     20
  7654 | MARTIN | SALESMAN  | 7698 | 28-SEP-81 00:00:00 | 1250.00 | 1400.00 |     30
  7698 | BLAKE  | MANAGER   | 7839 | 01-MAY-81 00:00:00 | 2850.00 |         |     30
  7782 | CLARK  | MANAGER   | 7839 | 09-JUN-81 00:00:00 | 2450.00 |         |     10
  7788 | SCOTT  | ANALYST   | 7566 | 19-APR-87 00:00:00 | 3000.00 |         |     20
  7839 | KING   | PRESIDENT |      | 17-NOV-81 00:00:00 | 5000.00 |         |     10
  7844 | TURNER | SALESMAN  | 7698 | 08-SEP-81 00:00:00 | 1500.00 |    0.00 |     30
  7876 | ADAMS  | CLERK     | 7788 | 23-MAY-87 00:00:00 | 1100.00 |         |     20
  7900 | JAMES  | CLERK     | 7698 | 03-DEC-81 00:00:00 |  950.00 |         |     30
  7902 | FORD   | ANALYST   | 7566 | 03-DEC-81 00:00:00 | 3000.00 |         |     20
  7934 | MILLER | CLERK     | 7782 | 23-JAN-82 00:00:00 | 1300.00 |         |     10
(14 rows)
```

6.2 Access Method Hints

The following hints influence how the optimizer accesses relations to create the result set.

Table 3-2 Access Method Hints

Hint	Description
`FULL(`*`table`*`)`	Perform a full sequential scan on *`table`*.
`INDEX(`*`table`*` [ `*`index`*` ] [...])`	Use *`index`* on *`table`* to access the relation.
`NO_INDEX(`*`table`*` [ `*`index`*` ] [...])`	Do not use *`index`* on *`table`* to access the relation.

In addition, the `ALL_ROWS`, `FIRST_ROWS`, and `FIRST_ROWS(`*`n`*`)` hints of Table 3-1 can be used.

Examples

The sample application does not have sufficient data to illustrate the effects of optimizer hints so the remainder of the examples in this section will use a banking database created by the `pgbench` application located in the Postgres Plus Advanced Server `dbserver\bin` subdirectory.

The following steps create a database named, `bank`, populated by the tables, `accounts`, `branches`, `tellers`, and `history`. The `-s 5` option specifies a scaling factor of five which results in the creation of five branches, each with 100,000 accounts, resulting in a total of 500,000 rows in the `accounts` table and five rows in the `branches` table. Ten tellers are assigned to each branch resulting in a total of 50 rows in the `tellers` table.

Note, if using Linux use the `export` command instead of the `SET PATH` command as shown below.

```
export PATH=/opt/EnterpriseDB/8.3/dbserver/bin:$PATH
```

The following example was run in Windows.

```
SET PATH=C:\EnterpriseDB\8.3\dbserver\bin;%PATH%

createdb -U enterprisedb bank
CREATE DATABASE

pgbench -i -s 5 -U enterprisedb -d bank

creating tables...
10000 tuples done.
20000 tuples done.
30000 tuples done.
        .
        .
        .
470000 tuples done.
480000 tuples done.
490000 tuples done.
500000 tuples done.
set primary key...
vacuum...done.
```

Ten transactions per client are then processed for eight clients for a total of 80 transactions. This will populate the `history` table with 80 rows.

```
pgbench -U enterprisedb -d bank -c 8 -t 10
        .
        .
        .
transaction type: TPC-B (sort of)
scaling factor: 5
number of clients: 8
number of transactions per client: 10
number of transactions actually processed: 80/80
tps = 6.023189 (including connections establishing)
tps = 7.140944 (excluding connections establishing)
```

The table definitions are shown below:

```
\d accounts

       Table "public.accounts"
  Column  |      Type      | Modifiers
----------+---------------+-----------
```

```
 aid      | integer        | not null
 bid      | integer        |
 abalance | integer        |
 filler   | character(84) |
Indexes:
    "accounts_pkey" PRIMARY KEY, btree (aid)

\d branches

       Table "public.branches"
  Column  |      Type      | Modifiers
----------+---------------+-----------
 bid      | integer        | not null
 bbalance | integer        |
 filler   | character(88) |
Indexes:
    "branches_pkey" PRIMARY KEY, btree (bid)

\d tellers

        Table "public.tellers"
  Column  |      Type      | Modifiers
----------+---------------+-----------
 tid      | integer        | not null
 bid      | integer        |
 tbalance | integer        |
 filler   | character(84) |
Indexes:
    "tellers_pkey" PRIMARY KEY, btree (tid)

\d history

              Table "public.history"
 Column |            Type             | Modifiers
--------+-----------------------------+-----------
 tid    | integer                     |
 bid    | integer                     |
 aid    | integer                     |
 delta  | integer                     |
 mtime  | timestamp without time zone |
 filler | character(22)               |
```

The `EXPLAIN` command shows the plan selected by the query planner. In the following example, `aid` is the primary key column, so an indexed search is used on index, `accounts_pkey`.

```
EXPLAIN SELECT * FROM accounts WHERE aid = 100;

                                  QUERY PLAN
-----------------------------------------------------------------------------
--
 Index Scan using accounts_pkey on accounts  (cost=0.00..8.32 rows=1
width=97)
   Index Cond: (aid = 100)
(2 rows)
```

The `FULL` hint is used to force a full sequential scan instead of using the index as shown below:

```
EXPLAIN SELECT /*+ FULL(accounts) */ * FROM accounts WHERE aid = 100;
```



```
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on accounts  (cost=0.00..14461.10 rows=1 width=97)
   Filter: (aid = 100)
(2 rows)
```

The `NO_INDEX` hint also forces a sequential scan as shown below:

```
EXPLAIN SELECT /*+ NO_INDEX(accounts accounts_pkey) */ * FROM accounts WHERE
aid = 100;

                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on accounts  (cost=0.00..14461.10 rows=1 width=97)
   Filter: (aid = 100)
(2 rows)
```

In addition to using the `EXPLAIN` command as shown in the prior examples, more detailed information regarding whether or not a hint was used by the planner can be obtained by setting the `client_min_messages` and `trace_hints` configuration parameters as follows:

```
SET client_min_messages TO info;
SET trace_hints TO true;
```

The `SELECT` command with the `NO_INDEX` hint is repeated below to illustrate the additional information produced when the aforementioned configuration parameters are set.

```
EXPLAIN SELECT /*+ NO_INDEX(accounts accounts_pkey) */ * FROM accounts WHERE
aid = 100;

INFO:  [HINTS] Index Scan of [accounts].[accounts_pkey] rejected because of
NO_INDEX hint.

INFO:  [HINTS] Bitmap Heap Scan of [accounts].[accounts_pkey] rejected
because of NO_INDEX hint.
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on accounts  (cost=0.00..14461.10 rows=1 width=97)
   Filter: (aid = 100)
(2 rows)
```

Note that if a hint is ignored, the `INFO:  [HINTS]` line will not appear. This may be an indication that there was a syntax error or some other misspelling in the hint as shown in the following example where the index name is misspelled.

```
EXPLAIN SELECT /*+ NO_INDEX(accounts accounts_xxx) */ * FROM accounts WHERE
aid = 100;

                                  QUERY PLAN
-----------------------------------------------------------------------------
--
 Index Scan using accounts_pkey on accounts  (cost=0.00..8.32 rows=1
 width=97)
```

```
   Index Cond: (aid = 100)
(2 rows)
```

6.3 Joining Relations Hints

There are three possible plans that may be used to perform a join between two tables:

- *Nested Loop Join* – The right table is scanned once for every row in the left table.
- *Merge Sort Join* – Each table is sorted on the join attributes before the join starts. The two tables are then scanned in parallel and the matching rows are combined to form the join rows.
- *Hash Join* – The right table is scanned and its join attributes are loaded into a hash table using its join attributes as hash keys. The left table is then scanned and its join attributes are used as hash keys to locate the matching rows from the right table.

The following table lists the optimizer hints that can be used to influence the planner to use one type of join plan over another.

Table 3-3 Join Hints

Hint	Description
`USE_HASH(`*`table`*` [...])`	Use a hash join with a hash table created from the join attributes of *`table`*.
`NO_USE_HASH(`*`table`*` [...])`	Do not use a hash join created from the join attributes of *`table`*.
`USE_MERGE(`*`table`*` [...])`	Use a merge sort join for *`table`*.
`NO_USE_MERGE(`*`table`*` [...])`	Do not use a merge sort join for *`table`*.
`USE_NL(`*`table`*` [...])`	Use a nested loop join for *`table`*.
`NO_USE_NL(`*`table`*` [...])`	Do not use a nested loop join for *`table`*.

Examples

In the following example, a join is performed on the `branches` and `accounts` tables. The query plan shows that a hash join is used by creating a hash table from the join attribute of the `branches` table.

```
EXPLAIN SELECT b.bid, a.aid, abalance FROM branches b, accounts a WHERE b.bid
= a.bid;

                                QUERY PLAN
--------------------------------------------------------------------------
 Hash Join  (cost=1.11..20092.70 rows=500488 width=12)
   Hash Cond: (a.bid = b.bid)
   ->  Seq Scan on accounts a  (cost=0.00..13209.88 rows=500488 width=12)
   ->  Hash  (cost=1.05..1.05 rows=5 width=4)
         ->  Seq Scan on branches b  (cost=0.00..1.05 rows=5 width=4)
(5 rows)
```

By using the `USE_HASH(a)` hint, the planner is forced to create the hash table from the `accounts` join attribute instead of from the `branches` table. Note the use of the alias, `a`, for the `accounts` table in the `USE_HASH` hint.

```
EXPLAIN SELECT /*+ USE_HASH(a) */ b.bid, a.aid, abalance FROM branches b,
accounts a WHERE b.bid = a.bid;

                                   QUERY PLAN
-----------------------------------------------------------------------------
---
 Hash Join  (cost=21909.98..30011.52 rows=500488 width=12)
   Hash Cond: (b.bid = a.bid)
   ->  Seq Scan on branches b  (cost=0.00..1.05 rows=5 width=4)
   ->  Hash  (cost=13209.88..13209.88 rows=500488 width=12)
         ->  Seq Scan on accounts a  (cost=0.00..13209.88 rows=500488
width=12)
(5 rows)
```

Next, the `NO_USE_HASH(a b)` hint forces the planner to use an approach other than hash tables. The result is a nested loop.

```
EXPLAIN SELECT /*+ NO_USE_HASH(a b) */ b.bid, a.aid, abalance FROM branches
b, accounts a WHERE b.bid = a.bid;

                                QUERY PLAN
--------------------------------------------------------------------------
 Nested Loop  (cost=1.05..69515.84 rows=500488 width=12)
   Join Filter: (b.bid = a.bid)
   ->  Seq Scan on accounts a  (cost=0.00..13209.88 rows=500488 width=12)
   ->  Materialize  (cost=1.05..1.11 rows=5 width=4)
         ->  Seq Scan on branches b  (cost=0.00..1.05 rows=5 width=4)
(5 rows)
```

Finally, the `USE_MERGE` hint forces the planner to use a merge join.

```
EXPLAIN SELECT /*+ USE_MERGE(a) */ b.bid, a.aid, abalance FROM branches b,
accounts a WHERE b.bid = a.bid;

                                   QUERY PLAN
-----------------------------------------------------------------------------
---
 Merge Join  (cost=69143.62..76650.97 rows=500488 width=12)
   Merge Cond: (b.bid = a.bid)
   ->  Sort  (cost=1.11..1.12 rows=5 width=4)
         Sort Key: b.bid
         ->  Seq Scan on branches b  (cost=0.00..1.05 rows=5 width=4)
   ->  Sort  (cost=69142.52..70393.74 rows=500488 width=12)
         Sort Key: a.bid
         ->  Seq Scan on accounts a  (cost=0.00..13209.88 rows=500488
width=12)
(8 rows)
```

In this three-table join example, the planner first performs a hash join on the `branches` and `history` tables, then finally performs a nested loop join of the result with the `accounts_pkey` index of the `accounts` table.

```
EXPLAIN SELECT h.mtime, h.delta, b.bid, a.aid FROM history h, branches b,
accounts a WHERE h.bid = b.bid AND h.aid = a.aid;

                                      QUERY PLAN
-----------------------------------------------------------------------------
---------
 Nested Loop  (cost=1.11..207.95 rows=26 width=20)
   ->  Hash Join  (cost=1.11..25.40 rows=26 width=20)
         Hash Cond: (h.bid = b.bid)
         ->  Seq Scan on history h  (cost=0.00..20.20 rows=1020 width=20)
         ->  Hash  (cost=1.05..1.05 rows=5 width=4)
               ->  Seq Scan on branches b  (cost=0.00..1.05 rows=5 width=4)
   ->  Index Scan using accounts_pkey on accounts a  (cost=0.00..7.01 rows=1
       width=4)
         Index Cond: (h.aid = a.aid)
(8 rows)
```

This plan is altered by using hints to force a combination of a merge sort join and a hash join.

```
EXPLAIN SELECT /*+ USE_MERGE(h b) USE_HASH(a) */ h.mtime, h.delta, b.bid,
a.aid FROM history h, branches b, accounts a WHERE h.bid = b.bid AND h.aid =
a.aid;

                                        QUERY PLAN
-----------------------------------------------------------------------------
--------------
 Merge Join  (cost=23480.11..23485.60 rows=26 width=20)
   Merge Cond: (h.bid = b.bid)
   ->  Sort  (cost=23479.00..23481.55 rows=1020 width=20)
         Sort Key: h.bid
         ->  Hash Join  (cost=21421.98..23428.03 rows=1020 width=20)
               Hash Cond: (h.aid = a.aid)
               ->  Seq Scan on history h  (cost=0.00..20.20 rows=1020
                   width=20)
               ->  Hash  (cost=13209.88..13209.88 rows=500488 width=4)
                     ->  Seq Scan on accounts a  (cost=0.00..13209.88
                         rows=500488 width=4)
   ->  Sort  (cost=1.11..1.12 rows=5 width=4)
         Sort Key: b.bid
         ->  Seq Scan on branches b  (cost=0.00..1.05 rows=5 width=4)
(12 rows)
```

6.4 Global Hints

Thus far, hints have been applied directly to tables that are referenced in the SQL command. It is also possible to apply hints to tables that appear in a view when the view is referenced in the SQL command. The hint does not appear in the view, itself, but rather in the SQL command that references the view.

When specifying a hint that is to apply to a table within a view, the view and table names are given in dot notation within the hint argument list.

Synopsis

```
hint(view.table)
```

Parameters

hint

Any of the hints in Table 3-2 or Table 3-3.

view

The name of the view containing *table*.

table

The table on which the hint is to be applied.

Examples

A view named, `tx`, is created from the three-table join of `history`, `branches`, and `accounts` shown in the final example of Section 6.3.

```
CREATE VIEW tx AS SELECT h.mtime, h.delta, b.bid, a.aid FROM history h,
branches b, accounts a WHERE h.bid = b.bid AND h.aid = a.aid;
```

The query plan produced by selecting from this view is show below:

```
EXPLAIN SELECT * FROM tx;

                                      QUERY PLAN
-----------------------------------------------------------------------------
---------
 Nested Loop  (cost=1.11..207.95 rows=26 width=20)
   ->  Hash Join  (cost=1.11..25.40 rows=26 width=20)
         Hash Cond: (h.bid = b.bid)
         ->  Seq Scan on history h  (cost=0.00..20.20 rows=1020 width=20)
         ->  Hash  (cost=1.05..1.05 rows=5 width=4)
               ->  Seq Scan on branches b  (cost=0.00..1.05 rows=5 width=4)
   ->  Index Scan using accounts_pkey on accounts a  (cost=0.00..7.01 rows=1
       width=4)
         Index Cond: (h.aid = a.aid)
(8 rows)
```

The same hints that were applied to this join at the end of Section 6.3 can be applied to the view as follows:

```
EXPLAIN SELECT /*+ USE_MERGE(tx.h tx.b) USE_HASH(tx.a) */ * FROM tx;

                                        QUERY PLAN

-----------------------------------------------------------------------------
-------------
-
 Merge Join  (cost=23480.11..23485.60 rows=26 width=20)
   Merge Cond: (h.bid = b.bid)
   ->  Sort  (cost=23479.00..23481.55 rows=1020 width=20)
         Sort Key: h.bid
```



```
         ->  Hash Join  (cost=21421.98..23428.03 rows=1020 width=20)
               Hash Cond: (h.aid = a.aid)
               ->  Seq Scan on history h  (cost=0.00..20.20 rows=1020
                   width=20)
               ->  Hash  (cost=13209.88..13209.88 rows=500488 width=4)
                     ->  Seq Scan on accounts a  (cost=0.00..13209.88
                         rows=500488 width=4)
   ->  Sort  (cost=1.11..1.12 rows=5 width=4)
         Sort Key: b.bid
         ->  Seq Scan on branches b  (cost=0.00..1.05 rows=5 width=4)
(12 rows)
```

In addition to applying hints to tables within stored views, hints can be applied to tables within subqueries as illustrated by the following example. In this query on the sample application `emp` table, employees and their managers are listed by joining the `emp` table with a subquery of the `emp` table identified by the alias, `b`.

```
SELECT a.empno, a.ename, b.empno "mgr empno", b.ename "mgr ename" FROM emp a,
(SELECT * FROM emp) b WHERE a.mgr = b.empno;

empno | ename  | mgr empno | mgr ename
-------+--------+-----------+-----------
  7902 | FORD   |      7566 | JONES
  7788 | SCOTT  |      7566 | JONES
  7521 | WARD   |      7698 | BLAKE
  7844 | TURNER |      7698 | BLAKE
  7654 | MARTIN |      7698 | BLAKE
  7900 | JAMES  |      7698 | BLAKE
  7499 | ALLEN  |      7698 | BLAKE
  7934 | MILLER |      7782 | CLARK
  7876 | ADAMS  |      7788 | SCOTT
  7782 | CLARK  |      7839 | KING
  7698 | BLAKE  |      7839 | KING
  7566 | JONES  |      7839 | KING
  7369 | SMITH  |      7902 | FORD
(13 rows)
```

The plan chosen by the query planner is shown below:

```
EXPLAIN SELECT a.empno, a.ename, b.empno "mgr empno", b.ename "mgr ename"
FROM emp a, (SELECT * FROM emp) b WHERE a.mgr = b.empno;

                            QUERY PLAN
-------------------------------------------------------------------
 Merge Join  (cost=2.81..3.08 rows=13 width=26)
   Merge Cond: (a.mgr = emp.empno)
   ->  Sort  (cost=1.41..1.44 rows=14 width=20)
         Sort Key: a.mgr
         ->  Seq Scan on emp a  (cost=0.00..1.14 rows=14 width=20)
   ->  Sort  (cost=1.41..1.44 rows=14 width=13)
         Sort Key: emp.empno
         ->  Seq Scan on emp  (cost=0.00..1.14 rows=14 width=13)
(8 rows)
```

A hint can be applied to the `emp` table within the subquery to perform an index scan on index, `emp_pk`, instead of a table scan. Note the difference in the query plans.

```
EXPLAIN SELECT /*+ INDEX(b.emp emp_pk) */ a.empno, a.ename, b.empno "mgr
empno", b.ename "mgr ename" FROM emp a, (SELECT * FROM emp) b WHERE a.mgr =
b.empno;

                                QUERY PLAN
---------------------------------------------------------------------------
 Merge Join  (cost=1.41..13.21 rows=13 width=26)
   Merge Cond: (a.mgr = emp.empno)
   ->  Sort  (cost=1.41..1.44 rows=14 width=20)
         Sort Key: a.mgr
         ->  Seq Scan on emp a  (cost=0.00..1.14 rows=14 width=20)
   ->  Index Scan using emp_pk on emp  (cost=0.00..12.46 rows=14 width=13)
(6 rows)
```

6.5 Conflicting Hints

This final section on hints deals with cases where two or more conflicting hints are given in a SQL command. In such cases, the hints that contradict each other are ignored. The following table lists hints that are contradictory to each other.

Table 3-4 Conflicting Hints

Hint	Conflicting Hint
`ALL_ROWS`	`FIRST_ROWS` - all formats
`FULL(`*table*`)`	`INDEX(`*table* `[` *index* `])`
`INDEX(`*table*`)`	`FULL(`*table*`)` `NO_INDEX(`*table*`)`
`INDEX(`*table index*`)`	`FULL(`*table*`)` `NO_INDEX(`*table index*`)`
`USE_HASH(`*table*`)`	`NO_USE_HASH(`*table*`)`
`USE_MERGE(`*table*`)`	`NO_USE_MERGE(`*table*`)`
`USE_NL(`*table*`)`	`NO_USE_NL(`*table*`)`

