
# EDB Postgres Advanced Server 9.5.Infor (9.5.75.1)
# Support for Infor LN 10.5+

**Release Notes**

**July 1, 2016**

**EDB Postgres Advanced Server, Version 9.5.Infor (9.5.75.1) Release Notes**
**by EnterpriseDB Corporation**


EnterpriseDB Corporation, 34 Crosby Drive Suite 100, Bedford, MA 01730, USA
**T** +1 781 357 3390 **F** +1 978 589 5701 **E** info@enterprisedb.com **www**.enterprisedb.com

# Table of Contents

Introduction

EDB Postgres Advanced Server v9.5Infor Features

- Shared Query Plan Cache
  - Overview
  - Cache Invalidation and Embargo
  - Configuration Parameters (GUCs)
    - edb_enable_shared_plan_cache
    - edb_shared_plan_cache_size
    - edb_shared_plan_cache_entries
    - edb_shared_plan_cache_expire_time
    - edb_shared_plan_cache_embargo
    - edb_shared_plan_cache_partitions

Installers for

Documentation

Platform Support and System Requirements

How to Report Problems

# 1 Introduction

EDB Postgres Advanced Server 9.5.Infor is a custom build of Advanced Server that introduces some new features to support the Infor LN 10.5 product. The versioning of the product start with 9.5.75.1 and will increment the fourth digit for minor releases and patch updates.

Other Infor products such as M3, Lawson & LSF are certified for use with the standard EDB Postgres Advanced Server 9.5 product.

Users of EDB Postgres Advanced Server v9.5.Infor should generally follow the standard EDB Postgres Advanced Server v9.5 documentation set except for additional information specified in these release notes.

# 2 EDB Postgres Advanced Server v9.5.Infor Features

The major features of this release are:

- Shared Query Plan Cache - Improve performance on high concurrency workloads where the same queries are repeatedley prepared by several concurrent users (RM37481)
    - Enables a reduction in the amount of time required to parse and plan statements prepared by several concurrent users.
    - See section 2.1 of these Release Notes for more details.
- Performance improvements in EDBLoader (RM37486)
    - Optimized several subsystems, mostly related to memory management and logging code, to improve performance on multi-terabyte data loads.
- Planner performance improvements (RM37479)
    - Reduces the performance penalty of planning queries against indexes with many dead tuples. Instead of finding the maximum (or minimum) value that is not known-dead, we now find the maximum (or minimum) value in the index.
- Protocol level cursor optimizations in OCL (RM37480)
    - Using the OCL driver, Advanced Server is now compatible with behavior in Oracle such that when a protocol-level cursor is closed, the locks held by that cursor are released.
- Suppress “duplicate key value” error from database server log (RM37483)
    - Infor applications are written in such a way that they will try to INSERT rows that violate constraints and then deal with failures. For example, an application may add row that violates a UNIQUE constraint, trap the

resulting error, and then UPDATE the existing row instead.
    - As a result of this activity, the database server log gets filled up with lots of error messages that they would like to omit. This enhancement is implemented as plugin that uses the server log hook to omit this error from the database sever log.
- Multi-host connect strongs in lippq to aid in failover (RM 37485)
    - When a libpq client attempts to connect to a server, libpq should attempt to connect to each host listed in the connection string until it finds a host willing to offer the service types (read-only or read/write) indicated in the connection string.
    - The format of a connection string that supports multiple hosts (and an optional service type) is:
      `postgresql:<host>[:<port>][,...][?readonly={0|1}]`
    - For example:
      `postgresql://localhost:5444,backupHost1:5444,backupHost2:5000/fp6bkit?readonly=0`
        - In the above example,
            - libpq will first attempt to connect to database fp6bkit on localhost:5444;
            - If that server is not answering (or is a standby), libpq then attempts to connect to backupHost1:5444.
            - If that fails, libpq attempts to connect to backupHost2:5000.
            - If no server is providing the requested service type, the connection attempt fails.
    - To use this feature in an HA group, each client should list all hosts (the master and all standbys) in the libpq connection string and specify "readonly=0". With such a connection string, libpq will only connect to the single master (because readonly=0); if the master dies, each client must try to reconnect (using the same connection string) and will eventually connecting to the newly promoted master.

## *2.1 Shared Query Plan Cache*

The shared query plan cache improves performance on high concurrency workloads where the same queries are repeatedley prepared by several concurrent users.

- The cache stores the parse tree and execution plan for prepared statements.
- This cache is shared by all processes so a parse tree/plan generated by one backend process may be used by a different backend process.
- The cache can only store plans for prepared statements
    - The prepared statement may be created by a PREPARE statement or by a client application when using a protocol-level prepare/execute API (such

as PQprepare()/PQexecPrepared()).

- The shared plan cache will not cache statements prepared by a procedural language.

### 2.1.1 Overview

When a statement is prepared, the server searches for a match in the shared query plan cache. A given statement will match a cache entry if the text of the SQL statement and a series of key configuration properties are identical.

If a match is found, the backend copies the parse tree and execution plan from the cache into the local prepared statement cache maintained by the backend process.

If no match is found, the backend prepares a local copy of the statement (that is, it generates the parse tree and execution plan), stores the parse tree and execution plan into the local prepared statement cache, and then adds the parse tree and plan to the shared plan cache.

It is important to understand that two statements match only if each of the above key components are identical. For example, the following 4 statements do not match:

```
SELECT * FROM customer WHERE balance = 100;
SELECT * from customer WHERE balance = 100;
SELECT /* comment goes here */ * FROM customer WHERE balance = 100;
SELECT * FROM customer WHERE balance = 101;
```

Also, two statements with identical SQL text will only match if prepared with the same search_path.

### 2.1.2 Cache Invalidation and Embargo

Individual cache entries may be invalidated because of auto-vacuum. If auto-vacuum changes the data distribution statistics for a given table, any cache entry that refers to that table will be removed from the shared plan cache.

The entire cache is emptied when any session executes a DDL (ALTER, CREATE, or DROP) command. After the cache is emptied, it is also disabled (or embargoed) for a short tunable period of time to prevent lock contention.

The duration of the embargo is controlled by the edb_shared_plan_cache_embargo configuration parameter and is specified as a number of seconds. The default value for this parameter is 60 (meaning that the cache is embargoed for at least 60 seconds after invalidation). If a session executes another DDL statement while the cache is embargoed, the embargo duration is reset to edb_shared_plan_cache_embargo seconds. In other

words, if edb_shared_plan_cache_embargo is set to 60 seconds and you execute a DDL statement, wait 30 seconds and then execute another DDL statement, the cache is embargoed for 90 seconds.

To understand the need for the embargo, consider the following scenario:

- A cache that contains many entries is invalidated due to a sequence of commands to DROP, CREATE and load data into tables being executed in session S1.
- Other sessions (S2, S3, S4, etc) are preparing statements, generating parse trees and execution plans, and trying to populate the cache.
- Meanwhile, session S1 continues issuing DDL commands, and invalidates the cache once again.

The tunable embargo time period allows the DBA to minimize the amount of cache repopulation and lock contention that is observed as a series of DDL commands is performed.

### 2.1.3 Configuration Parameters (GUCs)

The following configuration parameters control the shared query plan cache. More details on how to set and use configuration parameters in general can be found EDB Postgres Enterprise Edition Guide Chapter 2.1 - Configuration Parameters

#### 2.1.3.1 edb_enable_shared_plan_cache

A boolean value; when true, the cache is enabled (although it may be embargoed); when false, the cache is disabled
Default: true

#### 2.1.3.2 edb_shared_plan_cache_size

An integer value; determines the number of kilobytes to allocate for the shared plan cache
Default: 64MB

#### 2.1.3.3 edb_shared_plan_cache_entries

An integer value; determines the maximum number of entries that may be placed in the cache
Default: 8192

#### 2.1.3.4 edb_shared_plan_cache_expire_time

An integer value; determines the minimum lifetime (in seconds) of an entry in the cache. When a session tries to add a prepared statement to the cache and finds that the cache is full, that session will remove any entries that were added to the cache more than edb_shared_plan_cache_expire_time seconds ago. Keep in mind that all entries are

removed from the cache when a DDL statement is executed (regardless of the value of edb_shared_plan_cache_expire_time).
Default: 300 seconds (five minutes)

#### 2.1.3.5 edb_shared_plan_cache_embargo

An integer value; determines how long the cache is disabled after each DDL statement (See 2.1.2 Cache Invalidation for more information).
Default: 60 seconds

#### 2.1.3.6 edb_shared_plan_cache_partitions

An integer value; determines the number of partitions in the shared plan cache. By default, the shared plan cache is divided into 16 partitions in order to reduce lock contention (a single read/write lock controls access to each cache partition). You may increase this value for very large caches or if profiling shows contention waiting for locks in the shared plan cache.
Default: 16 partitions

# 3 Installing and Using EDB Postgres Advanced Server 9.5.Infor

EDB Postgres Advanced Server v9.5.Infor is packaged and delivered as a series of Interactive installers available via the EDB Support portal.

After your initial installation, please make sure you always update to the latest 9.5.75.y minor release to stay up to date with important critical patches. (Users of 9.5.Infor will be registered with EDB to ensure they are alerted to patches to the 9.5.Infor / 9.5.75.x product line.)

# 4 Documentation

Users of EDB Postgres Advanced Server v9.5.Infor should generally follow the standard EDB Postgres Advanced Server v9.5 documentation set except for additional information specified in these release notes.

http://www.enterprisedb.com/documentation

Please note that subscription holders can also access PDF versions of the documentation by logging into the EnterpriseDB website, and visiting the customer portal at:

http://www.enterprisedb.com/support

# 5 Platform Support and System Requirements

EDB Postgres Advanced Server v9.5.Infor supports 64 bit Linux platforms. This includes the following:

Interactive Installers:
RHEL / CentOS / OEL 6, 7
Ubuntu 14.04, Debian 7.6, SLES 11, 12

# 6 How to Report Problems

To report any issues you are having please contact EnterpriseDB's technical support staff:

- Email: support@enterprisedb.com
- Phone: +1-732-331-1320 or 1-800-235-5891 (US Only)