Infor Enterprise Server Technical Reference Guide for Enterprise DB Database Driver

Release 10.6.x

Publication Information

Release: Infor LN 10.6.x
Publication Date: March 26, 2018
Document code: ln_10.6.x_lndbedbdrvtrg__en-us

Contents

About this guide

This document describes the database driver that forms the interface between the LN application server layer and the EnterpriseDB database server. In this document, the database driver is referred to as the LN EnterpriseDB driver.

The information provided in this document applies to all EnterpriseDB versions. Exceptions are explicitly stated.

For previous Infor LN or Baan porting sets, see a previous version of this document.

Intended audience

This document is intended for anyone who wants to configure or customize the Infor Enterprise Server database driver for EnterpriseDB. This document assumes that you have operating system and database knowledge.

Related documents

You can find the documents in the product documentation section of the Infor Xtreme Support portal, at http://www.infor.com/inforxtreme.

For information on the installation procedure for EnterpriseDB PPAS and Infor LN software, see these documents:

- *Infor LN - Installation Guide*
- *Infor Enterprise Server - Technical Manual*
- *Infor LN - Performance, Tracing and Tuning Guide*

Contacting Infor

If you have questions about Infor products, go to the Infor Xtreme Support portal.

If we update this document after the product release, we will post the new version on this website. We recommend that you check this website periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1: LN Database Driver Overview

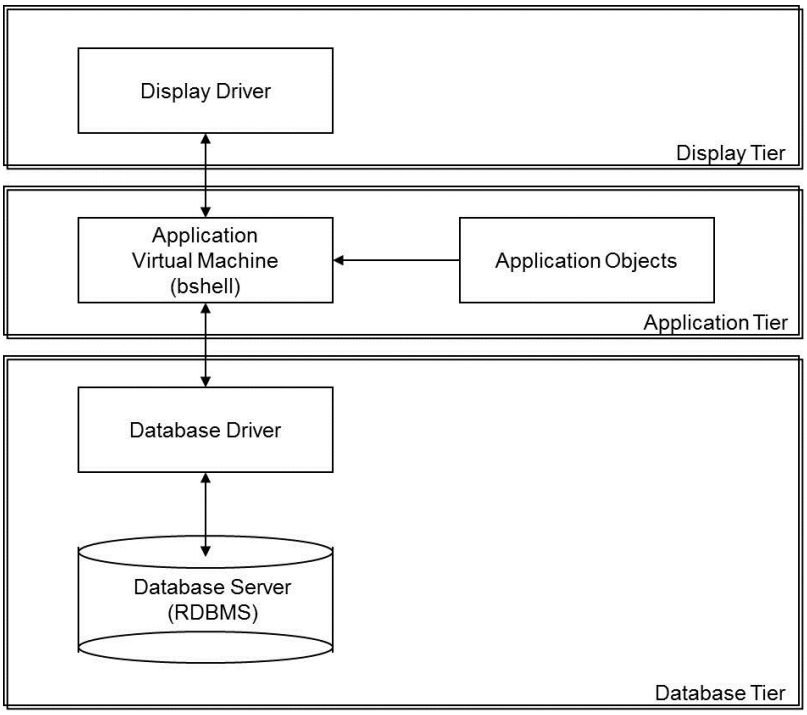
The database driver is an important part of Infor's commitment to an open-systems client/server architecture. Because the LN architecture includes the LN software and a third-party relational database management system (RDBMS), the driver provides an interface between the LN software and the various RDBMS products. The database driver allows the majority of the LN processing to be independent from the RDBMS.

LN architecture

LN supports a three-tier architecture that consists of a display tier, an application tier, and a database tier. The display tier provides presentation services for user interaction. The application tier consists of the LN application virtual machine and the application objects. The database tier includes the LN database driver and a third-party RDBMS product that acts as the database server. The following figure shows the LN architecture.

The emphasis of this document is the LN database driver, which is the interface between the LN applications and the RDBMS server. The database driver translates database requests from the LN application virtual machine to RDBMS-specific SQL requests that the driver sends to the database server. After the database server retrieves the requested information, the database driver then passes the data back to the LN application virtual machine.

To put the functions of the database driver into perspective, the following figure shows the three tiers of the total LN architecture:



Display tier

The display tier consists of the display driver that includes the LN user interface (UI). The display driver facilitates the communication between the user and the application tier. Data input from the user through the UI is relayed to the LN application virtual machine. The display driver displays data returned from the LN application virtual machine in graphical form.

Application tier

The application tier includes the application objects and the LN application virtual machine; together, the application objects and the application virtual machine provide much of the functionality of LN.

The application objects include the compiled LN applications and the data dictionary. The LN applications provide the functionality needed to implement the LN Enterprise Resource Planning (ERP) system.

These applications are written in Baan 3GL or Baan 4GL programming languages supported by the LN Tools package.

The data dictionary defines the data models used by the applications; the data dictionary includes information about the domains, schemas, and referential integrity rules used by LN.

The LN application virtual machine schedules and runs the application objects, sends and receives information to and from the display server, and initiates an instance of the database driver necessary for communication with the database server. A running database driver can support multiple connections

to a single RDBMS instance. If an LN installation stores data tables in multiple RDBMS products or instances, the application virtual machine must start one instance of the database driver for each RDBMS product or RDBMS instance with which it must communicate.

The LN application virtual machine is referred to as the LN shell or the bshell. Throughout this document, the virtual machine is referred to as the LN application virtual machine or the application virtual machine.

Database tier

The database tier consists of the LN database driver and the database server. The database driver provides a common interface between the LN application virtual machine and the database server. Communication between the application virtual machine and the database driver is the same, no matter which RDBMS product you use as the database server.

One database driver exists for each of the RDBMS products that LN supports.

Communication between the database driver and the database server is tailored to the RDBMS you use. The database driver communicates with the RDBMS through structured query language (SQL) statements and the native application programming interface (API) of the RDBMS.

The database server consists of one of these third-party RDBMS products:

- EnterpriseDB
- Oracle
- DB2
- Microsoft SQL Server

All LN application data is stored in a relational database managed by an RDBMS. You can have multiple RDBMS products in one LN installation, with some data in one database server and other data in another.

Data flow through the LN architecture

Note that the database driver provides an interface between the LN application virtual machine and the specific RDBMS server you use. The remainder of this section describes the flow of data through LN.

If a user performs an operation at a GUI workstation, the display server interprets the input and sends the information to the LN application virtual machine. Based on the information the virtual machine receives, the application virtual machine runs the appropriate application object.

If a running application object requires information stored in the database, the application virtual machine sends the request to the database driver. Data requests from the client applications are RDBMS-independent and made using LN SQL, which is an RDBMS-independent SQL language.

If the application virtual machine runs a database query from an application object, the virtual machine first determines whether a running database driver is available to process the query. If no database

driver is running, or if the running database driver instances are communicating with a database server other than the one storing the needed data, the application virtual machine starts a new instance of the database driver. The application virtual machine parses the LN SQL database query the virtual machine receives from the application object and sends an internal representation of the query to the database driver. The internal representation of the query the database driver receives is still RDBMS-independent.

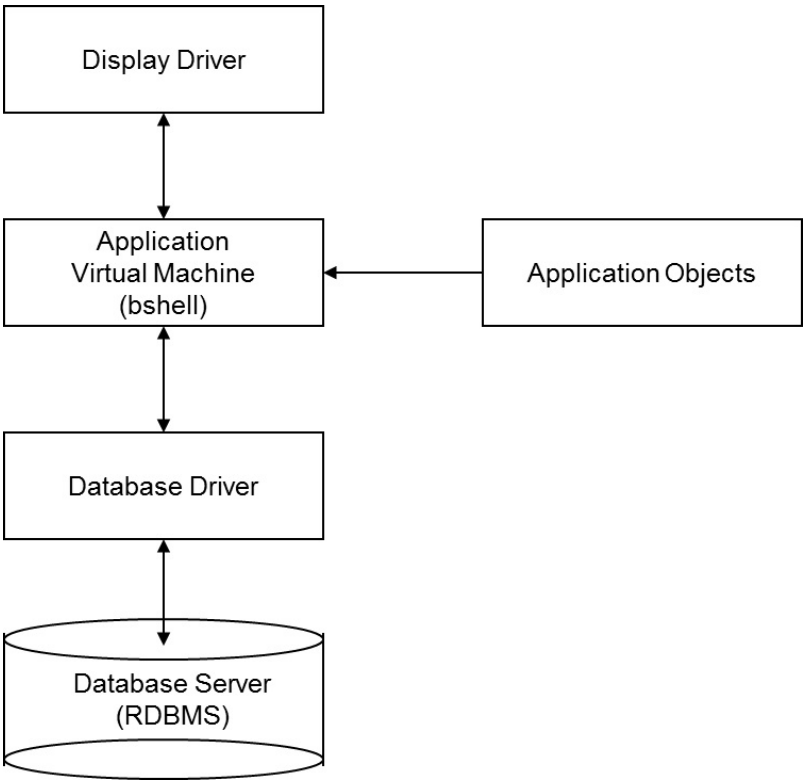
The database driver uses the SQL statements compatible with the specific RDBMS being used to translate the database query into an appropriate query. Each database driver takes advantage of the design of the particular RDBMS that the driver supports, so the resulting SQL statements are valid for the RDBMS and provide the best possible performance. The RDBMS specific SQL statements are then submitted to the RDBMS server, which processes the data request.

After the RDBMS processes the query, the RDBMS returns the data to the database driver. The database driver catches and handles any error conditions. The database driver then returns the data and status information to the application virtual machine, and then the driver provides the information to the application that requested the information. The application virtual machine can also send a message to the display server, which displays an appropriate message on the user's workstation.

LN hardware configurations

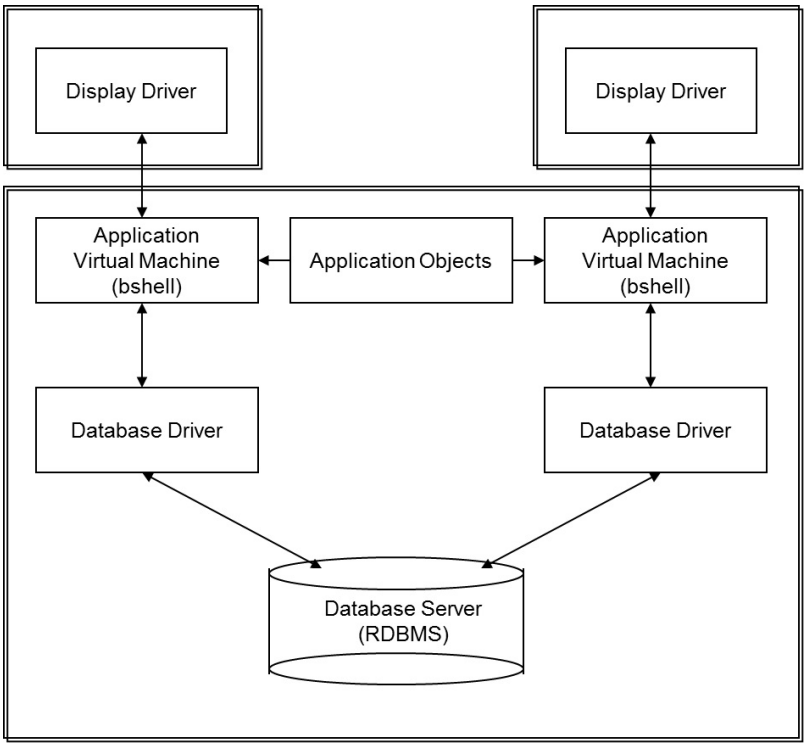
Several hardware configurations are supported for an LN implementation. These configurations include standalone mode and many variations of client/server mode. Available hardware, data storage requirements, and performance expectations determine the most appropriate hardware configuration.

Standalone mode refers to a configuration where all components of the LN architecture run on a single machine. In standalone mode, an end user can work from the host machine or from a thin client machine, such as an X-Terminal that runs BI. The following figure shows the standalone-mode configuration:



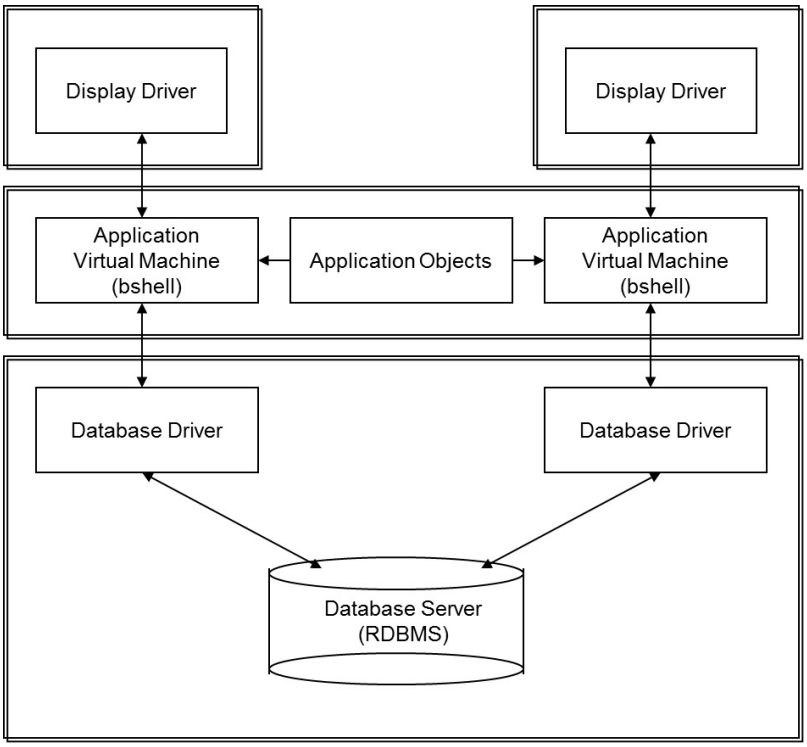
In a client/server configuration, the components of the LN architecture are distributed across two or more machines. Various client/server configurations exist; this section describes the most common configurations.

The simplest client/server configuration is a variation of standalone mode; in this configuration, the application tier, database driver, and RDBMS are on one machine, while the display drivers are distributed among the user workstations. An instance of the application virtual machine and at least one instance of the database driver starts for each user. All users have access to the same application objects and database servers; the following figure shows this configuration:

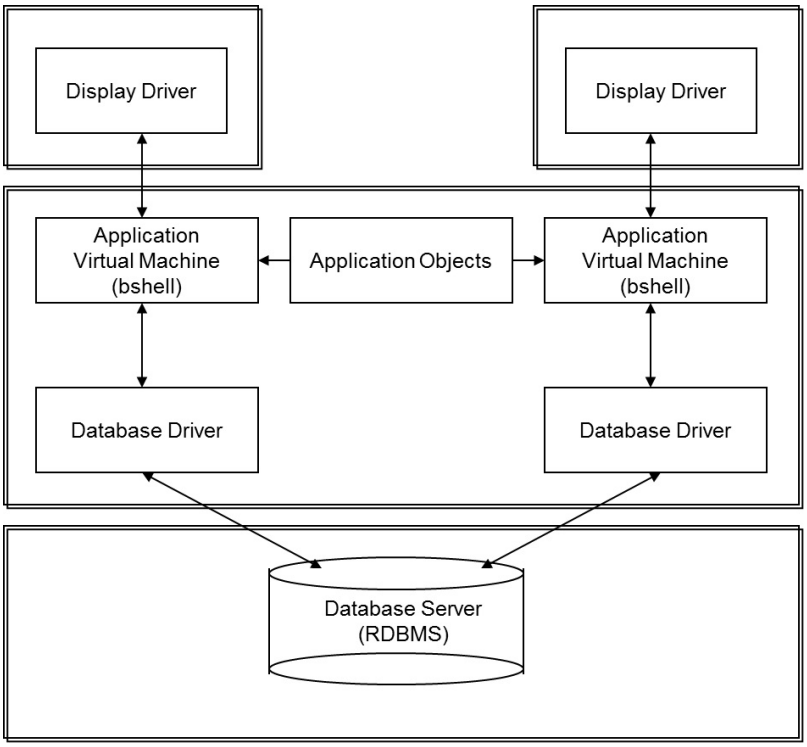


If two machines are available as servers, two configurations are commonly used. In both configurations, the display drivers reside on the user workstations. In the first configuration, the application tier is placed on one server, while the database driver and the database server are placed on another. As with the previous configuration, an instance of the application virtual machine and at least one instance of the database driver starts for each user.

All users have access to the same application objects and database servers; the following figure shows this client/server configuration. This configuration uses the LN method of client/server access between the application virtual machine and the database server.



An alternative configuration with two servers is to place the applications and the database driver on one server and the database server on another. End user workstations are linked to the machine with the application virtual machine. An instance of the application virtual machine and at least one instance of the database driver is started for each user. All users have access to the same application objects and database servers; the following figure shows this client/server configuration. This configuration uses the RDBMS's ability to provide client/server access.



You can also use various other configurations of client/server systems, such as dividing the application logic among multiple servers or use multiple servers for distributing the database.

Chapter 2: LN Database Organization

All application data used by LN is stored in database tables in the RDBMS. To keep the majority of the LN processing independent of the RDBMS, LN uses a data dictionary. The data dictionary includes domain, schema, and referential integrity information stored in a database-independent manner.

Many tables are required and a convention is available to name:

- Tables
- Columns in tables
- Indices to data in the tables.

We will describe the data dictionary and the naming conventions LN database drivers use to access data stored in the RDBMS. We also discuss how LN data types are mapped to EnterpriseDB data types.

LN data dictionary

A data dictionary is a catalog that provides information about the data in a database. You can think of a data dictionary as “data about data,” or metadata. You can use a data dictionary to find data that resides in a database table.

The LN database drivers maintain a data dictionary because the data that LN applications use can differ from the database tables defined in the RDBMS. The LN data dictionary maps LN data types, domains, schemas, and referential integrity information to the appropriate information in the RDBMS. To store or retrieve data in the RDBMS, the database driver maps data dictionary information to database table definitions.

LN data dictionary information can be stored in shared memory, where the information will be available to all running LN application virtual machines. The data dictionary information is shared among all the open sessions in a single database driver.

The database driver cannot directly use the LN data to create EnterpriseDB tables because not all LN data types exactly match EnterpriseDB data types. To create valid EnterpriseDB tables, the driver must perform some mapping or translation. To map the LN data dictionary with tables in EnterpriseDB, conventions are used for the table names, column names, and index names.

Table naming convention

The table name of an LN table stored in EnterpriseDB has this format.

```
t<DD Table name><Company number>
```

The explanation of the table name:

- DD Table Name

  The name of the table used in the data dictionary, which consists of a package designation, a module designation, and a table number, as follows:

  ```
  <DD Table name> = <package><module><table number>
  ```

  - Package: A two-letter code that refers to the LN package that created the table. For example, a table created by the LN Tools package has the package code `tt`.
  - Module: A three-letter code that refers to the LN module in a package that created the table. For example, a table that the LN Distribution Sales module creates has the module code `sls`.
  - Table number: A three-digit code that refers to a table that belongs to an LN module.
- Company Number:

  Used in LN to differentiate areas of functionality. A company must exist with the number 000. Additionally, several other company numbers can exist.

  For example, the data dictionary table `ttadv999` with company number `000` is created in EnterpriseDB as `tttadv999000`.

  Note:
  - For tables with the Multi language Application Data feature enabled, a secondary table exists, named as:

    ```
    s<DD Table name><Company number>
    ```
  - For tables with the Document Authorization (DBCM) feature enabled, another secondary table exists, named as:

    ```
    v<DD Table name><Company number>
    ```

Column naming convention

Each column in the LN data dictionary corresponds to one or more columns in an EnterpriseDB table.

The rules for column names are:

- General:

  If you create an LN column name in EnterpriseDB, the column name is preceded by the string `t$`. For example, the LN column with the name `cpac` is created in EnterpriseDB with the name `t$cpac`. To avoid reserved words, you can precede column names with `t$`. If a column name contains a period, the period is replaced by a dollar sign.

- When the Multi language Application Data feature is enabled, column names preceded by the string `t$$` are generated by the database driver.
- Long string columns:

  LN columns of type String can exceed the maximum length of character columns in a RDBMS.

  The maximum length in EnterpriseDB PPAS is at least 1GB. Therefor an LN column will never be split.

- Array columns:

  In the LN data dictionary, you can define array columns; an array column is a column with multiple elements in the column. The number of elements is called the depth. For example, you can define a column that contains a date as an array of three elements: a day, a month, and a year. In EnterpriseDB, the three elements of the array column are placed in separate columns. The names of these columns include a suffix with the element number. For example, an array column `date` becomes:

  - `t$date$1`: Element 1
  - `t$date$2`: Element 2
  - `t$date$3`: Element 3

- Array compression:

  The maximum number of EnterpriseDB PPAS columns is limited. If the number of LN columns exceeds the maximum number of EnterpriseDB PPAS columns, the database driver tries to compress (join) array columns to reduce the number of columns. All array elements of one array column are stored as one column in the EnterpriseDB PPAS database with the elements concatenated in binary format. To start, the driver compresses the array column that yields the highest number of columns. The driver continues to compress array columns until the number of columns is less than the maximum number of columns.

  The name of the compressed column in EnterpriseDB follows the same convention used for the other columns, such as:

  - `t$array`: Contains all elements of the compressed column.

Index naming convention

LN indexes are identified by a sequence number for each table, with the sequence numbers beginning with one. Each table has at least one index: the primary index. For each user, EnterpriseDB PPAS requires that all index names be unique; therefore, the table name and the index number are included in the index name.

Index names have this format:

```
t<DD Table Name><Company Number>$idx<Index Number>
```

For example, the index name for an LN table with name `ttadv999`, index number `1`, company number `000` is `tttadv999000$idx1`.

If an LN index is defined as a unique index, the EnterpriseDB PPAS index is created with the UNIQUE clause. Without the UNIQUE clause, duplicate indexes are created.

The EnterpriseDB driver uses the index name conventions to create and delete indexes. Index names allow you to generate index hints to help the EnterpriseDB PPAS cost-based optimizer (CBO) to choose the appropriate execution plan.

Note:

- Secondary tables, which are used by the Document Authorization feature, have the same indexes as the primary tables, but the index names have this format:

```
v<DD Table Name><Company Number>$idx<Index Number>
```

- When the Multilanguage Application Data feature is enabled for a table or a table has BLOB columns, this additional index is created:

```
[s|t]<DD Table Name><Company Number>$UUID
```

Data type mapping

This table shows the mapping between LN data types and their EnterpriseDB PPAS counterparts:

Mapping between LN and EnterpriseDB PPAS data types	
LN data type	**EnterpriseDB PPAS data type**
Byte	BIGINT
Enumerated	BIGINT
Integer	BIGINT
Long	BIGINT
UTC Date/Time	DATE (1)
Text	BIGINT
(Bit)Set	BIGINT
Float	DOUBLE PRECISION
Double	DOUBLE PRECISION
String	VARCHAR(n)
TSS string	VARCHAR(n)
Date	DATE (1)
Raw	RAW
Blob	BLOB
(2)	RAW

1. The empty date and utc timestamp is represented in EnterpriseDB PPAS as January 1, 4712 B.C 00:00:00.
2. The EnterpriseDB PPAS RAW type is used to store compressed array columns and UUIDs generated if the Multilanguage Application Data feature is used or if a table has one or more BLOB columns.

Additional constraints

Besides these naming conventions and data types, when you map LN data to EnterpriseDB PPAS data, these rules apply:

- All names the database driver generates are in lowercase characters and are not enclosed in double quotes; therefore, when storing the names in the EnterpriseDB PPAS dictionary, EnterpriseDB PPAS converts these names to uppercase. When EnterpriseDB PPAS retrieves names from the EnterpriseDB PPAS dictionary, the names display in uppercase.
- All columns that the LN EnterpriseDB driver creates have the NOT NULL constraint. LN applications do not support NULLS.

Chapter 3: Database Driver Internal Processing

The LN EnterpriseDB database driver converts RDBMS-independent database requests into requests specifically designed for EnterpriseDB. Some of the internal processing that occurs in the LN EnterpriseDB database driver is described.

These topics are discussed:

- Data integrity
  The features that ensure data integrity.
- Database driver SQL processing.
  The internal processing of an SQL statement in the driver.
- To set driver behavior
  The mechanisms that enable you to modify the default behavior of the database driver

Data integrity

Several features of the LN database driver help ensure data integrity; these features include locking mechanisms, methods used for ensuring referential integrity, and methods used for distributed databases. Additionally, to maintain data integrity while you minimize network traffic, you can use data buffering techniques.

This section provides an overview of the features that the LN EnterpriseDB database driver uses to ensure referential integrity to work with distributed databases, and to apply data buffering techniques. Locking strategies are described in detail in Database Driver Configuration and Tuning on page 35.

Referential integrity

Referential integrity preserves the defined relationships between tables when you maintain records. The LN database driver has a built-in mechanism to preserve referential integrity. The database driver does not depend on the underlying RDBMS to maintain referential integrity.

Distributed databases

An SQL query can require information from multiple tables. When tables are distributed, the tables can be physically located in separate databases. To access data from distributed tables, the query is split into multiple queries, with one query for each database. To service each query, the application virtual machine activates multiple database drivers. One active database driver is called upon to retrieve the data required from a single database. When all of the data the original query requires is gathered from the databases, the database driver joins the data and returns the data to the application virtual machine.

Data buffering

The application virtual machine can buffer and flush updates at the time of transaction commit, or, if required, earlier; this reduces the number of network round trips and data volumes.

If multiple rows are returned from a query, the rows are buffered and then sent back to the application virtual machine as one block. To minimize the amount of data transferred between the application virtual machine and the database driver, data reduction and compression is applied.

Database driver SQL processing

As described in LN Database Driver Overview on page 8, the application virtual machine sends RDBMS-independent database queries and update requests to the database driver. The database driver converts these RDBMS-independent database requests into SQL statements appropriate to the specific RDBMS being used. This section describes the SQL processing the LN EnterpriseDB database driver performs.

Because the LN database driver uses EnterpriseDB Call Interface (OCI) connector to communicate with EnterpriseDB PPAS.

Oracle Call Interface (OCI)

OCI is an application programming interface (API) that enables you to communicate with the database server. An OCI consists of a function library that you can call from an application program to run SQL statements and communicate with the data source.

The OCI functions in a way to let the LN EnterpriseDB database driver calls perform these actions:

- Connects to EnterpriseDB PPAS (opens the session).
- Allocates cursors.
- Parses an SQL statement.
- Binds input variables.
- Defines output variables.

- Runs an SQL statement.
- Fetches the resulting rows.
- Commits or aborts a transaction.
- Closes, unbinds, and drops a cursor.
- Disconnects from EnterpriseDB PPAS (closes the session).

The LN EnterpriseDB driver also uses these features of OCI:

- Array fetches (if enabled).
- Array inserts (if enabled and if possible).

SQL processing

The database-dependent layer of the LN EnterpriseDB database driver dynamically generates SQL statements. Because LN applications are dynamic, you cannot know in advance which tables will be used at run time; therefore, you cannot prepare the queries before run time.

In the LN EnterpriseDB database driver, the procedure to process SQL consists of several steps; these steps are described in this section.

If the LN EnterpriseDB driver receives a query from the application virtual machine, the query is translated into a format suitable for EnterpriseDB PPAS . To transfer the query to EnterpriseDB PPAS , you must use OCI function calls. In the LN EnterpriseDB database driver, you can allocate a cursor, and assign the query to the cursor. You must then parse the SQL statement, bind the input and output variables, and use the cursor to run the query. After you run the query, you perform a fetch operation and place the resulting column values in the bound output variables. The rows that EnterpriseDB PPAS returns are passed to the database independent layer of the LN EnterpriseDB database driver, which sends the results back to the application virtual machine.

If you must re-execute a statement, the cursor from the previous execution closes and the result rows are discarded, whether the re-execution is with the same input parameters or not. If new input values are required, the new values are assigned to the input parameter columns, and the query is re-executed.

For re-execution, no reparse of the statement or re-bind of input and output parameters is required, which improves the overall performance.

If array fetching is enabled, multiple rows are fetched in one call to the driver. To buffer multiple rows fetched in one operation, space is allocated in the driver. You can fetch multiple rows to the buffer, and, when requested, the rows are returned to the application virtual machine. If no rows are left in the buffer and more rows are requested, another array fetch operation is carried out.

You can also buffer inserts. If array inserting is enabled, the driver places the rows to be inserted in a buffer. If the buffer is full, or if necessitated by some other event, the rows are flushed to EnterpriseDB PPAS. The rows in the buffer are inserted with a multirow insert.

Note: To manually place data into the database, you can use the LN utility `bdbpost6.2`; you can use this utility to place data into a new database table or to append data to an existing database table. If you use bdbpost6.2, you can set particular options. For more information, see the *Infor Enterprise Server - Technical Manual*. If you use bdbpost6.2, by default the rows are buffered and flushed when the array buffer is full. You must specify the array size; otherwise, no buffering is carried out. To specify

the array buffer size in the tabledef file on a per-table basis or globally, you can use an environment variable. The following sections describe environment variables, resource variables, and storage files. The section on environment variables briefly describes the tabledef file.

Setting driver behavior

To configure the LN EnterpriseDB database driver, several facilities are available, with the most common being through driver resources; two other facilities are environment variables and the storage file. The driver resources and environment variables are described in more detail in Database Driver Resources and Environment Variable on page 45 and the storage file in Storage Parameter File Format and Driver Configuration Options on page 63.

Driver resources

The driver resources are parameters you can set to modify the behavior of the LN EnterpriseDB database driver; you set these parameters in a file called the resource file (`db_resource`). One resource file is available for all database drivers that run in an LN environment, where you can also find resources for all the database driver types. When you first invoke the driver, a database driver reads the parameters set in the resource file.

The resource file can contain many entries, with one entry for each line. Each entry is used to set a single resource parameter, with the resource name followed by a colon and then the value to which the resource is to be set. This is an example of the contents of a resource file that contains two entries:

```
pgclientpath:/opt/PostgresPlus/9.5AS

edb_lock_timeout:300
```

If you modify the behavior of the database driver, to take advantage of the characteristics of the database driver, you often must modify the behavior of the LN application virtual machine; therefore, two types of database driver resources are available: those you use to modify the behavior of the database driver, and those you use to modify the behavior of the application virtual machine. Driver resources you use to modify database driver behavior are called resources for the server. Driver resources you use to modify behavior in the application virtual machine are called resources for the client.

In a Windows environment, the resource file `db_resource` is located in the directory `%BSE%\lib\defaults`, where %BSE% refers to the directory on which the LN software environment is installed. In a UNIX environment, the resource file is located in the directory `$BSE/lib/defaults`, where $BSE refers to the directory on which the LN software environment is installed. If the database driver and the application virtual machine run on the same machine, only one `db_resource` file is created, which contains all the required resource parameters. If the database driver and the application virtual machine run on separate machines, one `db_resource` file must be located on the machine that runs the database driver that contains the server resources. And one `db_resource` file must be located on the machine that runs the application virtual machine that contains the client resources.

Besides the default resource file `db_resource`, to override resource values for specific users or groups of users, you can setup an alternative resource file. Specify the alternative resource file with the environment variables USR_DBS_RES and USR_DBC_RES. USR_DBS_RES specifies the path to a file that contains an alternative resource file for the server. You must set this file on the machine that runs the database driver. Use USR_DBC_RES to specify the path to a file that contains an alternative resource file for the client. You can set this file on the machine that runs the application virtual machine. Any driver resource set in the alternative resource file overrides the setting of the same driver resource in `db_resource`.

For more information about setting the database driver environment variables, see Environment variables on page 25.

Environment variables

To override driver resources, use environment variables. Usually, you configure a default set of resource parameters in the resource file; the administrator can override these default settings with environment variables.

Primarily, an environment variable corresponds to each resource parameter. The environment variable name is usually the uppercase equivalent of the resource parameter name.

As with the database driver resources, you can use some environment variables to modify the behavior of the database driver (server); you can use others to modify the behavior of the application virtual machine (client). If you must use a database driver environment variable for the server, you must set the database driver on the machine that runs the database driver to override the corresponding driver resource. If you must use a database driver environment variable for the client, you must set the variable on the machine that runs the application virtual machine to override the corresponding driver resource.

Server environment variables

To override the driver resources for all tables in a database, or for specific tables and company numbers in the database, use the environment variables that affect the database driver. You can set the database driver server environment variables in the following three ways:

- Use the LN **Database Definitions (ttaad4510m000)** and **Tables by Database (ttaad4111m000)** sessions.
- Manually modify the LN tabledef6.2 file.
- Use the standard operating system mechanism.

Note: Sessions that maintain the tabledef6.2 file will not preserve manually added changes.

To modify database driver behavior, Infor recommends that you use the LN **Database Definitions** session. If you must configure specific tables and companies for access with a specific database driver, ensure you use the **Tables by Database** session. These sessions cause environment variables for a particular database driver to override the defaults set in the resource file, and allow you to centrally maintain the environment variables.

The **Database Definitions** session maintains database driver configuration information in a file called tabledef6.2. This file is stored in the directory %BSE%\lib, which resides on the machine on which the database driver runs. Although Infor recommends that you use the **Database Definitions** session to maintain this file, advanced users can manually modify this file.

The format of the tabledef6.2 file is as follows:

```
<Table Name>:<Company Number>:<Driver Type>(<Environment Variable>= <Val►
ue>):<audit Y/N>
```

If you must specify multiple environment variables for a single table and company number, the variables are listed in the parentheses, separated by commas.

If you must specify all tables or all companies, you must use the asterisk (*) in place of a table name or company number. For example, you can make the following entry in the tabledef6.2 file:

```
tccom010:812:entdb(EDBPROF=0.4):N
```

In this example, all the queries on table tccom010812 that require at least 0.4 seconds are logged in the EDBPROF file. Note that this table is considered to have a different database definition from other tables. If an EnterpriseDB driver is already running, but accesses a different table, a separate driver starts for this table. Environment variables that appear in the driver specifications of the tabledef6.2 file are placed in the driver's environment before the variables are invoked, so the variables are available to the driver at startup.

If you must modify the default database driver resources for specific users, to set database driver environment variables for specific users, use the standard operating system. These environment variables override the settings created in the **Database Definitions** session for these users.

Client environment variables

To override the client resources that affect the application virtual machine, use database driver environment variables that affect the client. You must set these environment variables on the machine that runs the application virtual machine; they must be set using the standard operating system methods used for setting environment variables. Any client environment variables that are used override the equivalent resource variables set for the client in the db_resource file.

Storage parameter file

The storage parameter file allows you to specify the distribution of table and index data in various EnterpriseDB PPAS tablespaces.

To run DDL statements, such as a create table or create index statement, the database driver uses storage parameters.

A storage parameter file is defined for each database driver. The storage parameter file for the LN EnterpriseDB database driver is called edb_storage_param and is located in the UNIX directory

$BSE/lib/edb. For a detailed description of the format of the storage parameter file, see Storage Parameter File Format and Driver Configuration Options on page 63.

Chapter 4: Database Security

To maintain security, the LN EnterpriseDB database driver controls user access to the database and database objects. The LN database administrator (DBA) module enables the DBA to control access to the database using LN sessions. Using the DBA module makes DBA tasks easier and less prone to errors than directly using database driver tools. First is discussed how the LN EnterpriseDB database driver handles issues related to database security, and then the DBA module is described.

Database security

Database security consists of two aspects: object security and authentication. Object security refers to the process of determining whether a user who has access to the database is authorized to access particular database objects. Authentication refers to the process of determining whether a user is authorized to access the database. To ensure security, object security and authentication use the concept of groups. This section first describes the group concept, and then describes how the LN EnterpriseDB driver provides object security and authentication.

Groups

In any RDBMS, a group is defined as a collection of database users. All users assigned to a group are granted the same database privileges. After you define a group with a particular set of privileges, you can assign users to that group. Using groups simplifies the management of a large number of groups with common requirements.

An LN group consists of a database name and methods to provide object security and authentication in the database. The LN group name is the same name as the database that holds the LN data in the RDBMS. To provide object security and authentication, the LN group uses the mechanisms of the RDBMS.

An LN group is a superset of the usual RDBMS group, in that the group includes the RDBMS group and also the database name and an RDBMS login.

In EnterpriseDB PPAS , an LN group is made up of three components: a database schema, a login for authentication, and an EnterpriseDB role for object security. The EnterpriseDB database schema has the same name as the LN group. The login is the same name as the LN group and is assigned database owner (DBO) privileges in the database. Finally, an EnterpriseDB role is created, which becomes the

target for privileges granted on objects in the database. Users are associated with the EnterpriseDB role and, therefore, inherit the privileges granted to the EnterpriseDB role. The advantage of having a table assigned to a role is that the members of the group can share and operate on the same data in a single table.

For example, users **Maria** and **John** can both be assigned to LN group `erpdb`. Group `erpdb` owns the tables and grants select, insert, delete, and update privileges to the EnterpriseDB role. Therefore, users **Maria** and **John** inherit the select, insert, delete, and update privileges granted to the EnterpriseDB role, to access and manipulate LN group table data.

The LN user is shielded from the RDBMS groups. The database driver performs all the processing required to make use of the RDBMS groups. Only the database administrator must be concerned about the RDBMS groups and the LN DBA module that allows the administrator to easily maintain the RDBMS groups.

Object security

In EnterpriseDB, if a user creates an object such as a table, the user becomes the owner of the object, and only the owner can access the object. Other users can only access the object if they have been granted privileges to do so. In an LN environment, in which many users access the same tables in the EnterpriseDB PPAS database, a mechanism has been developed to allow multiple users to share these tables.

To allow various LN users to share the same EnterpriseDB table, you can use a group concept. An LN group maps users to a database in EnterpriseDB and ensures that members of the group have sufficient privileges to access data in the group's tables.

The LN EnterpriseDB driver uses an EnterpriseDB role to implement the LN group concept. Whenever the group user creates a new table, select, insert, delete, and update privileges are granted to the EnterpriseDB role. Any user associated with the role automatically inherits these privileges and can individually perform these operations on the group table.

If you add new users, you only must associate the users with the EnterpriseDB role. These users automatically inherit all privileges currently granted to the role without the need to grant privileges on every group object in the database to the user. If the user is dropped from the role, these privileges are revoked, and the user no longer has access to tables in that role. If you explicitly grant the user privileges to operate on the tables, if you drop the user from the role, you must also explicitly revoke these rights. If you grant privileges to the role, you can greatly reduce the overhead of adding users, which also provides flexibility and ease of maintenance.

In the DDL statements generated by the driver, object names are not qualified by the owner name. Ownership is determined by the session (group or user) in which the create table runs. If you create objects identified as belonging to the group, the user who creates the object must log onto EnterpriseDB as the group user; in this case, the group owns the table, and permissions are granted on the group to provide access to all group users.

Authentication

The database driver maps LN users to EnterpriseDB user accounts to allow these users to establish a connection to EnterpriseDB and access data. To prevent unauthorized users from accessing the database, non-mapped users cannot establish a connection to the database.

If you create a database, an administrator creates a login for the user and associates the user with a group in the database that has object privileges. The group user corresponds to the target database. The members that belong to this group inherit the group privileges and can establish a connection to the database either by unified login or by a valid password stored in encrypted form in the driver administration files.

To add or drop DBA from the EnterpriseDB role, use the LN Database Administration (DBA) module. Users authorized to access the database are registered in the LN driver administration files. The user name and password LN uses to log onto EnterpriseDB on behalf of the user are maintained in the UNIX file `$BSE/lib/edb/edb_users`.

You define all the LN users, their corresponding EnterpriseDB PPAS logon names and passwords, and the name of the group to which the names and passwords are assigned in the UNIX file $BSE/lib/edb/edb_users. The format of each entry in this file is:

```
<LN User>:<EDB User>:<Encrypted EDB User Password>:<LN Group Name>
```

The LN application virtual machine starts the LN EnterpriseDB driver on behalf of the user. From the UNIX file `$BSE/lib/edb/edb_users`, the driver identifies the EnterpriseDB user and the user's password and establishes the connection to EnterpriseDB.

The group logon procedure also includes a password, which is defined in the UNIX file `$BSE/lib/edb/edb_groups`. The format is:

```
<Group Name>:<Encrypted Group Password>
```

DBA module

The DBA module maintains the database administration files that the EnterpriseDB database driver uses. This module enables an administrator to register authorized users and give users access to data. To maintain the administration files the database driver requires at run time, a tool is provided with the LN EnterpriseDB database driver. The administration files are stored in the UNIX directory `$BSE/lib/edb`.

The DBA module implements the user and group administration functions for all LN database drivers. The EDB_MAINT utility is an executable program called by the DBA module that implements the functions required to make changes in EnterpriseDB. While you can call the EDB_MAINT utility from outside the DBA module, Infor does not recommend you do this because EDB_MAINT does not modify these users and groups' files.

The DBA module is described in more detail in the *Infor Enterprise Server Administrator's Guide (U8854 US)*.

Chapter 5: Database Driver Profiling and Statistics

The LN EnterpriseDB driver provides a facility for monitoring system performance. This facility includes:

- A profiling facility to gather timing information for SQL statements.
- A statistics facility to gather driver-wide statistics.
- A facility for debugging and troubleshooting issues.

Profiling

The database driver allows users to log timing aspects and statistics. This option is useful for tuning, because the information can help identify performance bottlenecks and can provide input into the tuning process.

The database driver's profiling option provides the user with a way to gather the timing of SQL statements being executed. However, logging all statements with their timings will result in a very large log file which cannot be properly analyzed.

You can define a logging threshold in which only statements that take more than a predefined number of seconds are logged.

With profiling, the following information is logged: the RDBMS request, the elapsed time, the user name, the date, and the time. The maximum precision that can be specified is 0.01 seconds.

To determine which table actions are most time consuming, you can set the EDBPROF environment variable to a number of seconds; for example, set EDBPROF as:

```
export EDBPROF=5.0
```

This sets EDBPROF to five seconds, which causes statements that take more than 5.0 seconds of elapsed time to be logged to the EDBPROF file in the current working directory of the driver.

To view statement execution time for individual tables, you can set the EDBPROF environment variable in the UNIX file `$BSE/lib/tabledef6.2`; for example, in this file, you can make this entry:

```
tccom010:812:entdb(EDBPROF=0.4):N
```

In this example, all the queries on table tccom010812 that require more than 0.4 seconds are logged in the EDBPROF file. Note that a separate driver starts for this table. The table is considered to have a different database definition.

Profiling example

Each phase in the SQL query processing that exceeds the profiling value is printed. The following is a sample EDBPROF file:

```
98-03-09[15:00:39]: Profiling value = 0.02 sec
------ Profiling value exceeded -----
<jim><bshellmain>:98-03-09[15:01:16.054]: Time (multi_fetch) : 0.033598 seconds
SQL statement:
SELECT /*+ FIRST_ROWS USE_NL(a,b,c) INDEX(a tttadv112000$idx1) */ a.t$pacc,a.t$
cpac,a.t$sequ,b.t$desc,c.t$expi FROM erpln.tttadv112000 a,erpln.tttadv130000 b,
erpln.tttadv200000 c WHERE c.t$cpac = :1 AND c.t$cmod = :2 AND c.t$cses = :3
AND c.t$vers = a.t$vers AND c.t$rele = a.t$rele AND c.t$cust = a.t$cust AND b.t
$cpac = :4 AND b.t$vers = a.t$vers AND b.t$rele = a.t$rele AND b.t$cust = a.t$
cust AND b.t$kdes = :5 AND b.t$rkey = :6 AND b.t$clan = :7 AND a.t$pacc = :8
AND a.t$cpac = :9 ORDER BY 1,2,3
-------------------------------------
------ Profiling value exceeded -----
<jim><bshellmain>:98-03-09[15:01:16.101]:
Time (parse) : 0.023754 seconds
SQL statement:
SELECT /*+ FIRST_ROWS INDEX(a tdctcs440000$idx9) */ a.t$adat,a.t$bfix,a.t$ccod,
a.t$cdat,a.t$cgrp,a.t$conf,a.t$cprj,a.t$csta,a.t$ctyp,a.t$delc,a.t$dsca,a.t$dti
m,a.t$ercs,a.t$erss,a.t$etim,a.t$fdat,a.t$fre1,a.t$fre2,a.t$fre3,a.t$fre4,a.t$f
re5,a.t$fre6,a.t$fre7,a.t$ider,a.t$idln,a.t$loca,a.t$modc,a.t$name,a.t$pact,a.t
$phse,a.t$pref,a.t$prgr,a.t$prib,a.t$prio,a.t$prjm,a.t$rdat,a.t$rtim,a.t$rtst,a
.t$rtyp,a.t$rusr,a.t$sact,a.t$scmp,a.t$sdat,a.t$solv,a.t$stat,a.t$ttim,a.t$txta,
a.t$txtp,a.t$txts,a.t$unit,a.t$uref,a.t$utyp,a.t$vref FROM erpln.tdctcs440000 a
WHERE a.t$delc = :1 AND (a.t$loca >= :2) AND (a.t$loca <= :3) AND (a.t$rtyp =
:4 AND a.t$unit = :5 AND a.t$uref = :6 AND a.t$utyp = :7 AND a.t$scmp = :8
AND a.t$vref = :9 AND a.t$csta = :10 AND a.t$loca > :11)
ORDER BY 39,50,51,52,42,53,8,26,24
-------------------------------------
```

The example shows two queries; the first has an execution time that exceeds the limit, while the second has a parse time that exceeds the limit.

To gather statistics

The database driver provides an option to gather driver-wide statistics on actions performed, such as:

- Number of cursors:
  - Opened.
  - Closed.
  - Current open.
- Number of parses, binds, executes, and fetches.
- Number of logons: Sessions.
- Number of inserts, updates, and deletes.
- Number of commits and rollbacks.

For each action, the cumulative elapsed time spent and the average time is logged. You can enable the statistics with the environment variable EDBSTAT. If you set the variable to zero, when the driver terminates, a statistics report is generated; in other words, it exits from LN Tools or the session. If a

value n greater than zero is specified, the driver logs an incremental report every n seconds (the driver must be active). The statistics report is written to the file EDBSTAT in the current directory.

These examples show how you can set EDBSTAT:

```
export EDBSTAT=0
export EDBSTAT=30
```

In the first example, EDBSTAT is set to zero; with this value, only a final report is generated. In the second example, EDBSTAT is set to 30, which logs a report every 30 seconds while the driver is active.

The following is a sample output of EDBSTAT. Because the report is generic for all databases, some information, such as the specific row actions, might not be appropriate for a particular database driver.

```
<3472> 2009-06-23[16:36:47]: Statistics [interval = 300]

DB-Cursor    Open     Close     Parse      Bind    Define   Execute     Fetch     Break
Count           7         7         7         7        13         9         5         5
Time(s)    0.0001    0.0001    0.0003    0.0001    0.0002    0.0100    0.0037    0.0000
Avg        0.0000    0.0000    0.0000    0.0000    0.0000    0.0011    0.0007    0.0000

Retained        #    Reused         %  Reparsed         %    Detach
Count           0         0       0.0         0       0.0         0

           BlobRd   BlobApp    BlobSz   BlobClr
Count          60        59        61         1
Time(s)    0.0198    0.3063    0.0085    0.0008
Avg        0.0003    0.0052    0.0001    0.0008

3/4GL       CrIdx     DrIdx     CrTbl     ClTbl     DrTbl     LkTbl     NrRow
Count           0         0         0         0         0         0         0
Time(s)    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
Avg        0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000

            Logon    Logoff    Commit  Rollback  ReadOnly
Count           1         1         1         1         0
Time(s)    0.0069    0.0014    0.0051    0.0723    0.0000
Avg        0.0069    0.0014    0.0051    0.0723    0.0000
```

Troubleshooting

The LN EnterpriseDB database driver provides a facility to troubleshoot problems. You can trace and store in a log file the actions that the driver performs; any errors that occur are also logged. The following sections describe how to log trace information and how to find and interpret the error log.

Logging database driver trace information

The database driver provides an option to trace online information about the actions being performed by the driver. The resulting log file contains debugging information that can help solve problems.

When tracing is enabled, the information stored in the log files includes:

- Table and index information (data dictionary).

- The SQL statements being executed.
- Values of the input and output bind variables.
- Other function-level debug statements.

To enable tracing, you can use the environment variable `DBSLOG`. Debugging information is appended to the file `dbs.log` in the UNIX directory `$BSE/tmp`.

To enable tracing, specify this command:

```
export DBSLOG=1570
```

To enable tracing for only categories of interest, several tracing categories are defined. For more details, see Database Driver Resources and Environment Variable on page 45.

Logging errors

The driver logs error messages in the log flies in the directory `$BSE/log`. From these log files, you can retrieve the following information:

- The user name, date, time, source file, and line number.
- The function called.
- The error code returned by the database.
- The database error description.
- The BDB error code returned to the application.
- Sometimes, the failing SQL statement.

If a database error occurs, the database driver attempts to map the error to a known or anticipated error condition. Generally, these mapped BDB errors have corresponding error numbers that fall in the range of 1 to 1,000. If a database-specific error occurs, the database driver maps the error to the BDB error code over 1,000 with this formula:

- abs(error_code) + 1000

Therefore, if the EDB OCI interface returns error ORA-1652, BDB error 2652 is returned to the application.

Usually, the log entries from the display driver, application virtual machine, and database driver contain sufficient information to determine the nature of and solution to the problem.

Whenever an error is encountered with an error code greater than 1,000, we recommend that you check the log entries from the database driver.

Chapter 6: Database Driver Configuration and Tuning

The LN EnterpriseDB database driver is designed to allow tuning for optimal performance. Several parameters used by the database driver are preset with default values. These values provide an acceptable balance between system performance and memory usage in most situations. Because every environment is different, the default values of these parameters can be modified to achieve more optimal performance. The LN EnterpriseDB database driver parameters and expected database driver behavior when changing these parameters are discussed.

Cursor management

The LN EnterpriseDB database driver has a resource variable that influences the cursor handling:

- **`retained_cursors`**

  Use this resource to reduce the number of inactive cursors, and therefore the number of open cursors.

  After all rows are fetched, the driver has a facility to place inactive cursors, which are cursors in Cancel status, in a cancelled list; this is so the cursors become candidates for assignment to a different query. However, a number of inactive cursors in this list are not available for this, and are defined by the resource retained_cursors.

  If the number of cursors in the cancel list exceeds the value of retained_cursors, a request for a new cursor is issued, and no reusable cursors are available, the least recently inactivated cursor is used for this new cursor. This cursor is disassociated from the original query and assigned to a new query, which performs the parsing and binding on this cursor. If the original query carries out a re-execute, the driver detects that the cursor is associated with another query and receives a new cursor and reparse, and then binds the query again.

  If you increase the value of retained_cursors, this can lead to less reparsing and rebinding of queries, which reduces CPU resources. However, the result is that the number of open cursors and memory increases.

Array interface

The LN EnterpriseDB database driver can use multirow features of the OCI interface for array fetches and array inserts. With the array interface, communication between the EnterpriseDB driver and the EnterpriseDB server is more efficient, multiple rows are fetched or inserted with a single OCI call. Because multiple rows are stored in a buffer in the LN EnterpriseDB database driver, more memory is consumed. Array interfacing is useful if you access a remote database, because the number of network round-trips is reduced.

The default number of rows fetched with a single OCI call is 5. To change this setting, you can specify this resource `edb_max_array_fetch`.

To disable the array interface, set `edb_max_array_fetch:1`

The default number of records buffered during insert and passed to EnterpriseDB with a single OCI call is 1; to change this value, you can specify the resource `edb_max_array_insert`.

During data loading using the `bdbpost` or `bdbreconfig` utility, the number of buffered records is specified by the `table_load_array_size` resource.

Query tuning

This section describes how to externally influence the query generation made by the EnterpriseDB driver.

Query hint generation

To improve performance, the EnterpriseDB driver generates hints in the native EnterpriseDB queries. When the resource `edb_hint_no_hints` is set, the driver will not generate any hint, except for queries with explicitly defined hints in the SQL query.

The hint generation tries to generate a database hint based on these origins:

- SQL query hints.
- Hints based on the ORDER BY clause.
- Hints based on the WHERE clause.

The hint origins are used in the order listed here, and the generation stops when a database hint is generated. Therefore, if a user generates a hint based on the ORDER BY clause, the hint generation no longer looks into the WHERE clause.

SQL Query Hints

The SQL query hints can be divided into three groups:

1. The "no hints" hint:

   If you use the "no hints" hint, the database driver does not generate a hint. Although no database hint is generated, the hint generation will not continue with the other hint origins.

2. Hints that influence the execution tree:

   This group consists of these hints:

   - The first rows and all rows hints.
   - The database dependent string literal hint.
   - The ordered hint.
   - The use index <n> on <table> hint.

3. Hints that influence driver behavior, such as array fetching:

   These hints do not influence the database hint generation in the driver, and are ignored. A hint can be generated based on other SQL hints, if present, or based on one of the other hint origins.

Hints based on the ORDER BY clause

The database drivers generate an index hint based on the ORDER BY clause, If these conditions are met:

- All items named in the ORDER BY clause must belong to the same table.
- All items must have the same direction. Do not use a mixture of ASC and DESC.
- Specify an index that matches the ORDER BY clause.

An index hint is generated for the best matching index. To determine the best matching index, you compare the ORDER BY clause with all indexes defined on the table.

A hint is generated for the index that has the highest number of matching fields, in the same order as the ORDER BY. You cannot use an index that has fewer fields than the ORDER BY, because the remaining ORDER BY fields still require an undesirable sort operation.

If multiple indexes have the same number of matching fields, the index with the lowest index number is used.

Hints based on the WHERE clause

The database drivers generate an index hint based on the WHERE clause, If these conditions are met:

- The query contains one single table, and therefore no joins.
- An index is defined that matches the WHERE clause.

To determine the best matching index, use these rules:

- Ignore those parts of the WHERE clause that cannot be used. Complex operations, such as pattern matching, are required, or result in multiple ranges.

  These operands are ignored:

  - LIKE expression [NOT] LIKE expression.
  - Exists [NOT] EXISTS expression.

- OR – lists expression OR expression.
- Set expression [NOT] IN expression.
- Negations expression != expression

  expression <> expression

  expression NOT BETWEEN expression

  expression NOT INRANGE expression

- If you use combined columns in a query, the combined columns are replaced by combined operands that contain the base columns of which the combined column exists.

  Base columns that do not have a counterpart are ignored.

  For example:

  ```
  AND_index3 >= { :aaa , :bbb }
  ```

  is handled as:

  ```
  AND { :col1, :col2, :col3, :col4 } >= { :aaa, :bbb }
  ```

  Note: The specification of the combined column name `_index<n>` does not imply the use of index <n> in EnterpriseDB. For the hint generation, the column `_index<n>` is a combined column that contains the base columns of that index.
- From the remaining parts of the query, the columns are determined that you can use for an index hint.
- Determine the matching indexes:

  For all indexes, the fields in the indexes are compared with the usable columns in the WHERE clause.

  Indexes whose initial field is not used are not matching.
- To determine the best matching index, calculate the total weight per index. The index with the highest total weight is selected.

  If multiple indexes have the same total weight, the index with the lowest index number is selected.
- The calculation starts with the first index field, and stops when an index field is not used in the WHERE clause. All remaining fields are not considered.

  For each column, the weight is determined. By default, the weight of the column is equal to the number of occurrences of the column in the pruned where clause.

  To tune the weight determination, use the resources `hint_idx_weight_equal` and `hint_idx_weight_range`, as shown in this example:

  The result of the expression: `<column weight>` * (`<hint_idx_weight_factor>`)$^{(position\ -1)}$ is added to the total weight.

Example of index selection

SQL query:

```
SELECT     tfacr200.ttyp, tfacr200.ninv
FROM       tfacr200
```

```
WHERE      tfacr200.ninv = :inv
AND        tfacr200.tdoc = :doc
AND        ( tfacr200.itbp BETWEEN :it1 AND :it2
             OR tfacr200.itbp = :it3 )
AND        tfacr200.line = :lin
AND        tfacr200.ttyp = :ttyp
```

Ignore unusable parts:

```
WHERE     tfacr200.ninv = :inv
AND       tfacr200.tdoc = :doc
AND       ( tfacr200.itbp BETWEEN :it1 AND :it2
            OR tfacr200.itbp = :it3 )
AND       tfacr200.line = :lin
AND       tfacr200.ttyp = :ttyp
```

Determine fields that are considered for the hint generation:

```
WHERE     tfacr200.ninv = :inv
AND       tfacr200.tdoc = :doc
AND       tfacr200.line = :lin
AND       tfacr200.ttyp = :ttyp
```

Determine the matching indexes:

```
Index 1:  ttyp  ninv  line  tdoc  docn  lino
Index 2:  itbp  ttyp  ninv  line  tdoc  docn  lino
Index 3:  year  btno  ttyp  ninv  line  tdoc  docn  lino
Index 4:  itbp  docd  ttyp  ninv  line  tdoc  docn  lino
Index 5:  itbp  ttyp  ninv  line  docd  tdoc  docn  lino
Index 6:  year  btno  tdoc  docn  lino  ttyp  ninv  line
```

An index hint in index 1 is generated because all other indexes do not match; this is because the initial index fields are not used in the WHERE clause.

Example of weight calculation

While the query in this example can look unwise, the query shows the weight calculation and the influence of the involved resources.

SQL query:

```
select* from dbtst180
where   projno > ' '
and     emstdate > 0
and     empno > 0
and     emendate = 0
and     emendate > 0
```

Indexes:

```
Index 1: empno projno stdate endate
Index 2: endate projno stdate empno
```

The column weight depends on the type of the expression. For equal expressions (<col> = <…>) the value of the resource `hint_idx_weight_equal` is taken. For range expressions (like >=) the value of the resource `hint_idx_weight_range` is taken:

```
hint_idx_weight_equal  = 1
hint_idx_weight_range  = 1
hint_idx_weight_factor = 1

_index1: empno(1.000) projno(1.000) stdate(1.000) endate(2.000)
   weight 5.000

_index2: endate(2.000) projno(1.000) stdate(1.000) empno(1.000)
   weight 5.000
```

An index hint for index 1 is generated. Note that index 2 is probably better for this query:

```
hint_idx_weight_equal  = 5
hint_idx_weight_range  = 1
hint_idx_weight_factor = 1

_index1: empno(1.000) projno(1.000) stdate(1.000) endate(2.000)
   weight 5.000

_index2: endate(6.000) projno(1.000) stdate(1.000) empno(1.000)
   weight 9.000
```

Note: The weight for column endate is reduced from 6 (`weight_range` + `weight_equal`) to 2 (`weight_range` + `weight_range`) because previous columns in the index definition only had matches on range expressions. In that case, the `weight_equal` is reduced to `weight_range`.

An index hint for index 2 is generated.

```
hint_idx_weight_equal  = 1
hint_idx_weight_range  = 1
hint_idx_weight_factor = 0.5

_index1: empno(1.000) projno(0.500) stdate(0.250) endate(0.250)
   weight 2.000

_index2: endate(2.000) projno(0.500) stdate(0.250) empno(0.125)
   weight 2.875
```

An index hint for index 2 is generated.

```
hint_idx_weight_equal  = 1
hint_idx_weight_range  = 0
hint_idx_weight_factor = 1

_index1: empno(0.000) [stopped]
   weight 0.000

_index2: endate(1.000) projno(0.000) [stopped]
   weight 1.000
```

An index hint for index 2 is generated.

Note: If `hint_idx_weight_equal` and `hint_idx_weight_range` are set to `0`, no index hint based on the WHERE clause is generated.

Concatenated expressions

The Infor application can use concatenated expressions, which operate on a combined column. Concatenated expressions that exist on combined columns are as follows:

- select >=
- select >
- select <=
- select <
- select between and

For example, an SQL statement can include a where clause such as:

```
WHERE comb >= {"tt", "adv", "000"}
```

In this example, comb is a combined column of columns c1, c2, and c3. This expression selects these ranges of rows:

- c1 = "tt" and c2 = "adv" and c3 >= "000"
- c1 = "tt" and c2 > "adv"
- c1 > "tt"

The EnterpriseDB driver can allow EnterpriseDB to solve the WHERE clause using three various techniques: nested, iterative, and filter. These techniques are introduced because the EnterpriseDB cost-based optimizer cannot efficiently handle these queries in all situations. For these queries, you can introduce full table scans and sort operations.

If you specify a different technique, the EnterpriseDB optimizer can make different decisions on how to run a query, and also provide workarounds for typical optimizer behavior. However, the optimizer behavior can change from version to version in EnterpriseDB; therefore, you can tune each version differently, which makes providing guidelines difficult.

To detect long running or bad performing queries, and then experiment with these various techniques, use the `EDBPROF` variable. For more information on `EDBPROF`, see Database Driver Resources and Environment Variable on page 45.

This list describes the nested, iterative, and filter techniques:

- The nested technique:

  The three conditions are ORed to this expression:

  - c1 = "tt" and c2 = "adv" and c3 >= "000" OR
  - c1 = "tt" and c2 > "adv" OR
  - c1 > "tt"

  This can be rewritten as:

  - c1 > "tt" OR
  - c1 = "tt" AND (c2 > "adv" OR

- c2 = "adv" AND (c3 >= "000"))

  The last expression has a nested AND/OR condition and is, therefore, referred to as the nested technique.

- The iterative technique:

  To resolve one query, multiple SQL statements are issued; these statements do not contain OR conditions, and therefore can be efficiently handled by EnterpriseDB. You can only use the iterative technique for unbounded queries.

- The iterative technique uses these three conditions:
  - c1 = "tt" and c2 = "adv" and c3 >= "000"
  - c1 = "tt" and c2 > "adv"
  - c1 > "tt"

  First, a query with Step 1 is issued. If the query does not return a row, the process continues with Step 2. If Step 2 does not return a row, the process continues with Step 3. Also, if one step has returned all rows, but more rows are required, the driver continues with the next step.

- The filter technique:

  This technique is related to the nested technique but has a different approach. This technique initially selects too many rows, but then filters out those rows that do not match the total WHERE clause. This technique selects based on the first column in a concatenated index and filters out rows with the NOT() operator. The query is solved as:
  - c1 >= "tt" AND NOT(c1 = "tt" AND (c2 < "adv" OR
  - c2 = "adv" AND (c3 < "000"))

  The NOT() expression is like an inverted nested query. These rows are filtered out of the initial set determined by the first condition: c1 >= "tt".

Specifying query tuning

You can specify the query tuning by table and by index.

On UNIX specify this file: `$BSE/lib/edb/edb_driver_param`.

For a list of the actual values you can specify, see "Storage Parameter File Format and Driver Configuration Options on page 63".

Optimistic and pessimistic reference checks

To optimize concurrency, the LN EnterpriseDB database driver supports optimistic and pessimistic reference checks. In lookup reference mode, if you perform inserts in a child table, the driver checks whether the reference exists in the parent table and locks the referenced record to ensure that another user cannot delete the record in the current transaction. This approach is called the pessimistic approach.

This approach blocks an insert of another user that references the same parent row, thereby affecting the concurrency. To avoid this problem, another approach is available in which you use a row in the parent table that is not locked, depending on the user's choice. This approach is called the optimistic approach. Because the record is not locked, another user can still perform an insert operation, which improves the concurrency.

The dbsinit resource variable controls this behavior.

Locking behavior

If you update a database table, an RDBMS must use a locking mechanism to ensure data integrity. To lock the physical data, EnterpriseDB PPAS uses row level locking; therefore, the smallest unit of data you can lock is the row.

To ensure data integrity, the LN EnterpriseDB database driver uses EnterpriseDB's locking mechanism. The database driver uses several locking strategies, including:

- Implicit and explicit locking.
- Delayed locking.
- Statement and lock timeouts.
- High level lock retries.

Delayed locking

The database driver uses delayed locking; therefore, before the driver performs an update, the driver checks each column to determine whether the related columns have been changed. If the columns have not changed, the driver does not carry out the update. This method reduces the workload on the RDBMS and the network traffic between the database driver and the RDBMS.

Statement and lock timeouts

If another session locks a resource, a SELECT FOR UPDATE statement waits for a predetermined time period (timeout duration). If a statement times out, you cannot determine whether the process timed out because of a resource lock or for some other unrelated reason, such as slow network throughput or poor response from the database server.

The timeout duration is configurable. If the database server resides on a separate machine from the driver, or if server performance is not optimal, you might be required to increase the timeout duration so that the database driver does not give up before the server or network has had an opportunity to deliver the results from a request. Note that statements that do not take locks, or statements that read through locks, can also timeout.

To specify the lock wait period for SELECT FOR UPDATE, INSERT, DELETE, UPDATE, and LOCK TABLE statements, use the EDB_LOCK_TIMEOUT environment variable or the `edb_timeout` resource variable.

The following is an example of how to set the timeout parameters:

```
edb_lock_timeout:600
```

Appendix A: Database Driver Resources and Environment Variable

You can use the database driver resources and environment variables as configuration parameters to modify the behavior of the EnterpriseDB database driver. Some of these resources are used by the client, while others are used by the server. In this context, the client is the LN application virtual machine and the server is the LN EnterpriseDB database driver.

The LN application virtual machine and the database driver can run on separate machines. Set the client resources on the machine that runs the LN application virtual machine. Set the server resources on the machine that runs the database driver. You must set the resources for the client and server on both machines.

To set the database driver resources and environment variables, see Setting driver behavior on page 24.

Summary of EnterpriseDB driver resources and environment variables

These types of resources and environment variables are available to use with the LN EnterpriseDB database driver:

- Client and server resources used by all LN database drivers.
- Client resources used by all LN database drivers.
- Server resources used by all LN database drivers.
- Resources used only by the LN EnterpriseDB database driver.
- Resources passed to the EnterpriseDB database server.

The tables provide a summary of each of these resources and environment variables. Detailed descriptions of each entry in the tables are available later.

Client and server resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
baan_sql_cacherows	BAAN_SQL_CACHEROWS	Defines the size of internal buffers in the query processor

Client and server resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
baan_sql_trace	BAAN_SQL_TRACE	Allows you to view SQL query information
enable_refmsg	ENABLE_REFMSG	Causes logging of denied updates of delete actions
rds_full	RDS_FULL	Sets maximum number of rows transferred in one block
use_shm_info	USE_SHM_INFO	Enables or disables shared memory use

Client resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
baan_sql_stmt_cache_size	BAAN_SQL_STMT_CACHE_SIZE	Defines the size of the query cache
bdb_debug	BDB_DEBUG	Sets the debugging link between client and server
bdb_driver	BDB_DRIVER	Sets database specifications
bdb_max_server_schedule	BDB_MAX_SERVER_SCHEDULE	Defines the mechanism to stop idle database drivers
ssts_set_rows	SSTS_SET_ROWS	Sets number of rows read ahead (single table single row)
-	USR_DBC_RES	Specifies alternative resource file for client

Server resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
bdb_max_sessions	BDB_MAX_SESSIONS	Defines the number of sessions per driver
bdb_max_session_schedule	BDB_MAX_SESSION_SCHEDULE	Defines the mechanism to close idle driver sessions
dbslog	DBSLOG	Allows you to perform driver profiling
-	DBSLOG_LOCK_PROF	Specifies the lock time after which locks are logged
dbslog_name	DBSLOG_NAME	Allows you to specify the file name for logging
dbsinit	DBSINIT	Deprecated
hint_index_weight_equal	HINT_INDEX_WEIGHT_EQUAL	Defines the equal-weight parameter for the query hint generation
hint_index_weight_range	HINT_INDEX_WEIGHT_RANGE	Defines the range-weight parameter for the query hint generation

Server resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
hint_index_weight_factor	HINT_INDEX_WEIGHT_FACOR	Defines the weight-factor parameter for the query hint generation
shm_max_dd_cache_size	SHM_MAX_DD_CACHE_SIZE	Defines the maximum size of the database driver DD cache
mle_join_type	MLE_JOIN_TYPE	Specifies join type between data table and shadow table, with translations, in a Multi Language Enabled (MLE) environment.
query_comments	QUERY_COMMENTS	Allows tracing of used connections in the database server.
table_load_array_size	TABLE_LOAD_ARRAY_SIZE	Defines the maximum number of rows inserted during data load.
-	USR_DBS_RES	Specifies alternative resource file for server

Resources used only by the LN EnterpriseDB database driver		
Resource name	**Environment variable**	**Description**
max_sql_input_binds	MAX_SQL_INPUT_BINDS	Allows to adjust the size of internal ‘where bind’ buffers.
edb_alter_session	EDB_ALTER_SESSION	Allows you to change EnterpriseDB session settings.
edb_max_array_fetch	EDB_MAX_ARRAY_FETCH	Defines the maximum number of rows fetched
edb_max_array_insert	EDB_MAX_ARRAY_INSERT	Defines the maximum number of rows inserted
edbprof	EDBPROF	Enables profiling
edbstat	EDBSTAT	Allows statistics to be gathered.
edb_hint_no_hints	EDB_HINT_NO_HINTS	Disable the generation of query hints
edb_timeout	EDB_LOCK_TIMEOUT	Sets the timeout value
retained_cursors	RETAINED_CURSORS	Defines the number of retained cursors

Resources used only by the LN EnterpriseDB maintenance program		
Resource name	**Environment variable**	**Description**
edb_default_tablespace	EDB_DEFAULT_TABLESPACE	Allows you to set a default tablespace for newly created EnterpriseDB users.

Resources passed to the LN EnterpriseDB database server		
Resource name	**Environment variable**	**Description**
pgclientpath	PGCLIENTPATH	Sets the location of EnterpriseDB PPAS
pghost	PGHOST	EnterpriseDB host connection parameter
pgport	PGPORT	EnterpriseDB port connection parameter
pgdatabase	PGDATABASE	EnterpriseDB dbname connection parameter

Detailed description of driver resources and environment variables

The driver resources are divided into two sections: sections generic to all LN database drivers, and those specific to the LN EnterpriseDB driver.

Each group of resources is shown in alphabetical order.

Generic driver resources

baan_sql_cacherows / BAAN_SQL_CACHEROWS	
Driver resource	baan_sql_cacherows
Environment variable	BAAN_SQL_CACHE_ROWS
Client/Server resource	Set for both client and server
Type	Integer
Default	71
Description	This variable influences the number of records that the query processor internally caches for sorting, aggregation functions, or prepared sets. If you exceed this limit, temporary files are generated. For optimal performance of the internally used hash functions, specify a prime number.

baan_sql_stmt_cache_size / BAAN_SQL_STMT_CACHE_SIZE	
Driver resource	baan_sql_stmt_cache_size
Environment variable	BAAN_SQL_STMT_CACHE_SIZE
Client/Server resource	Set for client only

baan_sql_stmt_cache_size / BAAN_SQL_STMT_CACHE_SIZE	
Type	Integer
Default	330
Description	This resource sets the number of inactive queries that must be retained for reuse.

baan_sql_trace / BAAN_SQL_TRACE	
Driver resource	baan_sql_trace
Environment variable	BAAN_SQL_TRACE
Client/Server resource	Set for client only
Type	Integer (Octal)
Default	0
Description	This variable is introduced to view the SQL query information being handled in client and server. If you set this variable, debug information is printed by the client to the log file (client) or `dbs.log` file (server). The information contains various categories you can enable separately, but most categories are not relevant for the audience of this document. The relevant values of the `baan_sql_trace` variable and the descriptions of these values are : • 0002000: Major query interface logging • 0004000: Detailed query interface logging

bdb_debug / BDB_DEBUG	
Driver resource	bdb_debug
Environment variable	BDB_DEBUG
Client/Server resource	Set for client only
Type	Integer (octal)
Default	0

bdb_debug / BDB_DEBUG	
Description	You can use this variable to generate debugging information about the communication between the client and the database driver. If you set this variable, the client prints debugging information to standard error (stderr). You can specify debug information in these categories: • 00001: server types • 00002: database actions • 00004: delayed lock actions • 00010: reference information • 00100: permission information To define multiple categories, you can add the octal values. To determine if a given category must be logged, the value is compared bitwise.

bdb_driver / BDB_DRIVER	
Driver resource	bdb_driver
Environment variable	BDB_DRIVER
Client/Server resource	Set for client only
Type	String
Default	None
Description	You can use this variable to set a database specification, usually found in the file `tabledef6.2`. If you set this variable, all the tables are accessed using the database driver specified, and `tabledef6.2` is read. You must specify the driver specified in the file `$BSE/lib/ipc_info`.

bdb_max_server_schedule / BDB_MAX_SERVER_SCHEDULE	
Driver resource	bdb_max_server_schedule
Environment variable	BDB_MAX_SERVER_SCHEDULE
Client/Server resource	Set for client only
Type	Integer
Default	3

bdb_max_server_schedule / BDB_MAX_SERVER_SCHEDULE	
Description	This variable defines the mechanism for terminating idle database drivers by the application virtual machine. If the database driver has no more open sessions, the application virtual machine can terminate the driver. You close an idle database driver after several schedule ticks. A schedule tick is generated whenever you end an LN session. At this point, all idle database drivers have a schedule counter incremented. When the value of the schedule counter reaches the value of `bdb_max_server_schedule`, the database driver ends.

bdb_max_sessions / BDB_MAX_SESSIONS	
Driver resource	bdb_max_sessions
Environment variable	BDB_MAX_SESSIONS
Client/Server resource	Set for server only
Type	Integer
Default	0 (unlimited)
Description	This variable defines the number of sessions per driver. If any driver has reached this threshold, a new driver starts that handles new sessions.

bdb_max_session_schedule / BDB_MAX_SESSION_SCHEDULE	
Driver resource	bdb_max_session_schedule
Environment variable	BDB_MAX_SESSION_SCHEDULE
Client/Server resource	Set for server only
Type	Integer
Default	3
Description	This variable defines the mechanism for closing idle sessions in the driver. If the client process has no more references (cursors or queries) to the session, the client can close the client process. The client closes an idle session after several schedule ticks. A schedule tick is generated whenever an LN session is ended. At this point, all idle sessions have a schedule counter incremented. If the value of the schedule counter reaches the value of `bdb_max_session_schedule`, the session closes. The default for `bdb_max_session_schedule` is three. If you set `bdb_max_session_schedule` to one, this results in fewer connections from the driver to the RDBMS, because whenever an LN session ends, the corresponding RDBMS session (logon) closes (logoff).

dbsinit / DBSINIT	
Driver resource	Dbsinit
Environment variable	DBSINIT
Client/Server resource	Set for server only
Type	Integer (octal)
Default	1
Description	With this variable you can set flags to specify the optimizations to be used. At this time, legal values are 0 (not set), 1. Other values are reserved and must not be used. To define multiple categories, add the octal values. To determine if a given category must be logged, the value is compared bitwise. This parameter is deprecated and removed in a future release.

dbslog / DBSLOG	
Driver resource	dbslog
Environment variable	DBSLOG
Client/Server resource	Set for server only
Type	Integer (octal)
Default	0
Description	This variable provides detailed debugging information about the online processing of the driver. The information is logged in the file dbs.log in the driver's current directory. You can specify these debugging categories: • 0000001: Data dictionary information on tables in the driver • 0000002: Query info (SQL Level 1) • 0000004: Query plan info (SQL Level 2) • 0000010: Row action information • 0000020: Table action information • 0000040: Transaction action information • 0000100: DBMS input/output data (SQL Level 2) • 0000200: Administration file info (SQL drivers) • 0000400: DBMS SQL statements • 0001000: General debug statements • 0004000: Data buffering info (communication) • 0100000: Lock retries logged (includes session name) • 0200000: Logs successful locks and longest lock duration in a transaction To define multiple categories, you can add the octal values. To determine if a given category must be logged, the value is compared bitwise.

DBSLOG_LOCK_PROF	
Driver resource	—
Environment variable	DBSLOG_LOCK_PROF
Client/Server resource	Set for server only
Type	Floating point number
Default	0
Description	Specifies the minimum duration of a lock that must be logged. Any locks of shorter duration are not logged. This variable specifies the minimum number of seconds, to a precision of milliseconds, that must elapse before a lock is logged. Lock time is calculated as the time from when the first record in a transaction is locked to the time of the commit or abort. This time is the longest time a record remains locked during a transaction. Note that you must set the appropriate dbslog categories.

dbslog_name / DBSLOG_NAME	
Driver resource	dbslog_name
Environment variable	DBSLOG_NAME
Client/Server resource	Set for server only
Type	String
Default	dbs.log
Description	You can use this variable to specify a file name where DBS logging information must be written. If a file already exists with the same name, the file is used for logging. If the file is locked during write operations, multiple servers can use the same log file.

enable_refmsg / ENABLE_REFMSG	
Driver resource	enable_refmsg
Environment variable	ENABLE_REFMSG
Client/Server resource	Set for server only
Type	Boolean
Default	0 (disabled)
Description	There are two valid values for this variable: 0 and 1. When set to 0, no log messages are generated. When set to 1, log messages are generated in the database driver log file. For example, when an update of a delete action was denied because of existing references.

hint_idx_weight_equal / HINT_IDX_WEIGHT_EQUAL	
Driver resource	hint_idx_weight_equal
Environment variable	HINT_IDX_WEIGHT_EQUAL
Client/Server resource	Set for server only
Type	Integer
Default	1
Description	This variable controls the index hint generation. For information on the influence of this variable on the index hint generation, see "Database Driver Configuration and Tuning on page 35".

hint_idx_weight_factor / HINT_IDX_WEIGHT_FACTOR	
Driver resource	hint_idx_weight_factor
Environment variable	HINT_IDX_WEIGHT_FACTOR
Client/Server resource	Set for server only
Type	Floating point number
Default	1.0
Range	0.0 … 1.0
Description	This variable controls index-hint generation. For information on the influence of this variable on the index-hint generation, see "Database Driver Configuration and Tuning on page 35".

hint_idx_weight_range / HINT_IDX_WEIGHT_RANGE	
Driver resource	hint_idx_weight_rangel
Environment variable	HINT_IDX_WEIGHT_RANGE
Client/Server resource	Set for server only
Type	Integer
Default	1
Description	This variable controls the index-hint generation. For information on the influence of this variable on the index-hint generation, see "Database Driver Configuration and Tuning on page 35".

rds_full / RDS_FULL	
Driver resource	rds_full

rds_full / RDS_FULL	
Environment variable	RDS_FULL
Client/Server resource	Set for both client and server
Type	Integer
Default	5
Description	This variable defines the maximum number of rows transferred between the LN application virtual machine and the driver as one block. Multiple blocks, and thus network round trips, are transferred if more rows are requested. You must set this variable to the same value for the client and server. The sql.set.rds.full() function or the “buffer <n> rows query overrule this default setting.

mle_join_type / MLE_JOIN_TYPE	
Driver resource	mle_join_type
Environment variable	MLE_JOIN_TYPE
Client/Server resource	Set for server only
Type	Integer
Default	0 (INNER)
Description	This variable determines the type of join used, in case of a Multi Language Enabled (MLE) environment, between the data table and the corresponding shadow table that contains the translations. The default join type (INNER) is recommended for normal circumstances. The database does not always generate an optimal execution plan. This can be the case when 5 data languages or more are used. To rectify this you can use a LEFT join type between the tables. This resource is implemented for MLE tables for all databases; it must be applied after extensive testing. When upgrading to a new database version, the use of this resource must be validated again. These values are supported: • 0: INNER (default) • 1: LEFT

query_comments/QUERY_COMMENTS	
Driver resource	query_comments
Environment variable	QUERY_COMMENTS
Client/Server resource	Set for server only
Type	Integer
Default	0

query_comments/QUERY_COMMENTS	
Description	This variable allows enabling tracing queries in the database server. Enabling this resource results in inserting comments for some types of statements, for example queries. The comments include additional information about the database driver its process ID, the LN user name and the LN session name. **Warning:** Enabling this resource severely impacts performance as it affects the query plan cache of the database server. Enabling this setting is for diagnostic purposes only.

shm_max_dd_cache_size / SHM_MAX_DD_CACHE_SIZE	
Driver resource	shm_max_dd_cache_size
Environment variable	SHM_MAX_DD_CACHE_SIZE
Client/Server resource	Set for server only
Type	Integer
Default	12582912 (12Mb)
Description	This variable defines the maximum number of bytes that is used to cache database driver data dictionaries in the shared memory cache. If you exceed this limit, no further information are cached. A value of 0 is interpreted as unlimited. See use_shm_info.

ssts_set_rows / SSTS_SET_ROWS	
Driver resource	ssts_set_rows
Environment variable	SSTS_SET_ROWS
Client/Server resource	Set for client only
Type	Integer
Default	3
Description	This variable defines the number of rows to be read ahead for a fetch request from the client. The default is three rows, that means that for one fetch request, three rows are read. For the next two fetch requests, rows are taken from the client row buffer or fetched from the database without re-executing the query.

table_load_array_size / TABLE_LOAD_ARRAY_SIZE	
Driver resource	table_load_array_size
Environment variable	TABLE_LOAD_ARRAY_SIZE
Client/Server resource	Set for server only

table_load_array_size / TABLE_LOAD_ARRAY_SIZE	
Type	Integer
Default	100
Description	This variable defines the number of rows that are inserted using array interfacing when using the `bdbpost` or `bdbreconfig` utility.

use_shm_info / USE_SHM_INFO	
Driver resource	use_shm_info
Environment variable	USE_SHM_INFO
Client/Server resource	Set for both client and server
Type	Boolean
Default	1 (enabled)
Description	You can use this variable to enable or disable the use of shared memory to each of the database driver data dictionaries. Two values are valid for this variable: 0 and 1. If set to 0, shared memory is disabled. If set to 1, shared memory is enabled.

USR_DBC_RES	
Driver resource	—
Environment variable	USR_DBC_RES
Client/Server resource	Set for client only
Type	String
Default	None
Description	This variable contains the file specification of an alternative resource file for the client. The file specification is based on the BSE directory and is in double quotes. When set, any resources in the alternative resource file override the same client resources set in `db_resource`.

USR_DBS_RES	
Driver resource	—
Environment variable	USR_DBS_RES
Client/Server resource	Set for server only
Type	String
Default	None

USR_DBS_RES	
Description	This variable contains the file specification of an alternative resource file for the client. The file specification is based on the BSE directory and is within double quotes. When set, any resources in the alternative resource file override the same server resources set in `db_resource`.

EnterpriseDB driver specific resources

max_sql_input_binds / MAX_SQL_INPUT_BINDS	
Driver resource	max_sql_input_binds
Environment variable	MAX_SQL_INPUT_BINDS
Client/Server resource	Set for server only
Type	Integer
Default	512
Description	This resource sets the maximum number of the input binds that a query can have. Only very complex permission schemes might get so many additional clauses in their queries that they need enlarged bind buffers.

edb_alter_session/EDB_ALTER_SESSION	
Driver resource	edb_alter_session
Environment variable	EDB_ALTER_SESSION
Client/Server resource	Set for server only
Type	String
Default	Not set
Description	After you set this resource, the EnterpriseDB driver issues an alter session <resource value> command directly after each logon to EnterpriseDB. You can use this command for support purposes to change the EnterpriseDB session settings for a particular driver without having to change the generic EnterpriseDB settings. For a description of the supported SQL syntax, refer to the EnterpriseDB Server SQL reference document.

edb_hint_no_hints/EDB_HINT_NO_HINTS	
Driver resource	edb_hint_no_hints

edb_hint_no_hints/EDB_HINT_NO_HINTS	
Environment variable	EDB_HINT_NO_HINTS
Client/Server resource	Set for server only
Type	Integer
Default	0
Description	When set, this resource disables the query hint generation of the EnterpriseDB driver. Only queries that have explicit query hints specified in the application source will get an EnterpriseDB hint. All other hints will be suppressed,

edb_max_array_fetch /EDB_MAX_ARRAY_FETCH	
Driver resource	edb_max_array_fetch
Environment variable	EDB_MAX_ARRAY_FETCH
Client/Server resource	Set for server only
Type	Integer
Default	5
Description	This resource defines the maximum number of rows immediately fetched from the EnterpriseDB RDBMS. The query hint “array size <n>” overrules this default setting. To disable the array interface, set `edb_max_array_fetch:1`.

edb_max_array_insert /EDB_MAX_ARRAY_INSERT	
Driver resource	edb_max_array_insert
Environment variable	EDB_MAX_ARRAY_INSERT
Client/Server resource	Set for server only
Type	Integer
Default	1
Description	This environment variable defines the maximum number of rows immediately inserted in EnterpriseDB. Note that you cannot always enable this option. For example, if you must check or update references, or if the application requires immediate response from the driver as to whether the insert is successful, you cannot perform an array insert. See table_load_array_size

edbprof / EDBPROF	
Driver resource	edb_prof

edbprof / EDBPROF	
Environment variable	EDBPROF
Client/Server resource	Set for server only
Type	Floating point
Default	Not set
Description	If you specify a value in this variable, each phase in the SQL query processing that exceeds the number of seconds specified will be logged. The maximum precision you can specify is 0.01 seconds.

edbstat / EDBSTAT	
Driver resource	edbstat
Environment variable	EDBSTAT
Client/Server resource	Set for server only
Type	Integer
Default	Not set
Description	This variable allows you to report database driver statistics. If set to a value n greater than 0, statistics are logged every n seconds while the driver is active. If set to 0, a statistics report is generated when the driver terminates.

edb_lock_timeout / EDB_LOCK_TIMEOUT	
Driver resource	edb_lock_timeout
Environment variable	EDB_LOCK_TIMEOUT
Client/Server resource	Set for server only
Type	String
Default	None
Description	This resource specifies the lock wait period for specific actions. For more information on locking behavior and setting timeouts, see "Statement and lock timeouts on page 43".

retained_cursors / RETAINED_CURSORS	
Driver resource	retained_cursors
Environment variable	RETAINED_CURSORS
Client/Server resource	Set for server only
Type	Integer
Default	50

retained_cursors / RETAINED_CURSORS	
Description	This resource sets the number of inactive cursors that must be retained in the list for reuse by session. For more information on cursor management, see "Cursor management on page 35". Note that all application sessions, which share the same database connection, share the same inactive cursor pool.

EnterpriseDB maintenance program specific resource

edb_default_tablespace / EDB_DEFAULT_TABLESPACE	
edb_maint resource	edb_default_tablespace
Environment variable	EDB_DEFAULT_TABLESPACE
Type	String
Default	Not set
Description	If LN adds a new user to the EnterpriseDB database, the default tablespace is set to PGDEFAULT. When this resource is set, an ALTER USER set DEFAULT TABLESPACE <string> command will be issued after the user is created.

Resources passed to the EnterpriseDB RDBMS

pgclientpath / PGCLIENTPATH	
Driver resource	pgclientpath
Environment variable	PGCLIENTPATH
Client/Server resource	Set for server only
Type	String
Default	Blank
Description	This resource must be set to the EnterpriseDB PPAS software location that contains the EnterpriseDB OCI connector

pghost / PGHOST	
Driver resource	pghost

pghost / PGHOST	
Environment variable	PGHOST
Client/Server resource	Set for server only
Type	String
Default	Blank (local host)
Description	Sets the EnterpriseDB host connect parameter. It contains the host to connect to.

pgport / PGPORT	
Driver resource	pgport
Environment variable	PGPORT
Client/Server resource	Set for server only
Type	String
Default	Blank (local host)
Description	This resource sets the EnterpriseDB port connect parameter

pgdatabase / PGDATABASE	
Driver resource	pgdatabase
Environment variable	PGDATABASE
Client/Server resource	Set for server only
Type	String
Default	erpln
Description	Sets the EnterpriseDB dbname connect parameter. It contains the database name to connect to. .

Appendix B: Storage Parameter File Format and Driver Configuration Options

File format: $BSE/lib/edb/edb_driver_param

The driver parameter file provides a way to influence the behavior of the database driver on a per table/index basis for:

- The ownership of the database tables. For more information, see Database Security on page 28.
- The query generation. For more information, see Query tuning on page 36.
- The refresh time of the record cache: this field is obsolete.

This is an example of an entry in the driver parameter file:

```
*:*:T: group:014::
```

In this example, the database driver creates tables owned by an LN group, and uses the iterative technique during the query generation.

A driver parameter file is defined for each database driver. The storage file for the LN EnterpriseDB database driver is called `edb_driver_param` and is located in the UNIX directory `$BSE/lib/edb`.

The file consists of zero or more entries, each of which consists of several fields separated by colons. The format of an entry in the storage file is (primary key in bold):

```
<table/module specification> : <company number> : <object type> : <group>
 : <table/index optimization> : <refresh time>
```

For information on each of the fields in an entry in the storage parameter file, see Parameter file field descriptions on page 64.

The drivers scan this file top-down. The first matching line is used, but not the best matching line.

File format: $BSE/lib/edb/edb_storage_param

The storage parameter file provides a way to specify the distribution of table and index data in various tablespaces. The database driver uses the storage parameters whenever you want to run a DDL

statement, such as a create table or create index statement. The following is an example of an entry in the storage parameter file:

```
*:*:T: TABLESPACE dataspace
```

In this example, the database driver adds the “<tablespace>” clause to the create statement during index or table creation. If the segment for a table or index is not specified, the table and index data are created in the default tablespace. If index data is to be separated, you must specify a tablespace.

A storage parameter file is defined for each database driver. The storage file for the LN EnterpriseDB database driver is called `edb_storage_param` and is located in the UNIX directory `$BSE/lib/edb`.

The entries in the `edb_storage_param` file consist of fields that are the same for the storage files for all database drivers and storage parameters specific to the EnterpriseDB database driver.

The file must consist of zero or more entries, each consisting of several fields separated by colons. The format of an entry in the storage file is (primary key in bold):

```
 <Table/Module Specification> : <Company Number> : <Object Type> :[<com▶
press specification>] <Storage Parameters>
```

Each of the fields in an entry in the storage parameter file are described in the section Parameter file field descriptions on page 64.

The drivers scan this file top-down. The first matching line is used, but not the best matching line.

Parameter file field descriptions

table/module specification		
Description	This field consists of a list of comma-separated table names or a module name to which the entry applies. An asterisk (*) indicates all tables.	
Example	ttadv000,ttadv999	Two specific tables
	ttadv	All tables in module tt and package adv
	tt	All tables in module tt
	*	All tables

company number		
Description	This field consists of a list of company numbers to which the entry applies. An asterisk (*) indicates all company numbers.	
Example	000,999	Companies 000 and 999
	*	All companies

object type		
Description	This field consists of a list of object (table or index) identifications to which the entry applies. You can specify the following options:	
	T	Table only
	I	All indexes
	I <index number>	Only specified index
	*	Both table and indexes
Fall back rules in the database drivers	If no specific index line (I<index number>) is defined, the value for the generic index line is used. If no generic index line is defined, the driver uses the value for the table line.	
Example	I1,I2	Only index 1 and 2
	T	Only for table

Compress specification	
Description	When this field is present the table or index is compressed. EnterpriseDB does not have a specific compression option.

Group	
Description	This field identifies the owner of the table. You must specify Group.
Note	Previous porting sets also supported the Private option. This option is obsolete.

table/index optimization	
Description	Specific flags related to indexes and tables can be specified. If specified on a “T” object entry, the flag defines the default for all indexes. You can use octal values to set the flags for a specific index or table. • 00: Use default initial technique (filter) • 03: Initial technique is nested • 04: Initial technique is iterative • 05: Initial technique is filter You can OR these flags to this value: • 0010: Generate statistics at creation of the table
Example	014 – Recommended value.

Refresh time	
Description	Ignored by the EnterpriseDB driver

Storage parameters	
Description	The specific database driver implementation defines these parameters, and often defines map-to-table and index creation options available in the host RDBMS. Several driver configuration options specific to the EnterpriseDB driver allow the user to customize table and index attributes at create time. These storage options are used to affect specified tables. For a complete description of the parameters you can add to the CREATE TABLE and CREATE INDEX commands, refer to the EnterpriseDB SQL Language Reference Manual.
Example	TABLESPACE DATA