Infor Enterprise Server Technical Reference Guide for DB2 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_lndbdb2drvtrg__en-us

Contents

About this guide

This document supplies technical reference information about the Infor Enterprise Server database driver for DB2 for UNIX and Windows operating systems.

The database driver is referred to as the Infor Enterprise Server DB2 database driver. The information is Infor-release independent and can be used for Infor Baan IVc, Infor Baan 5.0 and LN

For simplification, LN is used in this document. Unless specified, the information applies to all supported DB2 versions and any differences are stated.

The term ‘Infor Enterprise Server’ comprises the tools and porting set. However this document only handles issues related to the porting set, where the DB2 database driver is a part of.

The term UNIX refers to both UNIX and Linux. For details on other platform support, see the *Infor Enterprise Server Platform Support Matrix*, see Infor Xtreme KB 1183466.

Intended audience

This document is intended for anyone who wants to configure or customize the Infor Enterprise Server database driver for DB2. To understand this document, a basic knowledge of UNIX or Windows and DB2, and an understanding of database technology, is required.

This document covers the database driver for Infor ERP Baan IVc, Infor ERP 5.0c and Infor LN.

Related documents

- You can find the documents in the product documentation section, as described in Contacting Infor.

For information on the installation procedure for DB2 and LN software, see these documents:

- *Infor Baan IVc - Pre installation Guide for Windows*
- *Infor Baan IVc - Pre installation Guide for UNIX*
- *Infor Baan IVc - Deployment Guide for Windows and UNIX*
- *Infor LN - Installation Guide*
- *Infor Enterprise Server - Technical Manual*
- *Infor LN - Performance, Tracing and Tuning Guide*

- *Infor LN - Performance, Tracing and Tuning Guide for DB2 Server*

For detailed information regarding DB2, see IBM's information center for DB2.

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: Database Driver Overview

The database driver plays an important role in 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 is required to provide a seamless interface between the LN software and the various RDBMS products. The database driver enables the majority of the LN processing to be independent of the RDBMS.

Architecture

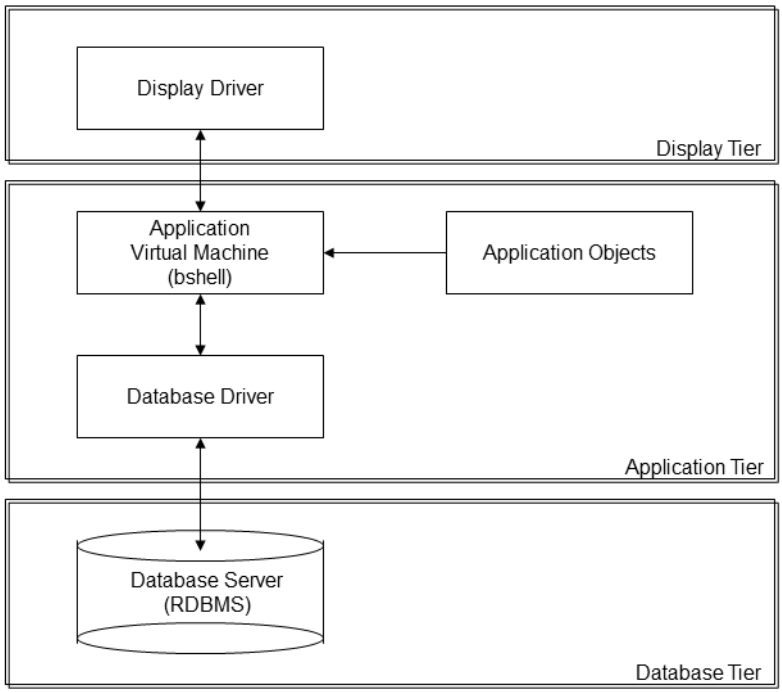
LN supports a three-tier architecture that consists of a display tier, an application tier, and a database tier. The display tier provides presentation services for user interaction. The application tier consists of the LN application virtual machine, the application objects, and the LN database driver.

The database tier consists of a third-party RDBMS product that acts as the database server..

The emphasis of this document is on the LN database driver, which is the database driver 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 virtual machine sends to the database server. After the database server retrieves the requested information, the database driver passes the data back to the LN application virtual machine.

To put the functions of the database driver into perspective, the following sections briefly describe each of the three tiers of the total LN architecture.

This diagram illustrates the 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, the LN application virtual machine, and the database driver. Together, the application objects and the application virtual machine provide much of the functionality of LN, whereas the database driver provides connectivity to the database server to store and retrieve application data.

The application objects include the compiled LN applications and the data dictionary. These applications are written in Baan 3GL or Baan 4GL, which are programming languages supported by Infor ERP Enterprise Server, and are compiled into an intermediate code interpreted at run time by the virtual machine.

The data dictionary defines the data models the applications use. 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 as required 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 the virtual machine must communicate.

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

The LN database driver is also a part of the application tier. 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 is used as the database server. One database driver is available for each of the RDBMS products that LN supports.

Database tier

Usually, the database tier consists of a third-party database server product. Communication between the database driver and the database server is tailored to the RDBMS in 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 four third-party RDBMS products: Oracle, IBM DB2, or 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 residing in one database server and some residing in another.

Data flow in the architecture

Note that the database driver provides an interface between the LN application virtual machine and the specific RDBMS server being used. The flow of data through the system is described later in this section.

When 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 application virtual machine receives, the virtual machine causes the appropriate application object to be executed.

When a running application object requires information that is 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, an RDBMS-independent SQL language.

When the application virtual machine executes 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 the running database driver instances communicate with a database

server other than the server that stores the required data, the application virtual machine starts a new instance of the database driver. The application virtual machine parses the LN SQL database query it 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 translates the database query into an appropriate database-specific query using SQL statements compatible with the RDBMS being used.

Each database driver takes advantage of the design of the particular RDBMS that the driver supports, so that 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.

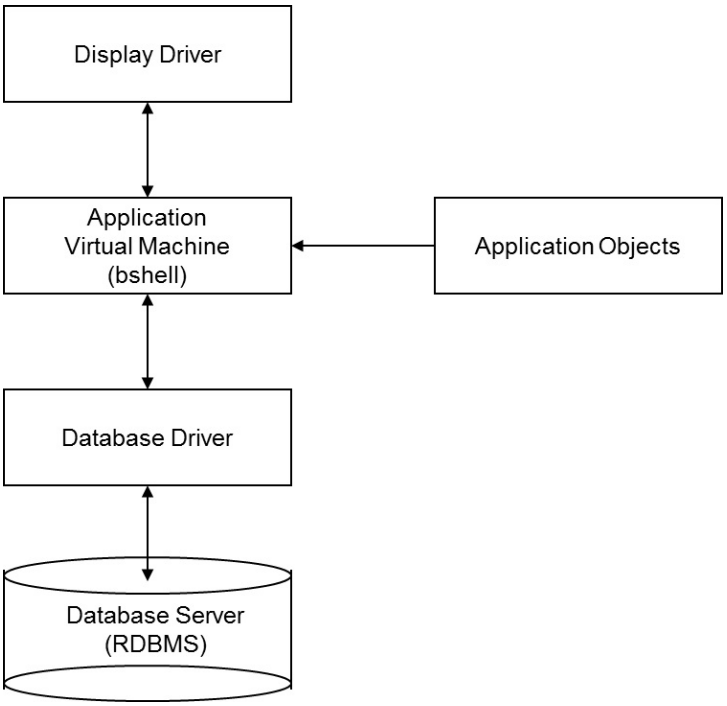
After the RDBMS processes the query, the RDBMS returns the data to the database driver. The database driver catches and handles any error conditions. The database driver then returns the data and status information to the application virtual machine, where the driver provides the information to the application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

Hardware configurations

Several hardware configurations are supported for an LN implementation, including 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, a user can work from the host machine or a thin client machine. However this is still possible, it is not commonly used anymore.

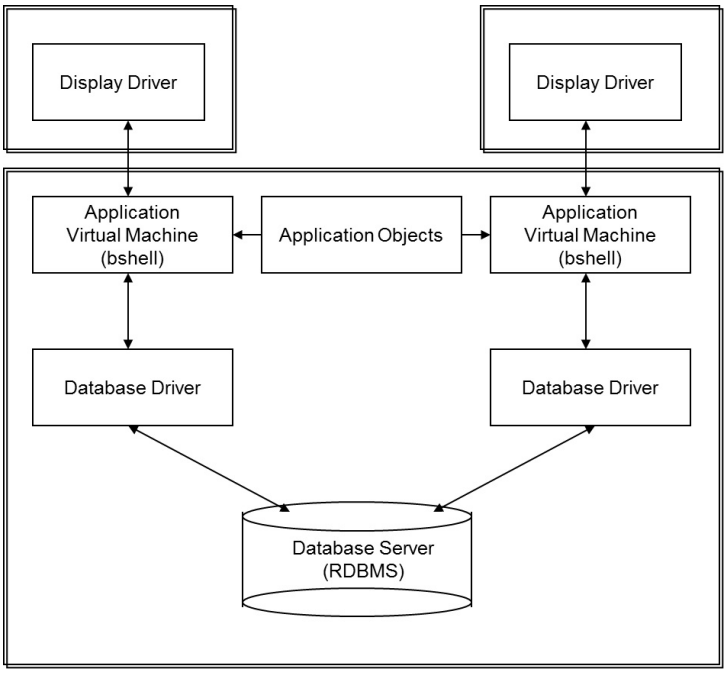
This figure illustrates the standalone mode configuration:



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

The simplest client/server configuration is sometimes thought of as 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 is started for each user. This configuration is also known as 2-tier or host-mode.

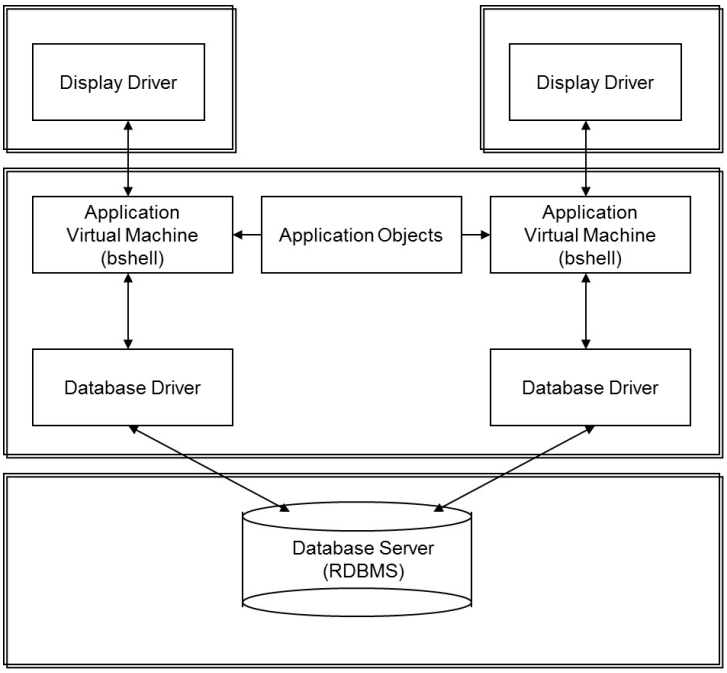
This figure illustrates this 2-tier or host-mode configuration:



If two machines are to be used as servers, two configurations are commonly used. In both configurations, the display drivers reside on the user workstations.

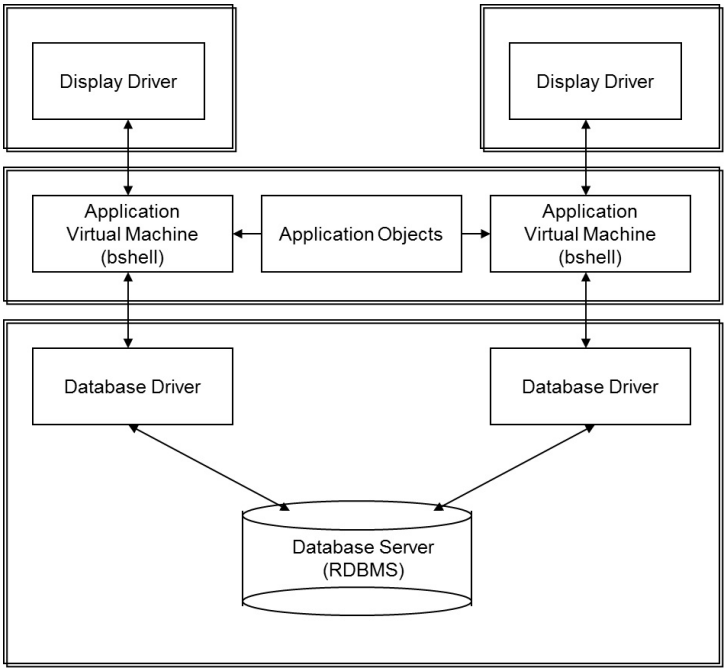
In the first configuration, the application tier and database driver are placed on one server, while the database server is placed on another. An instance of the application virtual machine and at least one instance of the database driver is started for each user. All users have access to the same application objects and database servers. This configuration is also known as 3-tier.

The following figure illustrates this client/server configuration. To provide client/server access, this configuration uses the RDBMS's networked client functionality.



In this second configuration, the application tier is placed on one server, while the database driver and 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 is started for each user. All users have access to the same application objects and database servers.

This client/server configuration is illustrated in the following figure. This configuration uses the LN method of client/server access between the application virtual machine and the database driver.



Many other configurations of client/server systems are available, including dividing the application logic among multiple servers or using multiple servers to distribute the database.

Note:

The LN installer does not support this configuration; therefore, you must manually set up this configuration.

Level 2 database drivers

Infor RDBMS database drivers are communicating with the RDBMS server in so called level 2 mode. This mode distinguishes itself from Level 1 in the way the LN SQL queries are processed and in the way the data is stored in the database. The Level 1 driver is obsolete.

A level 2 database driver sends more complex queries to the database server, which requires the RDBMS to do a larger portion of the work. An application program written for a Level 2 database driver typically uses set-oriented database access.

Chapter 2: Database Organization

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

Because many tables are required, a convention is used for naming tables, columns in tables, and indexes to data in the tables. The data dictionary and naming conventions that the LN database drivers use to access data stored in the RDBMS are described. Also is discussed how LN data types are mapped to DB2 data types.

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 the data, or metadata. You can use a data dictionary to find data that resides in a database table.

The LN database drivers maintains a data dictionary because the data that the 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. When storing or retrieving data in the RDBMS, the database driver maps data dictionary information to database table definitions.

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

The LN data dictionary cannot be directly used by the database driver to create DB2 tables because not all LN data types exactly match DB2 data types or limits. To create valid DB2 tables, the driver must perform some mapping or translation. When mapping the LN data dictionary to tables in DB2, conventions are used for the table names, column names, and index names.

Table naming convention

The table name of an LN table stored in DB2 has the following format:

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

This list describes each of the components of the table name:

- 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`.

- DD Table Name:

  The name of the table used in the data dictionary. The data dictionary table name consists of three letters followed by three digits. The letters refer to the application that uses the table, and the digits indicate the order in which the tables were created.

- Company Number:

  In LN, three-digit company numbers are used to differentiate areas of functionality. A company must exist with the number 000. Several other company numbers must also be present.

For example, the data dictionary table `adv999` with company number `000` is created in DB2 as `tt tadv999000`.

Note:

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

```
s<Package><DD Table name><Company number>
```

For tables with the Document Authorization (also known as DBCM) feature enabled, another secondary table exists named:

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

Column naming convention

Each column in the LN data dictionary corresponds to one or more columns in a DB2 table.

The rules for column names are:

- General:

  When an LN column name is created in DB2, the column name is preceded by the string `t_`. For example, the LN column with the name `cpac` is created in DB2 with the name `t_cpac`. By preceding column names with `t_`, you can avoid reserved words. If a column name contains a period, the period is replaced by the underscore character.

- Long string columns:

  LN columns of type string can exceed the maximum length of character columns in DB2. The DB2 data type CHAR has a limit of 254 characters. This limit is different (larger) for Unicode installations.

When an Infor ERP string column exceeds this limit, the column is split into segments with up to 254 characters each. The first 254 characters are placed in a column where the name of the column is extended with `_1`. The next 254 characters are placed in a column with a name extended by `_2`, etc. until all the characters of the string are placed in a column. For example, if an LN string column called `desc` contains 300 characters, these two DB2 columns are created:

- `t_desc_1`: size 254
- `t_desc_2`: size 46

- Array columns:

  In the LN data dictionary, array columns can be defined. An array column is a column with multiple elements in the column. The number of elements is called the depth. For example, a column that contains a date can be defined as an array of three elements: a day, a month, and a year. In DB2, 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 called **date** becomes:

  - `t_date_1`: element 1
  - `t_date_2`: element 2
  - `t_date_3`: element 3

  Note that if the element is of type string, and one element type exceeds the maximum DB2 character size of 254, the element is split, such as:

  - `t_str_1_1`: element 1, part 1
  - `t_str_1_2`: element 1, part 2

- Array compression:

  The maximum number of DB2 columns is 500. If the number of LN columns exceeds the maximum number of DB2 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 DB2 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 then continues to compress array columns until the number of columns is less than 500. The name of the compressed column in DB2 follows the same convention used for the other columns, such as:

  - `t_array`: contains all elements of the compressed column.

  Note that when a compressed array column is longer than 254 characters, the column is split into segments of 254 characters or less.

Index naming convention

LN indexes are identified by a sequence number for each table; the sequence numbers start from one. Each table has at least one index: the primary index. DB2 requires that index names must be unique. For this reason, the table name and index number are included in the index name.

Index names have tis format:

```
i<Table name>_<Index number>
```

For example, the index name for an LN table with name `ttadv999`, index number `1`, company number `000` is:

`ittadv999000_1a`

If an LN index is defined as a unique index, the DB2 index is created with the UNIQUE clause, which prevents duplicate rows from being created in the table.

Note:

- when a table has one or more BLOB columns this additional index is created:

```
i<DD table name><company number>_UUID
```

- Indices are at default bidirectional and have ‘a’ as suffix in the index name.

Data type mapping

This table shows the mapping between LN data types and their DB2 counterparts:

Mapping between LN and DB2 data types	
LN data types	**DB2 data types**
CHAR	SMALLINT
ENUM	SMALLINT
INT	SMALLINT
LONG	INTEGER
UTC DATE	DATE
UTC TIME	TIMESTAMP
TEXT	INTEGER
BITSET	INTEGER
FLOAT	DOUBLE
DOUBLE	DOUBLE
STRING(N)	CHAR(N)/VARGRAPHIC(N)
MULTI-BYTE STRING	CHAR(N)/VARGRAPHIC(N)
DATE	DATE
RAW	CHAR FOR BIT DATA
BLOB4	BLOB

- The VARGRAPHIC data type is used in Unicode mode and not available for Infor Baan IVc.

- The DB2 CHAR FOR BIT DATA type is used to store UUID columns, generated if a table contains a BLOB column (except in Baan IV where the UUID column is a string column).

In Unicode mode, DB2 driver uses the VARCHAR data type for the LN string type. Otherwise, the DB2 driver uses the CHAR data type. At LN application level, the data type is interpreted as normal string type, including any trailing spaces. Operations such as comparison and concatenation remain the same for earlier versions, regardless of the string data-type that is used at DB2 level.

LN converts multi-byte data into TSS (Triton Super Set) format for internal processing.

Multi-byte data can be stored in the database in TSS format or in the original native format. For storing in TSS or native format, the DB2 data type CHAR is used. Note that char data will be sorted on byte order because Infor databases are built with COLLATE USING IDENTITY. Also, the correct locale and code page must be specified when creating the Infor database for your multi-byte environment to function properly.

Unicode support

Besides native and multibyte support, the DB2 database driver supports Unicode, starting with porting set 8.4b. Unicode is supported only for LN.

The differences between native and multibyte:

- For native/multibyte installations, all string types use VARGRAPHIC instead of CHAR.
- The data type VARGRAPHIC is used because GRAPHIC has more limitations than the CHAR data type. Because GRAPHIC is equivalent to UTF-16 in the DB2 RDBMS, the limit of GRAPHIC data type is 127 characters (=254 bytes). The VARGRAPHIC data type is used to prevent performance loss caused by many long string splits.
- The VARGRAPHIC limit is 16336 characters (=32672 bytes). If this limit is exceeded, long string columns are spitted into parts.
- Unicode databases in DB2 created during installation are created with a Unicode aware collation. This is mandatory when using the DB2 driver in Unicode mode.

Additional data base objects

For the Document Authorization feature these data base objects are required:

- The table tttocm999000 that holds an entry for each checked out object;
- The sequence SEQ_rcd_seqn, which is needed for cursor stability.

Additional constraints

Besides the naming conventions and data types previously described, when you map LN data to DB2 data, these rules apply:

- All names generated by the database driver are in lowercase characters and are not enclosed in double quotation marks. DB2 treats object names with case sensitivity.
- All columns created by the DB2 driver have the NOT NULL constraint. LN applications do not support NULLS.
- If an LN index is defined as a unique index, the DB2 index is also created with the UNIQUE clause; otherwise, indexes that permit duplicates are not defined with the UNIQUE clause.

The date range for the LN application virtual machine is the same as the range for DB2, with one exception: the application virtual machine date 0 is mapped to the DB2 date 1. The application virtual machine date 1 is marked as an invalid date.

Chapter 3: Database Driver Internal Processing

The DB2 database driver converts RDBMS independent database requests into specifically-designed requests for DB2.

These LN DB2 driver internal issues are discussed:

- Data integrity
  Several features ensure the data integrity.
- Database driver SQL processing
  The internal processing of an SQL statement in the driver.
- Setting driver behavior.
  The mechanisms 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. Data integrity is also maintained while you use data buffering techniques to minimize network traffic. This section provides an overview of the features used by the LN DB2 driver to ensure referential integrity, to work with distributed databases, and to apply data buffering techniques.

Referential integrity

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

Data buffering

The application virtual machine can buffer updates and then flush the updates at the time of the transaction commit, or 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 sent back to the LN application virtual machine as one block. To compact the data, data reduction and compression is applied, which minimizes the amount of data transferred between the application virtual machine and database driver.

Database driver SQL processing

As discussed in 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 decides whether to convert these RDBMS-independent database requests into SQL statements appropriate to the specific RDBMS being used. This section details the SQL processing the LN DB2 driver performs. Because the LN DB2 driver uses the call-level interface to communicate with DB2, this chapter first describes this interface.

Call-level interface

To communicate with DB2, the DB2 driver uses the call level interface (CLI). CLI is a function library or a set of C functions that can be called from a C program to run SQL statements.

The CLI functions that are called by the DB2 driver perform for example these actions:

- Log onto DB2 (open session).
- Allocate a statement handle.
- Parse a SQL statement.
- Bind input variables.
- Bind output variables.
- Execute the SQL statement.
- Fetch the resulting rows.
- Break a query (asynchronous).
- Commit/abort transaction.
- Close a cursor.
- Log off from DB2 (close session).

The DB2 driver also uses these features of CLI:

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

SQL processing

The database-dependent layer of the LN DB2 driver dynamically generates SQL statements. Because LN applications are dynamic, the tables that will be used at runtime are not known in advance; therefore, you cannot prepare the queries before runtime.

In the LN DB2 driver, SQL statements are processed in several steps, which are described in this section.

If the LN DB2 driver receives a query from the application virtual machine, the query is translated into a format suitable for DB2, and is passed to DB2 by CLI function calls. In DB2, a statement handle/cursor is allocated, and the query is executed by assigning the statement handle to the query. The SQL statement is parsed, input and output variables are bound, and the query is executed using the statement handle. Sometimes, DB2 internally opens a server cursor for query execution. After the query is executed, a fetch operation is completed and the resulting column values are placed in the bound output variables.

The rows returned by DB2 are passed to the database-independent layer of the LN DB2 driver, which sends the results back to the application virtual machine.

If a statement must be re-executed, the cursor from the previous execution is closed and the resulting rows are discarded, regardless of whether the re-execution is with the same input parameters. If new input values are required, the new values are assigned to the input parameter columns, and the query is re-executed. However, for re-execution, no re-parse of the statement or re-bind of input and output parameters is required; this improves the overall performance.

If array fetching is enabled, multiple rows are fetched in one call to the driver. Space is allocated in the driver to buffer multiple rows fetched in one operation. Multiple rows can be fetched to the buffer, and are returned to the application virtual machine when requested. If no rows are left in the buffer and more rows are requested, another array fetch operation is executed.

Inserts can also be buffered. When array inserting is enabled, the driver places the rows to be inserted in a buffer. When the buffer is full, or if required because of another event, the rows are flushed to DB2. The rows in the buffer are inserted with a multi-row insert.

Note:

To manually enter data in the database, use the LN utility `bdbpost`. Use this utility to place data into a new database table or to append data to an existing database table. When you use `bdbpost`, you can set particular options.

If you use `bdbpost`, by default, the rows are buffered and are flushed when the array buffer is full. You must specify the array size, otherwise buffering is not carried out. Specify the array buffer size in the file `db2_storage_param` on a per table basis, or globally, using an environment variable or resource variable. The following sections describe the environment variables, resource variables, and parameter files.

Setting driver behavior

Several facilities can configure the LN DB2 driver, the most common being through driver resources. Two other facilities are environment variables and the parameter files. The driver resources and environment variables are described in more detail in Driver Resources and Environment Variables on page 43, while the parameter files are described in Parameter File Formats and Configuration Options on page 61.

Driver resources

The database driver resources are parameters you can set to modify the behavior of the LN DB2 driver. These parameters are set in a file called the resource file (`db_resource`). One resource file is available for all database drivers that run in an LN environment. Resources for the database driver offerings can be set there. The database driver reads the parameters set in the resource file when it is first invoked and uses these settings to configures itself.

The resource file can contain multiple entries, with one entry per 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 must be set.

The following is an example of the contents of a resource file that contains two entries:

```
db2_retained_cursors:50
db2_max_open_handles:100
```

When modifying the behavior of the database driver, you must frequently modify the behavior of the LN application virtual machine to take advantage of the characteristics of the database driver. Therefore, two types of database driver resources are available:

- Resources used to modify the behavior of the database driver.
- Resources used to modify the behavior of the database driver's client (for example the application virtual machine).

Driver resources used to modify database driver behavior are called resources for the server. Driver resources used to modify behavior in the application virtual machine are called resources for the client.

The `db_resource` file is located in the folder where the LN software environment is installed.

- On UNIX: `$BSE/lib/defaults`
- On Windows: `%BSE%\lib\defaults`

$BSE and %BSE% refer to the folder in which the LN software environment is installed.

When the database driver and the application virtual machine run on the same machine, one `db_resource` file contains all required resource parameters.

When the database driver and the application virtual machine run on separate machines, a `db_resource` file is required on the server that runs the:

- Database driver containing the server resources.
- Application virtual machine containing the client resources.

Besides the default resource file `db_resource`, you can set up an alternative resource file to override resource values for specific users or groups of users. The alternative resource file is specified with the environment variables USR_DBS_RES and USR_DBC_RES. You can use the USR_DBS_RES variable to specify the path to a file that contains an alternative resource file for the server. You must set this variable on the machine that runs the database driver. Specify the path to a file that contains an alternative resource file for the client in the USR_DBC_RES variable. Set this variable 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`.

Environment variables

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

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

When a database driver environment variable for the client is to be used, to override the corresponding driver resource, you must set the variable on the machine that runs the application virtual machine.

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. To set the database driver server environment variables, three methods are available:

- Use the LN **Database Definitions (ttaad4510m000)** session and **Tables by Database (ttaad4111m000**) session.
- Manually modify the LN tabledef6.x (tabledef6.1 for Baan IV, tabledef6.2 for Baan 5.0 and LN) file.
- Use the standard operating system mechanism.

The recommended method to modify database driver behavior is the LN Database Definitions session. When specific tables and companies must be configured for access with a specific database driver, 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.x, which is in the UNIX directory `$BSE/lib/` or the Windows directory `%BSE%\lib\`. This file resides on the machine where the database driver runs. While we recommend that you use the Database Definitions session to maintain this file, advanced users can manually modify this file. The format of the tabledef6.x file is as follows:

```
<table name>:<company number>:<driver type>(<env var>=<value>)
```

If multiple environment variables are to be specified for a single table and company number, these variables are listed in parentheses, and separated by commas. If all tables or all companies are to be specified, you use the asterisk (*) in place of the table name or company number. For example, you can make the following entry in the tabledef6.x file:

```
tccom010:812:db2(DB2PROF=0.4)
```

In this example, all the queries on table tccom010812, which require at least 0.4 seconds, are logged in the `DB2PROF` file. Note that this table is considered to have a different database definition from other tables.

If a DB2 driver is already running, but accesses a different table, a separate driver will be started for this table.

Environment variables that appear in the driver specifications of the tabledef6.x file are placed into the driver's environment before the file is invoked, so the variables are available to the driver at startup.

If the default database driver resources must be overridden for specific users, use environment variables. 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 the database driver environment variables that affect the client. You must set these environment variables on the machine that runs the application virtual machine, and they must be set using the standard operating system methods used to set environment variables. Any client environment variables used override the equivalent resource variables set for the client in the db_resource file.

Storage parameter file

The storage parameter file provides a way to specify the distribution of table and index data in various segments. The database driver uses storage parameters whenever you execute a DDL statement, such as a create table or create index statement. This is an example of an entry in the storage parameter file:

```
*:*:*:T_SPACE dataspace I_SPACE indexspace
```

In this example, the database driver adds the 'in `<tablespace name>` index in `<index tablespace name>`' clause to the create statement during table creation.

A storage parameter file is defined for each database driver. The storage parameter file for the LN DB2 driver is called `db2_storage_param` and is located in the UNIX directory `$BSE/lib/db2` or the Windows directory `%BSE%\lib\db2`. The format of the storage parameter file is described in Parameter File Formats and Configuration Options on page 61.

The driver scans storage parameter and driver parameter files top-down. The **`first`** matching line is used (not the **`best`** matching line).

Note:

Infor Baan IVc has one single file, called `db2_storage`, for both storage parameters and driver parameters, see Parameter File Formats and Configuration Options on page 61.

Chapter 4: Database Security

To maintain security, the LN DB2 driver controls user access to the database and database objects. The LN database administrator (DBA) module uses LN sessions to enable the DBA to control access to the database. Using the DBA module makes DBA tasks easier and less prone to errors than directly using database driver tools.

Security

Database security comprises 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 DB2 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 management of a large number of users with common requirements.

An LN group consists of a database name and methods for providing object security and authentication in the database. The LN group name is the same as the name of the database that contains 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 because an LN group includes the RDBMS group and also the database name and an RDBMS login.

In DB2, an LN group consists of three components: a database, a login (for authentication), and an operating system group (for object security). The DB2 database has the same name as the LN group. The login is the same name as the LN group, and is assigned database owner privileges in the database.

An operating system group is created, which becomes the target for privileges granted on objects in the database. Users are associated with the operating system group and, therefore, inherit the privileges granted to the group. The advantage of having a group table is that the members of the group can share and operate on the same data in a single table.

Note:

On Windows, the login for the LN group is always set to `baan`. The operating system group will have the same name as the DB2 database, and the user **`baan`** must belong to this group.

For example, users **`Maria`** and **`John`** can both be assigned to LN group `baandb`. Group `baandb` owns the tables and grants select, insert, delete, and update privileges to the operating system group. Therefore, users **`Maria`** and **`John`** inherit the select, insert, delete, and update privileges granted to the operating system group, 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 use the RDBMS groups. Only the database administrator must be concerned about the RDBMS groups. The administrator can easily maintain the RDBMS groups with the LN DBA module.

Object security

In DB2, if a user creates an object such as a table, the user becomes the owner of that object, and only the owner can access the object. Other users can only access the object if these users have been granted privileges to do so. In an LN environment, in which many users access the same tables in the DB2 database, a mechanism has been developed to allow users to share these tables.

To allow multiple LN users to share the same DB2 table, the group concept is used. An LN group maps users to a database in DB2 and ensures that members of the group have sufficient privileges to access data in the LN group's tables.

To implement the LN group concept, the LN DB2 driver uses an operating system group. Initially, the group is granted `connect` and `createtab` privileges to the database. Whenever the LN group user creates a new table, select, insert, delete, and update privileges are granted to the operating system group.

Any user associated with the operating system group automatically inherits these privileges and can individually perform these operations on the LN group table.

When new users are added, these users only have to be associated with the operating system group. These users automatically inherit all privileges currently granted to the operating system group, without the need to grant privileges on every group object in the database to the user. When the user is dropped from the operating system group, these privileges are revoked and the user no longer has access to tables in that operating system group. If the privileges to operate on the tables are explicitly granted to the user, these privileges must also be explicitly revoked when the user is dropped from the operating system group. The overhead of adding users is greatly reduced by granting privileges to the operating system group. This also provides flexibility and ease of maintenance.

Authentication

To determine whether a user is authorized to access a database, DB2 uses the operating system's authentication mechanism. When a database has been created, an administrator or DB2 database owner grants access permissions to an operating system group. Access permission can be granted for the entire database or for specific database objects.

LN users mapped to operating system users are permitted to establish a connection to DB2 with their own user name and password. To prevent unauthorized users from accessing the database, non-mapped users cannot establish a connection to the database.

The LN group user corresponds to the target database. The group name is the same as the database name. The members that belong to this group inherits the group privileges and can establish a connection to the database that uses a valid password stored in encrypted form in the driver administration files.

To add or drop a user from an LN group, use the LN Database Administration (DBA) module or the `db2v5_maint` utility. Note that any DBA module or `db2v5_maint` action will not affect the operating system user or group. This action only affects the LN administration files. For example, to change a user's password, you must change the user's operating system password and then change the password in LN using the DBA module or `db2v5_maint`.

Users authorized to access the database are registered in the LN driver administration files. The user name and password each LN user will use to log onto DB2 is maintained in the UNIX `$BSE/lib/db2_users` file or the Windows `%BSE%\lib\db2_users` file.

All the LN users and their corresponding DB2 logon names, passwords, and the name of the group to which the users are assigned are defined in the `db2_users` file. The format of each entry in this file:

```
<Infor ERP user>:<DB2 user>:<Encrypted DB2 User password>:
<Infor ERP group name>
```

The LN application virtual machine starts the LN DB2 driver on behalf of the user. From the `db2_users` file, the driver identifies the DB2 user and the user's password, and then establishes the connection to DB2.

The group logon procedure also includes a password, which is defined in the UNIX file `$BSE/lib/db2_groups` or the Windows file `%BSE%\lib\db2_groups`. The format for these files:

```
<Group name>:<Encrypted group password>
```

The group name is the same as the database name.

DBA module

The DBA module maintains the database administration files used by the DB2 database driver. This module allows an administrator to register authorized users and provide users with access to data. This tool is provided with the LN DB2 driver to maintain the administration files the database driver

requires at runtime. The administration files are stored in the directory `$BSE/lib/db2` for UNIX or `%BSE%\lib\db2` for Windows.

The DBA module implements the user and group administration functions for all LN database drivers. The `db2v5_maint.6.2` utility is an executable program, called by the DBA module that implements the functions required to make changes in DB2. You can call the `db2v5_maint.6.2` utility from outside the DBA module, but this is not recommended, because the users and groups tables are not modified by `db2v5_maint.6.2`.

For more information about the DBA module, see *Infor Enterprise Server - Administration Guide*.

Database maintenance db2v5_maint tool

The DBA module performs administration tasks by invoking `db2v5_maint`. Although the user can directly call `db2v5_maint`, we recommend that you enable the DBA module to invoke `db2v5_maint`.

This section describes the administrative tasks the `db2v5_maint` utility performs, and the options and arguments of these tasks. Unless the options are listed in brackets, all options listed must be specified. For options in braces, {}, you must specify one set of the options in the brackets. Options in square brackets [ ] are optional.

Four additional options, –b, -q, -n, and –F, are available for each function. These options are defined as:

- -b: Processes in batch mode.
- -q: Redirects error messages to a specified error file.
- -n: Will not update LN files `db2_users` or `db2_groups`.
- -F: Validates the user, group, or password, but will not make changes.

The administrative tasks the `db2v5_maint` utility performs are shown in this list:

- Adds a user to a group:

```
db2v5_maint [-n] –a<Infor ERP user> –m<DB2 user> –p<DB2 password>
–G<Group name> {-F –c<dba> -i<dba password> | –P<Group password>}
```

- Removes a user from a group:

```
db2v5_maint [-n] –d<Infor ERP user> –G<Group name>
{–P<Group password> | -F –c<dba> -i<dba password>}
```

- Adds a group:

```
db2v5_maint [-n] –A<Group name> –P<Group password>
[-F –c<dba> -i<dba password>]
```

- Removes a group:

```
db2v5_maint [-n] –D<Group name>
{–P<Group password> | -F –c<dba> -i<dba password>}
```

- Changes a user password:

```
db2v5_maint [-n] -r<Infor ERP user> -m<DB2 user> -p<new password>
```

- Changes a group password:

```
db2v5_maint [-n] -R<Group name> -P<new password>
```

- Checks a user password:

```
db2v5_maint -h -m<DB2 user> -p<DB2 password> -G<Group name>
```

- Displays version information:

```
db2v5_maint {-v | -V}
```

- Displays usage:

```
db2v5_maint {-U | ?}
```

- Sets environment variables:

```
db2v5_maint -O<list of environment variables and their settings>
```

Chapter 5: Database Driver Profiling and Statistics

The LN DB2 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

With the database driver you can log timing aspects and statistics. This functionality is useful when tuning your system. The information can help identify performance bottlenecks.

The database driver's profiling option provides a way to gather the timing of SQL statements being executed. However, logging all statements with their timings results in a log file too large to be correctly 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, set the DB2PROF environment variable to a number of seconds. For example, you can set DB2PROF as follows:

```
export DB2PROF=5.0
```

This command sets DB2PROF to five seconds, which causes statements that take more than 5.0 seconds of elapsed time to be logged to the file db2prof. The file db2prof is stored in the directory where the driver was started.

To view statement execution time for individual tables, the user can set the DB2PROF environment variable in the UNIX file $BSE/lib/tabledef6.2 or the Windows file %BSE%\lib\tabledef6.2. For example, you can make the following entry in the file:

```
tccom010:812:db2(DB2PROF=0.4)
```

In this example, all queries on table tccom010812, which require more than 0.4 seconds, are logged in the db2prof file. Note that a separate driver is started for this table. The table is considered to have a different database definition.

Profiling example

When you perform profiling, each phase in the SQL query processing that exceeds the profiling value is printed. The following example text provides a sample DB2PROF file:

```
1999-04-01[11:49:04]: Profiling value = 2.00 sec
------ Profiling value exceeded -----
<user001><tdsls4401m000>:1999-10-08[22:10:32.560]:
Time (exec) : 7.661545 seconds
SQL statement:
SELECT a.t_acti,a.t_akcd,a.t_amld,a.t_amod,a.t_bind,a.t_bkyn,a.t_bqco,a.t_bqua,a.t_ccty,
a.t_cdis_1,a.t_cdis_2,a.t_cdis_3,a.t_cdis_4,a.t_cdis_5,a.t_cfrw,a.t_chan,a.t_citg,a.t_ci
tt,a.t_clot,a.t_clyn,a.t_cmnf,a.t_comq,a.t_coop_1,a.t_coop_2,a.t_coop_3,a.t_copr_1,a.t_c
opr_2,a.t_copr_3,a.t_corn,a.t_corp,a.t_cors,a.t_cosn,a.t_cpcl,a.t_cpcp,a.t_cpln,a.t_cprj
,a.t_cpva,a.t_crcd,a.t_crte,a.t_csgs,a.t_ctcd,a.t_ctrj,a.t_cups,a.t_cuqs,a.t_cvat,a.t_cv
ps,a.t_cvqs,a.t_cwar,a.t_cwoc,a.t_damt,a.t_ddta,a.t_disc_1,a.t_disc_2,a.t_disc_3,a.t_dis
c_4,a.t_disc_5,a.t_dldt,a.t_dmde_1,a.t_dmde_2,a.t_dmde_3,a.t_dmde_4,a.t_dmde_5,a.t_dmse_
1,a.t_dmse_2,a.t_dmse_3,a.t_dmse_4,a.t_dmse_5,a.t_dmth_1,a.t_dmth_2,a.t_dmth_3,a.t_dmth_
4,a.t_dmth_5,a.t_dmty_1,a.t_dmty_2,a.t_dmty_3,a.t_dmty_4,a.t_dmty_5,a.t_dqua,a.t_drct,a.
t_dtrm,a.t_elgb,a.t_fcop_1,a.t_fcop_2,a.t_fcop_3,a.t_invd,a.t_invn,a.t_isss,a.t_item,a.t
_ldam_1,a.t_ldam_2,a.t_ldam_3,a.t_ldam_4,a.t_ldam_5,a.t_leng,a.t_lsel,a.t_lseq,a.t_oamt,
a.t_odat,a.t_ofbp,a.t_oltp,a.t_opol,a.t_opri,a.t_oqua,a.t_orno,a.t_pmde,a.t_pmse,a.t_pms
k,a.t_pono,a.t_prdt,a.t_pric,a.t_pror,a.t_ratd,a.t_ratf_1,a.t_ratf_2,a.t_ratf_3,a.t_rats
_1,a.t_rats_2,a.t_rats_3,a.t_ratt,a.t_rdta,a.t_revi,a.t_scmp,a.t_scon,a.t_sdsc,a.t_shpm,
a.t_sqnb,a.t_stad,a.t_stbp,a.t_stcn,a.t_stdc,a.t_stpr,a.t_thic,a.t_ttyp,a.t_txta,a.t_vco
p_1,a.t_vcop_2,a.t_vcop_3,a.t_widt,b.t_txts,c.t_cadr,d.t_kitm,d.t_dsca,d.t_citg FROM ((
baan.ttdsls401570 AS a LEFT  JOIN ( baan.ttdisa001570 AS b LEFT  JOIN  baan.ttcibd001570
 AS d ON d.t_item = b.t_item) ON b.t_item = a.t_item) LEFT  JOIN  baan.ttcmcs003570 AS c
 ON c.t_cwar = a.t_cwar) WHERE ((CAST(? AS INTEGER)  = CAST(? AS INTEGER)  AND a.t_oltp
= ?) OR (CAST(? AS INTEGER)  != CAST(? AS INTEGER)  AND a.t_oltp != ?)) AND (b.t_item =
a.t_item OR b.t_item IS NULL OR a.t_item IS NULL) AND (d.t_item = b.t_item OR d.t_item
IS NULL OR b.t_item IS NULL) AND (c.t_cwar = a.t_cwar OR c.t_cwar IS NULL OR a.t_cwar
IS NULL) AND a.t_orno >= ? AND a.t_orno <= ? ORDER BY 104,108,126  OPTIMIZE FOR 5 ROWS
```

Gathering statistics

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

- Number of cursors: opened, closed, or 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 DB2STAT. If the variable is set to zero, a statistics report is generated when the driver terminates (exit from Tools or 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 DB2STAT in the current directory.

These are examples of how you can set DB2STAT:

```
export DB2STAT=0
export DB2STAT=30
```

In the first example, DB2STAT is set to zero; with this value, only a final report is generated. In the second example, DB2STAT is set to 30; this logs a report every 30 seconds while the driver is active. The statistics report is logged in the file `db2stat` in the current directory.

This example provides a sample output of DB2STAT.

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
```

In the previous example, the first section refers to fetch cursor actions, the second section refers to row actions, the third section refers to lock actions, the fourth section refers to update cursors, the fifth section refers to table actions, and the sixth refers to transaction or database actions. The first section of the summary lists types of fetches and updates, the second section lists fetch optimization, and the third numbers the cursors that were forced closed, the open cursors, and sessions.

The tuning options to improve performance, as a result of studying the `db2stat` file:

- Index optimization.
- Extend refresh time.
- Separate data from indices in tablespaces.

To check I/O times and process information, you can also use the DB2 `db2evmon` utility. For more information, see IBM DB2 documentation.

Troubleshooting

The LN DB2 driver provides a facility for troubleshooting problems. You can trace the actions the driver performs and store them in a log file. Additionally, 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 the driver performs. The resulting log file contains debugging information that can help solve problems.

If 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, use the environment variable DBSLOG. Debugging information is appended to the `dbs.log` file in the current directory. To enable tracing, specify this command:

```
export DBSLOG=0560
```

To enable tracing strictly for categories of interest, several tracing categories are defined.

Logging errors

In a UNIX environment, the driver logs error messages in the log flies in the folder `$BSE/log`. The log files are called `log.db2` and `log.db2.mesg`.

In a Windows environment, the driver logs error messages under the application logs in the Windows Event Viewer. To see the messages, click **Start**, point to **Programs and Administrative Tools (Common)**, and click **Event Viewer**.

You can retrieve this information from the log files:

- 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, an attempt is made to map the database to some known or anticipated error condition. Generally, these mapped BDB errors have corresponding error numbers that fall within the range of one to 1,000. If a database specific error occurs, the error is mapped to the BDB error code over 1,000 with this formula:

- `abs(error_code) + 1000`

Therefore, if an error -1652 occurs, BDB error 2652 is returned to the application.

Usually, the log entries from the display driver, application virtual machine, and database driver contain enough information to determine the nature of and solution to the problem. Whenever an error is encountered with an error code greater than 1,000, you are advised to check the log entries from the database driver.

Chapter 6: Database Driver Configuration

The LN DB2 driver is designed to permit tuning for optimal performance. Several parameters used by the database driver are preset with default values to provide acceptable system performance. Because every environment is different, the default values of these parameters probably do not provide an optimal performance. You can specify the LN DB2 driver parameters to your own requirements.

Cursor management

The LN DB2 database driver has two resource variables that influence the cursor handling: `db2_max_open_handles` and `db2_retained_cursors`. This list describes these resource variables:

- `db2_max_open_handles`:

  This resource influences the number of statement handles (DB2 cursors) that are kept for reuse by the driver. The default value is 100.

  Whenever the application closes a query, most resources in the driver are freed. If the DB2 cursor being allocated is quickly reused, closing and opening of the cursor can be avoided. To avoid closing and re-opening cursors, the cursor is placed in a free list that contains all cursors, which are disassociated from a query. When a new cursor is requested, a free cursor is taken from the free list. If no free cursors are available, a new cursor is opened. In this way the number of times a cursor is closed and opened is reduced. When the number of all open cursors exceeds the value of `db2_max_open_handles`, the DB2 cursor is closed.

- `db2_retained_cursors`:

  With this resource, you set the number of inactive cursors and reduce the number of open cursors.

  If a query fetches all rows, a cancel is issued on the query. This means that DB2 is notified that no additional fetches will be performed. DB2 can now free particular query resources. After the cancel, you can repeat the query without reopening (parsing, binding) the cursor. The driver does not know if a cursor in cancel state is reused later. In a worst case scenario, the cursors are not reused, and the cursor continues to be reserved for the query.

  After all rows are fetched, the driver has a facility to place inactive cursors (in cancel state) in a cancel list. These inactive cursors become candidates for being assigned to a different query. The number of inactive cursors to be kept in this list is defined by the resource `db2_retained_cursors`, which defaults to 20.

  If more than 20 cursors are in the cancel list, and a request for a new cursor is issued, the free list is checked. If a cursor exists on the free list, the cursor is used for the new cursor.

If no cursors exist on the free list, 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 does parsing and binding on this cursor. When the original query is re-executed, the driver detects that the cursor was associated with another query. The driver gets a new cursor, and reparses and binds the query again.

If you increase the value of `db2_retained_cursors`, less reparsing and rebinding of queries occurs, which reduces CPU resources. The number of open cursors and memory is increased.

Array interface

The LN DB2 driver can use the DB2 array interface for array fetches and array inserts. With the array interface, communication between the LN DB2 driver and DB2 is more efficient, multiple rows are fetched or inserted simultaneously. Because multiple rows are stored in a buffer in the LN DB2 driver, more memory is consumed. Array interfacing is useful if you access a remote database, because this helps reduce the number of network round-trips.

To adjust the size of the buffers that hold array rows, you must set the `db2_max_array_size` resource variable or the `DB2_MAX_ARRAY_SIZE` environment variable.

You can enable/disable the array fetch interface with the environment variable `DB2_ARRAY_FETCH` or the resource variable `db2_array_fetch`. You must set the array size; otherwise, the result rows are not buffered.

You can enable the array insert interface with the environment variable `DB2_ARRAY_INSERT` or the resource variable `db2_array_insert`.

By default, array inserts are enabled when data is placed in the database tables using the `bdbpost` utility with the **`-f`** option.

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 performed by the DB2 driver.

Concatenated expressions

LN can use concatenated expressions, which operate on a combined column. The following are concatenated expressions that exist on combined columns:

- select >=

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

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

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

Here, for 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 DB2 driver can allow DB2 to solve the WHERE clause using three different techniques: nested, iterative, and filter. These techniques are introduced because the DB2 optimizer cannot efficiently handle these queries in all situations. Instead, full table scans and sort operations can be introduced for these queries.

If you specify a different technique, this causes the DB2 optimizer to make different decisions on how to execute a query.

This method also provides some workarounds for typical optimizer behavior; however, the optimizer behavior can change in different DB2 versions. Therefore, you might be required to tune each version differently, which makes giving guidelines difficult.

To detect long running or bad performing queries, use the DB2PROF variable; subsequently, you can experiment with these various techniques. For a description of DB2PROF, see "Driver Resources and Environment Variables on page 43".

This list describes the nested, iterative, and filter techniques; these techniques are set in the index optimization field of the db2_driver_param file. For more information, see "Parameter File Formats and Configuration Options on page 61":

- Nested technique:

  The three conditions are ORed to the following 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 referred to as the nested technique.

- Iterative technique:

  To resolve one query, multiple SQL statements are issued; these statements do not contain OR conditions, therefore, DB2 can efficiently handle these statements. You can only use the iterative technique for unbounded queries.

The iterative technique uses these conditions:

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

First, a query with condition 1 is issued. When the query fails to return a row, the query continues with condition 2. When condition 2 does not return a row, the query continues with condition 3. Additionally, if one condition returns all rows, but more rows are required, the driver continues with the next condition.

- The filter technique:

  This technique is related to the nested technique, but with 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".

Optimistic and pessimistic reference checks

To optimize concurrency, the LN DB2 driver supports optimistic and pessimistic reference checks. In lookup reference mode, when inserts are performed 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 it within the current transaction. This approach is called the pessimistic approach.

This approach blocks an insert of another user referencing the same parent row, thereby affecting the concurrency. To avoid this problem, an approach is also available in which a row in the parent table that is not locked is used, 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. Enabling of this option is configurable with the `dbsinit` resource variable.

Locking behavior

To read rows from the table, the LN DB2 driver uses the uncommitted read isolation level. Unless explicitly stated in the query, the queries that return more than a single row do not acquire any type of lock, shared or exclusive. Queries such as SELECT FOR UPDATE, INSERT, DELETE and UPDATE acquire exclusive locks when the SQL_TXN_REPEATABLE_READ isolation level is used. Only in case of lookup references are shared locks acquired when the SQL_TXN_COMMITED_READ isolation level is used. The locks are retained until the transaction is committed or aborted.

To ensure that queries do not wait indefinitely on locks, the DB2 database configuration parameter LOCKTIMEOUT must be set to something other than the value -1.

Appendix A: Driver Resources and Environment Variables

You can use the database driver resources and environment variables as configuration parameters to modify the behavior of the DB2 database driver. Some of these resources are used with the client, while others are used with the server. In this context, the client is the LN application virtual machine and the server is the LN DB2 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 can set the resources for the client and server on both machines.

For more information about how to set the database driver resources and environment variables, see Setting driver behavior on page 24

Summary of DB2 resources and environment variables

These types of resources and environment variables are available to use with the LN DB2 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 DB2 driver.

These tables provide a summary of each of these types of resources and environment variables. For detailed descriptions of each entry in the tables, see Detailed description of DB2 resources and environment variables on page 46.

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.
baan_sql_trace	BAAN_SQL_TRACE	Use this variable to view SQL query information.
rds_full	RDS_FULL	Sets maximum number of rows transferred in one block

Client and server resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
tt_sql_trace	TT_SQL_TRACE	Use this variable to view the SQL query information.
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**
bdb_debug	BDB_DEBUG	Sets a debugging link between the client and server
bdb_driver	BDB_DRIVER	Sets database specifications
bdb_max_server_schedule	BDB_MAX_SERVER_SCHED-ULE	Defines mechanisms for terminating idle database drivers
bdb_use_row_version		Use a row version column to optimize delayed row locking.
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
baan_sql_stmt_cache_size	BAAN_SQL_STMT_CACHE_SIZE	Defines the size of the query cache

Server resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
bdb_max_sessions	BDB_MAX_SESSIONS	Defines number of sessions per driver
bdb_max_session_schedule	BDB_MAX_SESSION_SCHED-ULE	Defines mechanism for closing idle driver sessions
dbslog	DBSLOG	Enables driver profiling
	DBSLOG_LOCK_PROF	Specifies lock time above which locks are logged
dbslog_name	DBSLOG_NAME	Enables the file name to be specified for logging
dbsinit		Deprecated
enable_refmsg	ENABLE_REFMSG	Causes logging of denied updates of delete actions

Server resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
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
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.

Resources used only by the LN DB2 driver		
Resource name	**Environment variable**	**Description**
db2_array_fetch	DB2_ARRAY_FETCH	Enables/disables array fetching
db2_array_insert	DB2_ARRAY_INSERT	Defines maximum number of rows for array inserts
	DB2_DUMP_MESG	Specifies location for message log
	DB2_EXPLAIN	Enables Explain Snapshot facility
db2_level1	DB2_LEVEL1	Obsolete.
db2_max_array_size	DB2_MAX_ARRAY_SIZE	Adjusts buffer size for array fetches
db2_max_open_handles	DB2_MAX_OPEN_HANDLES	Sets number of open cursors per driver connection
db2_max_ret_rows	DB2_MAX_RET_ROWS	Sets maximum number of rows returned from DB2
db2_opt_level	DB2_OPT_LEVEL	Sets query optimization level
db2_opt_rows	DB2_OPT_ROWS	Adjusts number of rows retrieved
	DB2_PRINT_ERROR	Enables additional debugging information to be dumped
	DB2PROF	Enables profiling
db2_retained_cursors	DB2_RETAINED_CURSORS	Sets number of cursors in the “break” state per driver connection.
	DB2STAT	With this variable you can gather statistics

Resources used only by the LN DB2 driver		
Resource name	**Environment variable**	**Description**
max_sql_buffer	MAX_SQL_BUFFER	Sets the maximum memory allocated for one SQL statement

Detailed description of DB2 resources and environment variables

This section provides detailed information about the LN DB2 driver resources and environment variables. The driver resources are divided into two sections: those generic to all LN database drivers, and those specific to the LN DB2 driver. Each group of resources is listed 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, internally, the query processor caches for sorting, aggregation functions, or prepared sets. When this limit is exceeded, temporary files are generated. For optimal performance of the internally used hash functions, a prime number must be specified.

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
Type	Integer
Default	330

baan_sql_stmt_cache_size / BAAN_SQL_STMT_CACHE_SIZE	
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	Does not apply to Baan IV and Baan 5.0 application sessions. This variable is introduced to view the LN SQL query information being handled in client and server. When this variable is set, 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 useable for the audience of this document. The `baan_sql_trace` variable has these relevant values: • 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. When set, the client prints debugging information to standard error (stderr). You can specify these categories of debugging information: • 00001: Server types • 00002: Database actions • 00004: Delayed lock actions • 00010: Reference information • 00040: TSS info from `$BSE/lib/tss_mbstore` (UNIX) or `%BSE%\lib\tss_mbstore` (Windows) • 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 bit-wise.

bdb_driver / BDB_DRIVER	
Driver resource	bdb_driver
Environment variable	BDB_DRIVER
Client/Server resource	Set for client only
Type	String
Default	None
Description	With this variable you can set a database specification, usually found in the file `tabledef6.2`. If you set this variable, all tables are accessed using the database driver specified, and `tabledef6.2` is not read. The driver specified must be defined in the UNIX file `$BSE/lib/ipc_info` or the Windows 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. Whenever the database driver has no more open sessions, the application virtual machine can terminate the database driver. Closing an idle database driver is done after several schedule ticks. A schedule tick is generated whenever an LN session is ended. 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 is terminated.

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 is started to handle any 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. Whenever the client process has no more references (cursors or queries) to the session, the client can close the process. After several schedule ticks, you can close an idle session. A schedule tick is generated whenever an LN session is ended. At this point, all idle sessions have a schedule counter incremented. When the value of the schedule counter reaches the value of `bdb_max_session_schedule`, the session is closed The default for `bdb_max_session_schedule` is 3. Setting `bdb_max_session_schedule` to 1 would result in fewer connections from the driver to the RDBMS. This is because whenever an LN session is ended, the corresponding RDBMS session (logon) is closed (logoff).

bdb_use_row_version	
Driver resource	bdb_use_row_version
Environment variable	-
Client/Server resource	Set for client only
Type	Integer
Default	0
Description	If this resource is set to 1, every table that is created is extended with an extra column named `rcd_vers`. The value in this column identifies a row version and is updated by every update or delete action. The column is used to optimize the delayed locking approach, also known as optimistic locking. The value is used to verify that the row was not modified since the delayed lock was placed. **Note:** Changing this resource means that you cannot use the existing tables anymore. A possible, though potentially time consuming, solution is to export all data, change the resource value, and then import all data again.

dbsinit	
Driver resource	dbsinit
Environment variable	—
Client/Server resource	Set for server only
Type	Integer (octal)
Default	0
Description	With this variable you can set flags to specify the optimizations to be used. At this time, legal values are 000 (not set) and 001. Other values are reserved and must not be used. A flag of 00001 specifies that when you check for references in parent tables, an optimistic approach should be used. The referenced row in the parent table is not locked, which improves the overall concurrency. If this flag is not set, optimistic reference checking is not used. For more information, see "Optimistic and pessimistic reference checks on page 41". To define multiple categories, you can add the octal values. To determine if a given category must be logged, the value is compared bit-wise. This parameter is deprecated and is removed in a future release.

dbslog / DBSLOG	
Driver resource	dbslog

dbslog / 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. These debugging categories can be specified: • 0000001: Data Dictionary info of tables within the driver • 0000010: Row action information • 0000040: Transaction action information • 0000100: DBMS input/output data • 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 must add the octal values. To determine if a given category must be logged, the value is compared bit-wise.

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 the appropriate dbslog categories must be set.

dbslog_name / DBSLOG_NAME	
Driver resource	—
Environment variable	DBSLOG_NAME
Client/Server resource	Set for server only
Type	String
Default	dbs.log
Description	You can use `dbslog_name` to specify a file name in which DBS logging information is written. If a file with the same name exists, this 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.

rds_full / RDS_FULL	
Driver resource	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. If more rows are requested, multiple blocks and network round trips are transferred. This variable must be set to the same value for client and server. This setting is ignored when the virtual machine and the driver are running in the same process (combo mode), which is the default mode.

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
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.

ssts_set_rows / SSTS_SET_ROWS	
Driver resource	ssts_set_rows
Environment variable	SSTS_SET_ROWS
Client/Server resource	Set for client only

ssts_set_rows / SSTS_SET_ROWS	
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
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.

tt_sql_trace / TT_SQL_TRACE	
Driver resource	tt_sql_trace
Environment variable	TT_SQL_TRACE
Client/Server resource	Set for both client and server
Type	Integer (octal)
Default	0

tt_sql_trace / TT_SQL_TRACE	
Description	Applies to Baan IV and Baan 5.0 application sessions. For newer application versions, it is replaced by `baan_sql_trace`. With TT_SQL_TRACE you can view the Infor SQL query information that is handled in the client and server. When this variable is set, debug information is printed by the client to the display. The server prints information only if the dbslog variable permits information printing. The information contains several categories, which you can enable separately. The categories are: • Evaluation trees • SQL statements • Bind variables • Timings • Communication debugging The values of the TT_SQL_TRACE variable and their descriptions: • 000040 (c) : Show queries with their QID • 000200 (c) : Show query execution times • 002000 (c) : Show calls of internal SQL functions • 004000 (c+s) : Show query execution tree • 010000 (s) : Show query evaluation plan • 020000 (s) : Show FullTableScan

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	Use this variable to enable or disable the use of shared memory to each of the database driver DDs. Two values are valid for this variable: 0 and 1. If this variable is set to 0, shared memory is disabled; if is 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

USR_DBC_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 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
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`.

DB2 driver specific resources

db2_array_fetch/DB2_ARRAY_FETCH	
Driver resource	db2_array_fetch
Environment variable	DB2_ARRAY_FETCH
Client/Server resource	Set for server only
Type	Boolean
Default	1 (enabled)
Description	This environment variable can enable or disable the array fetch interface. The valid values are 0 and 1. If this variable is set to 0, the fetch interface is disabled; if set to 1, the fetch interface is enabled. For more information, see Array interface on page 39.

db2_array_insert/DB2_ARRAY_INSERT	
Driver resource	db2_array_insert
Environment variable	DB2_ARRAY_INSERT
Client/Server resource	Set for server only

db2_array_insert/DB2_ARRAY_INSERT	
Type	Boolean
Default	0 (disabled)
Description	This environment variable can enable or disable the array insert interface. The valid values are 0 and 1. If this variable is set to 0, the array insert interface is disabled; if set to 1, the array insert interface is enabled. For more information, see table_load_array_size and Array interface on page 39.

DB2_DUMP_MESG	
Driver resource	—
Environment variable	DB2_DUMP_MESG
Client/Server resource	Set for server only
Type	String
Default	$BSE/log/log.db2 (UNIX) or %BSE%\log\log.db2 (Windows)
Description	With this environment variable you can specify the location of the `log.db2.mesg` file. By default, this file is stored in the `$BSE/log` (UNIX) or `%BSE%\log` (Windows) directory. This variable must be the fully qualified file name where error messages will be written.

DB2_EXPLAIN	
Driver resource	—
Environment variable	DB2_EXPLAIN
Client/Server resource	Set for server only
Type	Boolean
Default	0
Description	You can use this variable to enable the Enable Snapshot facility. You can set this variable to these values: • 0: No explain output. • 1: Set CURRENT EXPLAIN SNAPSHOT=yes will be sent to the server to enable the Explain Snapshot facility. The explained information is inserted in the SNAPSHOT column of the EXPLAIN_STATEMENT table. Note that the EXPLAIN_STATEMENT table must have been created earlier.

db2_max_array_size / DB2_ MAX_ARRAY_SIZE	
Driver resource	db2_max_array_size

db2_max_array_size / DB2_ MAX_ARRAY_SIZE	
Environment variable	DB2_MAX_ARRAY_SIZE
Client/Server resource	Set for server only
Type	Integer
Default	1
Description	If the array interface is enabled, this variable defines the maximum number of rows fetched at once from the RDBMS. For more information, see "Array interface on page 39".

db2_max_open_handles / DB2_MAX_OPEN_HANDLES	
Driver resource	db2_max_open_handles
Environment variable	DB2_MAX_OPEN_HANDLES
Client/Server resource	Set for server only
Type	Integer
Default	100
Description	Limits the number of open cursors the driver maintains on a per-connection basis. Each cursor represents one type of SQL statement. A maximum of 200, and a minimum of one open statement handles, are permitted per connection.

db2_max_ret_rows / DB2_MAX_RET_ROWS	
Driver resource	db2_max_ret_rows
Environment variable	DB2_MAX_RET_ROWS
Client/Server resource	Set for server only
Type	Integer
Default	0
Description	This variable specifies the maximum number of rows to be returned to the driver by the DB2 engine. The default is 0, which implies that all rows are returned.

db2_opt_level / DB2_OPT_LEVEL	
Driver resource	db2_opt_level
Environment variable	DB2_OPT_LEVEL
Client/Server resource	Set for server only
Type	Integer
Default	0

db2_opt_level / DB2_OPT_LEVEL	
Description	Sets the query optimization level for SQL queries. The default and recommended value is 0, which means that index scans are used. For possible query optimization class values and their respective meanings, refer to the DB2 documentation.

db2_opt_rows / DB2_OPT_ROWS	
Driver resource	db2_opt_rows
Environment variable	DB2_OPT_ROWS
Client/Server resource	Set for server only
Type	Integer
Default	5
Description	This option allows you to specify to DB2 that no more than N rows will be retrieved; this enables DB2 to optimize the fetch request. The number of rows retrieved is assumed not to exceed N rows. Based on this value, DB2 determines a suitable communication buffer size to improve performance.

DB2_PRINT_ERROR	
Driver resource	—
Environment variable	DB2_PRINT_ERROR
Client/Server resource	Set for server only
Type	Integer
Default	Not set
Description	You can set this environment variable to dump additional debugging information in the `$BSE/log` (UNIX) or `%BSE%\log` (Windows) directory on error conditions.

DB2PROF	
Driver resource	—
Environment variable	DB2PROF
Client/Server resource	Set for server only
Type	Floating point
Default	Not set

DB2PROF	
Description	If a value is specified in this variable, any statement that takes more than the number of seconds specified is logged. The maximum precision that can be specified is 0.01 seconds. This variable is used to determine which table actions are the most time-consuming.

db2_retained_cursors/DB2_RETAINED_CURSORS	
Driver resource	db2_retained_cursors
Environment variable	DB2_RETAINED_CURSORS
Client/Server resource	Set for server only
Type	Integer
Default	20
Description	This variable is used to specify the number of inactive cursors to retain. These cursors can be reused and, therefore, save prepare/bind overhead, but will also need more resources, such as memory, than if the cursors were closed and released.

DB2STAT	
Driver resource	—
Environment variable	DB2STAT
Client/Server resource	Set for server only
Type	Integer
Default	Not set
Description	This variable allows database driver statistics to be reported. If this variable is set to a value N greater than 0, statistics are logged every N seconds while the driver is active. If this variable is set to 0, a statistics report is generated when the driver terminates.

max_sql_buffer / MAX_SQL_BUFFER	
Driver resource	max_sql_buffer
Environment variable	MAX_SQL_BUFFER
Client/Server resource	Set for server only
Type	Integer
Default	32K
Description	(Level 2 only) To specify the maximum size of memory to be allocated for any one SQL statement, use this variable.

Appendix B: Parameter File Formats and Configuration Options

Two parameter files affect the behavior of the database driver: the storage parameter file and the driver parameter file. Their formats and the parameters used with these files are discussed here.

For additional information about the storage parameter file, see Storage parameter file on page 27 .

Parameter file naming

For Infor Baan IVc, there is one file, called `db2_storage`. For Infor Baan 5.0 and LN, the `db2_storage` file is split into two separate files: `db2_storage_param` and `db2_driver_param`. Infor Baan IVc only uses the `db2_storage` file.

Parameter file formats

The parameter files consist of one or more entries, each of which consists of several fields separated by colons.

Storage parameter file format

The format of an entry in the storage parameter file is:

```
<table/module specification>:<company number>:<object
type>:[<compress specification>]<storage parameters>
```

Driver parameter file format

The format of an entry in the driver parameter file is:

```
<table/module specification>:<company number>:<object type>:group:<table/in▶
dex optimization>
```

Storage file format

Infor Baan IVc uses the `db2_storage` file. This file is a predecessor to the storage and driver parameter files which are introduced in the Infor Baan 5.0 release.

The format of an entry in the `db2_storage` file is:

```
<table/module specification>:<company number>:<object type>:group:<table/in▶
dex optimization>:< refresh time (obsolete)>: <storage parameters>
```

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 package tt and module adv
	tt	All tables in package 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 these options:	
	T	Table only
	I	All indexes
	I <index number>	Only specified index
	*	Both table and indexes
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.	
Example	COMPRESS=1;	Compression is used for the selected table(s) or indexes (COMPRESS YES)

Group	
Description	This field identifies the owner of the table group must be specified.

Table/index optimization		
Description	Specific flags related to indexes and tables can be set. When specified on a “T” object entry, this flag defines the default for all indexes. To set the flags for a specific index or table, use the octal values in this table. The octal values in this table can be used to set the flags for a specific index or table:	
	0000	No optimization
	0200	Nested
	0400	Iterative
	01000	Filter

Storage parameters		
Description	These parameters are defined by the specific database driver implementation and often map to table and index creation options available in the host RDBMS. These DB2 storage parameters are defined:	
	T_SPACE	Set the tablespace in which the table is to reside.
	I_SPACE	Set the tablespace in which the indexes for a table are to reside. Note that this goes on the T entry and not the I entry.

Examples

Storage parameter file

Whenever a `CREATE TABLE` or `CREATE INDEX` is performed, the storage parameter file is scanned from the beginning. The first entry that matches the table or index is taken, so the order in which the entries are specified is important.

This example provides a sample of a storage parameter file:

```
ttadv999,ttadv000:000:T:T_SPACE userspace1 I_SPACE index1
tdsfc:505:T:
*:*:T:T_SPACE dataspace1 I_SPACE index2
*:*:I:
```

In this example, the tables ttadv999 and ttadv000 of company 000 are created in tablespace userspace1. The associated indexes are created in tablespace index1. All users who create tables in module tdsfc create tables and indexes in the default tablespace and use array sizes of five. All other tables and indexes are created in tablespaces dataspace1 and index2.

If the tablespace for a table or index is not specified, the table and index data are created in the default tablespace. To separate the index data, you must also specify a tablespace for the table.

Driver parameter file

The driver parameter file is read-in at driver startup to specify several run time settings. An example of a driver parameter file is:

```
ttadv999,ttadv000:000:T:group:0000:5:
tdsfc:505:T:
```

```
*:*:T:
*:*:I:
```

Storage parameter file performance tips

Tips you can use to improve performance:

- A default table and index entry that covers all users, tables, and company numbers must be present; this is an example of such a default table and index specification entry:

```
*:*:T:T_SPACE tablespace I_SPACE indexspace
*:*:I:
```

Conversion from previous porting sets

Existing Baan installations, installed on Infor Baan 5.0 and later before porting set 7.1a, do not have the storage or driver parameter files. These porting sets only contain the `db2_storage` file.

If the driver parameter file and the storage parameter file do not exist, the LN database driver tries to open the `db2_storage` file, so that existing installations continue to work.

One Baan session is available, the **Convert Table and Index Repository (ttdba0540m000)** session, that converts the storage file to the storage parameter file and the driver parameter file. To convert the storage file to the new format, use this session.

Note: This functionality is not available for Infor Baan IVc.