Infor Enterprise Server Technical Reference Guide for SQL Server Database Driver

Publication Information

Release: Infor Enterprise Server 10.4

Publication Date: January 20, 2015

Document Code: U8173M US

Contents

About this guide

This document describes the database driver that forms the interface between the Infor LN application server layer and the MSQL database server.

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

The term ‘Infor Enterprise Server’ comprises the tools and porting set. This document handles issues related to the porting set, where the MSQL database driver is a part of. Affected porting set versions are 8.7b and later versions.

For simplification, LN is used in this document and unless specified, the information applies to all supported MSQL versions. Any differences are stated.

For supported platforms and MSQL RDBMS versions, check the technical notes of the Infor Enterprise Server porting set.

When Infor Baan IVc is used, a minimum version of porting set 8.7b01 is required.

Intended audience

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

Related documents

You can find the documents in the product documentation section of Infor Xtreme, as described in "Contacting Infor".

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

- *Infor Baan IVc - Pre installation Guide for Windows (U9739)*
- *Infor Baan IVc - Deployment Guide for Windows and UNIX (U9673)*

- *Infor Baan 5.0 - Installation Guide for Microsoft SQL Server (U8991)*
- *Infor LN - Installation Guide (U9498)*
- *Infor Enterprise Server - Technical Manual (U8172)*
- *Infor LN - Performance, Tracing and Tuning Guide (U9357)*
- *Infor LN - Performance, Tracing and Tuning Guide for SQL Server (B0079)*

Contacting Infor

If you have questions about Infor products, go to the Infor Xtreme Support portal at http://www.infor.com/inforxtreme.

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

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

LN Database driver overview

1

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

Architecture

LN supports a three-tier architecture consisting of:

- A display tier
- An application tier
- A database tier.

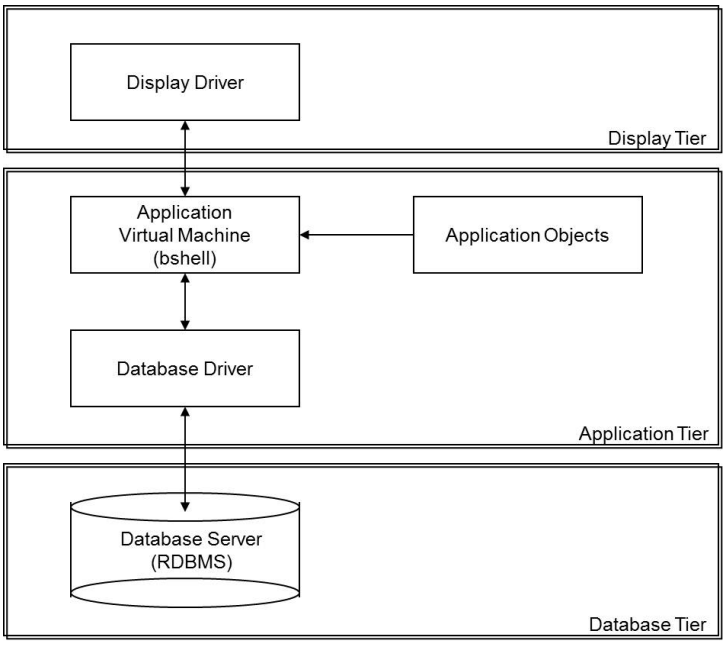
The display tier provides presentation services for user interaction.

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 figure below depicts the LN architecture.

The emphasis of this document is the LN database driver. The database driver is the interface between the LN applications and the RDBMS server. The database driver translates database requests from the LN application virtual machine to RDBMS specific SQL requests that are sent 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, each of the three tiers of the total LN architecture is described briefly in this diagram:



Display tier

The display tier consists of the display driver, which includes the LN user interface for Microsoft Windows (for example LN UI). The display driver facilitates the communication between the user and the application tier. Data input from the user is relayed to the LN application virtual machine; data returned from the LN application virtual machine is displayed to the user in graphical form by the display driver.

Application tier

The application tier includes the application objects and the LN application virtual machine as well as 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 for storing and retrieving application data.

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

These applications are written in Baan 3GL or Baan 4GL, which are programming languages, supported by the Infor Enterprise Server package and are compiled into an intermediate code which is interpreted at runtime by the virtual machine (VM).

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

The LN application virtual machine schedules and runs the application objects, sends and receives information to and from the display server, and initiates an instance of the database driver as necessary

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

The LN application virtual machine has traditionally been called the LN shell, or more often simply the bshell.

Throughout the remainder of this document, it is referred to as the LN application virtual machine or the application virtual machine.

The LN database driver is also typically 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. There is one database driver for each of the RDBMS products that LN supports.

Database tier

The database tier typically consists of a third party database server product. Communication between the database driver and the database server is tailored to the RDBMS being used. 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 different third party RDBMS products, for example; Oracle, Informix, IBM DB2, or Microsoft SQL Server. All LN application data is stored in a relational database managed by an RDBMS. It is possible to have multiple RDBMS products in one LN installation, with some data residing in one database server and some data residing in another.

Data flow in the LN architecture

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 in the following 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 it receives, the application virtual machine triggers the appropriate application object to be run.

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 are made using LN SQL, an RDBMS independent SQL language.

When the application virtual machine runs a database query from an application object, it first determines whether or not there is a running database driver available to process the query. If there is no database driver running, or if the running database driver instances are communicating with a database server other than the one storing the needed data, the application virtual machine starts a new instance of

the database driver. The application virtual machine parses the LN SQL database query it receives from the application object and sends an internal representation of the query to the database driver. The internal representation of the query that the database driver receives is still RDBMS independent.

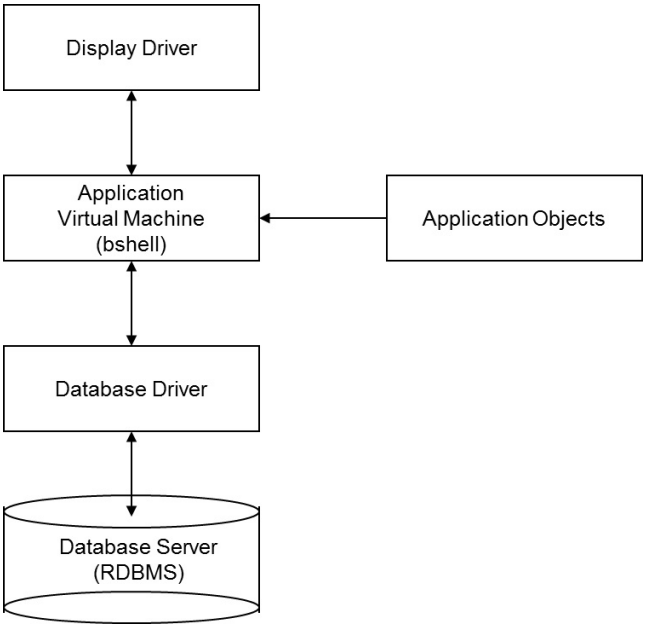
The database driver translates the database query into an appropriate database-specific query using SQL statements compatible with the RDBMS being used. Each database driver takes advantage of the design of the particular RDBMS that it supports so that the resulting SQL statements are valid for the RDBMS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

When the RDBMS has processed the query, it returns the data to the database driver. Any error conditions are caught and handled by the database driver. The database driver then returns the data and status information to the application virtual machine, where it provides the information to the application that requested it. The application virtual machine may also send a message to the display server, which displays an appropriate message on the user's workstation.

Hardware configurations

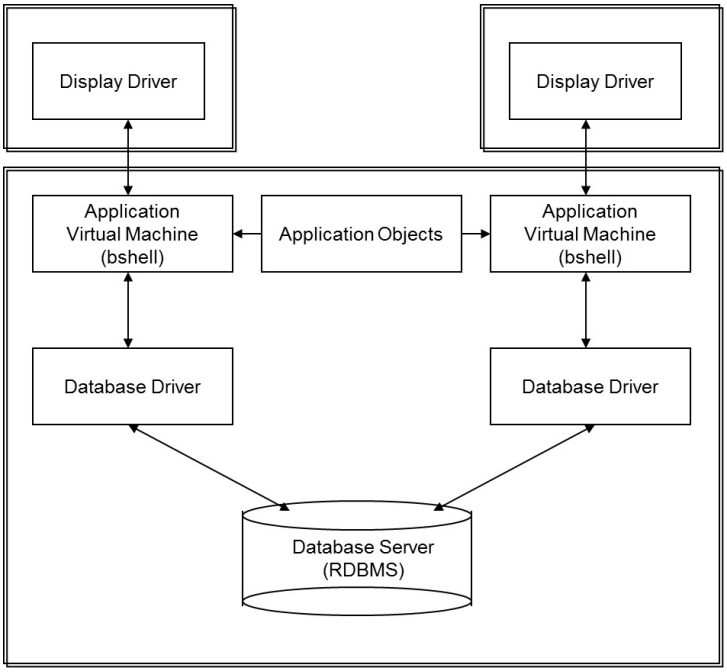
Several hardware configurations are supported for an LN implementation. These configurations include the stand alone mode and many variations of client/server mode. Available hardware, data storage requirements, and performance expectations determine the most appropriate hardware configuration.

Standalone mode refers to a configuration where all components of the LN architecture run on a single machine. In standalone mode, an end user can work from the host machine or from a thin client machine. The standalone mode configuration is illustrated in the diagram. However this is still possible, it is not commonly used anymore.



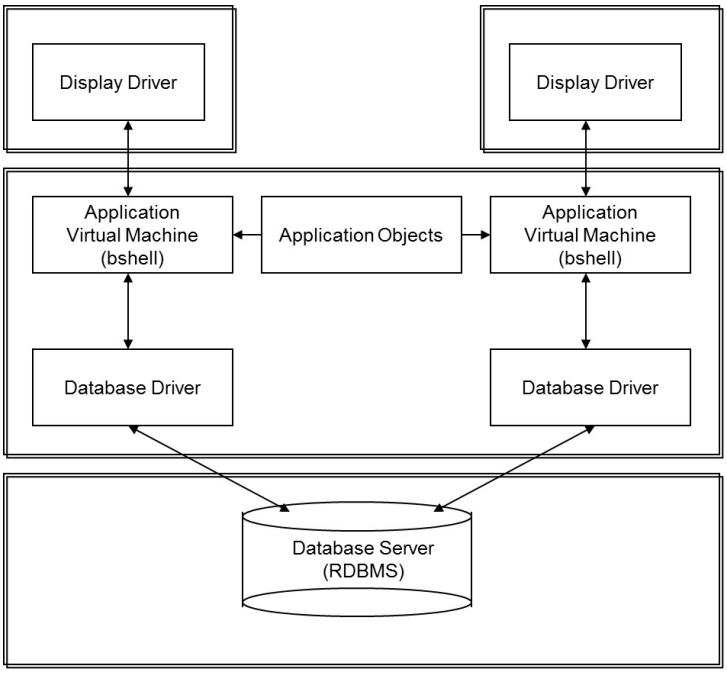
In a client/server configuration, the components of the LN architecture are distributed over two or more machines. There are many client/server configurations. The most common configurations are described here.

The simplest client/server configuration is sometimes thought of as a variation of the 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 illustrated in this figure:



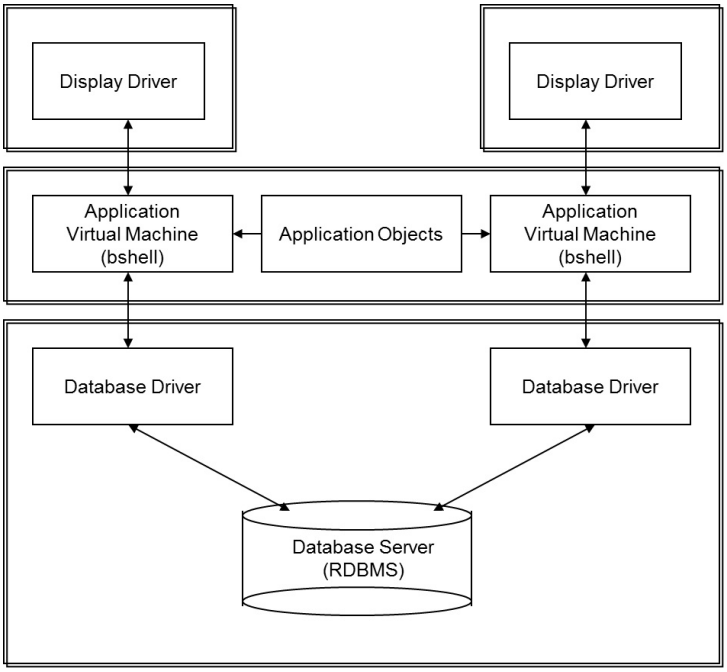
When two machines are available to be used as servers, two configurations are commonly used. In both configurations, the display drivers reside on the user workstations.

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 client/server configuration is illustrated in the figure below. This configuration uses the RDBMS's networked client functionality to provide client/server access. This configuration is also known as 2-tier or host-mode.



In the 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 figure below. This configuration uses the LN method of client/server access between the application virtual machine and the database driver. This configuration is also known as 3-tier.

Note: This configuration is not supported by the LN installer and must be set up manually.



There are many other configurations of client/server systems, including dividing the application logic among multiple servers or using multiple servers for distributing the database.

Database organization

2

How LN organizes the database.

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

Because many tables are required, a convention is used for naming tables, columns within tables, and indexes to data within the tables. This chapter describes the data dictionary and the naming conventions used by the LN database drivers to access data stored in the RDBMS. Also is discussed how LN data types are mapped to SQL Server data types.

Data dictionary

A data dictionary is a catalog that provides information about the data in a database. It can be thought of as “data about the data,” or metadata. A data dictionary can be used to describe data that resides within a database table.

LN maintains a data dictionary because the data used by the LN applications must be described in a database-independent manner. Since LN domains and data types may differ from those available in the database server, the database driver performs any mapping and translation necessary to store and retrieve the LN data in the database server. The LN data dictionary defines LN data types, domains, schemas, and referential integrity information and the database driver maps this information to the appropriate elements 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 cached in shared memory where it will be available to all running LN application virtual machines. The data dictionary information is shared among all the sessions open within a single database driver.

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

Table naming convention

The name of an LN table stored in SQL Server has the following format.

```
t<Package><Module><Table number><Company number>
```

The components of the table name.

- Package

  Package is a two-letter identifier that refers to the LN package where the table is defined. For example, a table defined in the LN Tools package has the package code **tt**.

- Module

  Module is a three-letter identifier for the subset of the package that defines the table.

- Table Number

  The table number is a three-digit number uniquely identifying the table within the module that it was defined. The number often indicates the order in which the table was defined in the module.

- Company Number

  Within LN, three digit company numbers are used to isolate datasets used for different purposes (for example; demo data, training data, production data). There must be a company with the number 000. Company 000 contains master data common to all companies. This data includes currencies and languages used. In addition to Company 000, there may be several other company numbers.

  For example, the table 999 in module adv with company number 000 is created in SQL Server as tttadv999000.

  Note:

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

    ```
    s<Package><Module><Table number><Company number>
    ```

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

    ```
    v<Package><Module><Table number><Company number>
    ```

Column naming convention

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

The rules for column names are:

- General

  When an LN column name is created in the SQL Server, it is preceded by the string **t_**. For example, the LN column with the name **cpac** is created in the SQL Server with the name **t_cpac**. By preceding column names by **t_**, reserved word collisions are avoided. If an LN column name contains a period [.], the period is replaced by the underscore [_] character.

- Long string columns

  SQL Server data type CHAR has a limit of 8000 characters. When an LN string column exceeds this limit, the column is split into segments with a maximum of 8000 characters each. The first 8000 characters are mapped to a column where the name of the column is extended with **_1**. The next 8000 characters are mapped to a column with a name extended by **_2**, and so on until all the characters of the string are mapped to a column. For example, if an LN string column called desc contains 8888 characters, these two SQL Server columns are created:

  - **t_desc_1**: size 8000
  - **t_desc_2**: size 888

- 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 containing a date can be defined as an array of three elements—a day, a month, and a year. In SQL Server, 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 SQL Server character size, it is split. For example:

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

- Array compression

  The maximum number of columns in a SQL Server is 1024 columns. If the number of LN columns exceeds the maximum number allowed in SQL Server, the database driver tries to compress (concatenate) array columns to reduce the total number of columns in the table. All elements of one array column can be stored as one column in the SQL Server table with the elements concatenated in binary format. The driver starts by compressing the array column that yields the highest number of columns. It continues compressing array columns until the number of columns is less than the maximum allowed. The name of the compressed column in the SQL Server follows the same convention used for the other columns. For example:

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

Index naming convention

LN indexes are identified by a sequence number for each table, with the sequence numbers starting from one. Each table has at least one index called the primary index. The SQL Server requires that for each database, all index names within each database must be unique. For this reason, the table name, index number, and the index type are included in the index name. The index type refers to the order, either ascending or descending.

Index names have this format:

```
I<Table name>_<Index number><Index type>
```

For example, the index name for an LN table with name ttadv999, company 000, index number 1, and index type ascending order is:

- Ittadv999000_1a

Note:

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

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

Data type mapping

This table shows the mapping between LN data types and their SQL Server counterparts.

Mapping between LN and MSQL data types		
LN data types	MSQL data types	
	For Infor Baan IVc & Baan 5.0	For Infor LN [1]
CHAR	Tinyint	Integer
ENUM	Tinyint	Integer
INT	Smallint	Integer
LONG	Integer	Integer
TIME	DateTime	DateTime
TEXT	Integer	Integer
BITSET	Integer	Integer
FLOAT	Real	Real
DOUBLE	Float	Float

Mapping between LN and MSQL data types		
STRING(N)	Char(n)	Char(n)
DATE	DateTime	DateTime
BLOB[2]	VARBINARY (max)	VARBINARY (max)

1 Default behavior on Infor LN, but configurable to the behavior on Infor Baan IVc/Baan 5.0 .

2 The SQL_BINARY 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).

By default, the Infor Enterprise Server SQL Server driver uses the VARCHAR data type for these releases:

- Infor Baan 5.2a and later with a 8.4.x porting set version or higher
- Infor Baan 5.0 and later with a 8.7.x porting set version

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 SQL Server level.

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 required for cursor stability.

Additional constraints

In addition to the above naming conventions and data types, the following rules apply when mapping LN data to SQL Server data:

- Because the binary sort order is selected during the installation, the SQL Server treats object names with case sensitivity.
- All columns created by the LN MSQL driver have the NOT NULL constraint. LN does not currently support NULL values in the data.
- The date range supported by the LN application virtual machine is not the same as the range for the SQL Server (the SQL Server is more restrictive). Therefore, some LN dates are not valid when stored with the LN MSQL driver. The LN date 0 is mapped to the earliest possible date in the SQL Server (01-Jan-1753). The earliest possible LN date is then 02-Jan-1753 and the latest is 31-Dec-9999.

Database driver internal processing

3

This chapter describes some of the internal processing that occurs within the LN MSQL database driver.

The LN MSQL database driver converts RDBMS independent database requests into requests targeted specifically to the SQL Server. First, some of the features that ensure data integrity are discussed. Next, the internal processing of a SQL statement within the driver is explained. The final section describes the mechanisms that allow modifying the default behavior of the database driver.

These LN MSQL database driver internal issues are discussed:

- Data integrity.
- Database driver SQL processing.
- Setting driver behavior.

Data integrity

Several features of the LN database driver help to ensure data integrity. These features include locking mechanisms, methods used for enforcing referential integrity, and methods used for distributed databases. Data integrity is maintained while minimizing network traffic by the use of data buffering techniques.

This section gives an overview of the features used by the LN MSQL database driver to maintain referential integrity, to work with distributed databases, and to apply data buffering techniques. Locking strategies are discussed in more detail in "Locking behavior" on page 42.

Locking

Locking in the database typically occurs when a record is modified, deleted or inserted. The lock persists until the transaction that the modification is part of is committed or aborted. This type of locking can be referred to as implicit locking, because the lock is taken implicitly by virtue of the database being modified. Records can also be locked explicitly by simply selecting them in a mode that causes the records to be locked even though they have not been modified. Locking allows the database server to

serialize changes to the content of the object being locked to prevent corruption of data. Explicit locking allows the application to obtain exclusive control over data.

Referential integrity

Referential integrity can be defined as the preservation of known relationships between data in tables when records are maintained. The LN database driver has a built-in mechanism for preserving referential integrity; it does not depend on the underlying RDBMS to maintain referential integrity.

Data buffering

Updates can be buffered by the LN application virtual machine and flushed at the time of transaction commit, or earlier when necessary. This can reduce the number of network round trips between the database driver and server.

When multiple rows are returned from a query, the rows can be buffered and then sent back to the LN application virtual machine as one block transfer. Data reduction and compression can be applied to compact the data, minimizing the amount of data transferred between the application virtual machine and the database driver.

Database driver SQL processing

As discussed in "LN Database driver overview" on page 9, the LN application virtual machine sends RDBMS independent read and update requests to the database driver. It is the function of the database driver to convert these RDBMS independent database requests into SQL statements that are appropriate for the specific RDBMS being used. This section details the SQL processing performed by the LN MSQL database driver.

Because the LN database driver uses the Microsoft Open Database Connectivity (ODBC) interface to communicate with the SQL Server, the ODBC interface will be described first.

The ODBC interface

ODBC is an application programming interface (API) used to communicate with the database server. ODBC is made up of a function library that can be called from an application program to execute SQL statements and communicate with the data source (for example. Microsoft SQL Server).

The ODBC functions called by the LN MSQL database driver perform these actions:

- Connect to Microsoft SQL Server (open session).

- Allocate a statement handle.
- Parse a SQL statement.
- Bind input variables.
- Define result variables.
- Execute a SQL statement.
- Fetch the resulting rows.
- Commit or abort a transaction.
- Close, unbind and drop a cursor.
- Disconnect from MSQL (close session).

The LN MSQL driver also uses these features of ODBC:

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

SQL processing

The LN MSQL database driver processes SQL statements in several steps. This section describes these steps.

SQL statements are dynamically generated by the database dependent (specific) layer of the LN MSQL database driver. Because the application virtual machine interprets code dynamically and the LN tools allow for modifications to the applications, it is not known in advance which tables are accessed at run time; therefore it is not practical to prepare the queries before run time.

When the LN MSQL driver receives a query from the application virtual machine, the query is translated into a format suitable for SQL Server. The text of the query is transferred to the SQL Server using ODBC function calls. The database driver makes a function call to the ODBC driver manager to allocate a statement handle, and the query is executed by assigning it to the statement handle and calling the ODBC query-execute function. In some cases, SQL Server opens a server cursor internally for query execution. After the query is executed, a fetch operation is done and the resulting column values are placed in the bound result variables. Control is returned to the database independent layer of the LN MSQL database driver, which sends the results back to the application virtual machine.

When a statement must be re-executed, the cursor from the previous execution is closed. The resulting rows are discarded (whether the re-execution is with the same input parameters or not). If new input values are required, the new values are assigned to the input parameter bind variables and the query is re-executed. For re-execution, no re-parse of the statement or re-bind of input and output parameters is typically required. Because binding is an expensive operation, avoiding it when possible can improve overall performance.

When array fetching is enabled, multiple rows can be fetched in one ODBC call to the server. Space is allocated within the driver to buffer multiple rows fetched in one operation. Multiple rows can be fetched to the buffer; and they are returned to the application virtual machine when requested. When no rows are left in the buffer and more rows are requested, another 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 when some other event necessitates it, the rows are flushed to the database. The rows in the buffer are inserted with a multi-row insert.

Note: Data can be placed into the database using the LN utility `bdbpost`. This utility is used to import data into a new database table or to append or replace data in an existing database table. Certain options can be set when using `bdbpost`, see:

- *Infor Enterprise Server - Technical Manual (U8172)*.

For example, when using `bdbpost`, the rows are buffered by default and are flushed when the array buffer is full. The array size must be specified; otherwise buffering is not done. The array buffer size can be specified globally, by using an environment variable or resource variable. The environment variables and resource variables are explained in the next section.

Note: The name of the parameter file differs per LN version. For more information see "Parameter file formats and configuration options" on page 69.

Setting driver behavior

There are several facilities available for configuring the LN MSQL database driver. The most common is through driver resources. Two other facilities for configuring the LN MSQL database driver are environment variables and the storage and parameter files. The driver resources and environment variables are described in more detail in ""Driver resources and environment variables" on page 49". The storage and parameter files are described in ""Parameter file formats and configuration options" on page 69".

Driver resources

The database driver resources are configuration parameters you can set to modify the behavior of the LN MSQL database driver. These parameters are set in the resource file (`db_resource`). One resource file is available for all database drivers that run in an LN environment. Resources for all 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 configure 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 containing two entries:

```
msql_opt_rows:10
msql_lock_timeout:60
```

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

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

For the most part, there is an environment variable corresponding 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 are used to modify the behavior of the database driver (server) and some are used to modify the behavior of the application virtual machine (client). When a database driver environment variable for the server is to be used, it must be set on the machine running the database driver to override the corresponding driver resource. When a database driver environment variable for the client is to be used, it must be set on the machine running the application virtual machine to override the corresponding resource.

Server environment variables

Environment variables that affect the database driver can be used to override the driver resources for all tables in a database or for specific tables and company numbers within the database. To set the database driver server environment variables, you can use:

- The LN Database Definitions (ttaad4510m000) session and Tables by Database (ttaad4111m000) session.
- The Infor Management console to modify environment variables for each LN instance.
- The standard operating system mechanism for setting environment variables.

The LN Database Definitions session is the recommended method to modify database driver behavior. When specific tables and companies are to be configured for access with a specific database driver, the Tables by Database session must be used. These sessions allow environment variables for a particular database driver to be written to the following table definitions files:

- `%BSE%\lib\tabledef6.1` for Infor Baan IVc
- `%BSE%\lib\tabledef6.2` for Infor Baan 5.0/LN

Environment variables present in the driver specifications in the table definitions file are passed to the driver and added to its environment after startup. Environment variables that correspond to resources override the defaults set in the resource file. This method allows the environment variables to be maintained centrally.

The Database Definitions session maintains database driver mapping and configuration information in the table definition file. This file is stored in the directory `%BSE%\lib` residing on the application server machine where the database driver runs. The format of entries in the table definition file is:

```
<table name>:<company number>:<driver type>(<environment
variable>=<value>)
```

The "`<driver type>`" and optional "(`<environment variable>=<value>`)" portions are collectively known as the driver specification. If multiple environment variables are to be specified for a single table and company number, they are listed within the parentheses and separated by commas. If all tables or all companies are to be specified, the asterisk (`*`) is used as a wildcard in place of table name or company number. For example, the following entry can be made in the table definition file:

```
tccom010:812:msql7(MSQL_LOCK_TIMEOUT=60,MSQL_DSN=SQLSRV)
```

In this example all the queries on table `tccom010812` are sent to an LN SQL Server driver. The MSQL_LOCK_TIMEOUT and MSQL_DSN settings specified are added into the driver's environment after it is started.

Note: Tabledef entries with different driver specifications are considered to have a different database definition from entries. If an MSQL driver is already running, but has a different driver specification, a separate driver is started for this entry. Environment variables that appear in the driver specifications of the table definition file are placed in the driver's environment after it is invoked. The variables are available to the driver at startup.

If the default database driver resources must be modified for specific users, the standard operating system method can be used to set database driver environment variables on a per-user basis. These environment variables override the resource settings for these users.

Client environment variables

Environment variables that affect the client can be used to override the client resources that affect the application virtual machine. Specify these environment variables on the machine running the application virtual machine. Use the Infor Manager or standard operating system methods for specifying the environment variables. Any client environment variables that are used override the equivalent resource variables set for the client in the `db_resource` file.

Storage parameter file

The storage parameter file provides a way to affect the distribution and configuration of table and index objects at create time. Storage parameters are used by the database driver whenever a DDL statement such as a create table or create index statement is executed. The following is an example of an entry in the storage parameter file:

```
*:*:T:FILEGROUP filegroupname
```

In this example, the database driver adds the “on `<filegroup name>`” clause to the create statement during table creation.

A storage parameter file is defined for each database driver. The storage parameter file is located in this directory: `%BSE%\lib\msql` For more information about the name of the file and the format of the storage parameter file, see ""Parameter file formats and configuration options" on page 69".

Driver parameter file

The driver parameter file provides a way to specify runtime parameters used by the database driver whenever a table is accessed. The following is an example of an entry in the driver parameter file.

```
*:*:*::00400::
```

In this example, the database driver is configured to use iterative query optimization technique.

A parameter file is defined for each database driver. The parameter file for the LN MSQL database driver is called `msql_driver_param` and is located in the directory `%BSE%\lib\msql`. The format of the parameter file is described in detail in "Parameter file formats and configuration options" on page 69.

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 `msql_storage`, for both storage parameters and driver parameters. For more information, see ""Parameter file formats and configuration options" on page 69".

Database security

4

The LN MSQL database driver maintains security by controlling user access to the database and user access to database objects. The LN database administrator (DBA) module allows the DBA to control access to the database using LN sessions. Using the DBA module makes DBA tasks easier and less prone to errors than using database driver tools directly. First is discussed how the LN MSQL database driver handles issues related to database security, then the DBA module is briefly described.

Security

There are two aspects of database security, object security and authentication. Object security refers to the process of determining whether or not a user who has access to the database is authorized to access particular database objects. Authentication refers to the process of determining whether or not a user is authorized to access (in other words, connect to) the database. Both object security and authentication use the concept of database groups or roles to ensure security. This section first describes the group concept, then describes how the LN MSQL driver authenticates and provides object security.

Groups

In LN, a group is defined as a collection of database users. All users assigned to a group are granted the same database privileges. Once a group is defined with a certain set of privileges, users can be assigned 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 within the database. The LN group name is the same as the name of the database that holds the LN data within the RDBMS. The LN group uses the mechanisms of the RDBMS to authenticate and provide object security.

For the Microsoft SQL Server, an LN group consists of three components:

- A database
- A login (for authentication)
- A SQL Server role (for object security)

The SQL Server database has the same name as the LN group. The login is also named the same as the LN group and is assigned database owner (DBO) privileges in the database. A SQL Server role is created whose name is derived from the LN group name. This role becomes the target for privileges granted on objects in the database. Users are associated with the SQL Server role and, as a result, inherit the privileges granted to the role. The advantage of having a group table is that all members of the group can share and operate on the same data in a single table. LN tables are typically owned by the group so that the tables and data can be shared amongst all users of the application.

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 SQL Server role. Therefore, users **Maria** and **John** inherit the select, insert, delete, and update privileges granted to the SQL Server role, to access and manipulate LN group table data.

The LN users are shielded from directly administering the role. The LN DBA sessions and the database driver do all the processing that is needed to make use of the role.

Only the administrator must be concerned about role administration. With the LN DBA module the administrator can easily maintain the role within the LN Tools.

Object security

If a user creates an object such as a table, in the SQL Server, the user becomes the owner of that object. Only the owner can access the object unless privileges are explicitly granted to other users. Other users can only access the object if they have been granted privileges to do so. In an LN environment where many users access the same tables in the SQL Server database, a mechanism has been developed to allow users to share these tables.

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

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

When new users are added, they only need to be associated with the SQL Server role. New users automatically inherit all privileges currently granted to the role without the need to grant privileges on every object in the database to the user. When the user is dropped from the role, these privileges are revoked. The user no longer has access to tables within that SQL Server group. If the privileges to operate on the tables were explicitly granted to the user, then they must also be explicitly revoked when the user is dropped from the LN group. The overhead of adding users is greatly reduced by granting privileges to the role and simply associating users with that role. The use of this method provides flexibility and ease of maintenance.

In the DDL statements generated by the driver, object names are not qualified by the owner name. Ownership is determined by the session (group or user) the create table is executed in. When creating objects identified as belonging to the group, the user creating the object logs into the SQL Server as the group user (the database driver handles this transparently).

In this case the table will be owned by the group and permissions will be granted to allow all group users access. When accessing objects, users connect to the SQL Server with their own login.

Authentication

The LN DBA module sessions are used to map LN groups and users to SQL Server logins to allow them to establish a connection to the SQL Server and access data. To prevent unauthorized users from accessing the database, non-mapped users cannot establish a connection to the database. When a database is created, an administrator or SQL Server database owner (DBO) creates a login for the user and associates the user with a SQL Server role in the database. The members that belong to this role inherit the role's privileges and are able to establish a connection to the database either via unified login or using a valid password stored in encrypted form in the driver administration files.

A user can be added to or dropped from an LN group by using the LN DBA (database administration) module sessions. Users who are authorized to access the database are registered in the LN driver administration files. The user name and password LN uses to log on to the SQL Server on behalf of the user are maintained in the file %BSE%\lib\msql\msql_users. Here, %BSE% refers to the LN software environment (BSE), the directory where the LN software was installed.

All the LN users and their corresponding SQL Server login names and passwords and the name of the LN group to which they are assigned are defined in the file %BSE%\lib\msql\msql_users. The format of each entry in this file is as follows:

```
<Infor ERP user>:<SQL Server login>:<Encrypted SQL Server user
password>:<Infor ERP group name>
```

The LN MSQL driver is started by the LN application virtual machine on behalf of the user. From the file %BSE%\lib\msql\msql_users the driver identifies the SQL Server login and password and establishes the connection to the SQL Server.

The group logon process also requires a password, which is defined in the file %BSE%\lib\msql\msql_groups. The file format is as follows:

```
<Infor ERP group name>:<Encrypted group password>
```

DBA module

The DBA module is used to maintain the database administration files used by all LN database drivers. The sessions in this module allow the administrator to register authorized users and give them access to data. A tool is provided with the LN MSQL database driver to maintain the administration files the database driver needs at run time. The administration files are stored in the directory `%BSE%\lib\msql`.

The DBA module implements the user and group administration functions for all LN database drivers. The `MSQL_MAINT` utility is an executable program called by the DBA module that implements the functions necessary to make changes in the SQL Server. We do not recommend that you call the `MSQL_MAINT` utility from outside the DBA module, because the users and groups files are not modified by `MSQL_MAINT`.

The DBA module is described in more detail in these documents:

- *Infor Baan IVc - Database Administration (DBA) on Windows (U7247)*.
- *Infor Baan 5.0 - Administration Guide (U7189 US)*.
- *Infor Enterprise Server - Administration Guide (U8854)*.

Database driver profiling and statistics

5

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

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

Profiling

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

The database driver's profiling option provides a method to gather the timing of SQL statements that are being executed. Logging all statements with their timings, results in a large log file that cannot be easily analyzed.

Define a logging threshold where only statements that take more than a given amount of time to complete are logged.

With profiling, this information is logged:

- RDBMS request
- Elapsed time
- User name
- Date
- Time

The maximum precision that can be specified is one hundredth of a second.

Set the `MSQLPROF` environment variable to a time threshold (in seconds), to determine which table actions are most time consuming. For example, specify `MSQLPROF` as:

```
SET MSQLPROF=1.0
```

This sets `MSQLPROF` to one second. Statements that take more than 1.0 seconds of elapsed time to complete are logged to the `MSQLPROF` file in the directory `%BSE%\tmp`.

Statement execution time can be viewed for individual tables by setting the `MSQLPROF` environment variable. For more information about the environment variables, see ""Driver resources and environment variables" on page 49".

Profiling mode

When the driver runs in profiling mode, each phase in the SQL query processing that exceeds the profiling value is printed. The following three event types are timed and can appear in the log file:

- The Parse event

  This represents the amount of time the RDBMS takes to parse an SQL statement.

- The Execute event

  This represents the amount of time the RDBMS takes to execute an SQL statement.

- The Fetch event

  This represents the amount of time it takes to retrieve data from the driver's internal buffer cache or the RDBMS.

A sample MSQLPROF file:

```
------ Profiling value exceeded -----
<nkumar><qptool>:2002-03-05[11:25:05.491]:
Time (parse) : 0.010000 seconds
SQL statement:
SELECT a.t_byte,a.t_date,a.t_double,a.t_float,a.t_int,a.t_long,a.t_string FROM
dbo.taabdb000001 a WITH ( INDEX=Itaabdb000001_1a)  WHERE (a.t_byte > ?)  OPTION (FAST 1)
-------------------------------------
------ Profiling value exceeded -----
<nkumar><qptool>:2002-03-05[11:25:05.521]:
Time (multi_exec) : 0.020000 seconds
SQL statement:
SELECT a.t_byte,a.t_date,a.t_double,a.t_float,a.t_int,a.t_long,a.t_string FROM
dbo.taabdb000001 a WITH ( INDEX=Itaabdb000001_1a)  WHERE (a.t_byte > ?)  OPTION (FAST 1)
-------------------------------------
------ Profiling value exceeded -----
<nkumar><qptool>:2002-03-05[11:25:05.521]:
Time (multi_fetch) : 0.000000 seconds
SQL statement:
SELECT a.t_byte,a.t_date,a.t_double,a.t_float,a.t_int,a.t_long,a.t_string FROM
dbo.taabdb000001 a WITH ( INDEX=Itaabdb000001_1a)  WHERE (a.t_byte > ?)  OPTION (FAST 1)
-------------------------------------
------ Profiling value exceeded -----
<nkumar><qptool>:2002-03-05[11:25:05.611]:
Time (multi_fetch) : 0.000000 seconds
SQL statement:
SELECT a.t_byte,a.t_date,a.t_double,a.t_float,a.t_int,a.t_long,a.t_string FROM
dbo.taabdb000001 a WITH ( INDEX=Itaabdb000001_1a)  WHERE (a.t_byte > ?)  OPTION (FAST 1)
-------------------------------------
------ Profiling value exceeded -----
<nkumar><qptool>:2002-03-05[11:25:05.691]:
Time (multi_fetch) : 0.000000 seconds
SQL statement:
SELECT a.t_byte,a.t_date,a.t_double,a.t_float,a.t_int,a.t_long,a.t_string FROM
```

```
dbo.taabdb000001 a WITH ( INDEX=Itaabdb000001_1a)  WHERE (a.t_byte > ?)  OPTION (FAST 1)
--------------------------------------
```

The data in the above sample file can be interpreted as follows:

- In this example, the number of seconds (profiling value) is 0.00. This means that all actions are logged to the file.
- Once a statement is executed, the result set can be fetched many times (until the result set is exhausted or the cursor is closed).

Gathering statistics

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

- Number of cursors (opened, closed, current open).
- Number of parses, binds, executes, fetches.
- Number of logons (sessions or connections).
- Number of inserts, updates, deletes.
- Number of commits, rollbacks.

For each action, the cumulative elapsed time and the average time spent is logged. The statistics can be enabled with the environment variable MSQLSTAT. When the variable is set to zero, a statistics report is generated when the driver terminates (exit from LN). When 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 MSQLSTAT in the directory %BSE%\tmp.

These examples show how to set MSQLSTAT:

```
SET MSQLSTAT=0
SET MSQLSTAT=30
```

In the first example, MSQLSTAT is set to zero. With this value, only a final report is generated. In the second example, MSQLSTAT is set to 30. This logs a report every 30 seconds while the driver is active.

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

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

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

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

           BlobRd   BlobApp    BlobSz   BlobClr
Count          60        59        61         1
Time(s)    0.0198    0.3063    0.0085    0.0008
```

```
Avg         0.0003    0.0052    0.0001    0.0008

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

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

Troubleshooting

The LN MSQL database driver provides a tracing facility for analyzing driver behavior. The actions performed by the driver can be traced and stored in a log file. In addition, any errors that occur are logged. The following sections explain how to trace log 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 that are being performed by the driver. The resulting log file contains debugging information that can help solve problems.

When tracing is enabled, the information stored in the log file 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.

Tracing can be enabled using the environment variable `DBSLOG`. Debugging information is appended to the file `dbs.log` in the directory `%BSE%\tmp`. Tracing can be enabled by running this command:

```
SET DBSLOG=0560
```

Several tracing categories are defined so that tracing can be enabled for only categories of interest. For more information, see ""Driver resources and environment variables" on page 49" .

Logging errors

In a Windows LN environment, the database driver logs its error messages in the Windows Application Event Log or in a log file in the `%BSE%\log` directory. The Application Event Log can be viewed using the Windows Event Viewer.

The log file is called `LOG.MSQL.MESG`. An alternate location for the error log can be specified through the environment variable `MSQL_DUMP_MESG`. Warning messages can be logged to the error log by setting the environment variable `MSQL_LOG_WARNINGS`. For detailed description of these two environment variables, see " "Driver resources and environment variables" on page 49" .

Information can be retrieved from these log entries:

- 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.
- The failing SQL statement, in some cases (in log.msql.mesg).

If a database error occurs, an attempt is made to map it to a known or anticipated error condition. Generally, these mapped (Baan database) BDB errors contain corresponding error numbers in the range of 1 to 1000. If a database specific error occurs, which does not have a corresponding BDB error code, it is mapped to an error code over 1000 with this formula:

```
abs(error_code) + 1000
```

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

In most cases, the log entries from the display driver, the LN application virtual machine, and the 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 1000, you must check the log entries from the database driver. All error log entries can be viewed in the Application Log by using the Windows Event Viewer.

Database Driver Configuration and Tuning 6

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

Cursor management

The LN MSQL database driver has one parameter that influences the cursor handling: `msql_retained_cursors`. The cursor type is fast-forward-only with auto-fetch enabled. You cannot change the cursor type.

With the `msql_retained_cursors` resource the number of inactive cursors and thus the number of open cursors can be modified.

When a query has fetched all rows, a `close` is issued on the corresponding cursor. This means that SQL Server is notified that no additional fetches are done. This gives SQL Server the chance to free certain query resources. After the cancel, the query can be re-executed without re-opening (parsing, binding) the cursor. The driver does not know if a cursor in cancel state is re-used later. In the worst case it is not re-used, 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.

A number of inactive cursors in this list that is not available for this purpose are defined by the resource `msql_retained_cursors`, which defaults to 20. If the number of cursors in the cancel list is less than 20, a cursor from the cancel list cannot be assigned to a different query. If more than 20 cursors are in the cancel list, and a request for a new cursor is issued, 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 re-executing the original query, the driver detects that the cursor is associated with another query gets a new cursor and re-parse and bind the query again. Increasing the value of `msql_retained_`

`cursors` leads to less re-parsing and re-binding of queries, which reduces CPU resources. The number of open cursors is increased which requires more memory on the application server.

Array interface

The LN MSQL database driver can use multi-row ODBC statements for array fetches and array inserts. With the array interface, communication between the MSQL driver and SQL Server is more efficient. Multiple rows can be fetched or inserted with a single ODBC call. Because multiple rows are stored in a buffer in the LN MSQL database driver, more memory is consumed. The array interface is especially useful when accessing a remote database, because it reduces the number of network round-trips and increases performance.

The array fetch interface is enabled by default and can be disabled with the resource variable `msql_array_fetch:0`. The array insert interface is also enabled by default and can be disabled with the resource variable `msql_array_insert:0`. Array inserts are always enabled when loading data using the `bdbpost` utility.

Set the `msql_max_arrsz` resource variable to adjust the size of the buffers that hold array rows.

Optimistic and pessimistic reference checks

To optimize concurrency, the LN MSQL database driver supports optimistic reference checking. In the 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 be sure that another user cannot delete it within the current transaction. This approach is called the pessimistic approach.

The pessimistic approach blocks an insert of another user referencing the same parent row, thereby affecting the concurrency. To avoid this problem there is another approach where the row in the parent table is not locked. This approach is called the optimistic approach. As the record is not locked, another user can still perform an insert operation, which improves the concurrency. Enabling of this option is configurable via the `dbsinit` resource variable. Pessimistic reference checking is not available for MSQL driver at present.

Locking behavior

To ensure data integrity, the RDBMS uses a locking mechanism when updating a database table. The LN MSQL database driver can direct the SQL Server to choose a particular lock level when executing a query. Before understanding the LN MSQL database driver locking strategy, you must first understand the SQL Server locking mechanisms.

SQL Server locking mechanism

There are several levels of granularity that commercial RDBMS products can use to lock the physical data including row-level locking and page-level locking. With row-level locking, the smallest unit of data that can be locked is the row. Any row within a database can be locked. In some cases, rows, even within the same table, can have a variable length.

With page-level locking, the smallest unit of data that can be locked is the page. A page is typically fixed in size and generally includes more than one row of data. The RDBMS can manage page-level locks more efficiently than row-level locks because page size is fixed and because page-level locking usually requires fewer locks than row-level locking. The disadvantage of page-level locking is that a page lock can affect more than the single row of data that is being modified. This can have an adverse impact on concurrency.

SQL Server offers the advantages of both row-level locking and page-level locking by introducing a dynamic locking strategy. This strategy allows the server to use page-level locking when there is no lock contention, but to de-escalate the page lock to a row-level lock if there is contention for the locked page.

LN MSQL database driver locking strategies

In most cases, the driver generates queries that contain optimizer lock hints to direct MSQL to choose a particular lock level when executing a query. In other cases, the driver does not supply these hints, but uses the default locking determined by the isolation level being used in the connection.

When a record is locked for update or delete by the LN MSQL database driver, the row must be selected with a lock before it is actually updated or deleted. The row is locked when the SELECT WITH LOCK statement is executed to make sure that another user does not change the row. When the SELECT WITH LOCK statement locks the row; it acquires a shared or exclusive lock depending on the isolation level and any lock hints used. If the process tries to acquire an exclusive lock on a row that is already locked by another process, the process waits until the locked resources are released or the lock timeout period expires. If a time-out occurs, the client either retries the same operation or rolls back the transaction.

The database driver uses a delayed locking strategy to improve concurrency. This means that before an update is executed, the driver checks each column to determine whether the related columns have been changed. If the columns are not changed, the update is not executed. This reduces both the workload on the RDBMS and the network traffic between the database driver and the RDBMS and improves concurrency by reducing locking in the server.

By default, the driver uses the ”read uncommitted isolation level” to acquire shared locks, and uses the exclusive lock for update and delete actions.

The driver typically uses an uncommitted read for normal read operations. Exclusive locks are required so that any locks are retained until the transaction is committed or aborted, even after the cursor is closed.

Several aspects of the LN MSQL database driver locking behavior can be configured. The following default characteristics of the LN MSQL database driver can be modified: the lock time-out interval, the

number of high-level lock retries, and the fill factor for indexes. Each of these locking behaviors is described in the following sections.

Statement and lock time-outs

A SELECT WITH LOCK statement waits for a predetermined time period (time out duration) if a resource is locked by another session. The time out duration is configurable. Note that statements that do not take locks, or statements that read through locks, can also time out. This condition is commonly referred to as a query timeout and is often attributable to poor database server response time or high network latency.

The lock wait period for SELECT FOR UPDATE, INSERT, DELETE and UPDATE statements can be specified using the `MSQL_LOCK_TIMEOUT` environment variable or the `msql_lock_timeout` resource variable. Note that care must be taken when experimenting with these options.

Isolation level

The LN MSQL driver uses the read uncommitted isolation level, which means that (multi-row) read requests do not acquire any type of lock (shared or exclusive) unless explicitly stated in the query syntax. Queries such as INSERT, DELETE, and UPDATE acquire an exclusive lock implicitly. A SELECT WITH LOCK request acquires an update lock. Only in case of lookup references are shared locks acquired. The locks are retained until the transaction is committed or aborted.

Index FILLFACTOR

Includes the "FILLFACTOR <n>" clause in the create index statement during index creation. Default fill factor of SQL Server is 0, which indicates a dynamic approach. Although a low fill factor value other than 0 can reduce the requirement to split pages as the index grows, the index will require more storage space and can decrease read performance. Even for a table oriented for many insert and update operations, the number of database reads typically outnumber database writes by a factor of 5 to 10. Therefore, specifying a fill factor other than the default can decrease database read performance by an amount inversely proportional to the fill factor setting. For example, a fill factor value of 50 can cause database read performance to decrease by two times. Read performance is decreased because the index contains more pages, thereby increasing the disk I/O operations required to retrieve the data.

Using the fill factor setting gives the most benefit when the number of rows in the table does not change (the table is static), but there is a significant amount of update activity on existing rows.

The First Free Numbers (tcmcs050) table is an example of such a table. The example shows an msql_storage_param (Infor Baan 5.0/LN) file entry with the fill factor set to 1 for table tcmcs050.

```
tcmcs050:*:T:FILLFACTOR 1
```

A low fill factor setting also helps reduce page and index splitting by allocating a greater number of pages for the data and indexes with ample extra space to accommodate new rows.

Note: A low fill factor setting has the negative side effect of consuming more space in the database (disk and buffers) and may also cause more I/O activity to satisfy reads.

Query Tuning

This section describes how to externally influence the query generation done by the MSQL driver.

Concatenated expressions

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

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

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

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

Here 'comb' is a combined column of say 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 MSQL driver can use one of the three different techniques: nested, iterative, and filter to let SQL Server solve the WHERE clause in different ways.

These techniques are introduced because the SQL Server optimizer is not able to handle these queries efficiently in all situations (full table scans and sort operations can be introduced for these queries).

Specifying a different technique causes the SQL Server optimizer to make different decisions on how to run a query. It provides some workarounds for typical optimizer behavior.

The MSQLPROF variable can be used to detect long running or bad performing queries. Then you can experiment with these different techniques. MSQLPROF is described in "Driver resources and environment variables" on page 49.

The following describes the nested, iterative, and filter techniques. These techniques are set in the index optimization field of the `msql_storage` (Infor Baan IV) or `msql_driver_param` (Infor Baan 5.0/ERP Enterprise/LN) file. See "Parameter file formats and configuration options" on page 69 for more information:

- The nested technique

  The three conditions are ORed to this expression:

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

  This can be rewritten as:

  - c1 > "tt" OR
  - c1 = "tt" AND (c2 > "adv" OR
  - c2 = "adv" AND (c3 >= "000"))

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

- The iterative technique

  Multiple SQL statements are issued to resolve one query. These statements do not contain 'OR' conditions, so they can be handled efficiently by the MSQL. The iterative technique can only be used for unbounded queries.

  The iterative technique uses these three conditions:

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

  First, a query with the first condition is issued. When it does not return a row, it continues with the second condition. When the second condition does not return a row, it continues with the third condition. In addition, when one condition has returned 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 has a different approach. It initially selects too many rows, but then filters out those rows that do not match the total WHERE-clause. It 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"))

  As can be seen, the NOT() expression is like an inverted nested query; these rows are filtered out of the initial set determined by the first condition 'c1 >= "tt"'.

Specifying query tuning

The query tuning can be specified per table and per index in the file `%BSE%\LIB\msql\msql_storage` (Infor Baan IVc) or `%BSE%\LIB\msql\msql_driver_param` (Infor Baan 5.0/LN). For the actual values that can be specified, see ""Parameter file formats and configuration options" on page 69" .

Driver resources and environment variables

A

You can use the database driver resources and environment variables as configuration parameters to modify the behavior of the MSQL database driver. Some of these resources are used by the client, while others are used by the server. In this context, the client is the LN application virtual machine and the server is the LN MSQL driver. The LN application virtual machine and the database driver can run on separate machines. Set the client resources on the machine that runs the LN application virtual machine. Set the server resources on the machine that runs the database driver. You must set the resources for the client and server on both machines.

To set the database driver resources and environment variables, see "Setting driver behavior" on page 26.

Summary of MSQL resources and environment variables

There are four types of resources and environment variables that can be used with the LN MSQL database driver:

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

The tables provide a summary of each of these types of resources and environment variables. Detailed descriptions of each entry in the tables can be found in "Detailed description of driver resources and environment variables" on page 52.

Client and server resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
baan_sql_cache_rows	BAAN_SQL_CACHEROWS	Only applicable to LN Defines the size of internal buffers in the query processor.
baan_sql_trace	BAAN_SQL_TRACE	Only applicable to LN. To view SQL query information.
rds_full	RDS_FULL	Sets maximum number of rows transferred in one block.
tt_sql_trace	TT_SQL_TRACE	Only applicable to Baan IV and Baan 5.0. To view SQL query information.
use_shm_info	USE_SHM_INFO	Not applicable to Baan IV Enables or disables shared memory use.

Client resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
baan_sql_stmt_cache_size	BAAN_SQL_STMT_CACHE_SIZE	Only applicable to LN Defines the size of the query cache
bdb_debug	BDB_DEBUG	Sets debugging link between client and server.
bdb_driver	BDB_DRIVER	Sets database specifications.
bdb_use_row_version		Use a row version column to optimize delayed row locking.
bdb_max_server_schedule	BDB_MAX_SERVER_SCHEDULE	Not applicable to Baan IV Defines mechanism for terminating idle database drivers.
ssts_set_rows	SSTS_SET_ROWS	Sets number of rows read ahead.
	USR_DBC_RES	Specifies alternative resource file for client.

Server resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
bdb_max_sessions	BDB_MAX_SESSIONS	Defines number of sessions per driver.
bdb_max_session_schedule	BDB_MAX_SESSION_SCHEDULE	Defines mechanism for closing idle driver sessions.

Server resources used by all LN database drivers		
Resource name	**Environment variable**	**Description**
dbslog	DBSLOG	Allows driver profiling.
	DBSLOG_LOCK_PROF	Specifies lock time above which locks are logged.
dbslog_name	DBSLOG_NAME	Allows file name to be specified for logging.
dbsinit		Deprecated.
enable_refmsg	ENABLE_REFMSG	Causes logging of denied updates of delete actions.
	USR_DBS_RES	Specifies alternative resource file for server.

Resources used only by the LN MSQL database driver		
Resource name	**Environment variable**	**Description**
msql_array_fetch	MSQL_ARRAY_FETCH	Enables array fetching.
msql_array_insert	MSQL_ARRAY_INSERT	Enables array inserts
msql_dsn	MSQL_DSN	Specifies data source to communicate with MSQL. This resource has priority over `msql_obdc_driver`.
	MSQL_DUMP_MESG	Specifies the filename where error messages will be written.
msql_join_hint	MSQL_JOIN_HINT	Configures driver to send join hint to SQL Server.
msql_level1	MSQL_LEVEL1	Obsolete
msql_lock_timeout	MSQL_LOCK_TIMEOUT	Specifies a timeout value for SQL statements blocked by locks.
msql_log_warnings	MSQL_LOG_WARNINGS	Configures driver to log warning messages.
msql_max_arrsz	MSQL_MAX_ARRSZ	Adjusts buffer size for array fetch and insert.
msql_max_sql_buffer	MSQL_MAX_SQL_BUFFER	Obsolete.
msql_network_packetsize	MSQL_NETWORK_PACKETSIZE	Sets the network packet size.
msql_odbc_driver	MSQL_ODBC_DRIVER	Sets a name for the used ODBC driver.

Resources used only by the LN MSQL database driver		
Resource name	**Environment variable**	**Description**
msql_odbc_perf_stat	MSQL_ODBC_PERF_STAT	Starts or stops ODBC's performance data logging.
	MSQLPROF	Configures profiling.
msql_serverhost	MSQL_SERVERHOST	Specifies host name for MSQL instance.
	MSQLSTAT	Allows statistics to be gathered.
msql_opt_rows	MSQL_OPT_ROWS	Adjusts number of rows retrieved.
msql_retained_cursors	MSQL_RETAINED_CURSORS	Sets number of breaked cursors per driver connection.
msql_odbc_custom_parameters	MSQL_ODBC_CUSTOM_PARAMETERS	Customizable connection String Keywords with SQL Server Native Client.

Detailed description of driver resources and environment variables

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

Each group of resources is shown in alphabetical order.

Generic driver resources

baan_sql_cacherows / BAAN_SQL_CACHEROWS	
Driver resource	baan_sql_cacherows
Environment variable	BAAN_SQL_CACHEROWS
Client/Server resource	Set for both client and server
Type	Integer
Default	71
Description	Does not apply to Baan IV and Baan 5.0 application sessions. This variable influences the number of records that are internally cached by the query processor for sorting, aggregation functions or

baan_sql_cacherows / BAAN_SQL_CACHEROWS	
	prepared sets. When this limit is exceeded, temporary files are generated. A prime number must be specified for optimal performance of the internally used hash functions.

baan_sql_stmt_cache_size / BAAN_SQL_STMT_CACHE_SIZE7	
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
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 SQL query information being handled in client and server. When this variable is set, the client prints debug information to the log file (client). The information contains various categories you can enable separately, but most categories are not relevant. The available values of the `baan_sql_trace` variable and their descriptions are shown in these rows: • 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)

bdb_debug / BDB_DEBUG	
Default	0
Description	This variable is used 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). These categories of debugging information can be specified: • 00001: server types • 00002: database actions • 00004: delayed lock actions • 00010: reference information • 00040: TSS info from `%BSE%\lib\tss_mbstore` • 00100: permission information Multiple categories can be defined by adding the octal values. The value is compared bitwise to determine if a given category must be logged.

bdb_driver / BDB_DRIVER	
Driver resource	bdb_driver
Environment variable	BDB_DRIVER
Client/Server resource	Set for client only
Type	String
Default	None
Description	This variable is used to set a database specification, usually found in the table definition file, tabledef6.1 Baan IV) or tabledef6.2 (Baan 5.0/LN. When this variable is set, all tables are accessed using the database driver specified and the table definition file is not read. The driver specified must be defined in the `%BSE%\lib\ipc_info` file.

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
Description	Does not apply to Baan IV application sessions. This variable defines the mechanism for terminating idle database drivers by the application virtual machine. Whenever the database driver has no more open sessions, it can be terminated by the appli-

bdb_max_server_schedule / BDB_MAX_SERVER_SCHEDULE	
	cation virtual machine. Closing an idle database driver is done after a number of schedule ticks. A schedule tick is generated whenever an LN session is ended. At this point, all idle database drivers will 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, it can be closed by the client. Closing an idle session is done after a number of schedule ticks. A schedule tick is generated whenever an LN session is ended. At this point, all idle sessions will 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 three. Setting `bdb_max_session_schedule` to one results in fewer connections from the driver to the RDBMS 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

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 very 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	01
Description	This variable allows flags to be set to specify the optimizations to be used. At this time, legal value is 001. Other values are reserved and must not be used. A flag of 001 specifies that an optimistic approach must be used when checking for references in parent tables. The referenced row in the parent table is not locked, improving the overall concurrency. If this flag is not set, optimistic reference checking is not used. See ""Optimistic and pessimistic reference checks"". Pessimistic reference checking is not available for LN MSQL driver currently. Multiple categories can be defined by adding the octal values. The value is compared bitwise to determine if a given category must be logged. This parameter is deprecated and is removed in the next release.

dbslog / DBSLOG	
Driver resource	dbslog
Environment variable	DBSLOG
Client/Server resource	Set for server only
Type	Integer (octal)
Default	0
Description	This variable provides detailed debugging information about the online processing of the driver. The information is logged in the file `dbs.log` in the driver's current directory. These debugging categories can be specified: • 0000001: Data Dictionary information of tables within the driver • 0000010: Row action information • 0000020: Table 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 • 0002000: Query processing info (for tt_sql_trace info) • 0004000: Data buffering info (communication) • 0100000: Lock retries logged (includes session name) • 0200000: Logs successful locks and longest lock duration in a transaction Multiple categories can be defined by adding the octal values. The value is compared bit wise to determine if a given category must be logged.

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 will not be 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

DBSLOG_LOCK_PROF	
	of the commit or abort. This is the longest time a record remains locked during a transaction. Please 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	Represents a file name where DBS logging information is to be written. If there is already a file with the same name, it 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 it is set to 1, a log message is generated in the database driver log file when an update of a delete action was denied because of existing references. When it is set to 0, no log messages are generated.

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. Multiple blocks (and thus network round trips) are transferred if more rows are requested. This variable must be set to the same value for both client and server. This setting is ignored when the virtual ma-

rds_full / RDS_FULL	
	chine and the driver are running in the same process (combo mode), which is the default mode.

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
Description	Applies to Baan IV and Baan 5.0 application sessions. For newer application versions, it is replaced by `baan_sql_trace`. TT_SQL_TRACE is used to view the Infor SQL query information handled in the client and server. If this variable is set, the client prints debug information to the display. The server only prints information if this is permitted by the dbslog variable. The information contains various categories which you can enable separately. The categories are: • Evaluation trees • SQL statements • Bind variables • Timings • Communication debugging The available values of the TT_SQL_TRACE variable and the descriptions are: • 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 • 040000 (c+s): Show qp tokens

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

ssts_set_rows / SSTS_SET_ROWS	
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, which means that for one fetch request, three rows will be read. For the following two fetch requests, rows will be taken from the client row buffer or fetched from the database without re-executing the query.

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	Does not apply to Baan IV. This variable can be used to enable or disable the use of shared memory to each of the database driver DDs. There are two valid values for this variable: 0 and 1. When it is set to 0, shared memory is disabled. When it 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
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

USR_DBS_RES	
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`.

MSQL driver specific resources

Resource names are case sensitive. If you use a spelling checker, ensure that the resource names are not changed.

msql_array_fetch / MSQL_ARRAY_FETCH	
Driver resource	msql_array_fetch
Environment variable	MSQL_ARRAY_FETCH
Client/Server resource	Set for server only
Type	Boolean
Default	1 (enabled)
Description	This environment variable is used to enable or disable the array fetch interface. The valid values are 0 and 1. When set to 0, the array fetch interface is disabled. When set to 1, it is enabled. For more information, see ""Array interface" on page 42" .

msql_array_insert / MSQL_ARRAY_INSERT	
Driver resource	msql_array_insert
Environment variable	MSQL_ARRAY_INSERT
Client/Server resource	Set for server only
Type	Boolean
Default	1 (enabled)
Description	This environment variable is used to enable or disable the array insert interface. The valid values are 0 and 1. When set to 0, the array insert interface is disabled. When set to 1, it is enabled. Note that this option can be disabled by the driver, even if this resource is enabled. For example, if references must be checked or updated or the application requires immediate response from the driver as to whether the insert is successful, no array insert is done. For information, see ""Array interface" on page 42".

msql_dsn / MSQL_DSN	
Driver resource	msql_dsn
Environment variable	MSQL_DSN
Client/Server resource	Set for server only
Type	String
Default	None
Description	Allows specification of a data source name to be used by the driver for communication with SQL Server. This resource has priority over `msql_obdc_driver` resource.

MSQL_DUMP_MESG	
Driver resource	—
Environment variable	MSQL_DUMP_MESG
Client/Server resource	Set for server only
Type	String
Default	Not set
Description	This environment variable allows the location of the `log.msql.mesg` file to be specified. By default, it is stored in the `%BSE%\log` directory. This variable must be the fully qualified filename where error messages are written.

msql_join_hint / MSQL_JOIN_HINT	
Driver resource	msql_join_hint
Environment variable	MSQL_JOIN_HINT
Client/Server resource	Set for server only
Type	String
Default	NONE
Description	This variable allows the driver to send a join hint to SQL Server, which enforces a join strategy between two tables. Multiple join types in the same FROM clause of a query are not supported. **Caution:** In general, do not use join hints. Using this setting without proper understanding can cause severe performance problems. The possible values and their descriptions are shown in these rows: • NONE: No join hint will be sent

msql_join_hint / MSQL_JOIN_HINT	
	• LOOP: Looping strategy • HASH: Hashing strategy • MERGE: Merging strategy

msql_lock_timeout / MSQL_LOCK_TIMEOUT	
Driver resource	msql_lock_timeout
Environment variable	MSQL_LOCK_TIMEOUT
Client/Server resource	Set for server only
Type	Integer
Default	10
Description	Determines the timeout value, in seconds, for queries blocked by locks in the database server. The default is 10 which means “wait for 10 seconds for the lock to be released before giving up”. When set to -1, the driver waits indefinitely for the locks to be released and when set to 0, the driver does not wait.

msql_log_warnings / MSQL_LOG_WARNINGS	
Driver resource	msql_log_warnings
Environment variable	MSQL_LOG_WARNINGS
Client/Server resource	Set for server only
Type	Boolean
Default	0 (disabled)
Description	This variable enables or disables the logging of warning messages. The warnings are logged to the `%BSE%\LOG\msql.log.mesg` or to the file specified by `MSQL_DUMP_MESG` environment variable. The valid values are 0 and 1. When 0 is set, logging is disabled and when 1 is set, logging is enabled.

msql_max_arrsz / MSQL_MAX_ARRSZ	
Driver resource	msql_max_arrsz
Environment variable	MSQL_MAX_ARRSZ
Client/Server resource	Set for server only
Type	Integer
Default	5
Description	If `msql_array_insert` is enabled, this variable defines the maximum number of rows inserted at once into the RDBMS. If `msql_`

msql_max_arrsz / MSQL_MAX_ARRSZ	
	array_fetch is enabled, this variable defines the maximum number of rows fetched at once from the RDBMS. For more information about the array interface, see ""Array interface"".

msql_network_packetsize / MSQL_NETWORK_PACKETSIZE	
Driver resource	msql_network_packetsize
Environment variable	MSQL_NETWORK_PACKETSIZE
Client/Server resource	Set for server only
Type	Unsigned integer
Default	0 (using Microsoft SQL server's default)
Description	This variable allows the use of a different network packet size and can affect performance. For more information, check Microsoft SQL documentation about network packet size.

msql_odbc_driver / MSQL_ODBC_DRIVER	
Driver resource	msql_odbc_driver
Environment variable	MSQL_ODBC_DRIVER
Client/Server resource	Set for server only
Type	String
Default	NONE (empty string)
Description	This variable shows which ODBC driver must be used. If this variable is not set, the driver checks which ODBC drivers are available on the local system. It is mandatory to install the SQL Server Client software, to avoid errors.

msql_odbc_perf_stat / MSQL_ODBC_PERF_STAT	
Driver resource	msql_odbc_perf_stat
Environment variable	MSQL_ODBC_PERF_STAT
Client/Server resource	Set for server only
Type	Boolean
Default	0 (disabled)
Description	This variable is used to enable or disable ODBC's performance data logging. The data includes values such as the number of server round trips, average fetch time, average cursor size, number of selects, number of prepares, current connection count, maximum number of connections opened, bytes sent, and bytes received. The valid values are 0 and 1. Logging is disabled when set to 0. When set to 1, the

msql_odbc_perf_stat / MSQL_ODBC_PERF_STAT	
	performance data is logged to the `odbcP<process id>.log` file at `%BSE%\LOG`.

msql_opt_rows / MSQL_OPT_ROWS	
Driver resource	msql_opt_rows
Environment variable	MSQL_OPT_ROWS
Client/Server resource	Set for server only
Type	Integer
Default	The same value as the msql_max_arrsz default.
Description	This option allows you to specify to SQL Server that the first n rows must be retrieved fast. This will allow SQL Server to optimize the fetch request. Based on this value, SQL Server determines a suitable communication buffer size to improve performance. This variable can have a value of n, whose possible values and their descriptions are given in these rows: • n < 0: No optimization hint • 0: Use default (= the default value for msql_max_arrsz) • n > 0: OPTION(FAST n) query hint

msql_odbc_custom_parameters / MSQL_ODBC_CUSTOM_PARAMETERS	
Driver resource	msql_odbc_custom_parameters
Environment variable	MSQL_ODBC_CUSTOM_PARAMETERS
Client/Server resource	Set for server only
Type	String
Default	Not set
Description	**Note:** This variable can be used to add connection string keywords with SQL Server Native Client. Using this parameter is at your own risk. A connection keyword can change the behavior of SQL server database and result in errors. Infor does not provide support on issues caused by the use of these parameters.

MSQLPROF	
Driver resource	—
Environment variable	MSQLPROF
Client/Server resource	Set for server only
Type	Floating point

MSQLPROF	
Default	Not set
Description	When a value is specified in this variable, any statement that takes more than the number of seconds specified will be 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.

msql_retained_cursors / MSQL_RETAINED_CURSORS	
Driver resource	msql_retained_cursors
Environment variable	MSQL_RETAINED_CURSORS
Client/Server resource	Set for server only
Type	Integer
Default	20
Description	This resource sets the number of inactive (breaked) cursors that must be retained in the list for reuse. These cursors can be reused and, therefore, save prepare/bind overhead. The cursors must also have more resources such as memory than if they were closed and released. For more information, see ""Cursor management" on page 41" .

msql_serverhost / MSQL_SERVERHOST	
Driver resource	msql_serverhost
Environment variable	MSQL_SERVERHOST
Client/Server resource	Set for server only
Type	String
Default	None
Description	Allows specification of a host name for the driver to locate the SQL Server instance to be used.

MSQLSTAT	
Driver resource	—
Environment variable	MSQLSTAT
Client/Server resource	Set for server only
Type	Integer
Default	Not set

MSQLSTAT	
Description	This variable allows database driver statistics to be reported. If it is set to a value n greater than 0, statistics are logged every n seconds while the driver is active. If it is set to 0, a statistics report is generated when the driver terminates.

Parameter file formats and configuration options

B

There are two parameter files that influence the behavior of the database driver:

- The storage parameter file.
- The driver parameter file.

The formats and the parameters used with these two files are discussed. Additional information about the storage parameter file can be found in:

- "Storage parameter file" on page 29
- "Array interface" on page 42
- "Index Fill factor and indexes"

For additional information about the driver parameter file, see

- "Driver parameter file" on page 29

Parameter file naming

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

Parameter file formats

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

Storage parameter file format

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

```
<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:<query optimization>:<refresh time (obsolete)>:<driver
parameters> (obsolete)
```

Storage file format

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

The format of an entry in the `msql_storage`:

```
<table/module specification>:<company number>:<object type>:group:<query
 optimization>:<refresh time (obsolete)>:<storage parameters>
ttadv999,ttadv000:000:T:group:00400::
```

In this example, for the tables' ttadv999 and ttadv000 of company 000, the database driver is configured not to use index hints while fetching the data.

The implementation of the new storage and driver parameter files is a simple division of information in the `msql_storage` file. If you are still using the `msql_storage` file in your implementation, all references to either the storage or device parameter file in this technical reference manual must be adapted.

Parameter file field descriptions

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

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

object type		
Description	This field consists of a list of object (table or index) identifications to which the entry applies. You can specify 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) (DATA_COMPRESSION = PAGE)

group (only for msql_storage file, not for msql_storage_param file)	
Description	This field identifies the owner of the table. The value group must be specified.

query optimization	
Description	Specific flags related to indexes and tables can be specified. When specified on a “T” object entry, it defines the default for all indexes. These octal values can be used to set the flags for a specific index or table:
	0000 — no optimization 00200 — nested 00400 — iterative 01000 — filter
	Default values can be defined for each table entry. Optimizations are explained in more detail in "Query tuning".
Example	00400

refresh time	
Description	This is an obsolete parameter which is ignored by the driver.
Example	-

storage parameters	
Description	These are defined by the specific database driver implementation and often map to table and index creation options available in the host RDBMS. The storage parameters for the MSQL database driver can be specified in either all upper case or all lower case letters. The following MSQL storage parameters are defined: • FILL FACTOR <n>Includes the fill factor <n> clause in the create index statement during index creation. • FILEGROUP <file group name>Includes the “on <file group name>” clause in the create table or create index statement during object creation.
Example	FILEGROUP primary

driver parameters	
Description	This is an obsolete parameter which is ignored by the driver.
Example	

Setting parameters in the storage parameter file

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

This is an example of a storage parameter file:

```
ttadv999,ttadv000:000:T:FILEGROUP mydatafilegroup
ttadv999,ttadv000:000:I:FILEGROUP myindexfilegroup
*:*:T:011:FILEGROUP mydatafilegroup
*:*:I:FILEGROUP myindexfilegroup
```

In the storage parameter file example, the tables ttadv999 and ttadv000 of company 000 are created in the file group `mydatafilegroup`. The associated indexes, except index 1, is created in file group `myindexfilegroup` and index 1 is a clustered index. All other tables and indexes are created in the default file group.

Note: If the file group for a table or index is not specified, the table and index data are created in the default (usually primary) file group. If you want to physically separate the index data, you must specify a different file group.

Setting parameters in the driver parameter file

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

```
*:*:T::00400::
*:*:I::
```

For group objects—table or index—a single entry must be created, consisting of all LN users in the group.

Write permission for the device parameter file must be controlled, and only the DBA can have write permission.

Conversion from previous porting sets

Existing Infor installations, installed on Infor Baan IVc or LN porting sets before porting set 7.1a, may not have the storage or driver parameter files. In these porting sets there is just the `msql_storage` file.

When the driver parameter file and the storage parameter file do not exist, the LN database driver tries to open the `msql_storage` file for compatibility.

The Convert Table and Index Repository (ttdba0540m000) session that converts the storage file to the storage parameter file and the driver parameter file. Use this session to convert the storage file.