Infor Enterprise Server Technical Reference Guide for Oracle Database Driver

Release 10.7.x

Publication Information

Release: Infor LN 10.7.x
Publication Date: January 21, 2019
Document code: ln_10.7.x_lndboradrvtrg__en-us

Contents

About this guide....6
Intended audience....6
Related documents....6
Contacting Infor....7
Chapter 1: LN Database Driver Overview....8
LN architecture....8
Display tier....9
Application tier....9
Database tier....10
Data flow through the LN architecture....10
LN hardware configurations....11
Chapter 2: LN Database Organization....16
LN data dictionary....16
Table naming convention....17
Column naming convention....17
Index naming convention....19
Data type mapping....19
Additional constraints....20
Chapter 3: Database Driver Internal Processing....21
Data integrity....21
Referential integrity....21
Distributed databases....22
Data buffering....22
Database driver SQL processing....22
Oracle Call Interface (OCI)....22
SQL processing....23

Setting driver behavior....24
Driver resources....24
Environment variables....25
Storage parameter file....26
Chapter 4: Database Security....28
Database security....28
Groups....28
Object security....29
Authentication....30
DBA module....30
Chapter 5: Database Driver Profiling and Statistics....32
Profiling....32
Profiling example....33
To gather statistics....33
Troubleshooting....34
Logging database driver trace information....35
Logging errors....35
Chapter 6: Database Driver Configuration and Tuning....37
Cursor management....37
Array interface....38
Query tuning....38
Query hint generation....38
Concatenated expressions....43
Specifying query tuning....44
Locking behavior....45
Implicit and explicit locking....45
Delayed locking....46
Statement and lock timeouts....46
Appendix A: Database Driver Resources and Environment Variable....47
Summary of Oracle driver resources and environment variables....47
Detailed description of driver resources and environment variables....51
Generic driver resources....51
Oracle driver specific resources....62
Oracle maintenance program specific resource....68

Resources passed to the Oracle RDBMS....68
Appendix B: Storage Parameter File Format and Driver Configuration Options....71
File format: $BSE/lib/ora/ora_driver_param....71
File format: $BSE/lib/ora/ora_storage_param....71
Parameter file field descriptions....72
Appendix C: To Connect to Oracle....75
Connection parameters....75
Globalization support....77
NLS parameters....77
Collation and Sorting....78
Appendix D: Conversion from CHAR to VARCHAR2 strings....80
Difference between the CHAR and VARCHAR2 data type 80
Conversion from CHAR to VARCHAR....81

About this guide

This document describes the database driver that forms the interface between the LN application server layer and the Oracle database server. In this document, the database driver is referred to as the LN Oracle driver. The information is Infor-release independent and can also be used for Infor Baan IVc and Infor Baan 5.0

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

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

Intended audience

This document is intended for anyone who wants to configure or customize the Infor Enterprise Server database driver for Oracle. The database driver and the Oracle database server can run on a UNIX system or a Windows system. This document assumes that you have operating system and database knowledge.

Related documents

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

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

- *Infor Baan IVc - Pre installation Guide for Windows*
- *Infor Baan IVc - Pre installation Guide for UNIX*
- *Infor Baan IVc - Deployment Guide for Windows and UNIX*
- *Infor Baan 5.0 - Installation Guide for Oracle on UNIX*
- *Infor Baan 5.0 - Installation Guide for Oracle on Windows*
- *Infor LN - Installation Guide*
- *Infor Enterprise Server - Technical Manual*
- *Infor LN - Performance, Tracing and Tuning Guide*
- *Infor LN - Performance, Tracing and Tuning Guide for Oracle Server*

Contacting Infor

If you have questions about Infor products, go to Infor Concierge at https://concierge.infor.com/ and create a support incident.

If we update this document after the product release, we will post the new version on the Infor Support Portal. To access documentation, select **Search > Browse Documentation**. We recommend that you check this portal periodically for updated documentation.

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

Chapter 1: LN Database Driver Overview

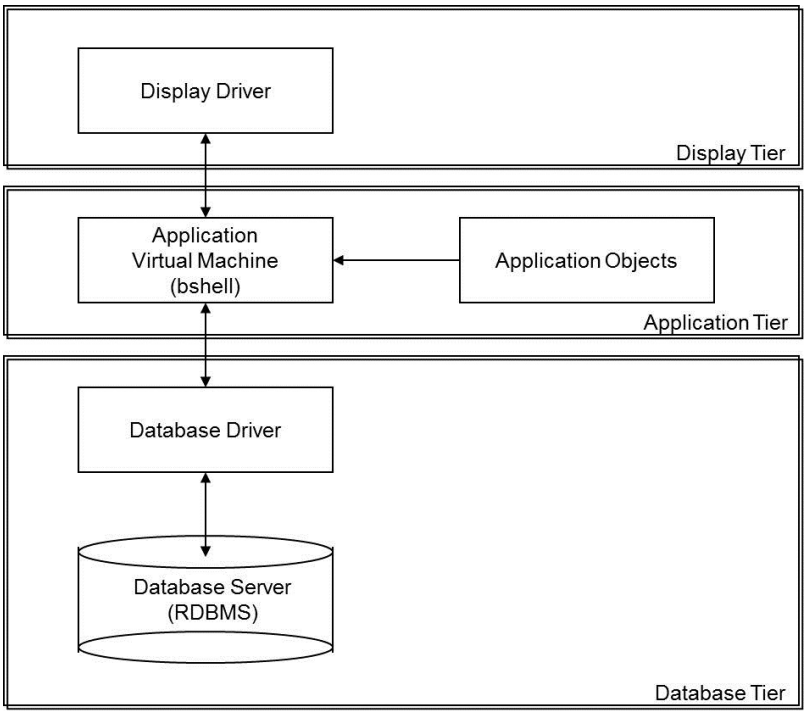
The database driver is an important part of Infor's commitment to an open-systems client/server architecture. Because the LN architecture includes the LN software and a third-party relational database management system (RDBMS), the driver provides an interface between the LN software and the various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

LN architecture

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

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

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



Display tier

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

Application tier

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

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

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

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

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

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

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

Database tier

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

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

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

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

- Oracle
- DB2
- Microsoft SQL Server

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

Data flow through the LN architecture

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

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

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

If the application virtual machine runs a database query from an application object, the virtual machine first determines whether a running database driver is available to process the query. If no database driver is running, or if the running database driver instances are communicating with a database server

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

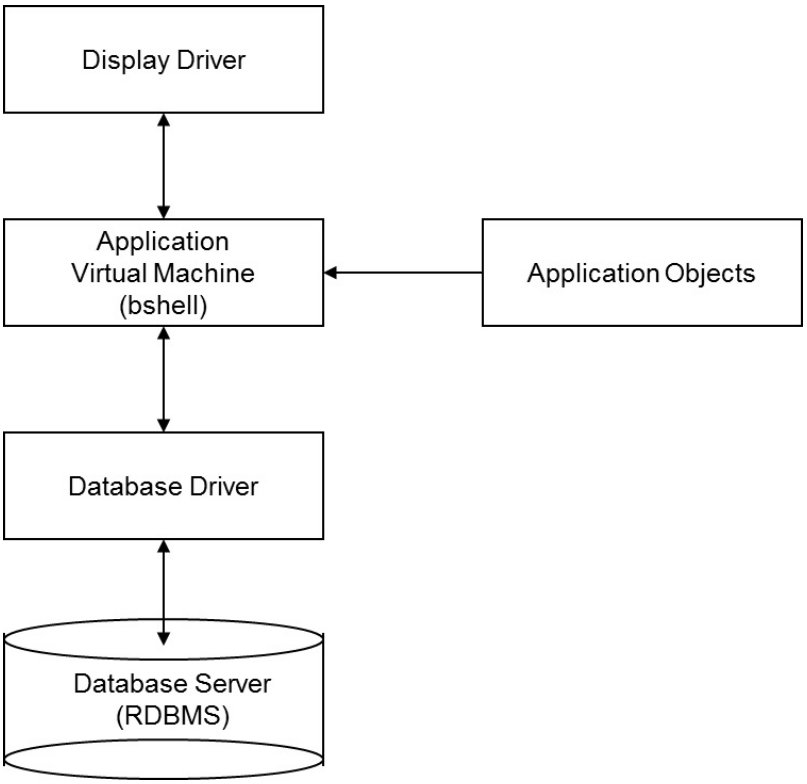
The database driver uses the SQL statements compatible with the specific RDBMS being used to translate the database query into an appropriate query. Each database driver takes advantage of the design of the particular RDBMS that the driver supports, so the resulting SQL statements are valid for the RDBMS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

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

LN hardware configurations

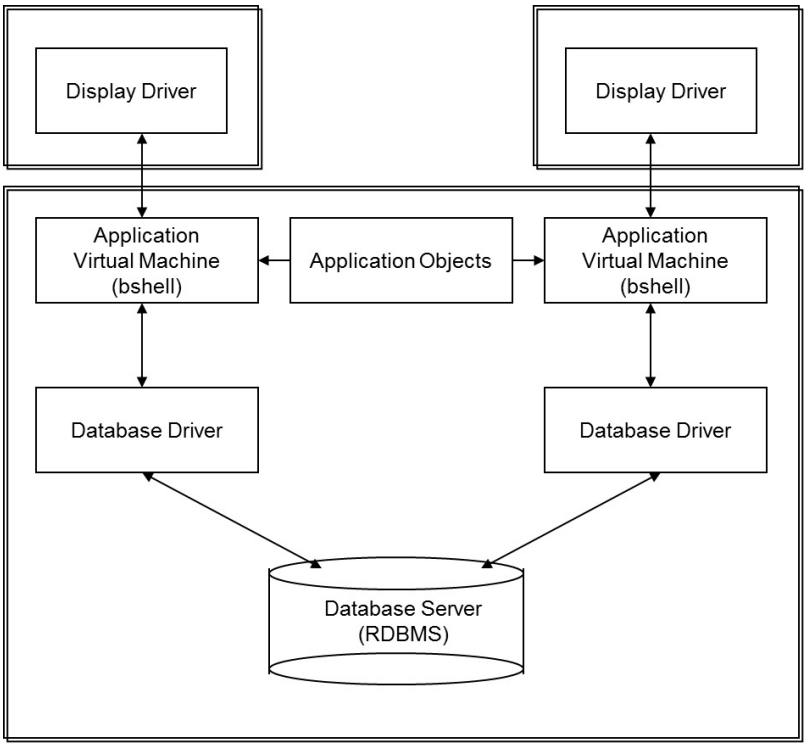
Several hardware configurations are supported for an LN implementation. These configurations include standalone mode and many variations of client/server mode. Available hardware, data storage requirements, and performance expectations determine the most appropriate hardware configuration.

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



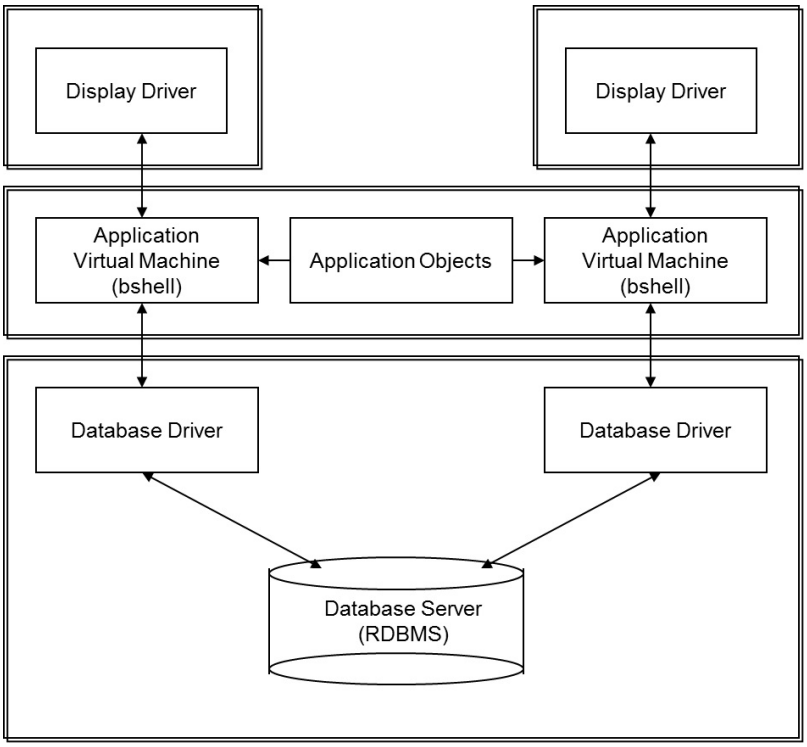
In a client/server configuration, the components of the LN architecture are distributed across two or more machines. Various client/server configurations exist; this section describes the most common configurations.

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

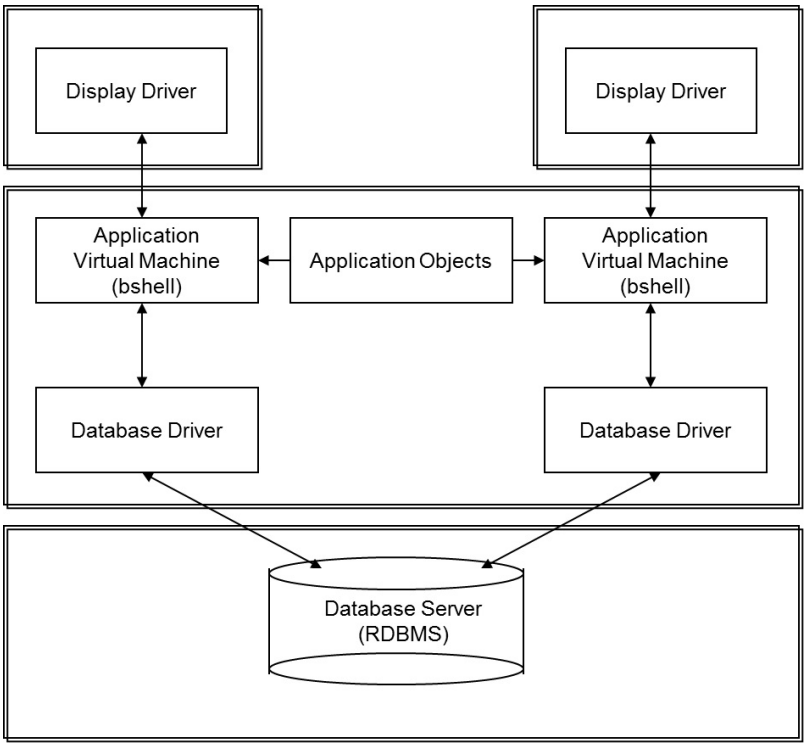


If two machines are available as servers, two configurations are commonly used. In both configurations, the display drivers reside on the user workstations. In the first configuration, the application tier is placed on one server, while the database driver and the database server are placed on another.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

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



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



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

Chapter 2: LN Database Organization

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

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

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

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

LN data dictionary

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

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

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

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

Table naming convention

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

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

The explanation of the table name:

- DD Table Name

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

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

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

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

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

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

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

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

Column naming convention

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

The rules for column names are:

- General:

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

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

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

  The maximum length depends on the version of Oracle and the Oracle data type you use.

  In Oracle 11g and 12c, the Oracle data type CHAR has a limit of 2,000 characters. If an LN string column exceeds this limit, the column is split into segments with up to 2,000 characters each. The first 2,000 characters are placed in a column in which the name of the column is extended with `#1`. The next 2,000 characters are placed in a column with a name extended with `#2`, and so on until all the characters of the string are placed in a column.

  The Oracle data type VARCHAR2 has a limit of 4000 characters.

  For example, if an LN string column called `desc` contains 2,500 characters, these two Oracle columns are created in Oracle:

  - `t$desc#1`: size 2000
  - `t$desc#2`: size 500
- Array columns:

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

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

  Note: If the element is of type String, and one element type exceeds the maximum Oracle character size, the element is split, such as:
  - `t$str$1#1`: Element 1, part 1.
  - `t$str$1#2`: Element 1, part 2.
- Array compression:

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

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

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

  Note: If a compressed array column exceeds the maximum length of a RAW Oracle column, the column is split into segments, as previously described.

Index naming convention

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

Index names have this format:

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

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

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

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

Note:

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

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

- When the Multilanguage Application Data feature is enabled for a table or a table has BLOB columns, the following 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 Oracle counterparts:

Mapping between LN and Oracle data types	
LN data type	**Oracle data type**
Byte	NUMBER
Enumerated	NUMBER
Integer	NUMBER
Long	NUMBER
UTC Date/Time	DATE (1)
Text	NUMBER

Mapping between LN and Oracle data types	
LN data type	**Oracle data type**
(Bit)Set	NUMBER
Float	NUMBER
Double	NUMBER
String	CHAR(n)/VARCHAR(n) NCHAR(n)/NVARCHAR(n) (2 or 3)
TSS string	CHAR(n)/VARCHAR(n) NCHAR(n)/NVARCHAR(n) (2 or 3)
Date	DATE (1)
Raw	RAW
Blob	BLOB
(4)	RAW

1. The empty date and utc timestamp is represented in Oracle as January 1, 4712 B.C 00:00:00.
2. The LN Oracle driver can use either the ANSI-compliant CHAR(n) Oracle data type or the VARCHAR2(n) Oracle data type. This is controlled by the resource ora_use_varchar.

   For more information on the conversion from a CHAR(n) based database to a VARCHAR2(n) based database, see Conversion from CHAR to VARCHAR2 strings on page 80.
3. If LN is installed in Unicode mode, the LN Oracle driver uses the Oracle NCHAR or NVARCHAR2 data type.
4. The Oracle RAW type is used to store compressed array columns and UUIDs generated if the Multilanguage Application Data feature is used or if a table has one or more BLOB colums (except in Infor Baan IVc, where the UUID column is a string column).

Additional constraints

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

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

Chapter 3: Database Driver Internal Processing

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

These topics are discussed:

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

Data integrity

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

This section provides an overview of the features that the LN Oracle database driver uses to ensure referential integrity to work with distributed databases, and to apply data buffering techniques. Locking strategies are described in detail in Database Driver Configuration and Tuning on page 37.

Referential integrity

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

Distributed databases

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

Data buffering

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

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

Database driver SQL processing

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

Because the LN database driver uses Oracle Call Interface (OCI) to communicate with Oracle, the following section describes OCI.

Oracle Call Interface (OCI)

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

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

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

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

The LN Oracle driver also uses these features of OCI:

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

SQL processing

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

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

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

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

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

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

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

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

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

Setting driver behavior

To configure the LN Oracle database driver, several facilities are available, with the most common being through driver resources; two other facilities are environment variables and the storage file. The driver resources and environment variables are described in more detail in Database Driver Resources and Environment Variable on page 47 and the storage file in Storage Parameter File Format and Driver Configuration Options on page 71.

Driver resources

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

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

```
oracle_home:/usr/oracle/product/11gR2

ora_timeout:{300,300,300,300,300}
```

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

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

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

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

Environment variables

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

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

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

Server environment variables

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

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

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

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

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

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

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

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

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

```
tccom010:812:oracle8(ORAPROF=0.4):N
```

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

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

Client environment variables

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

Storage parameter file

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

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

A storage parameter file is defined for each database driver. The storage parameter file for the LN Oracle database driver is called ora_storage_param and is located in the Windows directory %BSE%\lib\ora or the UNIX directory $BSE/lib/ora. For a detailed description of the format of the

storage parameter file, see Storage Parameter File Format and Driver Configuration Options on page 71.

Chapter 4: Database Security

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

Database security

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

Groups

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

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

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

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

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

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

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

Object security

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

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

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

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

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

Authentication

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

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

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

You define all the LN users, their corresponding Oracle logon names and passwords, and the name of the group to which the names and passwords are assigned in the Windows file %BSE%\lib\ora\ora_users or the UNIX file $BSE/lib/ora/ora_users. The format of each entry in this file is as follows:

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

The LN application virtual machine starts the LN Oracle driver on behalf of the user. From the Windows file %BSE%\lib\ora\ora_users or the UNIX file $BSE/lib/ora/ora_users, the driver identifies the Oracle user and the user's password and establishes the connection to Oracle.

The group logon procedure also includes a password, which is defined in the Windows file %BSE%\lib\ora\ora_groups or the UNIX file $BSE/lib/ora/ora_groups. The format is as follows:

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

Oracle can also use the operating system authentication to permit users to connect to the Oracle database. To enable this, you can create automatic logons (ops$ accounts). After the user logs onto the operating system, the user can connect to Oracle without supplying the user name and password. This facility is also available for the Oracle driver. You can configure the ops$ prefix with the OS_AUTHENT_PREFIX resource; this resource must be equal to the value of the OS_AUTHENT_PREFIX in the init<SID>.ora parameter file. The value defaults to ops$. The OS_AUTHENT_PREFIX resource is described in Database Driver Resources and Environment Variable on page 47.

DBA module

The DBA module maintains the database administration files that the Oracle database driver uses. This module enables an administrator to register authorized users and give users access to data. To maintain the administration files the database driver requires at run time, a tool is provided with the LN

Oracle database driver. The administration files are stored in the Windows directory %BSE%\LIB\ORA or the UNIX directory $BSE/lib/ora.

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

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

Chapter 5: Database Driver Profiling and Statistics

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

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

Profiling

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

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

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

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

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

```
SET ORAPROF=5.0
```

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

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

```
tccom010:812:oracle8(ORAPROF=0.4):N
```

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

Profiling example

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

```
98-03-09[15:00:39]: Profiling value = 0.02 sec
------ Profiling value exceeded -----
<jim><bshellmain>:98-03-09[15:01:16.054]: Time (multi_fetch) : 0.033598 seconds
SQL statement:
SELECT /*+ FIRST_ROWS USE_NL(a,b,c) INDEX(a tttadv112000$idx1) */ a.t$pacc,a.t$
cpac,a.t$sequ,b.t$desc,c.t$expi FROM baan5.tttadv112000 a,baan5.tttadv130000 b,
baan5.tttadv200000 c WHERE c.t$cpac = :1 AND c.t$cmod = :2 AND c.t$cses = :3
AND c.t$vers = a.t$vers AND c.t$rele = a.t$rele AND c.t$cust = a.t$cust AND b.t
$cpac = :4 AND b.t$vers = a.t$vers AND b.t$rele = a.t$rele AND b.t$cust = a.t$
cust AND b.t$kdes = :5 AND b.t$rkey = :6 AND b.t$clan = :7 AND a.t$pacc = :8
AND a.t$cpac = :9 ORDER BY 1,2,3
-------------------------------------
------ Profiling value exceeded -----
<jim><bshellmain>:98-03-09[15:01:16.101]:
Time (parse) : 0.023754 seconds
SQL statement:
SELECT /*+ FIRST_ROWS INDEX(a tdctcs440000$idx9) */ a.t$adat,a.t$bfix,a.t$ccod,
a.t$cdat,a.t$cgrp,a.t$conf,a.t$cprj,a.t$csta,a.t$ctyp,a.t$delc,a.t$dsca,a.t$dti
m,a.t$ercs,a.t$erss,a.t$etim,a.t$fdat,a.t$fre1,a.t$fre2,a.t$fre3,a.t$fre4,a.t$f
re5,a.t$fre6,a.t$fre7,a.t$ider,a.t$idln,a.t$loca,a.t$modc,a.t$name,a.t$pact,a.t
$phse,a.t$pref,a.t$prgr,a.t$prib,a.t$prio,a.t$prjm,a.t$rdat,a.t$rtim,a.t$rtst,a
.t$rtyp,a.t$rusr,a.t$sact,a.t$scmp,a.t$sdat,a.t$solv,a.t$stat,a.t$ttim,a.t$txta,
a.t$txtp,a.t$txts,a.t$unit,a.t$uref,a.t$utyp,a.t$vref FROM baan5.tdctcs440000 a
WHERE a.t$delc = :1 AND (a.t$loca >= :2) AND (a.t$loca <= :3) AND (a.t$rtyp =
:4 AND a.t$unit = :5 AND a.t$uref = :6 AND a.t$utyp = :7 AND a.t$scmp = :8
AND a.t$vref = :9 AND a.t$csta = :10 AND a.t$loca > :11)
ORDER BY 39,50,51,52,42,53,8,26,24
-------------------------------------
```

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

To gather statistics

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

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

- Number of commits and rollbacks.

For each action, the cumulative elapsed time spent and the average time is logged. You can enable the statistics with the environment variable ORASTAT. If you set the variable to zero, when the driver terminates, a statistics report is generated; in other words, it exits from LN Tools or the session. If a value n greater than zero is specified, the driver logs an incremental report every n seconds (the driver must be active). The statistics report is written to the file ORASTAT in the current directory.

These examples show how you can set ORASTAT:

```
SET ORASTAT=0
SET ORASTAT=30
```

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

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

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

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

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

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

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

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

Troubleshooting

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

Logging database driver trace information

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

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

- Table and index information (data dictionary).
- The SQL statements being executed.
- Values of the input and output bind variables.
- Other function-level debug statements.

To enable tracing, you can use the environment variable `DBSLOG`. Debugging information is appended to the file `dbs.log` in the Windows directory `%BSE%\tmp` or the UNIX directory `$BSE/tmp`. On a Windows system, if you invoke logging remotely with `rexec`, the log is stored in the Windows file `%BSE%\tmp`.

To enable tracing, specify this command:

```
SET DBSLOG=1570
```

To enable tracing for only categories of interest, several tracing categories are defined. For more details, see Database Driver Resources and Environment Variable on page 47.

Logging errors

In a Windows environment, the database driver logs error messages in the Windows application Event Log or the log files in the `%BSE%\LOG` directory. In a UNIX environment, the driver logs error messages in the log flies in the directory `$BSE/log`. From these log files, you can retrieve the following information:

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

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

- abs(error_code) + 1000

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

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

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

Chapter 6: Database Driver Configuration and Tuning

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

Cursor management

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

- **`retained_cursors`**

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

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

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

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

Array interface

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

The default number of rows fetched with a single OCI call is 5. To change this setting, you can specify the resource ora_max_array_fetch. To disable the array interface, set `ora_max_array_fetch:1`.

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

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

Query tuning

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

Query hint generation

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

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

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

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

SQL Query Hints

Query hints are part of the SQL language since Infor Baan 5.0.

The SQL query hints can be divided into three groups:

1 The "no hints" hint:

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

2 Hints that influence the execution tree:

This group consists of these hints:

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

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

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

Hints based on the ORDER BY clause

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

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

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

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

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

Hints based on the WHERE clause

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

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

To determine the best matching index, use these rules:

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

  These operands are ignored:

  - LIKE expression [NOT] LIKE expression.
  - Exists [NOT] EXISTS expression.
  - OR – lists expression OR expression.

  - Set expression [NOT] IN expression.
  - Negations expression != expression

    expression <> expression

    expression NOT BETWEEN expression

    expression NOT INRANGE expression

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

  Base columns that do not have a counterpart are ignored.

  For example:

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

  is handled as:

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

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

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

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

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

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

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

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

Example of index selection

This example is based on the Baan 5.0b data model.

SQL query:

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

Ignore unusable parts:

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

Determine fields that are considered for the hint generation:

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

Determine the matching indexes:

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

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

Example of weight calculation

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

SQL query:

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

Indexes:

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

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

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

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

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

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

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

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

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

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

An index hint for index 2 is generated.

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

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

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

An index hint for index 2 is generated.

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

_index1: empno(0.000) [stopped]
   weight 0.000
```

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

An index hint for index 2 is generated.

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

Concatenated expressions

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

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

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

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

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

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

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

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

To detect long running or bad performing queries, and then experiment with these various techniques, use the `ORAPROF` variable. For more information on `ORAPROF`, see Database Driver Resources and Environment Variable on page 47.

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

- The nested technique:

  The three conditions are ORed to this expression:

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

- c1 > "tt"

  This can be rewritten as:

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

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

- The iterative technique:

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

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

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

- The filter technique:

  This technique is related to the nested technique but has a different approach. This technique initially selects too many rows, but then filters out those rows that do not match the total WHERE clause. This technique selects based on the first column in a concatenated index and filters out rows with the NOT() operator. The query is solved as:

  - c1 >= "tt" AND NOT(c1 = "tt" AND (c2 < "adv" OR
  - c2 = "adv" AND (c3 < "000"))

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

Specifying query tuning

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

On Windows specify this file: `%BSE%\lib\ora\ ora_driver_param`

On UNIX specify this file: `$BSE/lib/ora/ora_driver_param`.

For a list of the actual values you can specify, see "Storage Parameter File Format and Driver Configuration Options on page 71".

Locking behavior

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

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

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

Implicit and explicit locking

If the LN Oracle database driver performs an update, the database can perform the update with implicit or explicit locking.

With implicit locking, before you carry out the UPDATE or DELETE, you do not perform a SELECT FOR UPDATE.

The rows are locked when Oracle performs the update. The UPDATE and DELETE statements do not have a NOWAIT option; therefore, if the resource is locked, the statement and the session to which the statement belong wait until the lock is released or a timeout occurs. If a timeout occurs, the client retries the same operation or rolls back the transaction. If you run an INSERT statement with implicit locking, if the table that the row is to be inserted into is being held with an exclusive lock, the INSERT waits until the lock is released or until a timeout occurs. If a timeout occurs, the client retries the same operation or rolls back the transaction.

With explicit locking, a SELECT FOR UPDATE statement is issued prior to the UPDATE or DELETE, which explicitly locks the required rows. You can run the SELECT FOR UPDATE with the NOWAIT option. With the NOWAIT option, if another process locks the rows, Oracle does not wait for the lock to be released, but returns immediately with an error condition. To use explicit locking for an INSERT statement, you can generate a LOCK TABLE IN ROW SHARE MODE to ensure that the insert will not be blocked. The LOCK TABLE enables the NOWAIT option. With the NOWAIT option, Oracle does not wait for a lock to be released, but returns immediately with an error condition.

Whenever possible, the database driver requests that Oracle use implicit locking. If implicit locking is not possible, the driver requests explicit locking. To change the driver default behavior to use explicit locking, the user can set a flag in the ora_init resource; for more information about this resource, see Database Driver Resources and Environment Variable on page 47.

Delayed locking

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

Statement and lock timeouts

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

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

To specify the lock wait period for SELECT FOR UPDATE, INSERT, DELETE, UPDATE, and LOCK TABLE statements, use the ORA_TIMEOUT environment variable or the ora_timeout resource variable. The following table specifies the default and minimum timeouts. Note that when you experiment with these options, you must do so with care. Timeout values are specified in increments of seconds.

Action	Default timeout	Minimum timeout
select for update	0 (NOWAIT)	0
Insert	None	1
Update	None	1
Delete	None	1
lock table	0 (NOWAIT)	0

You can specify the timeouts in the resource ora_timeout as follows:

```
Ora_timeout:{[value], [value], [value], [value], [value]}
Action:       select   insert   update   delete   lock table
```

Here, all parameters are optional and you can specify integers to indicate the timeout for the particular action. If no parameter is specified, the default behavior for the actions is performed.

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

```
ora_timeout:{0, 5, 10, 15, 0}
```

Note: On Windows keep the default ora_timeout values, as by the lack of out-of-band interrupts on the Oracle client/server communication the timeout is not handled instantly.

Appendix A: Database Driver Resources and Environment Variable

You can use the database driver resources and environment variables as configuration parameters to modify the behavior of the Oracle 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 Oracle database driver.

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

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

Summary of Oracle driver resources and environment variables

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

- Client and server resources that are used by all LN database drivers.
- Client resources that are used by all LN database drivers.
- Server resources hat are used by all LN database drivers.
- Resources that are used only by the LN Oracle database driver.
- Resources that are passed to the Oracle database server.

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

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

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

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

Server resources that are used by all LN database drivers		
Resource name	**Environment variable**	**Description**
bdb_max_sessions	BDB_MAX_SESSIONS	Defines the number of sessions per driver
bdb_max_session_schedule	BDB_MAX_SESSION_SCHEDULE	Defines the mechanism to close idle driver sessions
dbslog	DBSLOG	To perform driver profiling
-	DBSLOG_LOCK_PROF	Specifies the lock time after which locks are logged
dbslog_name	DBSLOG_NAME	To specify the file name for logging
dbsinit	DBSINIT	Deprecated

Server resources that are used by all LN database drivers		
Resource name	**Environment variable**	**Description**
hint_index_weight_equal	HINT_INDEX_WEIGHT_EQUAL	Defines the equal-weight parameter for the query hint generation
hint_index_weight_range	HINT_INDEX_WEIGHT_RANGE	Defines the range-weight parameter for the query hint generation
hint_index_weight_factor	HINT_INDEX_WEIGHT_FACOR	Defines the weight-factor parameter for the query hint generation
max_sql_input_binds	MAX_SQL_INPUT_BINDS	To adjust the size of internal ‘where bind’ buffers.
shm_max_dd_cache_size	SHM_MAX_DD_CACHE_SIZE	Defines the maximum size of the database driver DD cache
mle_join_type	MLE_JOIN_TYPE	Specifies join type between data table and shadow table, with translations, in a Multi Language Enabled (MLE) environment.
query_comments	QUERY_COMMENTS	To trace used connections in the database server.
table_load_array_size	TABLE_LOAD_ARRAY_SIZE	Defines the maximum number of rows that are inserted during data load.
-	USR_DBS_RES	Specifies alternative resource file for server

Resources that are used only by the LN Oracle database driver		
Resource name	**Environment variable**	**Description**
ansi_outer_join	ANSI_OUTER_JOIN	To alter the generated SQL syntax for outer-joins.
baan_oracle_prefetch	BAAN_ORACLE_PREFETCH	To alter Oracle’s pre-fetch behavior.
connection_pooling	CONNECTION_POOLING	To choose between shared or dedicated Oracle sessions.
ora_alter_session	ORA_ALTER_SESSION	To change Oracle session settings.
ora_init	ORA_INIT	Defines several driver behaviors
ora_max_array_fetch	ORA_MAX_ARRAY_FETCH	Defines the maximum number of rows that are fetched
ora_max_array_insert	ORA_MAX_ARRAY_INSERT	Defines the maximum number of rows that are inserted

Resources that are used only by the LN Oracle database driver		
Resource name	**Environment variable**	**Description**
ora_parallel_degree	ORA_PARALLEL_DEGREE	Enables parallel index creation and statistics generation
oraprof	ORAPROF	Enables profiling
orastat	ORASTAT	To gather statistics.
ora_hint_no_hints	ORA_HINT_NO_HINTS	Disable the generation of query hints
ora_timeout	ORA_TIMEOUT	Sets the time-out value
ora_use_varchar	ORA_USE_VARCHAR	Use the Oracle VARCHAR2/NVARCHAR2 data type for string data
os_authent_prefix	OS_AUTHENT_PREFIX	Defines prefix for Oracle's operating system authentication
retained_cursors	RETAINED_CURSORS	Defines the number of retained cursors
sql_trace	SQL_TRACE	Enables Oracle's trace facility

Resources that are used only by the LN Oracle maintenance program		
Resource name	**Environment variable**	**Description**
ora_default_tablespace	ORA_DEFAULT_TABLESPACE	To set a default tablespace for newly created Oracle users.
ora_temporary_tablespace	ORA_TEMPORARY_TABLESPACE	To set a temporary tablespace for newly created Oracle users

Resources that are passed to the LN Oracle database server		
Resource name	**Environment variable**	**Description**
nls_comp	NLS_COMP	Sets Oracles `nls_comp` parameters
nls_lang	NLS_LANG	Sets Oracles NLS parameters
nls_sort	NLS_SORT	The type of sort to use
oracle_home	ORACLE_HOME	Sets Oracle's `ORACLE_HOME`
oracle_local_template	ORACLE_LOCAL_TEMPLATE	Oracle8 only: The default connect template to connect to a local database
oracle_service_name	ORACLE_SERVICE_NAME	Specifies the Oracle database to connect to
oracle_sid	ORACLE_SID	Sets Oracle's `ORACLE_SID`

Detailed description of driver resources and environment variables

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

Each group of resources is shown in alphabetical order.

Generic driver resources

baan_sql_cacherows / BAAN_SQL_CACHEROWS	
Driver resource	`baan_sql_cacherows`
Environment variable	`BAAN_SQL_CACHE_ROWS`
Client/Server resource	Set for both client and server
Type	Integer
Default	71
Description	Does not apply to Baan IV and Baan 5.0 application sessions. This variable influences the number of records that the query processor internally caches for sorting, aggregation functions, or prepared sets. If you exceed this limit, temporary files are generated. For optimal performance of the internally used hash functions, specify a prime number.

baan_sql_stmt_cache_size / BAAN_SQL_STMT_CACHE_SIZE	
Driver resource	`baan_sql_stmt_cache_size`
Environment variable	`BAAN_SQL_STMT_CACHE_SIZE`
Client/Server resource	Set for client only
Type	Integer
Default	330
Description	Does not apply to Baan IV and Baan 5.0 application sessions. 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`

baan_sql_trace / 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. If you set this variable, debug information is printed by the client to the log file (client) or `dbs.log` file (server). The information contains various categories you can enable separately, but most categories are not relevant for the audience of this document. The relevant values of the `baan_sql_trace` variable and the descriptions of these values are: • 0002000: Major query interface logging • 0004000: Detailed query interface logging

bdb_debug / BDB_DEBUG	
Driver resource	`bdb_debug`
Environment variable	`BDB_DEBUG`
Client/Server resource	Set for client only
Type	Integer (octal)
Default	0
Description	This variable allows generating debugging information about the communication between the client and the database driver. If you set this variable, the client prints debugging information to standard error (`stderr`). You can specify these categories of debugging information: • 00001: server types • 00002: database actions • 00004: delayed lock actions • 00010: reference information • 00100: permission information To define multiple categories, you can add the octal values. To determine if a given category must be logged, the value is compared bitwise.

bdb_driver / BDB_DRIVER	
Driver resource	`bdb_driver`
Environment variable	`BDB_DRIVER`

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

bdb_max_server_schedule / BDB_MAX_SERVER_SCHEDULE	
Driver resource	`bdb_max_server_schedule`
Environment variable	`BDB_MAX_SERVER_SCHEDULE`
Client/Server resource	Set for client only
Type	Integer
Default	3
Description	This variable defines the mechanism for terminating idle database drivers by the application virtual machine. If the database driver has no more open sessions, the application virtual machine can terminate the driver. You close an idle database driver after several schedule ticks. A schedule tick is generated whenever you end an LN session. At this point, all idle database drivers have a schedule counter incremented. When the value of the schedule counter reaches the value of `bdb_max_server_schedule`, the database driver ends.

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

bdb_max_session_schedule / BDB_MAX_SESSION_SCHEDULE	
Driver resource	`bdb_max_session_schedule`

bdb_max_session_schedule / BDB_MAX_SESSION_SCHEDULE	
Environment variable	BDB_MAX_SESSION_SCHEDULE
Client/Server resource	Set for server only
Type	Integer
Default	3
Description	This variable defines the mechanism for closing idle sessions in the driver. If the client process has no more references, cursors or queries, to the session, the client can close the client process. The client closes an idle session after several schedule ticks. A schedule tick is generated whenever an LN session is ended. At this point, all idle sessions have a schedule counter incremented. If the value of the schedule counter reaches the value of bdb_max_session_schedule, the session closes. The default for bdb_max_session_schedule is three. If you set bdb_max_session_schedule to 1, this results in fewer connections from the driver to the RDBMS. This is because whenever an LN session ends, the corresponding RDBMS session (logon) closes (logoff).

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

dbsinit / DBSINIT	
Driver resource	dbsinit
Environment variable	DBSINIT

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

dbslog / DBSLOG	
Driver resource	`dbslog`
Environment variable	`DBSLOG`
Client/Server resource	Set for server only
Type	Integer (octal)
Default	0
Description	This variable provides detailed debugging information about the online processing of the driver. The information is logged in the file `dbs.log` in the driver's current directory. You can specify these debugging categories: • 0000001: Data dictionary information on tables in the driver • 0000002: Query info (SQL Level 1) • 0000004: Query plan info (SQL Level 2) • 0000010: Row action information • 0000020: Table action information • 0000040: Transaction action information • 0000100: DBMS input/output data (SQL Level 2) • 0000200: Administration file info (SQL drivers) • 0000400: DBMS SQL statements • 0001000: General debug statements • 0002000: Query processing info (for tt_sql_trace info) • 0010000: Log Connections and Cursors • 0004000: Data buffering info (communication) • 0100000: Lock retries logged (includes session name) • 0200000: Logs successful locks and longest lock duration in a transaction To define multiple categories, you can add the octal values. To determine if a given category must be logged, the value is compared bitwise.

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

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

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

hint_idx_weight_equal / HINT_IDX_WEIGHT_EQUAL	
Driver resource	`hint_idx_weight_equal`
Environment variable	`HINT_IDX_WEIGHT_EQUAL`
Client/Server resource	Set for server only
Type	Integer
Default	1
Description	This variable controls the index hint generation. For information on the influence of this variable on the index hint generation, see Database Driver Configuration and Tuning on page 37.

hint_idx_weight_factor / HINT_IDX_WEIGHT_FACTOR	
Driver resource	`hint_idx_weight_factor`
Environment variable	`HINT_IDX_WEIGHT_FACTOR`
Client/Server resource	Set for server only
Type	Floating point number
Default	1.0
Range	0.0 … 1.0
Description	This variable controls index-hint generation. For information on the influence of this variable on the index-hint generation, see Database Driver Configuration and Tuning on page 37.

hint_idx_weight_range / HINT_IDX_WEIGHT_RANGE	
Driver resource	`hint_idx_weight_range`
Environment variable	`HINT_IDX_WEIGHT_RANGE`
Client/Server resource	Set for server only
Type	Integer
Default	1
Description	This variable controls the index-hint generation. For information on the influence of this variable on the index-hint generation, see Database Driver Configuration and Tuning on page 37.

rds_full / RDS_FULL	
Driver resource	`rds_full`

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

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

mle_join_type / MLE_JOIN_TYPE	
Driver resource	`mle_join_type`
Environment variable	`MLE_JOIN_TYPE`
Client/Server resource	Set for server only
Type	Integer
Default	0 (INNER)

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

query_comments/QUERY_COMMENTS	
Driver resource	`query_comments`
Environment variable	`QUERY_COMMENTS`
Client/Server resource	Set for server only
Type	Integer
Default	0
Description	This variable allows enabling tracing queries in the database server. Enabling this resource results in inserting comments for some types of statements, for example queries. The comments include additional information about the database driver its process ID, the LN user name and the LN session name. **Warning:** Enabling this resource severely impacts performance as it affects the query plan cache of the database server. Enabling this setting is for diagnostic purposes only.

shm_max_dd_cache_size / SHM_MAX_DD_CACHE_SIZE	
Driver resource	`shm_max_dd_cache_size`
Environment variable	`SHM_MAX_DD_CACHE_SIZE`
Client/Server resource	Set for server only
Type	Integer
Default	12582912 (12Mb)

shm_max_dd_cache_size / SHM_MAX_DD_CACHE_SIZE	
Description	This variable defines the maximum number of bytes that is used to cache database driver data dictionaries in the shared memory cache. If you exceed this limit, no further information are cached. A value of `0` is interpreted as unlimited. See `use_shm_info`.

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

table_load_array_size / TABLE_LOAD_ARRAY_SIZE	
Driver resource	`table_load_array_size`
Environment variable	`TABLE_LOAD_ARRAY_SIZE`
Client/Server resource	Set for server only
Type	Integer
Default	100
Description	This variable defines the number of rows that are inserted using array interfacing when using the `bdbpost` or `bdbreconfig` utility.

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

tt_sql_trace / TT_SQL_TRACE	
Description	Applies to Baan IV and Baan 5.0 application sessions. For newer application versions, it is replaced by `baan_sql_trace`. This variable is introduced to view the Infor SQL query information being handled in client and server. If you set this variable, debug information is printed by the client to the display. The server only prints information if the `dbslog` variable allows it. The information contains various categories you can enable separately, such as evaluation trees, SQL statements, bind variables, timings, and communication debugging. The possible 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

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

USR_DBC_RES	
Driver resource	—
Environment variable	`USR_DBC_RES`
Client/Server resource	Set for client only
Type	String
Default	None

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

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

Oracle driver specific resources

ansi_outer_join / ANSI_OUTER_JOIN	
Driver resource	`ansi_outer_join`
Environment variable	`ANSI_OUTER_JOIN`
Client/Server resource	Set for server only
Type	Boolean
Default	1
Description	This resource is deprecated, and is removed in a future release. This variable influences the generated Oracle SQL for queries that contain outer joins. If you enable this variable, the database driver generates Oracle SQL that complies with the ANSI syntax. If the variable is disabled, the driver generates the Oracle proprietary syntax. This resource must be enabled when the applications uses the Document Authorization (DBCM) feature or the Multi Data Language (MLE) feature.

baan_oracle_prefetch / BAAN_ORACLE_PREFETCH	
Driver resource	`baan_oracle_prefetch`
Environment variable	`BAAN_ORACLE_PREFETCH`
Client/Server resource	Set for server only
Type	Integer
Default	`1`: Equal to the Oracle default
Description	This variable influences the number of records that are pre-fetched by the Oracle client libraries used by the Oracle driver. For more detailed information, see the Oracle documentation: *OCI_ATTR_PREFETCH_ROWS*

connection_pooling / CONNECTION_POOLING	
Driver resource	`connection_pooling`
Environment variable	`CONNECTION_POOLING`
Client/Server resource	Set for server only
Type	Integer
Default	1
Description	This resource controls the sharing of Oracle sessions between LN sessions. When enabled, all read-only database access shares a single Oracle session. Any access that causes a change or a lock in the database temporarily switches to another Oracle session. When disabled all LN sessions have their own dedicated Oracle session.

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

ora_hint_no_hints/ORA_HINT_NO_HINTS	
Driver resource	`ora_hint_no_hints`
Environment variable	`ORA_HINT_NO_HINTS`
Client/Server resource	Set for server only
Type	Integer
Default	0
Description	When set, this resource disables the query hint generation of the Oracle driver. Only queries that have explicit query hints specified in the application source get an Oracle hint. All other hints are suppressed,

ora_init / ORA_INIT	
Driver resource	`ora_init`
Environment variable	`ORA_INIT`
Client/Server resource	Set for server only
Type	Integer (octal)
Default	`0011000`: Specifies fixed char type and explicit inserts.
Description	This resource defines several driver behaviors. To select multiple behaviors, you can add the octal values. • `0001000`: Use fixed char type The resource `ora_use_varchar` has precedence over this setting. • `0010000`: Explicitly lock for inserts (action: `LOCK TABLE`).

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

ora_max_array_insert / ORA_MAX_ARRAY_INSERT	
Driver resource	`ora_max_array_insert`

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

ora_parallel_degree / ORA_PARALLEL_DEGREE	
Driver resource	ora_parallel_degree
Environment variable	ORA_PARALLEL_DEGREE
Client/Server resource	Set for server only
Type	Integer
Default	4
Description	If you specify a value greater or equal to 1, the database driver adds a PARALLEL <n> clause to the generated CREATE INDEX statements. The value is also passed to the SQL function that generates statistics for a table. This resource is introduced in porting set 9.0a.01

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

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

ora_timeout / ORA_TIMEOUT	
Driver resource	ora_timeout
Environment variable	ORA_TIMEOUT
Client/Server resource	Set for server only
Type	String
Default	None
Description	This resource specifies the lock wait period for specific actions. For more information on locking behavior and setting timeouts, see Statement and lock timeouts on page 46.

ora_use_varchar / ORA_USE_VARCHAR	
Driver resource	ora_use_varchar
Environment variable	ORA_USE_VARCHAR
Client/Server resource	Set for server only
Type	Integer
Default	0
Description	This variable controls the mapping between the LN string data type and the Oracle string data type. If this variable is set to 0, string columns are created and handled as the CHAR or NCHAR Oracle data type. If this variable is set to 1, string columns are created and handled as the VARCHAR2 or NVARCHAR2 Oracle data type. Because these Oracle data types have different functional behaviors, the LN Oracle driver interfaces differently to Oracle depending on the value of this resource. See Conversion from CHAR to VARCHAR2 strings on page 80.

os_authent_prefix / OS_AUTHENT_PREFIX	
Driver resource	`os_authent_prefix`
Environment variable	`OS_AUTHENT_PREFIX`
Client/Server resource	Set for server only
Type	String
Default	`ops$`
Description	With this variable, you can define the prefix for Oracle's operating system authentication. You can define the prefix in the Oracle `init<SID>.ora` parameter file also.

retained_cursors / RETAINED_CURSORS	
Driver resource	`retained_cursors`
Environment variable	`RETAINED_CURSORS`
Client/Server resource	Set for server only
Type	Integer
Default	50
Description	This resource sets the number of inactive cursors that must be retained in the list for reuse by session. For more information on cursor management, see Cursor management on page 37. Note that all application sessions, that share the same database connection, share the same inactive cursor pool.

sql_trace / SQL_TRACE	
Driver resource	`sql_trace`
Environment variable	`SQL_TRACE`
Client/Server resource	Set for server only
Type	String
Default	Not set
Description	With this resource you can enable the Oracle trace facility. To enable the facility, set the value to True.

Oracle maintenance program specific resource

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

ora_temporary_tablespace / ORA_TEMPORARY_TABLESPACE	
Ora_maint resource	ora_temporary_tablespace
Environment variable	ORA_TEMPORARY_TABLESPACE
Type	String
Default	Not set
Description	If a new user is added to the Oracle database, the default temporary tablespace is set to SYSTEM. If this resource is set, an ALTER USER TEMPORARY TABLESPACE <string> command will be issued after the user is created.

Resources passed to the Oracle RDBMS

nls_comp / NLS_COMP	
Driver resource	nls_comp
Environment variable	NLS_COMP
Client/Server resource	Set for server only
Type	String
Default	Not set (Set by the driver based on mode and Oracle version)
Description	The value of this resource must be ‘binary’ for SingleByte and MultiByte installations. For Unicode mode, it must be 'linguistic’. For more information, see To Connect to Oracle on page 75.

nls_lang / NLS_LANG	
Driver resource	nls_lang
Environment variable	NLS_LANG
Client/Server resource	Set for server only
Type	String
Default	Not set (Set in db_resource during installation)
Description	The value of this resource must be set to the same value as the Oracle instance the driver connects to. Check the Oracle table v$nls_parameters for the appropriate values for your database instance. The following is an example of how to use this parameter: nls_lang:american_america.WE8MSWIN1252 When operating in Single Byte or MultiByte mode you cannot set the AL32UTF8 character set. Unicode mode does not have this restriction. For more information, see To Connect to Oracle on page 75.

nls_sort / NLS_SORT	
Driver resource	nls_sort
Environment variable	NLS_SORT
Client/Server resource	Set for server only
Type	String
Default	binary
Description	If the database instance is created with nls settings that imply a sorting other than binary, you must specify this resource. The Oracle RDBMS does not handle nls during the evaluation of the WHERE clause of queries; therefore, unexpected results will occur if nls_sort is not set to binary. If unicode mode is enabled, nls_sort must have a value such as: `uca0700_ducet_vn` You can sort nls_sort by session; however, because the table indexes can only be created in one sorting order, this makes no sense. Changing the value of nls_sort requires the rebuild of all LN indexes.

oracle_home / ORACLE_HOME	
Driver resource	oracle_home
Environment variable	ORACLE_HOME
Client/Server resource	Set for server only
Type	String

oracle_home / ORACLE_HOME	
Default	Not set (usually set in the tabledef6.2 file)
Description	Sets Oracle's ORACLE_HOME environment variable. For more information, see To Connect to Oracle on page 75.

oracle_local_template / ORACLE_LOCAL_TEMPLATE	
Driver resource	oracle_local_template
Environment variable	ORACLE_LOCAL_TEMPLATE
Client/Server resource	Set for server only
Type	String
Default	For more information, refer to To Connect to Oracle on page 75.
Description	The value of this resource is used as the connect string to Oracle when the Net8™ BEQ adapter is used. For more information, see To Connect to Oracle on page 75.

oracle_service_name / ORACLE_SERVICE_NAME	
Driver resource	oracle_service_name
Environment variable	ORACLE_SERVICE_NAME
Client/Server resource	Set for server only
Type	String
Default	Not set.
Description	The value of this resource is used as the connect string to Oracle. For more information, see To Connect to Oracle on page 75.

oracle_sid / ORACLE_SID	
Driver resource	oracle_home
Environment variable	ORACLE_HOME
Client/Server resource	Set for server only
Type	String
Default	Not set (Normally set in the tabledef6.2 file)
Description	Sets Oracle's ORACLE_SID environment variable. For more information, see To Connect to Oracle on page 75.

Appendix B: Storage Parameter File Format and Driver Configuration Options

File format: $BSE/lib/ora/ora_driver_param

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

- The ownership of the database tables. For more information, see Database Security on page 28.
- The query generation. For more information, see Query tuning on page 38.
- The refresh time of the record cache: this field is obsolete.

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

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

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

A driver parameter file is defined for each database driver. The storage file for the LN Oracle database driver is called `ora_driver_param` and is located in the Windows directory `%BSE%\lib\ora` or the UNIX directory `$BSE/lib/ora`.

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

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

For information on each of the fields in an entry in the storage parameter file, see Parameter file field descriptions on page 72.

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

File format: $BSE/lib/ora/ora_storage_param

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

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

```
*:*:T: TABLESPACE dataspace PCTFREE 5
```

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

A storage parameter file is defined for each database driver. The storage file for the LN Oracle database driver is called `ora_storage_param` and is located in the Windows directory `%BSE%\lib\ora` or the UNIX directory `$BSE/lib/ora`.

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

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

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

Each of the fields in an entry in the storage parameter file are described in the section Parameter file field descriptions on page 72.

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

Parameter file field descriptions

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

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

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

Compress specification		
Description	When this field is present the table or index is compressed.	
Example	COMPRESS=1;	Table compression is used for the selected table(s) or index(es).

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

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

Refresh time	
Description	Ignored by the oracle driver

Storage parameters	
Description	The specific database driver implementation defines these parameters, and often defines map-to-table and index creation options available in the host RDBMS. Several driver configuration options specific to the Oracle driver allow the user to customize table and index attributes at create time. These storage options are used to affect specified tables. For a complete description of the parameters you can add to the CREATE TABLE and CREATE INDEX commands, refer to the Oracle SQL Language Reference Manual.
Example	TABLESPACE DATA PCTFREE 5 INITRANS 3

Appendix C: To Connect to Oracle

The Oracle driver of LN is named oracle8; this driver can be used for Oracle 8i and later. The oracle8 driver can be used for all supported Oracle versions.

For an up-to-date list of supported Oracle versions, see the technical notes, release notes, of the porting set.

The `-v` option of the oracle8 driver also shows this information:

```
% ora8_srv6.2 -v
-------------------------------------------------------
Product ver.: 9.3a
Port no.    : PA.5044.64-bit
Date        : Mon Oct  1 11:09:32 CEST 2018
Uname       : Linux nlbalbld3 3.0.101-71-default #1 SMP
Machine-id  : LINUX_INTEL
OS-release  : LINUX3.0.101
CFLAGS      :
LOADFLAGS   :
DBMS ver.   : Oracle 11.2/12.1/12.2/18 (shared library based)
-------------------------------------------------------
Copyright © 2018. Infor All rights reserved. www.infor.com
```

The Oracle driver can technically handle the listed Oracle versions.

Note that these versions are not always officially supported (anymore) by Infor and Oracle.

See Infor Support Portal KB 1183466 for the actual support matrix.

The oracle8 driver can distinguish between the listed Oracle versions, and adapts itself to the installed Oracle version.

Connection parameters

oracle_home

You must set this resource.

The `oracle_home` resource variable must contain a path to a valid Oracle client installation.

oracle_service_name

Setting this resource is optional for local Oracle databases. If not set, you must set the resource `oracle_sid`.

If the oracle service name is set, this resource must contain the name of a valid service configured for Oracle Net in the tnsnames.ora file. The value is used as the connect string to Oracle.

oracle_sid

This resource is required if `oracle_service_name` is not set.

The value of oracle_sid is used with `oracle_local_template` to generate a connect string to a local database instance.

oracle_local_template

The `oracle_local_template` resource contains a template of the connect string used to connect to Oracle if the `oracle_service_name` resource is not set.

Before use, some special characters in this template are replaced by the actual values of some resource values: the symbol @ is replaced with `oracle_sid`, and ? is replaced with `oracle_server_name`.

Depending on the platform and installed Oracle version, one of these build-in templates is used:

```
(DESCRIPTION =                              Oracle 11g/12c + Windows
  (SDU=32768)(TDU=32768)
  (ADDRESS =
    (PROTOCOL = beq)
    (PROGRAM  = oracle )
    (ARGV0    = oracle@ )
    (ARGS='(DESCRIPTION=(LOCAL=YES)(SDU=32768)(TDU=32768)(ADDRESS=(PROTO
COL=beq)))')
    (ENVS='ORACLE_SID=@')
  )
  (CONNECT_DATA = (SID = @))
)
(DESCRIPTION =                                      Oracle 11g/12c + Unix
  (SDU=32768)(TDU=32768)
  (ADDRESS =
    (PROTOCOL = beq)
    (PROGRAM  = ?/bin/oracle )
    (ARGV0    = oracle@ )
    (ARGS='(DESCRIPTION=(LOCAL=YES)(SDU=32768)(TDU=32768)(ADDRESS=(PROTO
COL=beq)))')
    (ENVS='ORACLE_SID=@')
  )
  (CONNECT_DATA = (SID = @))
)
```

Globalization support

NLS parameters

Oracle has many NLS parameters, but only a small number affect the behavior of the LN application.

Most parameters are irrelevant, because the Oracle driver interfaces on such a low level that these parameters have no effect.

For example, the Oracle driver handles date and time values in the internal (7-byte) Oracle format. All Oracle conversions affected by the NLS date and time parameters are not used.

The `nls_parameters` with an affect are mentioned later. For a more detailed description of these parameters, see the Oracle documentation.

nls_characterset

This parameter is determined at the creation time of the database instance. Each Oracle (client) application has a unique setting. If the `nls_characterset` of the Oracle instance does not match with the Oracle client, Oracle converts between the character sets. If a character is not available, a replacement character, such as the base character without accents `?` or `¿` display.

While this feature seems beneficial, for an application such as LN that wants to retrieve the data from the RDBMS exactly as the data was added, data corruption must be avoided. You can force the Oracle client application (the LN Oracle driver) to use the same `nls_characterset` as the Oracle instance.

The only way to set the `nls_characterset` for a client application is to specify the `nls_lang` variable. Note that this also affects the `nls_language` and `nls_territory` parameters, because the definition of `nls_lang` is:

```
nls_lang = <nls_territory>_<nls_language>.<nls_characterset>
```

The LN Oracle driver verifies the `nls_characterset` parameter, and an error is logged in case of a mismatch. To avoid data corruption, the driver stops.

nls_comp

To avoid the problems as described in the section Collation and Sorting on page 78, the Oracle driver always sets this parameter to LINGUISTIC when operating in Unicode mode. When operating in Single Byte or MultiByte mode the parameter is set to BINARY.

nls_language

This parameter is implicitly set if `nls_lang` is specified.

This parameter does not affect the LN application; however, Oracle error strings written in the error log file of the database driver appear in this language.

nls_sort

The default value depends on other NLS parameters, and is determined at Oracle installation time.

The parameter must be set to a supported value as described in Collation and Sorting on page 78.

nls_territory

This parameter is implicitly set if `nls_lang` is specified.

This parameter does not affect LN.

nls_nchar_characterset

This parameter is only relevant if the Oracle driver runs in Unicode mode. Preferably, set this value to AL16UTF16 (UTF-16 encoding).

Collation and Sorting

Oracle issues

Oracle has offered language-dependent sorting for some time. In practice, you could not use this sorting option.

The major flaw in Oracle's sorting architecture was that the sorting operation was carried out on the result set, while the database engine was completely unaware of the sorting.

Example (NLS_SORT=german):

```
Table vendor:
                                Name    id
                                übels   10
                                conn    20
                                selmer  30
                                willson 40
SQL> select *
from vendor
order by name;
                                Name    id
                                conn    20
                                selmer  30
                                übels   10
                                willson 40
SQL> select *
from vendor
where name between 'a' and 'z'
order by name;
                                Name    id
                                conn    20
                                selmer  30
                                willson 40
```

The second query is missing the record ‘übels’, which the database engine rejected because ü is binary larger than z.

From Oracle 9i onwards, Oracle offers functionality such as functional indexes and the `nls_comp` parameter that solve this problem.

For more information, see the Oracle documentation, such as the Oracle 11g Database Globalization Support Guide, Release 1 (B28298-02)

LN porting set

Another essential requirement is that the sorting order in the porting set must be identical to the sorting order in the underlying database.

If the porting set uses a sorting order other than the RDBMS, undefined behavior occurs; this implies you cannot use all sorting and collation orders offered by the RDBMS vendor.

Supported sorting orders

Currently, only these sorting and collation orders are supported:

- **`binary`**:
  This order must be used if the porting set Single byte and Multibyte setups.
- **`infor_generic_m`**:
  This order is used if the porting set runs in Unicode mode. This collation is only supported for Oracle versions up to Oracle 11.2.
- **`UCA0610_DUCET_S3_VN_BN_NY_EN_FN_HN_DN_MN`**:
  This order is used if the porting set runs in Unicode mode for Oracle 12.1.
- **`UCA0700_DUCET_VN`**
  This order is used if the porting set runs in Unicode mode for Oracle 12.2 onwards.

Other sorting and collation orders might be added in future releases.

Configuration

This resource variable `nls_sort` affects the sorting of the result sets returned from the Oracle RDBMS. In theory, the variable can differ by Oracle session. In practice, this is not possible, because the table indexes must be created for this sorting order to have a reasonable performance.

The nls_sort variable is set in the `$BSE/lib/defaults/db_resource` file by the LN installer.

If you change the value of `nls_sort`, all table indexes created by the oracle driver must be rebuilt by a complete reorganize tables operation from the LN tools.

Appendix D: Conversion from CHAR to VARCHAR2 strings

This appendix provides information on the Oracle database conversion from CHAR to VARCHAR2 strings.

Difference between the CHAR and VARCHAR2 data type

The LN Oracle driver strongly relies on the ANSI-compliant string comparison semantics, which are blank-padded semantics. Unlike other database vendors, Oracle has chosen to implicitly link the comparison behavior to the data type.

The Oracle CHAR/NCHAR data types comply with the ANSI-comparison semantics, while the Oracle VARCHAR2/NVARCHAR2 data types do not comply.

From porting set 8.6a onwards, you can use the VARCHAR2/NVARCHAR2 data type for LN. The resource `ora_use_varchar` controls this behavior. For the driver to operate correctly in varchar mode, several criteria must be met.

To preserve the blank-padded semantics, the LN Oracle driver must ensure that all data in Oracle is trimmed during comparison and padded during concatenation. When needed, the driver generates additional RTRIM() or CAST() operators in the Oracle query text.

Another issue is that the Oracle VARCHAR2 data type handles an empty string as a NULL value, which can lead to incorrect results. NULL values supply a different outcome in boolean expressions than empty strings.

To avoid incorrect values, the driver generates additional COALESCE() or NVL() calls in the Oracle query text.

Note:

For correct query evaluation when using the VARCHAR2/NVARCHAR2 data type, these constraints on the data in the database must be met:

- An empty string must be represented by a string of exactly 1 space.
- All other string data must never have trailing spaces.

It is important that all external integrations that change LN data comply with these constraints.

Without these data constraints and driver modifications done in porting set 8.6 or later, you cannot carry out operations such as comparison and concatenation in a predefined way, with unpredictable results.

Mixing CHAR and VARCHAR2 data types leads to a violation of the constraints mentioned earlier and to unpredictable results.

Conversion from CHAR to VARCHAR

The conversion from CHAR/NCHAR to VARCHAR2/NVARCHAR2 requires a complete database export/import cycle using LN tools.

When sufficient disk space is available and downtime is not a problem, complete these steps:

1. Make a complete database export using `bdbpre`.
2. Set the ora_use_varchar:1 resource.
3. Import the database using `bdbpost`.

You can use Oracle tools for this migration, but full knowledge of these tools is required.

Note: An ALTER TABLE command to change the data type from CHAR to VARCHAR2 does not shrink the data size. To achieve this, more complex conversions are required.

Ensure that after conversion to VARCHAR2, the data constraints mentioned in the previous section hold.

A table with VARCHAR2 columns which violates these constraints can be repaired with this UPDATE statement:

```
UPDATE <table>
       SET <column> = COALESCE(RTRIM(<column>),' ')
```

Tables with NVARCHAR2 columns used in Unicode mode require a slightly different statement:

```
UPDATE <table>
       SET <column> = COALESCE(RTRIM(<column>),N' ')
```