Lawson Administration: Data Access Using IBM DB2

Version 10.0.x
Published November 2012

Important Notices

The material contained in this publication (including any supplementary information) constitutes and contains confidential and proprietary information of Infor.

By gaining access to the attached, you acknowledge and agree that the material (including any modification, translation or adaptation of the material) and all copyright, trade secrets and all other right, title and interest therein, are the sole property of Infor and that you shall not gain right, title or interest in the material (including any modification, translation or adaptation of the material) by virtue of your review thereof other than the non-exclusive right to use the material solely in connection with and the furtherance of your license and use of software made available to your company from Infor pursuant to a separate agreement, the terms of which separate agreement shall govern your use of this material and all supplemental related materials ("Purpose").

In addition, by accessing the enclosed material, you acknowledge and agree that you are required to maintain such material in strict confidence and that your use of such material is limited to the Purpose described above. Although Infor has taken due care to ensure that the material included in this publication is accurate and complete, Infor cannot warrant that the information contained in this publication is complete, does not contain typographical or other errors, or will meet your specific requirements. As such, Infor does not assume and hereby disclaims all liability, consequential or otherwise, for any loss or damage to any person or entity which is caused by or relates to errors or omissions in this publication (including any supplementary information), whether such errors or omissions result from negligence, accident or any other cause.

Without limitation, U.S. export control laws and other applicable export and import laws govern your use of this material and you will neither export or re-export, directly or indirectly, this material nor any related materials or supplemental information in violation of such laws, or use such materials for any purpose prohibited by such laws.

Trademark Acknowledgements

Publication Information

Release: 10.0.x

Publication date: November 6, 2012

Document Number: IBM_10.0.x_W_03

Contents

Data Access Overview

1

This chapter contains the following topics:

Configuring the Database

This section contains the following topics:

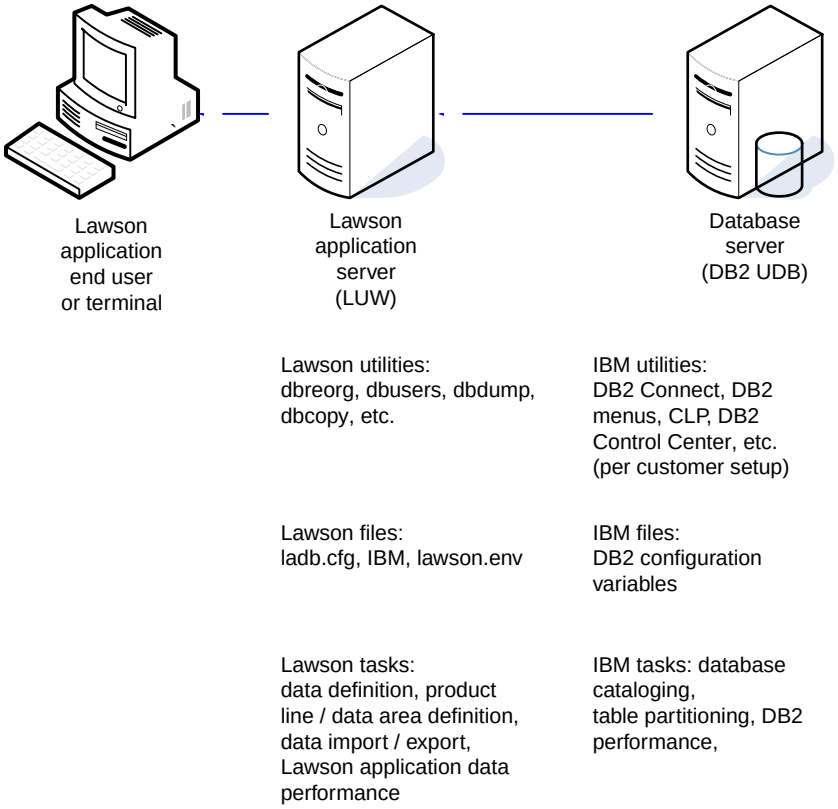
Installing and Configuring the Database

Install the database and complete the preliminary configuration before installing the Environment. Create databases to hold GEN, LOGAN, and applications data using tools provided by the database vendor.

Make the following decisions. Will you:

- Use multiple instances of the database?
- Put Lawson applications and the database on the same system?
- Use SMS or DMS tablespaces?
- Use bufferpools?

Figure 1. Machine diagram: DB2 installed on a UDB server



Record key values such as the installation path (*database*_HOME), database name, and the system ID (SID) when installing the database. These values are required during and after installation.

The *Lawson Administration: Data Access* guide for your database indicates the required database versions, additional required or optional products to install, and any required database settings that you need to set during installation.

Configuring Database Users

The Lawson database driver (ibmdb) can access the database as the **lawson** user or as the individual user who logged on and made a request for data. Create a method of access that fits your security and auditing needs any time after installing the database.

When configuring database users, do the following:

- Using tools provided by the database vendor, create the database user(s) that you want to access the database and assign them any privileges outlined in this guide.
- Create the same user(s) in the Environment, using Lawson Security and Resource Management or Lawson User Administration.
- Create a database service using Lawson Security and Resource Management.
- Edit the database driver configuration file (ORACLE, IBM, or MICROSOFT) and add the appropriate parameter settings for your method of access.

Configuring the Database Server (ladb)

The Database Server (**ladb**) analyzes and manages user requests, determines which database to access, and starts the database-specific driver (ibmdb). The Database Server Configuration file (ladb.cfg) contains the parameters that define the maximum number of processes, files, and servers; the maximum size of files; and other values. You use a text editor to update the ladb.cfg file. Consider the following:

- Determine your data-processing requirements for open dictionaries and files, the maximum number of user processes that can access the database, and how you want to configure driver reuse. Set the values in the ladb.cfg file appropriately.
- Some values are preset by installation programs, based on information you enter. You may need to make ongoing adjustments based on your data processing requirements. Any values in the configuration file can be updated at any time. Changes are implemented by restarting the database driver (ladb).

Managing Data

This section contains the following topics:

Implementing Dictionary Modifications

When you make structural changes to the product line or data areas, do a full dictionary build and reorganization for the product line and all associated data areas and data IDs. Build the dictionary and reorganize the database after you make a structural change to the dictionary or any of the following changes:

- Create a new data area, based on an existing product line.
- Assign a database space to the data area or product line.
- Add new files.
- Add new system codes.
- Add new fields to a database file.
- Change the definition of existing fields.
- Delete unused fields.

When multiple data areas exist, **blddbdict** creates or rebuilds the structure dictionary first, then the data area dictionaries. If all dictionaries generate without error, they are moved to the appropriate files. When you make changes to one or more data areas, do a data area build and reorganization of the specific data area dictionaries.

To implement dictionary changes, consider the following steps:

1. Create or modify the database definition using **dbdef**, **editda**.

   Note: If you are creating a new data area, you will need to create the database driver configuration file as well.

2. Build the data dictionary using **blddbdict**.
3. Create and verify a list of database changes using the **`dbreorg -lc`** options.
4. Reorganize the database to implement the changes.

   The **dbreorg** process affects tables for which the structure has changed.
5. Verify successful reorganization.
6. If you have added or removed fields from the database, modify the associated application programs and recompile the product line. These steps are documented in the *Application Developer Workbench* guide.

Finding and Fixing Dictionary Issues

For Lawson programs to work properly with Lawson tables stored in the third-party database, the Lawson dictionary descriptions must match the expected translations in the database. Mismatches in these definitions can lead to performance problems or even program failure (column definition mismatches). Use the Verify Dictionary utility when you suspect dictionary mismatches such as missing

objects or mismatched columns. Use the Build Data Definition Language utility to correct these issues or others, when you do not want to use the reorganization process.

Use the Verify Dictionary (**verifyibm**) utility to check the two definitions. If problems are identified, use the Build Data Definition Language utility (**bldibmddl**) to modify an object (table or index) that needs rebuilding or modification. These utilities are used from a Lawson command prompt.

Copying, Importing, and Exporting Lawson Data

The following table lists several data transfer scenarios, along with suggestions on the utilities to use to accomplish the task. Use these utilities when you need to move, export, or copy some portion of the Lawson database.

Use	To copy
dbdump -d and **dbload**	All the data from a specific data area or data ID of a product line into the same data area or data ID in a different product line.
expsysdb and **impexp**	Data from all data areas of a product line (or all files of a given application) to the same data areas in a different product line, including any related attachment files.
dbcopy	To copy data from one data area to another.
An application import/export program or custom program and **impexp**	Data from only a few or several selected data area files, with data meeting a desired set of selection criteria.
CSV file (created by either **rngdbdump** or by some other method) and **importdb**	CSV-formatted data area files to a Lawson database file. CSV files are also called comma-delimited files.
lashow from the displayed report; choose the CSV File option from the **Send (F8)** menu.	Lawson report data to a spreadsheet.
Paint, generate, compile, and run the desired report using Environment 4GL, and display the report. From **lashow**, choose the CSV File option from the **Send (F8)** menu.	Transfer custom report data to a spreadsheet.
gendbload	All the data from a specific data area or data ID of a product line into the same data area or data ID in a different product line, overriding the product line.

These commands, the parameters they accept, and other considerations are explained in this guide, the "Product Line Maintenance" chapter in the *Lawson Administration: Server Setup and Maintenance* guide, or the *System Utilities Reference Guide*.

Physical Data Storage Management

A Lawson database space is the storage area for your database files. After you define one or more database spaces using the Lawson **dbdef** utility, assign them to the product line, data area, system code, or table. Before using the database spaces, do the following:

- Estimate disk usage.
- Create database spaces.
- Assign portions of the database to a database space.
- Implement the dictionary changes.

You can assign database spaces at any time. Doing so, however, requires making it possible to stop the database prior to reorganizing the database. When reorganizing the database, do the following:

1. Determine where to store data and how much space you need.

   Note: A helpful tool in estimating the table and index space used by a data area is the Disk Usage utility .

2. Create the database space(s) you want with **dbdef**.
3. Assign product lines, system codes, or files to the database space(s) you created.
4. Build the data dictionary using **blddbdict**.
5. Reorganize the database using **dbreorg**, which implements the changes.

Database Installation and Configuration

2

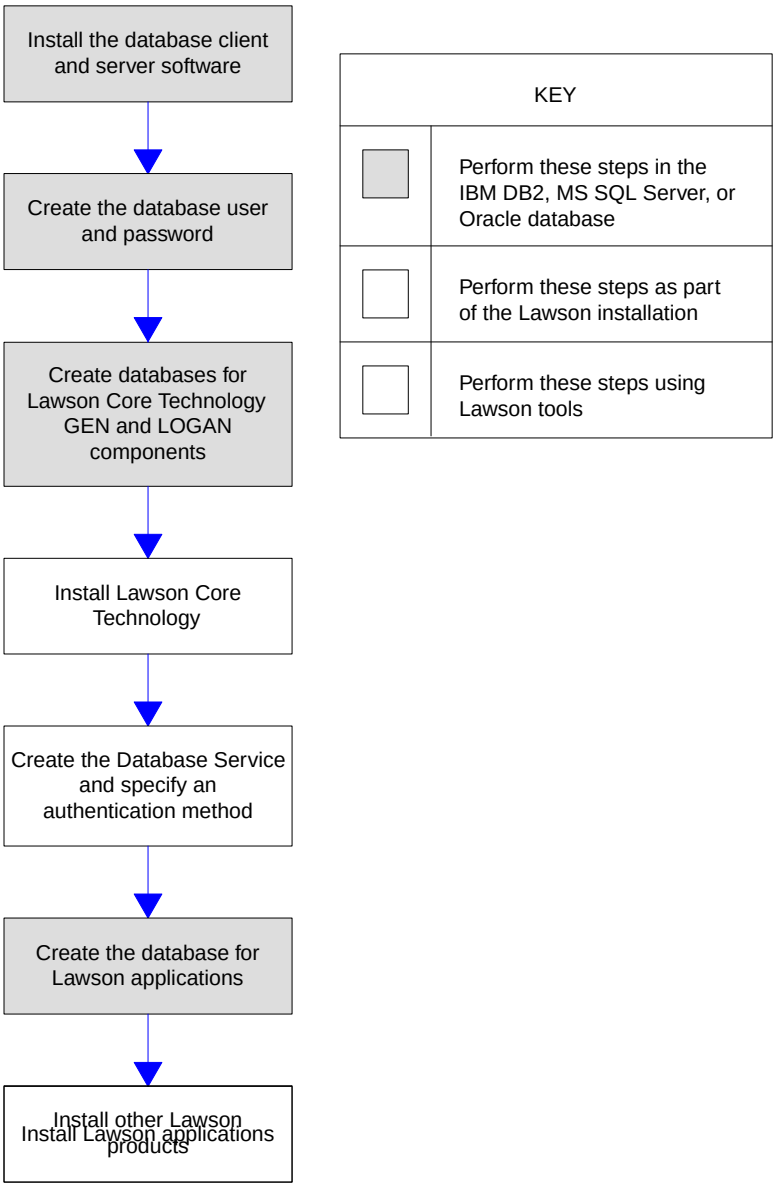
This chapter contains the following topics:

Database Installation and Configuration Overview

The tasks described in this chapter are required before installing Lawson products, and in some cases, before installing the IBM DB2 database. The following flow chart illustrates the installation and setup flow.

Warning: Lawson cannot assume any responsibility for the results (damage of data and/or structure) of improperly using non-Lawson tools. Please refer to the third-party tools documentation when attempting to access Lawson data via any non-Lawson tool.

Figure 2. Process overview: Installing and configuring the IBM DB2 database for Lawson.



Lawson Database Interface Architecture

This section contains the following topics:

Lawson System Tiers

The open architecture of the Lawson system is achieved through a multi-tiered software model. Lawson customers can use a variety of user interface options, hardware and operating system environments, and relational database management systems.

Lawson supplies the interface components that run the system and allow the tiers to communicate. Interface components include the set of executable programs libraries, configuration files, data, and dictionaries used to support the other Lawson tiers and interface to third-party products.

The Presentation Tier

The presentation tier is a Lawson user interface, called S3 for Workspace, which users view at the desktop through a standard web browser. Lawson also provides a character-based user interface called, Lawson Desktop (LID), which is used by developers and system administrators to manage the Lawson system.

The Lawson Applications Tier

Lawson Applications provide business functionality within the Lawson system. They provide processes for specific business tasks users need to perform with their business data. This tier handles the actual data processing.

The Lawson Core Technology Tier

The Lawson Core Technology includes a repository for system data. It also includes administrative programs or utilities that isolate Lawson applications and extensions from the operating systems, databases, and third-party software they are used with. The product is designed to require minimal maintenance. Lawson Core Technology products and utilities include the following.

- Lawson Security

  Lawson Security is a role-based, rules-driven security system that is administered through the Lawson Security Administrator graphical user interface. Because it is based on the roles people in your organization have, Lawson Security reflects the way your organization runs its business. Its use of a rules builder enables you to develop a highly flexible and complex, yet relatively easy to maintain security system.

- Resource Management

  The Lawson Resource Management application is a centralized information repository. It is a place to consolidate directory-type information about people, organizations, and things, and share that

information securely across the enterprise. The Resource Management application interfaces with Lawson workflow systems, security systems, web servers, and other software. It also makes your corporate resource data available to a wide variety of users and systems.

- Core Administrative Services

  Lawson provides set of system utilities that allow you to manage the Environment. With these utilities, you can start and stop the system or its components and manage system processes initiated by Lawson.

- Lightweight Data Access Services (LDAS)

  The database services interface provides communication between the relational database management system (RDBMS) and the other tiers. The interface is comprised of a database driver and database administration utilities (collectively known as LDAS).

  The Lawson Database Server (**ladb**):

  - receives, analyzes, and manages data requests,
  - starts the RDBMS-specific Database Driver (ibmdb), and
  - translates Lawson 4GL APIs into native RDBMS SQL commands.

- The GEN and LOGAN Databases

  Lawson architecture stores operational *metadata* (information about your Lawson system) used to operate and maintain the system in the GEN and LOGAN databases.

- Additional Administrative and Program Modification Support

  The Environment includes other support such as Internet Object Services, Lawson batch job administration tools, program modification support, and a broader range of administrative tools and utilities for managing your Lawson system.

Where Do Lawson Files Reside?

Files used by Lawson programs are stored in several directories. The directory names are *environment variables* that define the paths for the directories. The required directories are designated at installation time. The installation program also creates any required subdirectories in each directory. Other directories may be set as environmental variables for your convenience. The path names attached to these variables must be unique, although different environments can share the same **%GENDIR%** directory.

Important: These files, directories and environment variables are stored in the application server and not in the database server.

What Is the Lawson Environment Variable Naming Standard?

An environment variable name is shown as **%VARIABLE%**.

For example, environment variables and subdirectories are shown as:

%GENDIR%/java/jar

%LAWDIR%/*dataarea*

GENDIR

%GENDIR% is the directory path in which utilities and other files are stored. Original copies of configuration files are also stored here.

Subdirectories

%GENDIR%/bin	Executables
%GENDIR%/system	Server configuration file templates
%GENDIR%/install	Install scripts, logs, and output files created at install time
%GENDIR%/bpm	Files needed for Process Server

Default path

/gen

LAWDIR

%LAWDIR% is the directory path where the product lines and server configuration files are located.

Subdirectories

%LAWDIR%/system	Server configuration and log files
%LAWDIR%/*dataarea*	Product line information
%LAWDIR%/*dataarea*	Data area information
%LAWDIR%/bpm	Process Server information

Default path

/apps

LADBDIR

%LADBDIR% is the directory path where the dictionary files are located (data is stored in the RDBMS).

Default path

/DB

LAUNTDIR

%LAUNTDIR% is the directory where user home directories and session log files are located.

A user's home directory is created the first time the user logs in. The name of a user's home directory is based on the user's name and domain. For example, if a user's domain is DOMAINNAME and the user's name is USERNAME, the user's home directory is named *DOMAINNAME_USER NAME*

Default path

/launt

WEBDIR (optional)

The **%WEBDIR%** directory is an optional variable you can set for the base directory on the web server in which certain presentation layer directories and files reside. Lawson documentation sometimes uses this term to refer to the web server document root directory, as shown in the following table.

Subdirectories

%WEBDIR%/lawson/*MyPortal*	Web server components for S3 for Workspace. The location of *MyPortal* is defined at installation time.
%WEBDIR%/lawson/*MyStudio*	Web server components for the Lawson Design Studio. The location of *MyStudio* is defined at installation time.

Default path

DocumentRootDirectory

The Database Services Interface

The database services interface provides communication between the RDBMS and the other two tiers. It consists of the following components:

- The Database Server (**ladb**)
- The database APIs
- The database drivers

The Database Server (ladb)

The Database Server (**ladb**) is the main server engine. It analyzes and manages user requests, determines which database to access, and spawns database drivers. The **ladb** also notifies other servers of terminated database users.

Database (4GL) APIs

Presentation and application programs use Application Program Interfaces (APIs) provided by the Lawson system to communicate with the database. These APIs perform a set of functions such as finding a single record, summing each of three columns in a table, or deleting a set of records.

The Database Driver

The Database Server (**ladb**) communicates with the relational database management system (RDBMS) through database drivers. The driver translates Lawson 4GL Database APIs into native RDBMS SQL commands. The database driver is specific to the RDBMS you use. The following database driver is associated with the IBM DB2 server:

Database Server	Database Driver
IBM DB2	ibmdb

The Database Utilities

Several utilities are connected with the database driver to provide additional data access and management services. They are:

- **verifyibm**

  Compares Lawson dictionary descriptions to the expected translations in IBM DB2 to ensure the IBM DB2 database definitions match the Lawson dictionary.

- **bldibmddl**

  Generates SQL statements for creating or deleting selected tables, indexes, and stored procedures. The utility can operate on specific tables, all tables in a system code, or all tables in a data area.

- **bldibmsec**

  Duplicates Lawson table level security in the IBM DB2 database.

- **ibmdu**

  Calculates the space needed for an initial number of records in the database, using information found in the Lawson dictionary and both **`law_dba_table`** and **`law_dba_index`**.

What Is the Process Chain?

The process chain is a description of how information is processed from the time you request data through a Lawson application until the point when the information is returned to you.

The chain begins when you start a Lawson application. The program sends a message to **ladb**, which then determines the kind of database for each of the tables used by that program. If **ladb** encounters an IBM DB2 table, **ladb** starts a process to manage the information requests from that program.

When you ask the application program to process information, the program sends a message, known as a database call, to the database driver. The database call consists of some or all of the following:

- Identification and security information.
- The database command (**FIND**, **STORE**, **DELETE**, and so on).
- The product line and table to use.
- Key columns to find the data.
- Data to insert or update.
- Needed columns.
- The preferred index.

The database driver builds a SQL statement to send the database call to the IBM DB2 database server. This server processes the call by checking the appropriate database tables and then sending your data (or an error message if something goes wrong) back through the database driver to your application program.

The Role of the Environment in Data Management

The Environment plays an important role in holding and managing data.

Active Data Repository

The Lawson data model includes an active data repository that makes system-wide information available to all tiers in the Environment during runtime.

The repository includes metadata on:

- Presentation definitions
- Information relationships
- Process definitions
- Logic definitions
- Security definitions
- Application server definitions
- Open application interfaces

Files in the repository include the product line dictionary file and the data area dictionary file for all product lines in the Environment as well as the GEN dictionary. The compiled programs holding the form and object rules are also in the repository. The business logic in the active repository ensures data integrity before the database is updated by Lawson application programs.

The GEN Product Line

The Environment has its own product line called **GEN**. It holds the data needed for administration, such as user profiles, product line names, and batch job definitions.

The GEN product line includes the following:

- Administrative forms, which enable administrators to perform tasks such as job scheduling, security setup, and report printing.
- User forms, which are created by customers to perform specific tasks, such as enabling users to change their own passwords.
- GEN data and data dictionary, which the system uses to manage features such as user profiles, product line names, and batch job definitions.

Creating the Database

This section contains the following topics:

Database Installation and Configuration Requirements

This document provides the guidelines and requirements for configuring the database that will be used for the system.

Caution: Lawson does not provide instructions for installing IBM DB2 products; installing the database is the responsibility of the customer. Lawson cannot assume any responsibility for the results of an improper installation. Use the documentation provided with the IBM DB2 database for installation instructions.

Required Database Components and Settings

Relational Database Versions

Supported versions for your relational database are documented in the installation guide for your S3 Lawson System Foundation product and platform.

Operating System for the Database Client

The database interface layer utilizes third-party database client software to access data from the relational database server. The database client and server may run on different operating systems, utilizing connection software provided by the third-party database vendor. Lawson supports any configuration that the third-party relational database vendor certifies and supports between their database client and server. However, to be supported, the client software must be running on an operating system version that supports Lawson software products.

Additional Required Database Products

When installing IBM DB2 products, you are given a list of products and features from which to choose. While the Lawson run-time system does not require that you install additional products, the following are needed for installation, set up, or performance analysis of the Lawson database:

- IBM DB2 Command Line Processor (CLP)

  This utility dynamically executes SQL requests and/or IBM DB2 commands between local and remote databases and can create the database and table spaces.

  Note: There are two methods of using the CLP. One method is prefixing all CLP commands with **db2**. The second method is using the CLP in interactive mode by typing **db2** and pressing **Enter**. In the interactive mode, type CLP commands without prefixing them with **db2**. However, the interactive mode does not allow you to do piping or other operating system functions at the CLP command, nor does it allow you to recall commands.

- IBM DB2 Control Center

Data at Rest Encryption Support

Lawson supports data at rest encryption for a variety of database and platform versions. For details, see the installation guide and release notes for your S3 Lawson System Foundation product and platform.

The IBM Database Encryption Expert is required to implement Data at Rest Encryption of tables. You can find out more about this component at http://www-306.ibm.com/software/data/db2imstools/database-encryption-expert/. You must create the encrypted table spaces before you install Lawson products.

Before you start The IBM Database Encryption Expert server, version 1.1.1, requires a dedicated server (32-bit Red Hat Linux, Update 4). The IBM Database Encryption Expert database agent is platform independent, but must be able to communicate with the IBM Database Encryption Expert server.

For more information, see "Encryption of Data at Rest" on page 39.

Required Database Settings

- User and Group Naming Conventions

  IBM DB2 requires a maximum of eight characters in a database, user, or group name. The first character of user IDs must be A-Z, #(X'7B'), $ (X'5B'), or @ (X'7C').

  For more information on user naming conventions, see the IBM DB2 *Security Administrator's Guide*.

- The following settings are required for required for Lawson internet applications data processing:

Code page	1252
Code set	IBM-1252
Sort order	Binary (identity)

- Lawson does not support the UTF-8 character set.

Database Preinstallation Tasks

Your databases are configured by the installation programs, based on answers you gave during installation. Before you install the IBM DB2 database, S3 Lawson System Foundation, or Lawson applications, perform the following tasks and collect the information shown in this section.

Perform this task	Collect this information
Add environment variables and required values to the %PATH% variable. The choices you see in the installation program depend on what the installation program detects in the **%PATH%** variable on the Lawson server machine. **1** Define an environment variable for the **%DB2HOME%** directory. **%DB2HOME%** must equal *DB2INSTANCE***/sqllib** (the **sqllib** subdirectory in the home directory of the DB2 instance owner). **2** Verify that the database location is in the **%PATH%** variable. **Note:** If you plan to install GEN, LOGAN, and/or Lawson applications in different databases, all the directories must be in your path. **3** Verify that the directory containing the command to start the database client appears in your **%PATH%**. For information and instructions, see the documentation provided with the database and/or the operating system.	The installation location of the GEN, LOGAN, and application databases. The **%DB2HOME%** directory path for each instance.
Determine the initial number of simultaneous database connections The installation program asks for the number of simultaneous database connections that you allow. A database connection is one driver job, or one open data area for each running program. The default value is 500; you may need to increase this value significantly. Use this value to calculate the starting values of **`USERS`**, **`FOREIGN`**, and **`IFILES`** parameters in the Lawson ladb.cfg file. Individually tune these values as required after installation.	The number of simultaneous database connections.
Determine the local alias for the DB2 database The DB2 client must be installed and configured to connect to the DB2 server. This value is the alias name as cataloged on the local system. – or – **Determine the name of the database on the DB2 database server**	The local alias – or – The name for the DB2 database

Perform this task	Collect this information
Determine database user name (ID) and password • The login name (user name) and password for the database user who accesses Lawson data at run-time. These values are mapped to the **LOGINNAME** and **PASSWORD** parameters in the IBM database driver configuration file.	The login name and password for the database user.
Determine the schema to use when creating tables Use the schema name to keep the data for this Lawson data set separate from any other data stored in the database. All tables are created with this name as the table owner. Use a unique schema name for each Lawson data area. Lawson recommends that you use the format *EnvironmentName_DataAreaName*; for example, **810TEST_GEN** and **810TEST_LOGAN**. The schema name cannot exceed 18 characters in length.	The schema (owner) for tables.
Determine how to configure and name IBM DB2 table spaces Table spaces for tables and indexes must be defined in IBM DB2.	The data and index table space names for each product line.

Planning for Your DB2 Database

Cataloging your DB2 database is done differently depending on your system setup. If your Lawson applications and your DB2 database are on the same system, the **CREATE DATABASE** command does the following:

- Creates a system or local database directory if neither exists
- Catalogs an entry in the server's system database directory. The resulting entry will contain the database name, alias, and path.
- Catalogs the database in the server's local or volume database directory on the path indicated by the PATH parameter or in the default database path defined in the **dbm** configuration file.
- Updates the client's system database directory with the database name and alias if a remote client issued the command.

The node directory is not needed on the server in this case, as everything is on the same machine.

If your Lawson applications and your DB2 database are on different systems, the applications server becomes a client to the database server. The **CREATE DATABASE** command on the server creates the necessary System and Volume Database Directory entries in the database server. In order for the client to communicate with the server, the following entries must be made on the client:

- System Database Directory (via the catalog database command)
- Node Directory (via the catalog **tcpip** node command)
- Optional entries in the **/etc/services** file

For more information, see "Cataloging the DB2 Database" on page 28..

User Naming Conventions

The following table explains the user or group naming conventions required by IBM DB2. Apply the correct user naming conventions appropriate for your platform.

Platform	Description
Windows	IBM DB2 requires a maximum of 8 characters in a user or group name. The first character of user IDs must be A-Z, #(X'7B'), $ (X'5B'), or @ (X'7C').

For more information on user naming conventions, see the IBMDB2 *Security Administrator's Guide*.

Limits

A complete listing of IBMDB2 limits for is available in the *DB2 SQL Reference*.

The complete listing of IBMDB2 limits for OS/390 is available in the *DB2 for OS/390 SQL Reference*.

Creating the DB2 Database

This section assists you in creating the database.

You can use either the DB2 CLP or the DB2 Control Center to create the database in DB2. Steps for both methods are provided in this topic.

The **CREATE DATABASE** command creates the database. By default, this command creates three Automatic Storage table spaces named **SYSCATSPACE**, **TEMPSPACE1**, and **USERSPACE1**. The command also creates necessary catalog entries.

When you create the database, use these settings:

Code page	1252
Code set	IBM-1252
Sort order	Binary (identity)

To create the database using the CLP

1. Log in with **SYSADM** or **SYSCTRL** authority to create the database.
2. At the command prompt, type **db2** and the **CREATE DATABASE** command. For example, type:

   db2 CREATE DATABASE *database_name* **AUTOMATIC STORAGE YES ON** *path* **USING CODESET IBM-1252 TERRITORY US COLLATE USING IDENTITY PAGESIZE 4096**

   Important: The **CREATE DATABASE** command must be issued using the **IDENTITY** collating sequence, where strings are compared byte by byte.

3. Press **Enter**.

To create the database using the Create Database Wizard

1. Start the Control Center.
2. Attach to the instance as a user with **SYSADM** or **SYSCTRL** authority.
3. To start the Create Database Wizard, right-click the Databases folder and choose Create Database -> Standard...
4. On the Region tab, select **Codeset IBM-1252** and a collating sequence of **Identity**.
5. On the summary page, verify that the codeset and collating sequence are correct by clicking Show Command.
6. Click Finish.

Cataloging the DB2 Database

If your database server and applications are on different systems, use this procedure to catalog your DB2 database.

To catalog your DB2 database

1. Ensure that you know the name of the database you plan to catalog, and the node name and port number of the database server.
2. Using the **db2** utility or Command Line Processor, catalog the node by issuing the following command:

   db2 CATALOG TCPIP NODE *NodeName REMOTE DatabaseServerName* **SERVER {***Port#* **| /etc/services** *Name***} REMOTE_INSTANCE** *Instance_Name*

3. Catalog the database by issuing the following command:

   db2 CATALOG DATABASE *DatabaseName* **AS ALIAS AT NODE** *NodeName*

4. Update the **/etc/services** file.

To verify that your DB2 database is cataloged

- At the command prompt, type:

  db2 list database directory

  If cataloging is successful, an entry for your database appears in the contents of the database directory.

Creating Table Spaces

The following instructions assist you in creating table spaces:

To create table spaces

1 Log into the IBM DB2 database as an administrator.

Note: Using bufferpools is optional depending on your site, but highly recommended by Lawson.

2

Tip: Lawson recommends that you specify a pagesize of 8K or higher for the BUFFERPOOL.

When you create a system temporary tablespace, Lawson recommends specifying a BUFFERPOOL of 32K or higher.

Use the **CREATE BUFFERPOOL** command to create bufferpools by running a script similar to the following:

```
CREATE BUFFERPOOL DATPOOL1 SIZE -1;
(uses BUFFPAGES size in db cfg)
CREATE BUFFERPOOL IDXPOOL SIZE 40000;
```

3 Use the **CREATE TABLESPACE** command to create a table space by running a script similar to the following:

```
CREATE REGULAR TABLESPACE DATTS PAGESIZE 8 K
 MANAGED BY DATABASE USING
 ( FILE '/sqldata/db2v6/db2tc/instdb_datts1.dbf' 2560)
 EXTENTSIZE 16 OVERHEAD 18.6 PREFETCHSIZE 16
 TRANSFERRATE 1.0;

CREATE REGULAR TABLESPACE IDXTS PAGESIZE 4 K MANAGED
 BY DATABASE USING
( FILE '/sqldata/db2v6/db2tc/inst1db_idxts1.dbf' 2560)
 EXTENTSIZE 16 OVERHEAD 18.6 PREFETCHSIZE 16
 TRANSFERRATE 1.0;
```

4 Use the **ALTER TABLESPACE** command:

```
ALTER TABLESPACE datts bufferpool datpool1
ALTER TABLESPACE idxts bufferpool idxpool
```

IBM DB2 Configuration Parameters

The following table lists key IBM DB2 configuration parameters, the recommended values at which to set them, and where they are maintained.

Parameter	Description	Recommended value	Maintained using
APPLHEAPSZ	**Tip:** Increase the value of this parameter if your applications receive an error indicating that there is not enough storage in the application heap. The Application Heap Size parameter defines the number of private memory pages available for use by the database manager on behalf of a specific agent or subagent. The heap is allocated when an agent or subagent is initialized for an application. The amount allocated is the minimum needed to process the request. As the agent or subagent requires more heap space to process larger SQL statements, the database manager needs to consider allocating additional memory up to the maximum specified by this parameter.	512	**`$ db2 update db cfg for`** *`DatabaseName`* **`using APPLHEAPSZ 512`**

Parameter	Description	Recommended value	Maintained using
LOCKLIST	The **LOCKLIST** parameter indicates the amount of storage that is allocated to the lock list. There is one lock list per database and it contains the locks held by all applications concurrently connected to the database. Locking is the mechanism that the database manager uses to control concurrent access to data in the database by multiple applications. Both rows and tables can be locked.	1500	**$ db2 update db cfg for** *DatabaseName* **using LOCKLIST 1500**
DBHEAP	There is one database heap per database, and the database manager uses it on behalf of all applications connected to the database. It contains control block information for tables, indexes, table spaces, and buffer pools, as well as for the event monitor buffers, log buffer, and catalog cache. Increase the value of this parameter if your applications receive an error indicating that there is not enough storage available in the database heap to process a statement.	2000	**$ db2 update db cfg for** *DatabaseName* **using DBHEAP 2000**

Parameter	Description	Recommended value	Maintained using
DB2_RR_TO_RS	When the DB2_RR_TO_RS parameter is set to YES, the RR isolation level is downgraded to RS for user tables. RR semantics are no longer provided in the database manager instance. If your application does not require RR semantics, this registry variable can be used to reduce the next-key lock contention problems that can sometimes occur under RR.	YES	db2set command
Tablespaces	This parameter allows you to select whether to have the database automatically handle configuration settings or if these will be manually administered.	Managed by Database	**CREATE DATABASE***DatabaseName*

Configuring Lawson System Foundation for the Database Interface

This section contains the following topics:

What Is the Lawson Database Driver Configuration File?

Every product line and data area must have an associated IBM database driver configuration file, stored in the following location:

%LAWDIR%/*dataarea*/IBM

This file contains information that specify how the Lawson database drivers interact with the IBM database. When a database driver starts, it reads the database driver configuration file. The parameters in the file are exported into the environment of each database driver.

A database driver configuration file is created for a product line by the Lawson installation program based on the responses entered for the installation questions. Data areas are not created by the installation program, so you must create a new IBM file for each new data area you create. For more information, see "How Do I Create a New Data Area?" on page 57.

Changes in the Lawson IBM database driver configuration file can take effect without stopping or starting the database. You need to idle the drivers using the **dbadmin** utility, however. For more information, see "Editing the Lawson Database Driver Configuration File" on page 37.

Each new connection to the database requires an ibmdb process. Using **latm**, however, can cause the same ibmdb process to be used if you use an application repeatedly within a specified amount of time.

Note: The database driver configuration file is in the application server and not in the database server.

The following shows a sample Lawson IBM database driver configuration file.

```
LAWGATENAME=ibmdb.exe
DBNAME=law3gl
DB2INSTANCE=db2v7
LOGINNAME=lawson
PASSWORD=lawsonpw
```

Setting Batch or Online Connect Parameters

There are two ways to create a Lawson IBM Configuration file. Specify one set of connect parameters for the entire system or specify two sets of parameters, one for batch programs and one for online programs.

Example 1: Single connection parameters for online and batch programs (AIX client shown)

```
LAWGATENAME=ibmdb.exe
DBNAME=law3gl
LOGINNAME=lawson
PASSWORD=lawsonpw
DB2INSTANCE=db2v7
```

Example 2: Separate connection parameters for online and batch programs (AIX client connecting to an IBM DB2 O/S 390 server shown)

```
LAWGATENAME=ibmdb.exe
DB390NAME=law3gl
LOGINNAME=lawson
PASSWORD=lawsonpw

DB2INSTANCE=db2v7
```

```
DBNAME=lawgl
LIBPATH=/opt/db2v7/sqllib/lib:/usr/lib:/lib

[BATCH]
DB2INSTANCE=db2v7b
DBNAME=lawglb
LIBPATH=/opt/db2v7b/sqllib/lib:/usr/lib:/lib
```

What Are the Variables in the Database Driver Configuration File?

The following topics describe the variables in the IBM database driver configuration file:

- "Parameters that Specify Your Database Implementation" on page 34
- "Parameters that Configure Record Caching" on page 36

Parameters that Specify Your Database Implementation

The parameters in the following table configure the database.

Parameter	Description
LAWGATENAME	ibmdb.exe The name of the database driver executable, used between the Lawson applications and the IBM DB2 database.
DBNAME	Required. The database name. Enter the name as catalogued on the local system of the IBM DB2 database used to store Lawson tables and indexes for the data area. If the database resides on the OS/390, you might also need to specify the **DB390NAME**. The **DBNAME** variable in IBM DB2 uses only 8 characters.
DB2INSTANCE	Required. The name of the IBM DB2 instance to be used for a specific data area. The suggested entry is **DBINSTANCE=db2**. This parameter defaults to **DBINSTANCE=db2**.

Parameter	Description
LOGINNAME	Required. Enter this parameter when using the **USE_CFG_FILE** login procedure . Do not use the **LOGINNAME** parameter for any other login procedure. This is the user name used when the database driver connects to the IBM DB2 database. The default is **lawson**.
PASSWORD	Required. Enter this parameter when using the **USE_CFG_FILE** login procedure . Do not use the **PASSWORD** parameter for any other login procedure. This is the password for **LOGINNAME**, which the database driver uses when it connects to IBM DB2. The default is **lawson**.
SERVICENAME	This parameter is required when Lawson Security and Resource Management are implemented to manage users and secure data. This parameter specifies the name of the database service for this data area. If you do not enter a database service, the login procedure defaults to **USE_CFG_FILE**. You can either specify a **LOGINNAME** and **PASSWORD** in the IBM file, or configure the system so that each individual user accesses the database with a unique user name.

Parameter	Description
SCHEMA	Required. This parameter is the database owner. If the tables were not created by the user named in **LOGINNAME**, set **SCHEMA** to the name of the user who created the tables. If **SCHEMA** is not specified in the IBM file, the database driver (ibmdb) uses the following methods to determine a value for the table owner: • If defined in the IBM file, the **LOGINNAME** value from the IBM database driver configuration file defaults for the **SCHEMA** value. • If the **LOGINNAME** parameter is not defined, the database login name specified for the database service defaults to the **SCHEMA** value when using the **USE_PRIVILEGED_ID** or **USE_USER_AND_PRIVILEGED_ID** database authentication options. • If your system uses the **USE_USER_ID** database authentication option for Lawson Security or your system does not implement Lawson Security, the **lawson** user defaults for the **SCHEMA** value. If **SCHEMA** is not specified in the IBM file, the database utilities (**bldibmddl, bldibmsec, verifyibm**, or **ibmdu**) use the following methods to determine a value for the table owner: • If defined in the IBM file, the **LOGINNAME** value from the IBM database driver configuration file defaults for the **SCHEMA** value. • If the **LOGINNAME** and **SCHEMA** parameters are not defined, the database login name must be specified on the command line when running one of these utilities.
DB390NAME	Optional. If the name of the default database in **DB2/390** is not the same as **DBNAME**, set **DB390NAME** to the name of the default database.
DBPATH	The directory where the IBM DB2 database executable runs.

Parameters that Configure Record Caching

Use the following parameters to enable record caching.

Parameter	Description
ARRAYBUFSIZE	Optional. Indicates the maximum number of rows to be fetched at a time from the database server. The default is 10 rows. The setting for this variable in the database driver configuration file applies to all files in the data area or data ID unless a variable specific to a program has been created. When a variable specific to a program has been created, it overrides the setting in the database driver configuration file.
INSERTBUFSIZE	Optional. Indicates the maximum number of rows to be buffered in the database driver before an array insert flushes data out to the database. This parameter must be set to a value greater than 1 to enable array inserts. Lawson recommends a setting of 10 for normal use. However, when you perform a **dbcopy** or **dbreorg**, particularly when a large amount of data has been inserted into the database, increase the size of this parameter. Lawson recommends increasing it to 50 in these cases. Return the parameter to your normal setting when the **dbcopy** or **dbreorg** completes. For more information, see "Editing the Lawson Database Driver Configuration File" on page 37. As with **ARRAYBUFSIZE**, you can set this variable to control a specific program. The program-specific setting overrides the setting in the database driver configuration file.

Editing the Lawson Database Driver Configuration File

This procedure describes how to edit the configuration parameters in the Lawson IBM configuration file.

To edit the Lawson Database Driver Configuration File

1 Use the text editor to open the Lawson IBM configuration file for your product line or data area. The file location is:

%LAWDIR%/*productline*|*dataarea***/IBM**

Note: Lawson recommends using a text editor other than Notepad, as it may place unprintable characters in the IBM configuration file.

2 Change the desired parameters. For more information, see "What Are the Variables in the Database Driver Configuration File?" on page 34.

3 Save and exit the file.

4 Clear the active database drivers so that the changes become effective. Type:

```
dbadmin idleALL
```

Securing the Lawson Database Driver Configuration File

If your security and data access strategy includes determination of the database user and password in the database driver configuration file, you should restrict knowledge of the IBM DB2 Lawson user name and password to the database administrator and anyone else in your organization who must perform database administration tasks. Use this procedure to secure the IBM database driver configuration file.

To secure the Lawson IBM Database Driver Configuration File

1 In Windows Explorer, right-click the IBM configuration file, found in this location:

%LAWDIR%/*productline***|***dataarea***/IBM**

Note: The Lawson database driver configuration file in each product line must be secured.

2 Select **Properties > Security**.

3 Use the check boxes to specify Full Control access to the file for the users System and Administrators. This prevents any other users from viewing the contents of this file.

4 Click OK to save the changes.

Configuring Data At Rest Encryption

This section contains the following topics:

Encryption of Data at Rest

Encryption of data at rest provides data security by allowing you to encrypt sensitive data such as social security numbers and credit card numbers, which are saved in a database. This feature ensures that data is kept safe if the storage medium is stolen. It also helps in addressing security-related regulatory compliance issues.

Before applying encryption of data at rest in Lawson, you must configure your security using the IBM Database Encryption Expert.

Lawson database configuration is determined by how you configure encryption in IBM DB2. If you encrypt all tables in the IBM DB2 instance, no additional Lawson configuration is needed. If you encrypt a single table space in IBM DB2, then all Lawson files that you wish to encrypt must be assigned to that table space. You must define a tablespace as encrypted when it is created. You cannot make an existing tablespace encrypted using the ALTER TABLESPACE statement.

Note: You can assign the database space to a product line or data area, a system code, or to individual files. The database space assignment for a file overrides the definition for the system code, and the data area definition overrides the system code definition.

Configuring Database Users and Authentication

This section contains the following topics:

Database User Authentication Overview

This section provides a conceptual overview of how database authentication works and the parameters available for configuring it.

Database Service

A *service* is a Lawson component that facilitates single sign-on. Depending on your needs and how you choose to configure your system, you or your Lawson installer might need to create a service for your Lawson application database. This would be the case if you want to make use of some specialized authentication options that are available.

However, many customers might choose to have authentication options contained in their database driver configuration file and to secure that file through operating system file permissions. This is how database authentication was configured in prior release of Lawson and might still be satisfactory for your installation.

If you want passwords to be encrypted and to make use of Lawson support for the specialized authentication options that might be available through your RDBMS, create a database service and user identities for the service. Read the sections that follow for descriptions of the available database authentication options.

Database User Authentication Options

The method that authenticates database users is the login procedure. Several options are available and choosing the right one for your installation is an important decision. The available login procedures are described in more detail below. When you run the **ssoconfig** utility to create a database service, specify the login procedure.

USE_CFG_FILE

This database authentication method has the following characteristics:

- It allows a single privileged user to log into the database.
- The privileged user name/password (for example, **dblawson/dblawson**) is stored in clear text in the database driver configuration file (also known as the capital or CAP file) as the login name and password.
- It is the default login procedure, because it enables you to be up and running with the least amount of configuration.

The **USE_CFG_FILE** login procedure is the default because it does not require any initial setup. A database service is not a requirement for this configuration.

Lawson installers might use this method of authentication, for example, to perform smoke tests before installation is complete. Customers might also choose to run in this configuration if they have no need for database auditing or password encryption. If you choose to use this method, make sure you secure the database driver configuration file through file permissions.

If you use this method, no SSO configuration is required. All additional information in this document related to setting up a database service and creating identities does not apply to you. Refer to the Lawson documentation for the RDBMS system that you use for instructions about how to configure the database driver configuration file for a privileged user name and password.

USE_PRIVILEGED_ID

This database authentication method is the same as **USE_CFG_FILE** except that the privileged user and password are stored in Resource Management (LDAP repository) and the password is encrypted. To implement this method, create a database service.

USE_USER_ID

This database authentication method has the following characteristics:

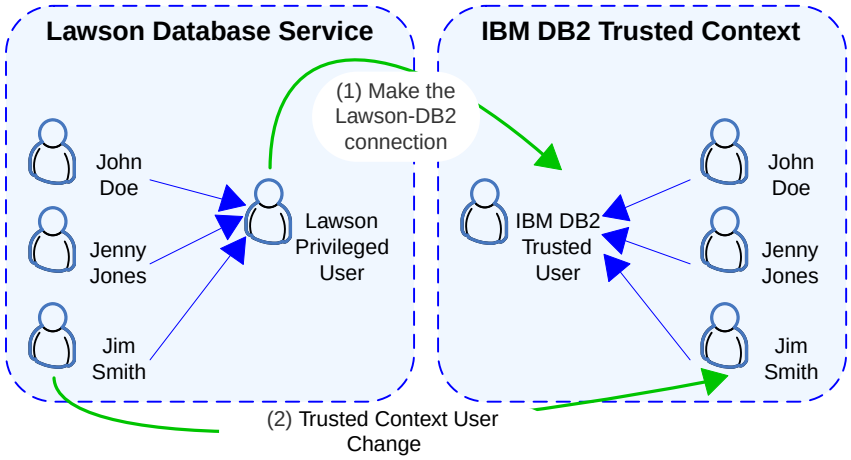
- Users log into the database using their specified database login names and passwords.
- The user names and passwords are stored in Resource Management (LDAP repository). Passwords are encrypted.

- Each user must have an “identity” on the database service. (Typically, you create this identity when you add the user to the system.)
- To implement this authentication method, create a database service.

USE_USER_AND_PRIVILEGED_ID

This database authentication method supports the IBM DB2 v9.5 or higher Trusted Context connections feature, to attach to the database as a single (privileged) user and then switch to another user on the connection. If you need more information about how this feature works, consult your IBM DB2 documentation, *Create Trusted Context*, in the Security section of the DB2 Information Center.

Figure 3. Diagram: Lawson user and IBM DB2 trusted context configuration



- Each individual user must have an identity on the Lawson database service. Typically, you create this identity when you add the user to the Lawson system.
- Each user name must be defined as part of the IBM DB2 trusted context.
- Individual users log in to Lawson using their user names which are linked to the Lawson privileged user.
- The Lawson privileged user (known as the trusted user in IBM DB2) is used to establish the trusted context connection.

To implement this authentication method, create a Lawson database service and a trusted context in DB2.

The privileged and unique user IDs and passwords are stored in Resource Management (the LDAP directory). Passwords are encrypted.

Database Driver Configuration File

This file (sometimes referred to as the capital file or CAP file) contains configuration parameters for the Lawson database driver. These parameters include the name of the database service. The database service name is configured in the format:

SERVICENAME=*YOUR_DATABASE_SERVICE_NAME*

Creating or Changing a Database Service

This procedure describes two methods for creating or updating a database service. Before creating a service:

- Determine the method you want to use to create the service.
- Determine the login procedure you want to use.
- Know the password for the **ssoconfig** utility.
- If you are updating an existing service, know the name of the service you want to update.

To create or change a database service

1. From the command line (in the Environment on which you want to create the new service), type:

   `ssoconfig –c`

2. At the prompt, type your **ssoconfig** password. (You might have to wait a few moments for this prompt to appear.)
3. From the list of functions, type **5** to select **Manage Lawson Services**.
4. From the list of functions, select **1** if you are adding a new service or select **2** if you are updating an existing service.
5. You are prompted to type a name for the service. Do one of the following:
   - If you are creating a new service, type any name that you want to use. Update the database driver configuration file with this name. Be sure to jot it down for future reference.
   - If you are updating an existing service, type the name of the service exactly.
6. From the prompt for authentication type, type **4** for DB.
7. At the prompt “Choose the method to use for sign on at the DB tier", make a selection from the following list. This is the login procedure. Type the number that corresponds to the login procedure you have decided to use:
   - **`(1) USE_PRIVILEGED_ID`**
   - **`(2) USE_CFG_FILE`**
   - **`(3) USE_USER_ID`**
   - **`(4) USE_USER_AND_PRIVILEGED_ID`**

   For a new service, the message "The service is created." appears. For an update of an existing service, the message "The service is modified" appears.

Activating the Database Service: Next Steps

Now that the service has been created, some additional procedures are necessary to activate it. Some of these steps apply to all systems; some apply only to customers who are using database auditing.

These steps are:

- Update the database driver configuration file as appropriate depending on the login procedure you have chosen.

  For more information, see "Updating the database driver configuration file" on page 43.

- Link identities to the Single Sign-on system depending on the login procedure you have chosen.

  For more information, see "Linking user identities to the database service" on page 43.

Updating the database driver configuration file

The database driver configuration file must be updated to contain the name of the database service. Use the service name you configured when you created the service. The required syntax for the database driver configuration file is:

SERVICENAME=*YOUR_DB_SERVICE*

For more information about the database driver configuration file, see *Lawson Administration: Data Access* for the RDBMS system you use.

Linking user identities to the database service

If you use the **USE_USER_AND_PRIVILEGED_ID**, **USE_PRIVILEGED_ID**, or **USE_USER_ID** login procedures, link all users to the database service by creating an identity for each user on the service. The procedures for adding and updating users describe how to do this. Typically, you create identities for users on the database service (and all other services they need identities for) when you add them to the Lawson repository (LDAP directory).

Configuring the Database Server

3

This chapter contains the following concepts and procedures:

Configuring the Database Server

This section contains the following topics:

The Lawson Database Server Configuration File (ladb.cfg)

The Lawson Database Sever Configuration file (**ladb.cfg)** contains the parameters used by the Database Server (**ladb**). These parameters define the maximum number of processes, files, and servers; the maximum size of files; and other values for parameters used by **ladb**.

ladb.cfg Parameter Reference

Database administrators can tune databases as they see fit. If the following parameters do not work for your database, you can adjust them to fit your needs. For good performance, you must constantly monitor current performance, make adjustments, and measure the results of any changes.

Keep a record of all changes you make to the default table parameters. If you find it useful to change the table or system parameters, inform your Lawson support staff.

ladb.cfg Parameters

Location: **`%LAWDIR%/system/ladb.cfg`**

The following table contains variables that affect all database types.

Parameter	Lawson setting	Description
DICTS	15	Maximum number of open dictionaries.
FILES	1500	Maximum number of open tables. This value is set assuming all Lawson applications exist in this Environment. This setting pre-configures for a larger than anticipated load. If only a small set of applications exists, this value may be set to a smaller number.
UFILES	450	Maximum number of tables for each user. This parameter should be set to the same value as the **IFILES** parameter.
USERS	500	Maximum number of user processes. Adjust as needed to meet your needs. If you change this number, you must also reset the kernel parameters. **Note:** The **FOREIGN** parameter needs to be the same value as **USERS**.
REUSE	OFF	Specify how idle drivers may be used by **ladb** for new connections. Valid values for setting the REUSE flag are • ON (TRUE or 1 may also be used) Allow any available idle connection to be reused by **ladb** when a new file is opened in the same data area. **Note:** REUSE can only be set to ON (TRUE/1) when LAWSONUID is set to **lawson**. • OFF (FALSE or 0 may also be used). Allow drivers to be reused only by the same user. Drivers are stopped when the user's connection to **ladb** is dropped. The REUSE flag can also be reset by the **dbadmin** utility while **ladb** is running. For more information, see "Utilities for Managing ladb" on page 47.
IDLETIME	1	How many minutes a database driver can be idle before it will be stopped. **ladb** checks for drivers to drop at one-minute intervals. Valid values for the **IDLETIME** flag are any integer value.
FOREIGN	500	Maximum number of foreign servers. Should be set to the same value as **USERS**.

Parameter	Lawson setting	Description
IFILES	450	Number of tables for each interface.
LAWSONUID	0	Indicates whether to execute foreign servers as **uid lawson** instead of the calling process's uid (0 = no, 1 = yes).

Parameters That Configure Timed Statistics

Use the parameters in the following table to enable timed statistics.

Parameter	Description
TIMESTATS	Optional. Enables the timed statistics feature. Allows values of **TRUE** (on) or **FALSE** (off).
USERFILTER	Optional. Requires that **TIMESTATS** is set to **TRUE**. The name of the user to monitor. If a user name has no program listed next to it, all programs being run by that user are monitored. **Note:** If you have modified the **USERFILTER** parameter with a prefix to identify a user as an operating system user (**USERFILTER=OSID:***username*) or a web user (**USERFILTER=LOBS:***username*), remove the prefix (**OSID:** or **LOBS:**) from this parameter.
PROGRAMFILTER	Optional. Requires that **TIMESTATS** is set to **TRUE**. The program to monitor when using timed statistics. If the program has no user entry preceding it, all instances of this program are monitored.
DATAAREAFILTER	Optional. Requires that **TIMESTATS** is set to **TRUE**. The data area to monitor when using timed statistics.
TIMESTATSDIR	Optional. Requires that **TIMESTATS** is set to **TRUE**. The directory for the output files from the timed statistics monitor. This must be an absolute path. The default for IBM DB2 is **tmp**.

Utilities for Managing ladb

The following utilities are useful when working with **ladb**.

The dbmode Utility

The Set or Display Database Mode utility (**dbmode**) enables administrators to switch Lawson into one of three modes: single-user, multi-user, and no-new-user. This ensures that users do not access the system during maintenance tasks, for example, when running Build Dictionary (**blddbdict**) and Reorganize Database (**dbreorg**). When anyone executes the **dbmode** utility, their user ID is reported in the ladb.log file. Only the user who puts **ladb** into single-user mode or no-new-user mode can set the mode back to multi-user mode.

The dbusers Utility

Use the **dbusers** utility to view database processes, see who is logged in, and verify that **ladb** is running. Use this utility prior to stopping a server program.

The dbadmin Utility

The Database Service Administration utility (**dbadmin**) performs administrative tasks for the Lawson Database Service (**ladb**) without shutting down the service to change the ladb.cfg file. For more information, see "The Lawson Database Server Configuration File (ladb.cfg)" on page 44.

Using the **dbadmin** utility, an administrator can perform the following tasks:

- Shut down idle database drivers

  With driver **REUSE** enabled, the database driver process remains running and connected to associated RDBMS database. The administrator can shut down any idle database driver process not currently in use.

- Enable or disable driver **REUSE**

  The **ladb** service can be configured to reuse database driver processes if the driver does not have any open database files. This option is enabled or disabled when you issue the **`startladb`** command and the **REUSE** parameter is in the ladb.cfg file. Using the **dbadmin** utility, the administrator can toggle the driver **REUSE** flag without stopping the **ladb** service.

- Enable or disable timed statistics collection

  The timed statistics feature gives you information on the amount of processing time used for each function performed on a specific table for each user and program combination. This information is useful when you want to understand performance tuning issues.

  Choose the timed statistics feature to monitor statistics for a specific user, program, or combination of user and program. You can also set this feature to monitor all users and programs, but if there are many users accessing the database, timed statistics might produce a large amount of output.

- Get the current value or set ladb.cfg configuration file variables.

The laps Utility

The Lawson Process Status (**laps**) reporting tool provides system administrators with commands that display Lawson-specific process information including parent-child relationships, session and process groups, process creation times, process IDs, and other valuable process information.

For more information, see the *System Utilities Reference Guide*.

Stopping and Starting the Lawson Database Server

It might be necessary to stop and start servers for routine system maintenance and for troubleshooting. When you make a change to a configuration file, you often need to stop and restart the affected server to ensure that the change takes effect.

You can stop or start servers individually or as part of the procedure of shutting down and starting up the Lawson system.

To stop and start all of the Lawson servers, use the **`stoplaw`** and **`startlaw`** commands. The **`startlaw`** and **`stoplaw`** commands control all servers.

Important: Always shut down the application server (such as WebSphere,) before issuing a **`stoplaw`** or **`stopladb`** command and restart the application server before issuing a **`startlaw`** or **`startladb`** command.

The following chart lists the server stop and start routines as well as other information associated with each server.

Server	Stop and start routine	.cfg file	Log file
ladb	stopladb startladb	ladb.cfg	**`%LAWDIR%/system/ladb.log`**
All Lawson servers	stoplaw startlaw	N/A	Log file for specific server

To stop the database server

1. Shut down the application server.
2. At a command prompt, type **`stopladb`** or **`stoplaw`** to stop all Lawson servers, including the database server.

To start the database server

1. Start the application server.
2. At a command prompt, type **`startladb`** or **`startlaw`** to start all Lawson servers, including the database server.

Managing the Data Dictionary

4

This chapter describes the following data dictionary management tasks:

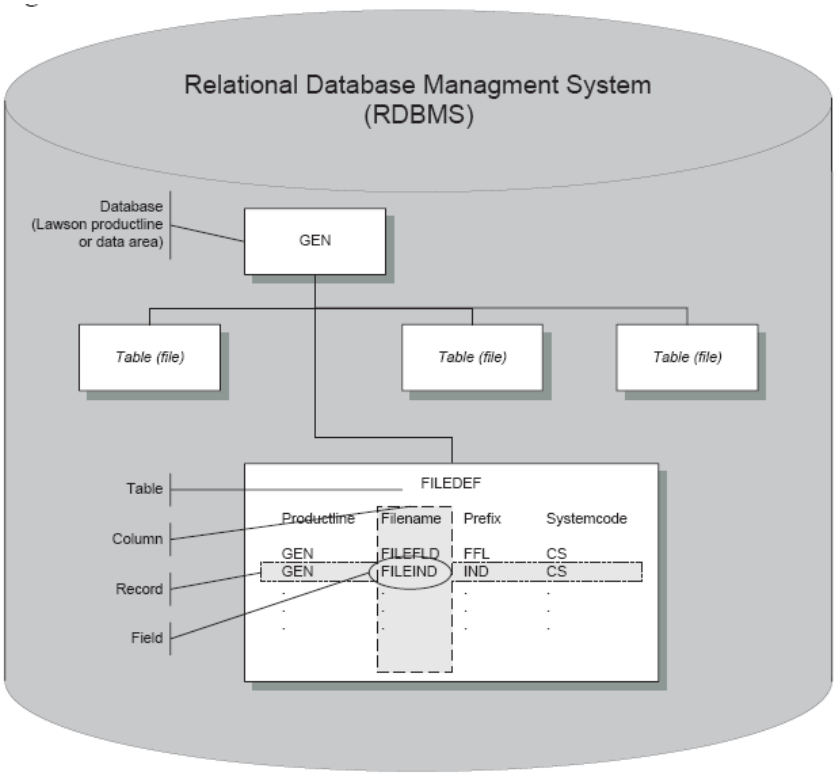
Lawson Data Dictionary Structure

This section contains the following topics:

Understanding Lawson Data Organization

Lawson uses the following strategy to organize data within the IBM DB2 database. The figure and the table that follows illustrate both Lawson and RDBMS terminology for database components.

Figure 4. Illustration: Lawson Database Structure



RDBMS Term	Lawson Term	Description
Database	Product line Data area	An RDBMS *database* is a collection of information organized into interrelated tables of data and specifications about that data. A Lawson *product line* includes the RDBMS database and the Lawson programs that access that data. The GEN and LOGAN product lines contain a relatively small number of programs that you use to manipulate product line data. S3 Lawson System Foundation offers a larger set of programs to access application product line data. On a high level, the product line definition has two components, structure and data. A *data area* is created from a product line, but it includes only the data structure when created. Data can be copied into a data area from the product line, imported from another source, or entered manually through Lawson programs. Lawson programs are shared between a product line and all data areas created from that product line.

RDBMS Term	Lawson Term	Description
Table	File	A *table* in an RDBMS is made up of logically arranged row and column definitions that define an entity (object). A Lawson *file* is equivalent to an RDBMS table. For example, in the preceding figure, the FILEDEF table in the GEN database contains data about Lawson files and the fields contained in the files.
Column	Field	Each table or file consists of *fields*, which are individual units of data.
Column	Element	An RDBMS *column* describes the attributes of an object, or piece of data. A column has a name and data type. In the Lawson database definition, the term *element* is also used for the concept of a column. A Lawson element describes the attributes of an object, such as a user ID or an expiration date. For example, in the preceding figure, the Filename column has the following attributes: alphanumeric, 8-character, database file name. Each field that belongs to the Filename column must conform to those attributes.
Row	Record	A Lawson or an RDBMS *record* or *row* is the organization of multiple, related fields. For example, in the preceding figure, a record is a group of related Product line, Filename, Prefix, and System Code fields.

Product Line Structural Definition

A product line is a complete set of programs. The information that defines the product line is defined by using the Lawson Database Definition utility (**dbdef**). There is one database definition for each product line, data area, and data ID. The database definition is stored in the GEN database. The GEN database contains information about all of the product lines, data areas, and data IDs defined in that environment. On a high level, the product line definition has two components, structure and data.

The **structural definition** includes:

- System codes
- Attachment space
- Elements and fields

- Derived fields including string and group fields
- Conditions, compute, and array value fields
- Text for elements and fields
- Values and translations
- Database relations, conditions, and subtypes
- Database rules, states, and events

Most of the above are described in the Lawson *Lawson Doc for Developers: Application Development Workbench* manual.

The **data definition** includes location and file sizes.

The file sizes are defined and stored in the data area and data ID portion of the product line. Data IDs are sub-areas of data areas with a set of data all their own. Together with the structural component, the data areas and data IDs make up the product line. An entire product line or individual data areas and data IDs may be secured from user access.

Database Spaces

Note: To skip directly to the procedures, see "Defining Database Spaces" on page 68.

A database space is the storage area for your database files. You are required to set up database spaces so the Lawson system knows where to store your data.

You can assign database spaces to each data area, system code, and file. When you assign a space to a file, the data for that file is stored in the location defined by that database space. If a file does not have a database space, the data is stored in the system code's database space. If the system code does not have a database space, the data area's database space is used.

Indexes, Index Keys, and Conditions

Indexes are sets of fields that the Lawson system uses to access and order records in a file. Every file must have at least one index. Fields that make up an index are called **index key** fields.

Using an index in a query makes the search much faster because the query looks only at records defined by the index. When a user makes a request through a Lawson form that involves searching the database to return a list of records, the system constructs a database call. Specifying an index and at least one value for an index key tells the system to search only records that match those values, which makes the query more efficient. The system returns records in the order defined by the index.

Conditions are statements that can be applied to files, indexes, or relations to limit the number of records returned in a search. An index that has a condition applied to it contains only records that meet the criteria defined in the condition.

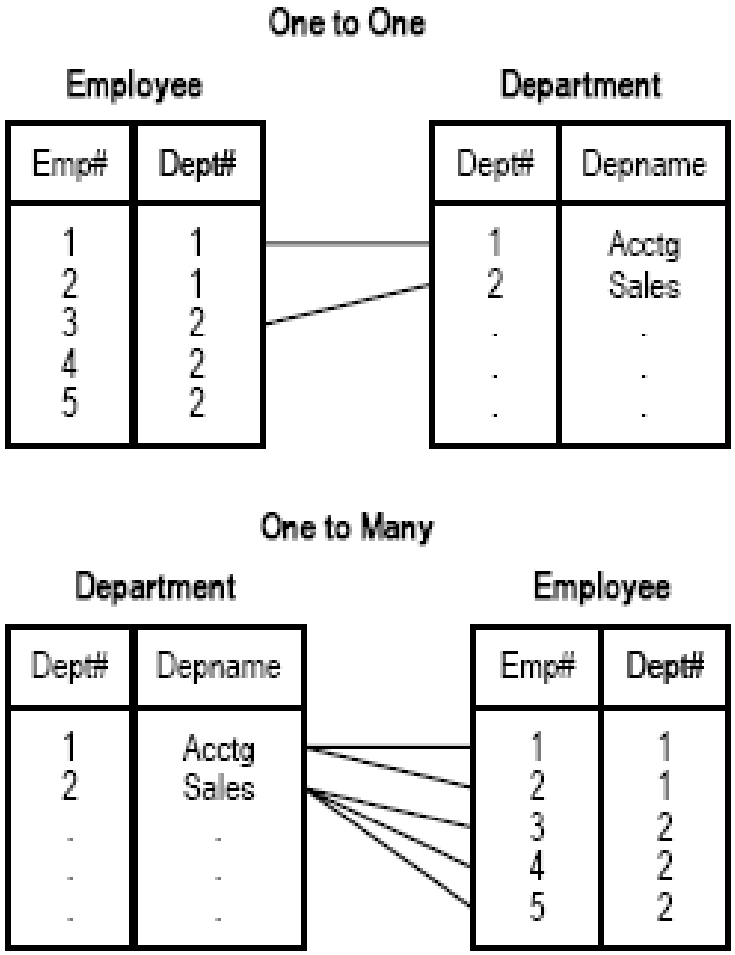
Relations

Relations connect one or more records from one file to one or more records in another file so data can be shared between the files when needed. The system searches for and return records from more than one file if the files are related.

There are two main types of relations:

- A one-to-one relation connects one record from a primary file to one record from a secondary file. For example, a one-to-one relation could exist from the EMPLOYEE file to the DEPARTMENT file, indicating that one employee belongs to one department. The DEPARTMENT field, which supplies the department number and occurs in both files, relates and connects the files.
- A one-to-many relation connects one record from the primary file to many records in the secondary file. For example, a one-to-many relation could exist from the DEPARTMENT file to the EMPLOYEE file, indicating that the data for a particular department also has related data for multiple employees.

Figure 5. Illustration: Relations



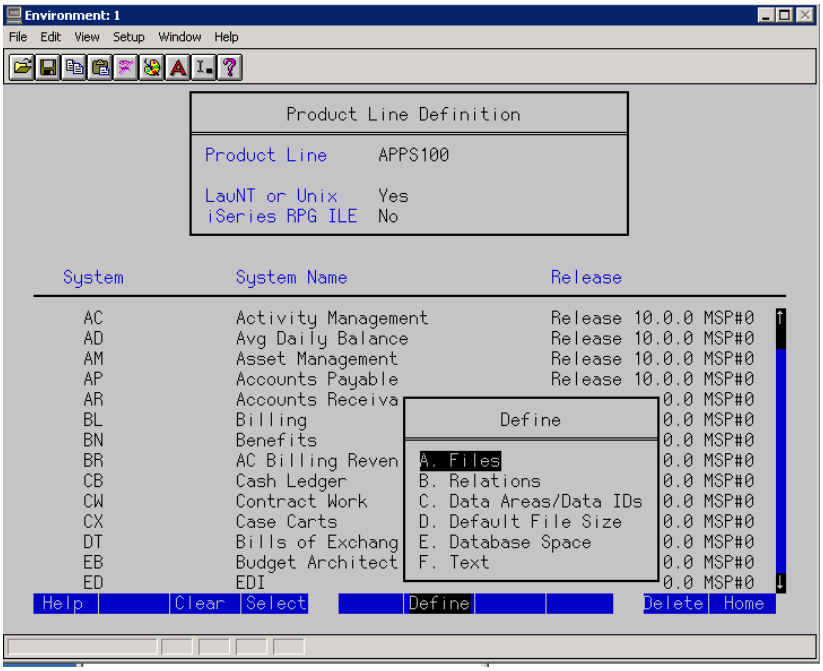
Creating or Modifying Product Lines and Data Areas

This section contains the following topics:

What Is the Database Definition (dbdef) Tool?

Database definition is the process of defining the files for a database as well as the fields, indexes, rules, and help text for each file. The main tool for defining databases is the Database Definition tool (**dbdef**), shown in the following illustration with its Define menu options.

Figure 6. Form clip: The Database Definition (dbdef) and Define Menu



The first step in defining a database is to define the product line, which is a complete set of programs. Once a product line is defined, update the structural components as needed.

The structural definitions are described in the *Application Development Workbench* manual.

The data definition includes location and file sizes. File sizes are defined and stored in the data area and data ID portion of the product line. Data IDs are sub-areas of data areas with a set of data all their own. Together with the structural component, the data areas and data IDs make up the product line. An entire product line or individual data areas and data IDs may be secured from user access.

Modifying Product Lines Using Database Definition

Caution: This procedure does not provide all the steps required to create a new product line. Lawson recommends using the Product Line Copying procedures provided in the *Lawson Administration: Server Setup and Maintenance* guide, or the Lawson installation process documented in the *Lawson Applications Installation Guide* to create a new Lawson product line.

This procedures lets you modify product lines using Database Definition — for example, if you want to modify a field, system code, or Lawson other database definition component.

To modify a product line

1 At the command prompt, type:

dbdef *productline*

- or -

From the Database Administration menu, choose **Database Definition**.

Tip: More detailed information about dbdef is provided in the *Application Developer Workbench* guide.

2 In the Product Line field on the Product Line Definition form, type or select a product line.

3 In the detail area of the form, type or select the following data.

System	Two-character system code.
System Name	Up to 20 characters describing the system.
Release	Release level of the system. All programs should have a release level of 7.0 or greater. Environment utilities identify the version as a period with one or more numbers on both sides. Other text in the field is ignored.

4 Press **Enter**.

Select **Define** or press **F6** to further define database space and other areas of the product line and system.

5 Rebuild the dictionary. Type:

blddbdict *productline*

For more information, see "What Is the Process for Implementing Database Modifications?" on page 73.

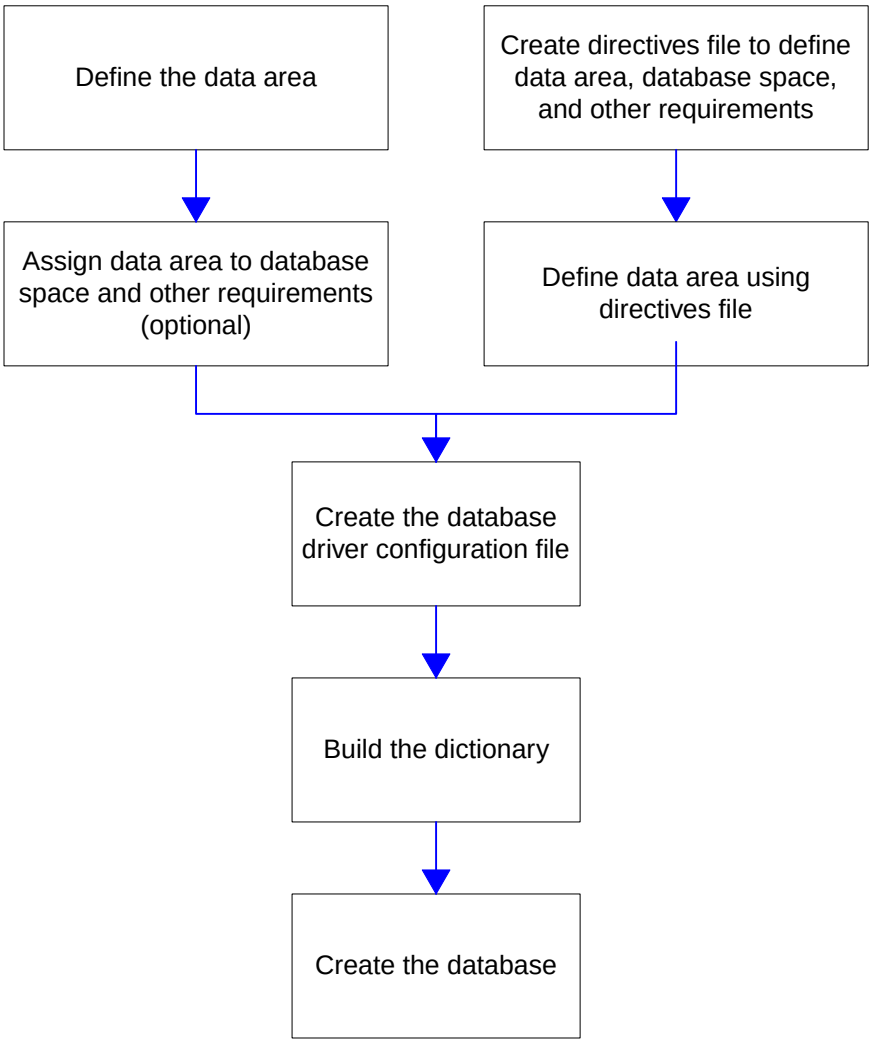
6 Reorganize the database. Type:

dbreorg *productline*

How Do I Create a New Data Area?

The processes for creating a new data area are shown in the following illustration:

Figure 7. Process overview: Creating a new data area



When creating a new data area, you can

- define the data area using Database Definition (dbdef)

  For more information, see "Defining Data Areas and Data IDs Using Database Definition" on page 58.

  – or –

- create a directives file for the Edit Data Area (editda) utility to specify the data area

  For more information, see "Specifying Data Area Attributes Using the editda Utility" on page 59.

Defining Data Areas and Data IDs Using Database Definition

Use this procedure to define data areas and data area IDs using Database Definition (**dbdef**). This procedure also describes how to assign database spaces to data area or to system codes within a data area, and to set file sizes and database spaces for files within a data area.

To define a data area

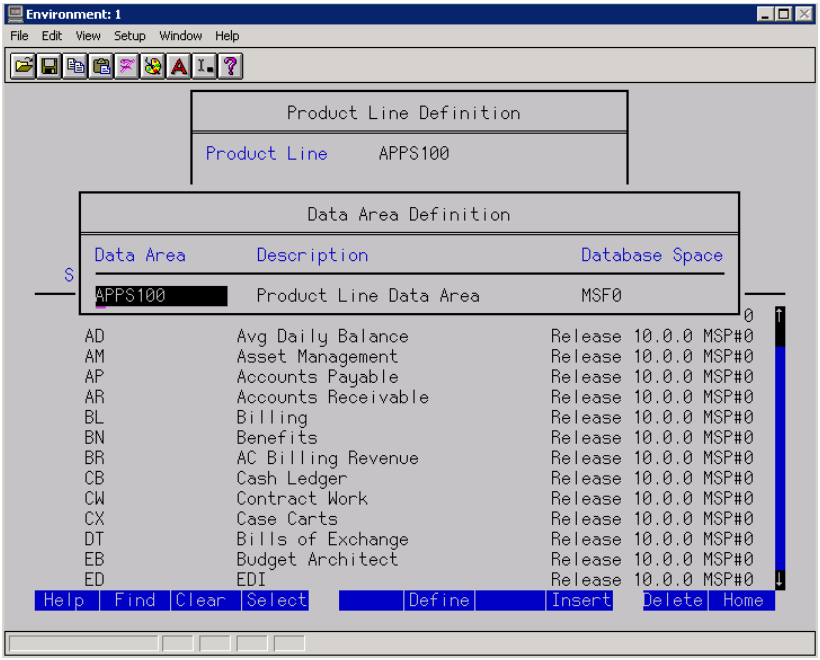
1 At the command prompt, type:

dbdef -p *dataarea*

2 Select **Define** or press **F6** and choose Data Areas/Data IDs.

3 On the Data Area Definition form, type a data area name and description. If one or more data areas already exist, tab through the fields until you get to a blank line.

The name of the data area must be unique, containing no other product line, data area, or data ID with the same name.



4 In the Database Space field, choose Select or press F4 to select a name.

5 Save your changes and exit Database Definition.

6 Create a database driver configuration file for this data area. For more information, see "Editing the Lawson Database Driver Configuration File" on page 37.

The file location must be:

%LAWDIR%/*dataarea***/IBM**

7 Rebuild the dictionary. Type:

blddbdict -p *dataarea*

For more information, see "What Is the Process for Implementing Database Modifications?" on page 73.

8 Reorganize the database. Type:

dbreorg -p *dataarea*

Edit Data Area Utility (editda)

The Edit Data Area utility (**editda**) is a command line tool for managing some aspects of data areas. This includes creating new data areas and data IDs and performing routine metadata changes for data areas in a product line. The utility has two components: a command with associated parameters, and a directives file that specifies the changes made by the utility.

Use **editda** to manage the following data area attributes:

- Add a data area or data ID.
- Assign a database space to a data area.
- Specify a default database space for a system code.
- Specify an initial record count and a record count increment for a file.
- Enable log changes (used for the Resource Management and Upgrade While Active products).
- Enable or disable **varchars**.

After you use the **editda** utility, run **blddbdict** and **dbreorg** to implement the changes.

Variable Length Character Strings (Varchars)

Enable **varchars** for all non-indexed columns in a table or for all columns in a table, indexed or non-indexed. By default, **varchars** are disabled. The **editda** utility enables **varchars** on a table-by-table basis. Once **varchars** are enabled, trailing spaces are truncated for eligible strings.

IBM DB2 specifies **varchars** for long strings even if **varchars** have not been enabled for Lawson data. However, string length varies only if you enable **varchars** in Lawson.

Note: Do not use **varchars** for tables that are updated frequently, because doing so could cause performance problems. Instead, use **varchars** for tables with static data in character (**char**) fields.

Specifying Data Area Attributes Using the editda Utility

The following are the basic steps for running the **editda** utility manually from a command line (not from within a script).

To run editda from the command line

1 Create a text file that specifies the attributes you wish to change. This file is called a directives file.

The position of attribute specifications within the directives file is important.

For more information, see "Directives File Syntax" on page 61.

2 Run the **editda** utility. Type:

editda -c *productline directives_file*

where *productline* is the product line containing the data area and *directives_file* is the path to the directives file.

For other command options, see "editda Syntax" on page 60.

3 If you are creating a new data area, you must create a database driver configuration file. For more information, see "Editing the Lawson Database Driver Configuration File" on page 37.

The file location must be:

%LAWDIR%/*dataarea***/IBM**

4 Rebuild the dictionary. Type:

blddbdict -p *dataarea*

5 **Tip:** Keep the directives file you created in step 1 as a record of changes made to a data area.

Reorganize the database. Type:

dbreorg -p *dataarea*

editda Syntax

The following table shows the parameters for the **editda** command.

Program Option	Description
V	Print version information.
c	Change (or create) all data areas in the product line as per the directives file.
l	List data area changes.
d	Delete the data area. The user is prompted (**Y**/ **N**) to confirm the delete. • **Y** = Yes, continue with the delete. • **N** = No, do not continue with the delete. The **Y** option (see below) suppresses the prompt step.
a	Show all files and system codes in output. (By default **editda** shows changed files only.)

Program Option	Description
v	Show actual value for defaults in output. (By default **editda** shows asterisks.)
productline	The product line target for the directives file (*dir_file*).
dir_file	Path to a text file that contains directives for execution of the **editda** command.
Y	Suppress the prompt for a delete (**-d**) command. Use this when deleting data areas from a script.

The following are examples of uses of the **editda** command.

Example: Change Data Area Settings

Changes data areas specified in the directives file according to the values in the directives file.

editda -c [-*lav***]** *productline dir_file*

If **-l** is specified, changes are written to standard output. If **-a** is specified, all files and system codes appear in the output. If **-v** is specified, actual values for defaults appear in the output.

Example: List Data Area Settings

Sends a list to standard output of the values for data area settings. The output is the same format as the directives file. For more information, see "Directives File Syntax" on page 61.

editda -l [-*av***]** *dataarea*

If **-a** is specified, all files and system codes appear in the output. If **-v** is specified, actual values for defaults appear in the output.

Example: Delete a Data Area

Deletes the specified data area or data ID from GEN, all directories, and dictionaries.

editda -d -[*Y***]** *dataarea*

If you do not use the **-Y** option, users are prompted to make sure they want to continue with the delete. If you use the **-Y** option, the prompt is skipped.

Directives File Syntax

A directives file is a text file that contains specifications for data area attributes. The following shows all possible attributes that can be updated using a directives file. The first letter of each line below (**D, S, F, I**) is a directive and following each directive are the attributes (*DataArea*, *FileName*, etc.) controlled by that directive.

```
D DataArea DBSpace
S SystemCode DBSpace
F FileName RecsInit RecsInc DBSpace LogChanges UseVarchars
I DataID
```

At a minimum, a directives file must contain a **D** directive to specify which data area is to be updated. There is no maximum number of directives that can be included.

Example: Database Space Assignment

```
D new_data_area my_space_1
```

This example shows a simple directives file that creates a new data area (**new_data_area**) and assigns it to a database space (**my_space_1**).

Example: Initial and Incremental Record Counts

```
D data_area_1
F timerecord 10000 2000
D data_area_2
F employ_table 10000 2000
```

This example shows a directives file that updates two data areas. In **data_area_1**, the initial record and record count increments are updated for the table “timerecord.” In **data_area_2**, the initial record and record count increments are updated for **employ_table**. The initial counts are set to 10,000 and the number of records to increment by is set to 2000 for both tables.

Attribute Considerations

An attribute can be specified by its position in the directive ("Using Attribute Positions" on page 62) or by using the attribute name ("Using Attribute Names" on page 63).

Both types of specifications can be contained in the same directive, but any positional specifications must appear before named attributes.

The types of values that can be entered for each attribute are described in "Attribute Values" on page 63.

- A hyphen (-) indicates that a particular attribute remains unchanged.
- An asterisk (*) indicates that the attribute should be set to the default.

Using Attribute Positions

When you specify the attribute by its position in the directive line, consider the following:

- Every attribute in the directive must be in the correct position for **editda** to interpret what needs to be changed.
- Use one white space or a comma to separate attributes.

```
F timerecord 10000 - myspace - *
```

In the above example, the following actions are specified:

- Set **RecsInit** to 10,000
- Set **Database Space** to myspace
- Do not update **LogChanges** (indicated by the hyphen character)
- Reset **UseVarchars** to the default (indicated by the asterisk character)

Using Attribute Names

The attribute name method offers greater flexibility in specifying attributes. An attribute name specification uses the syntax of **`attribute_name=value`**. For example, using the short form for RecsInit, you could specify an initial record count with an entry like **`init=10000`**.

When specifying attributes using their name, you can

- Use shortened attribute names
- Omit attributes that do not change
- Specify attributes in any order within the directive

```
F timerecord VC=*,init=10000,space=myspace
```

The above command shows an equivalence specification that has the following results:

- **`UseVarchars`** (**`VC`**) is reset to the default.
- **`RecsInit`** (**`init`**) is set to **10,000**.
- **`Database Space`** (**`space`**) is set to **myspace**.

For more information, see "Attribute Name Variations" on page 64.

Attribute Values

Following is a list of all attributes that can be updated through the **`editda`** utility and what possible values can be assigned to the attributes.

Note: You need to define *SystemCode*, *FileName*, and *DBSpace* in **`dbdef`** before changing those attributes with **`editda`**.

Attribute	Value
`DataArea`	Any name.
`DataID`	Any name.
`SystemCode`	Any existing system code.
`FileName`	Any existing file (table).
`DBSpace`	Any existing database space.
`RecsInit` (Initial Record Count)	An integer.
`RecsInc` (Record Count Increment)	An integer.

Attribute	Value
LogChanges	For Environment release 8.0.1 and above, most customers do not need to change this attribute. Values are: • **N**=No logging (default) • **Y**=Log for Resource Management • **U**=Log for upgrade • **A**=Log for Resource Management and upgrade
UseVarchars	Enable **varchars** for all non-indexed columns: • **K**=Enable **varchars** for all columns (indexed and non-indexed) • **N**=Disable **varchars** for all columns

Attribute Name Variations

Equivalence specifications (**attribute_name=value syntax**) can make use of shortened names. The following table shows the acceptable alternatives.

Attribute	Accepted names and variations			
DataAreaName	**DATAAREANAME**	**NAME**		
DataID				
SystemCode	**SYSTEMCODE**	**CODE**	**NAME**	**SC**
FileName	**NAME**	**FILENAME**	**FN**	
DBSpace	**DBSPACE**	**SPACE**	**S**	
RecsInit (Initial record count)	**RECSINIT**	**INIT**		
RecsInc (Record count increment)	**RECSINC**	**INC**		
LogChanges	**LOGCHANGES**	**LC**		
UseVarchars	**USEVARCHARS**	**VARCHARS**	**VC**	

Example: Comparison of Attribute Specifications

Each of the directives in this example accomplishes the same updates on the timerecord file:

- Set the initial record count to 10000
- Make the database space myspace
- Return the varchars option to the default setting

Attributes are specific by the position in the line; white spaces separate the attributes:

```
F timerecord 10000 - myspace - *
```

Attributes are specific by the position in the line; commas separate attributes:

```
F timerecord,10000,,myspace,,*
```

Attributes are specified with a mix of positions and names; position specifications come before attribute names, and white space separates attributes:

```
F timerecord 10000 space=myspace VC=*
```

Attributes are specified with attribute names; unchanged attributes omitted:

```
F timerecord s=myspace,VC=*,init=10000
```

Automatic Data Area Generation Script

The Automatic Data Area Generation script creates new data areas within a single database server instance. Data areas share existing database spaces. Since this script works directly with the database, first create a configuration file with specifications for the database you have installed, then run the script to generate the data areas. Add the database values specific to your installation.

The data area tables use the data area name as the database schema name.

To configure the ASP script for your database installation

1 Copy the script to a configuration file.

At a command prompt, type:

`cp %GENDIR%/bin/add_asp_da.pl %LAWDIR%/system/add_asp_da.cfg`

The data area files are created using the data area name as the schema name.

2 Locate the section for the IBM DB2 database and modify the following parameters:

Value	Description
`db2instance`	The name of the IBM DB2 instance.

3 Save the script.

To run the ASP data area generation script

- At the command line, type:

  add_asp_da *productline dataarea dbspace dbname username password*

  where the following parameters apply.

This parameter	Indicates
productline	The product line in which the data area is created.
dataarea	The data area to create.
dbspace	The database space within the product line to assign to the data area.
dbname	The name of the database to use for the data area. *dbname*, *username*, *password* remain fixed. Each data area is created using a data area name as a schema.
username	The name to use when connecting to the database.
password	The password to use when connecting to the database.

Enabling and Disabling Variable Length Character Strings

This procedure explains how to enable the use of variable character columns for a particular table.

To enable or disable the use of varchars

1 Create a directives text file to enable **varchars**. The minimum information required for the file is two lines in the following format:

  D *dataarea*

  F *file_name* **UseVarchars =** *value*

  where *value* is **Y**, **K**, or **N**.

  - **Y** enables **varchars** in all non-indexed columns in the data area.
  - **K** enables **varchars** in all columns in the data area, indexed and non-indexed.
  - **N** disables **varchars** in all columns in the data area.

2 Run the **editda** utility. Type:

  editda -c *productline dir_file*

where *productline* is the product line containing the data area and *dir_file* is the path to the directives file.

3 Rebuild the dictionary. Type:

`blddbdict -p` *`dataarea`*

4 Reorganize the database. Type:

`dbreorg -p` *`dataarea`*

It is a good idea to keep the directives file you created in step 1 as a record of the changes made to a data area.

Setting Database Spaces

This section contains the following concepts and procedures:

Database Spaces

A database space is the storage area for your database files. After defining one or more database spaces using the utility **dbdef**, assign them to the product line, data area, system code, or table.

When you assign a space to a file, the data for that file is stored in the location defined by that database space.

- If a *file* does not have a database space, the system stores data in the system code's database space.
- If a *system code* does not have a database space, the system uses the data area's database space.

When the IBM DB2 database is running on the OS/390 platform, the **`Database Space`** name in **dbdef** refers to the **`Stogroup`** name. It is possible to spread the indexes of a single file across multiple indexes.

When you assign a database space to a file, the data for that file is stored in the location defined by that database space. If a file does not have a database space, the data is stored in the system code's database space. If the system code does not have a database space, the system uses the data area's database space.

Make as many different combinations of database spaces for tables and their indexes as you wish by simply defining database spaces for each permutation you need. Keep in mind that all indexes on the same table share the same tablespace in **dbdef**.

For more information on the **dbdef** utility, see the *Lawson Doc for Developers: Application Development Workbench* manual.

Defining Database Spaces

This procedure lets you define a database space in order to designate where data is stored.

Need More Details? Check out the following concepts:

- "Database Spaces" on page 67

To define a database space

1. In the Product Line field on the Product Line Definition form of the Database Definition utility (**dbdef**), type or select any product line or data area. You will assign the database space to the product line or data area after the space is defined.
2. Press **Define (F6)** and choose Database Space.

   The Database Space Definition form appears.

   The fields that appear vary depending on the database type. The following table describes the valid values for each database type.

Field	Description
Database Space	Name that identifies the database space you are defining.
Type	Type of database you use. Valid entries are IBM DB2, or Empty.
Cluster Node	Optional. The cluster node you define the space for. The default is the cluster node you are currently on. The cluster node name must be valid.
Add Locally	Not implemented.
DB Space	The location where your database files are stored. You must specify a location here or in the field below.

Field	Description
Index Location or Index DB Space	The location where your index files are stored. The default is the value in the Location field (directly above this field). You must specify a location here.
Condition Name	Not implemented.
Is View	Specifies whether the files in this space are views of non-Lawson tables in the Lawson database. The Reorganize Database (**dbreorg**) utility does not reorganize views. Setting this field to Yes allows you to define a file to the Lawson system, but dbdef will not create a file with that name. This allows the Lawson system to access the data in an external file, via Selects and Drills to it from within Lawson, etc.
MetaData Imported	Enter Yes or No. The default is No. If MetaData Imported is Yes, the view names and field names in those views will be V_<OriginalFileName>. **Note:** This field is only accessible if the value for the Is View field is Yes.

3 Press **Enter** twice.

You are now ready to assign the database space to the product line or data area. For more information, see "Modifying Product Lines Using Database Definition" on page 56. For more information, see "Defining Data Areas and Data IDs Using Database Definition" on page 58.

To delete a database space

Before you start If you have assigned the database space to a product line, data area, or file, you must delete the database space on the Product Line Definition form and the File Sizes Definition form before you can delete it on the Database Space Definition form. It is good practice to move any existing tables out of the database space before deleting the database space.

1 On the Database Space Definition form, type or select the database space you want to delete.

2 Press **Delete (F9)**.

3 Press **Enter**.

How Is Database Space Saved Using the Empty Database Type?

You might have empty Lawson tables at your site. For example, you may not use a Lawson application that populates a particular table. If a table is not being used at your site, use the **dbdef** utility to change the table to type **`Empty`**. This frees up disk space that had been allocated for the table.

Before you perform this procedure for any table, be sure that the table is actually empty at all times. The Payroll application, for example, includes tables that contain records only at certain times in a payroll cycle. The application automatically deletes the records when they are no longer needed. A table of this kind should *not* be set to an empty database space.

If you are not sure whether a table is being used, use the **count** utility to determine whether it has records. Run the **count** utility over a period of time before setting a table to empty.

If a table that has been set to **`Empty`** is actually being used, you receive the message “Empty DB Space does not allow Stores.”

Note: It is theoretically possible that you might have an entire system code that is not used. If you are certain that none of the tables in a particular system code contains records, use the procedure referred to above to assign the system code to an empty DB space.

For more information, see "Assigning a Table to an Empty Database Space" on page 70.

Assigning a Table to an Empty Database Space

This procedure explains how to use the **dbdef** utility to create an empty database space and assign an unused table to the empty space.

Before you start Before you perform this procedure, be certain that the table you specify to an empty database space does not have records at any time. For more information, see "How Is Database Space Saved Using the Empty Database Type?" on page 70.

To create an empty database space

1. At a command prompt, type:

   `dbdef`
2. Select the product line and then select **Define** or press **F6**.
3. At the Define form, select **E** (Database Space).
4. Type a name such as "Empty” for the new Database Space.
5. Tab to **Type** and press **F4**.
6. Select **Empty**.
7. Select OK to complete the change.

To assign a table to the empty database space

1. From the product line selection form in **dbdef**, select the product line which contains the table you want to set to an empty database space, and then select **Define** or press **F6**.
2. Select **Option C** (Data Area/Data IDs).
3. At the Data Area definition form, select **Define** or press **F6** again.
4. Select **Option B** (File Sizes and Database Space).
5. Select **Find** or press **F2** and type the name of the file you want to assign to the Empty Database Space.
6. Choose **Select** or press **F4** to select the empty database space you defined. For more information, see "To create an empty database space" on page 70.
7. Select OK to complete the change.
8. Build the dictionary and reorganize the database to implement the change. For more information, see "Reorganizing the Database" on page 79.

How Is Database Space Saved Using a Virtual Index?

The virtual index feature allows you to turn off physical storage for specified indexes.

Important: Virtual indexes are intended to be used in situations where the maximum number of indexes supported by **dbdef** has been exceeded. While virtual indexes conserve physical storage, they are built at run-time, thus they must be built every time they are used by a form or report to access data. This generally has a negative impact on system performance.

A virtual index should not be created for any data set that is accessed frequently. If you need to add an index to a table, and need to use a virtual index, review all the indexes for that table in order to select the index that will impact your system performance the least.

One scenario that supports use of a virtual index is when you are certain an index will not be used in your system. For example, if index ABCSET was designed for use between two applications and you do not have the second application installed, index ABCSET could be set to virtual.

Application developers or database designers create virtual indexes using the Database Definition (**dbdef**) utility.

When Can I Create a Virtual Index?

- A unique index can also be a virtual index only if there is another non-virtual index that enforces the uniqueness constraint. In other words, the virtual index must be contained within another unique index.
- A conditional index cannot be a virtual index.
- A primary index cannot be deployed as a virtual index.

How Are Virtual Indexes Handled by the Database Interface?

- The **blddbdict** utility and the database driver (**ibmdb**) both process the **Virtual** flag.

  Note: Since performance may be impacted when processing without indexes in place, comments are generated with SQL code so the database administrator can easily determine whether a query should be using an index. This applies only to SQL databases that support hints; if index hints are turned off, no comments are generated.

- The **verifyibm** utility displays an informational message when a virtual index occurs in a table definition.
- The **bldibmddl** utility does not create DDL for virtual indexes.

Creating a Virtual Index for a Table

Before you start A virtual index saves physical storage, but can negatively impact runtime system performance. Carefully evaluate the need for the virtual index and the possible impact to your system. For more information, see "How Is Database Space Saved Using a Virtual Index?" on page 71.

1. On the File Definition form of the Database Definition tool (**dbdef**), select the desired product line and file.
2. Press **Define (F6)**.
3. Select Indexes.
4. In the **Virtual** field of the Index Definition form, enter **Yes**.
5. Press OK twice to save the index.

Implementing Dictionary Modifications

This section contains the following topics:

What Is the Process for Implementing Database Modifications?

This section describes the procedure required to build the data dictionary and reorganize the database, after you create or modify product lines. You can use this procedure with a single product line or a product line with multiple data areas and data IDs, depending on the options you specify during the process.

Multiple data area reorganizations can occur simultaneously, as long as the product line dictionary is up-to-date. There are two options for building and reorganizing the dictionary for a product line with multiple data areas:

- Full Product Line Dictionary Build and Reorganization

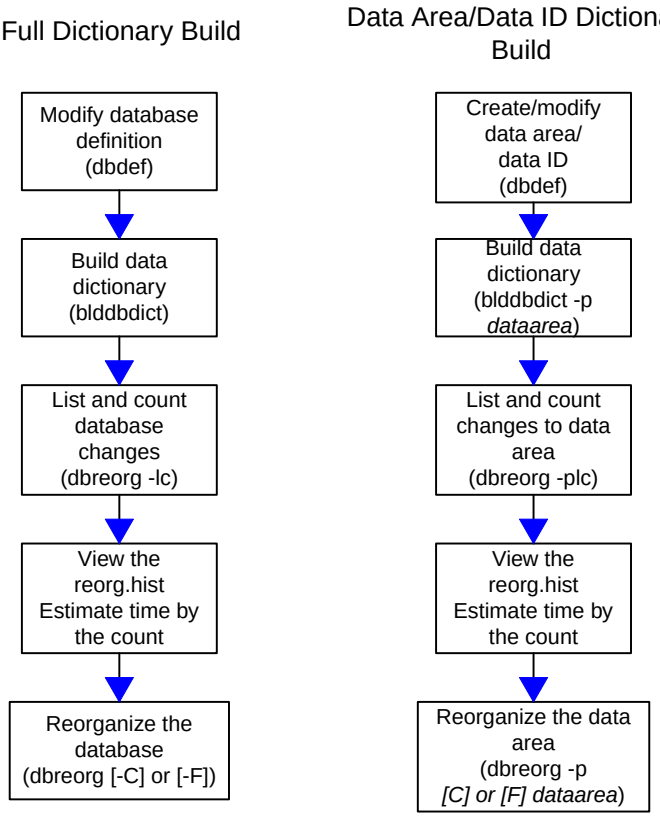
  When you make structural changes to the product line, such as adding a system code, you must do a full dictionary build and reorganization for the product line and all associated data areas and data IDs.

  When multiple data areas exist, **blddbdict** creates or rebuilds the structure dictionary first, then the data area dictionaries. If all dictionaries generate without error, they are moved to the appropriate files.

- Data Area or Data ID Dictionary Build and Reorganization

  When you make changes to data area information, such as file size or location, to one or more data areas, you must do a data area/data ID build and reorganization of the specific data area/data ID dictionaries.

Figure 8. Process flow: Full dictionary and data area/data ID dictionary builds



The following steps describe the procedures shown in the preceding figure:

1. Create or modify the database definition using **dbdef**.

   At this point, you can make the following changes:

   - add new fields
   - change the definition of existing fields
   - delete unused fields
   - add new files
   - add new system codes
   - assign a database space to the data area or product line

2. Build the data dictionary.
3. Create and verify a list of database changes.
4. Reorganize the database to implement the changes.

   The **dbreorg** process rebuilds only the tables for which the structure has changed.

5. Verify successful reorganization.
6. If you have added or removed fields from the database, modify the associated application programs and recompile the product line.

How Does the Database Reorganization Utility Work?

Database reorganization is the process of applying the changes you have made to the data dictionary to the database. Reorganize the database whenever you modify the structure of the database through the Database Definition utility (**dbdef**) and build the data dictionary (**blddbdict**).

After a database reorganization, the database tables match the database definition. Tables are restructured in alphabetical order, one table at a time. As part of the reorganization, the current data dictionary becomes the old dictionary and the new or future dictionary becomes the active dictionary. The original dictionary is overwritten.

There are three modes for database reorganization: Safe, Fast, and Conservative. The following table shows how each mode works.

Mode	Description
Safe (-S)	Table data is saved in a dump file before the utility alters the table directly. If an alteration fails, **dbreorg** switches to Conservative mode for the failed table and reload from the dump file.

Mode	Description
Fast (-F)	Bypasses the step of saving the table's data in a dump file and alters tables and indexes directly in the database. If an alteration fails, **dbreorg** switches to Conservative mode for the failed table and perform a dump/load.
Conservative (-C)	Requires longer processing time but ensures that no data is corrupted. **dbreorg** performs the following steps: **1** Export the contents of the data table to a flat file. **2** Drop the tables and indexes. **3** Create the table with the new definition. **4** Load data from the flat file into the newly defined table. **5** Build indexes. **6** Remove the flat file. **Important:** If you are reorganizing to move the application data from one database to another, **dbreorg** automatically uses the **-C** option or conservative mode. For example, if you previously stored your data in the Lawson database and now you are moving your data to a relational database, **dbreorg** uses this mode.

Other Database Reorganization Utility Options

This section describes the full syntax of the **dbreorg** utility.

dbreorg [-SCFYdin] [-W *WFFile***] {{-p** *dataarea***} | *Productline*}**

dbreorg [-G [*GCFile***]] [-W** *WFFile***] {{-p** *dataarea***} | *Productline*}**

dbreorg [-lcs] {{-p *dataarea***} | *Productline*}**

dbreorg [-x] {{-p *dataarea***} | *Productline*}**

Note: Multiple file names can be separated with a space, tab, carriage return, or comma.

-Y	Yes. Do not prompt to continue in Fast mode. (Requires **-F**.)
-d	No physical database exists.

-G *file*	Garbage collection. Reorganize only file(s) specified in the garbage collection file. If no garbage collection file is specified, all files are reorganized. The garbage collection file is a list of files to be reorganized. File names specified in the garbage collection file are separated by any of the following: <cr> (carriage return character), spaces, tabs, or commas. **Important:** The previous syntax for specifying garbage collection is supported for a limited time to give clients an opportunity to update scripts that use the old syntax. Migrate scripts that include garbage collection as soon as possible.
-i	Rebuild Data ID dictionaries.
-l	List only. List files to be reorganized but do not perform a reorganization. The **-lc** options together provide details about what the **dbreorg** does.
-c	Show file changes. Requires **-l**.
-s	List number of records in the file to be reorganized but do not perform the reorganization. Requires **-l**.
-n	Do not run **srgen**.
-p *DataArea*	Reorganize only the specified data area or product line.
-x	Examine the control file.
-W *file*	Use work file list.

dbreorg Files

The Reorganize Database utility (**dbreorg**) uses three files:

- The reorg.hist log file.
- The reorg.rwrk work file.
- The reorg.cntl control file.

When you run **dbreorg**, it writes the changes it has made in the data tables to the log file, reorg.hist. For the data area named *test*, the log file is located in %LADBDIR%/TEST.

The %LAWDIR%/*dataarea*/reorg.rwrk work file is a flat file that contains the data from a single table. **dbreorg** writes to the work file during the unload phase and reads from it during the load phase. If **dbreorg** completes successfully, it deletes the work file when it terminates.

You can choose any location for the work file. The default location is *dataarea*. If the disk on which the work file resides becomes full during the reorganization, **dbreorg** prompts you for the path and name of another work file to which it can write the additional data.

As **dbreorg** executes, it writes information to the control file, reorg.cntl, describing what step the utility is currently executing. The reorg.cntl file contains a description of all the work that needs to be done and all the work that has been done during the reorganization, providing error detection. When **dbreorg** successfully completes, it removes the reorg.cntl file. The control file is located in the %LADBDIR%/*dataarea* directory.

If **dbreorg** terminates abnormally before it has finished reorganizing the product line, it does not remove the control file. If you fix the cause of the failure and run **dbreorg** again, it can pick up where it left off by reading the information in the reorg.cntl file.

Troubleshooting the Database Reorganization Process

Before you reorganize a database, always create and view a list of the potential changes that will result to identify possible problems.

- The **`-l`** option lists the files to be reorganized without actually reorganizing them.
- The **`-c`** option lists file changes. It compares the current dictionary with the new dictionary. To run **`dbreorg -lc`**, both versions of the dictionary must be present. That is, **blddbdict** must have been run a second time to create a new or future dictionary.
- The **`-s`** option lists the number of records to be changed. The count of changes helps you estimate the amount of time it takes for the reorganization.

Note: Both the **`-c`** and **`-s`** options require the **`-l`** option.

The **dbreorg** utility stores the list of changes and the number of changes in the reorg.hist log file, located in *dataarea*. There is one log file for each data area.

When reorganizing a database, the **dbreorg** utility creates the reorg.cntl control file in *dataarea*, as well as one or more work files that contain application data from the table currently being reorganized. By default, the work file is stored in *dataarea* and is named *dataarea*/reorg.rwrk.

Caution: Never attempt to remove, edit, or change these files. To do so makes the files unusable, causing the recovery of the database reorganization to fail. Should this happen, file recovery requires using the database backup.

The **dbreorg** utility uses the reorg.cntl file to restart at the table where it failed, in the correct phase (Unload, Load, or Index).

- If **dbreorg** fails during the **Unload** phase, restarting unloads data at the first row of the table.
- If **dbreorg** fails during the **Load** phase, restarting drops the table, recreates it, and then starts reloading data from the first record in the temporary work file.
- If **dbreorg** restarts during the **Index** phase, it recreates indexes.

These features help you correct errors that caused the **dbreorg** to fail. Rerun **blddbdict** to rebuild the input dictionary, if necessary, and restart **dbreorg**

Only remove the control and work files when a database reorganization fails and cannot be corrected or recovered. If the backup of the database is reloaded, putting the database back to the status prior to the reorganization, the reorg.cntl file and work files must be removed. This removal prevents subsequent database reorganizations from using the work and control files from a previous reorganization.

Caution: Failing to remove the control and work files after reloading from backup could cause subsequent reorganizations to either fail or to run to completion for only part of the database reorganization, resulting in lost application data.

In a normal state, there are no reorg.cntl or work files present. They exist only if **dbreorg** is active or if **dbreorg** has failed and error correction has begun, restarting the **dbreorg** process.

If a dbreorg Fails

The method of recovering from a failed database reorganization prevents dictionary definitions from becoming inconsistent. In some previous releases, dictionary definitions could become inconsistent when, for example, a **dbreorg** failed and, before the dictionary problem that caused the failure was resolved, a new version of the dictionary was created. If this situation occurred (which would be most likely to happen only in a large company where several administrators and developers make changes to a database), the dictionary that the failed **dbreorg** was based on would no longer exist and the problems that caused the failure could not be isolated.

To prevent inconsistencies, **dbreorg** creates a saved dictionary, which is a copy of the **+**dictionary (the new, in-progress dictionary). The saved dictionary contains definitions of tables that have been reorganized successfully. If the **dbreorg** process completes normally, the saved dictionary is deleted as part of the **dbreorg** process. If **dbreorg** fails, the saved dictionary, which contains definitions of successfully reorganized tables, exists in the product line. The definitions of failed tables remain in the current dictionary.

The reorg.cntl file contains a record for each table that **dbreorg** attempted to reorganize that tells which dictionary contains the current definition of the table. The existence of a reorg.cntl file and a saved dictionary file indicate that the active dictionary is in an unstable state. The saved dictionary file contains information that **dbreorg** makes use of the next time it is run. Do not delete the saved dictionary.

The saved dictionary naming convention includes a unique version stamp. If another **dbreorg** is run while the dictionary is in an unstable state, a new saved dictionary with a unique version ID exists in the product line. The saved dictionary uses the naming convention **SD989697777**, where **SD** stands for "saved dictionary" and where **986997777** is an example of a version number assigned to the file.

If a data reorganization fails, investigate the cause of the failure. After correcting the error, rerun **dbreorg**. The next time you successfully rerun **dbreorg**, all saved dictionary files are deleted by **dbreorg**.

Reorganizing the Database

This procedure runs the Reorganize Database (**dbreorg**) utility to implement modifications that have been made to product lines or data areas.

If you change an existing product line or data area, run the Reorganize Database utility (**dbreorg**).

Note: If you are adding a new product line or data area, run the Create Database utility (**dbcreate**) first. See the *System Utilities Reference Guide* for syntax and instructions for **dbcreate**.

Need More Details? Check out the following concepts:

- "What Is the Process for Implementing Database Modifications?" on page 73
- "How Does the Database Reorganization Utility Work? " on page 74

To implement modifications to a product line

1 Build the data dictionary. At the command prompt, type:

blddbdict *productline*

- or -

From the Database Administration menu, choose **Dictionary Maintenance**, and then **Build Dictionary**. Type the name of the product line and press **Enter**.

2 Create a list of changes that will occur when you reorganize the product line. At the command prompt, type:

dbreorg -lc *productline*

3 View the list of changes, located in:

%LADBDIR%/*productline*/reorg.hist

4 If the changes listed are correct, reorganize the product line. At the command prompt, type:

dbreorg *productline*

To implement modifications to data areas

Note: When you make only location and extension changes, only the data area dictionaries where the changes occurred need to be rebuilt and those data areas reorganized.

1 To build a data area dictionary, at the command prompt, type:

blddbdict -p *dataarea*

- or -

To build all data area dictionaries within a product line, type:

blddbdict -q *productline*

From the Database Administration menu, choose Dictionary Maintenance, and then Build Dictionary. Select the **-p** option to build only one product line/data area and press **Enter**.

2 Create a list of changes that will occur when you reorganize the product line. At the command prompt, type:

dbreorg -lc *productline*

- or -

dbreorg -lcp *dataarea*

3 View the list of changes. Open the following file:

%LADBDIR%/*productline*/reorg.hist

4 If the changes listed are correct, reorganize the product line. At the command prompt, type:

dbreorg -p *dataarea*

Finding and Fixing Dictionary Issues

This section contains the following topics:

What Is the Verify Database Utility?

The Lawson repository contains descriptions of all Lawson tables and indexes. When these objects are stored in IBM DB2, the database driver (ibmdb) translates the Lawson description for storage in the IBM DB2 database. For Lawson programs to work properly with Lawson tables stored in the IBM DB2 database, Lawson dictionary descriptions must match the expected translations in IBM DB2. Mismatches in these definitions can lead to performance problems or even program failure (column definition mismatches).

The **verifyibm** utility checks for two kinds of differences when it compares the Lawson dictionary with the IBM DB2 definitions:

- **Missing objects**

  Any table or index defined in the Lawson dictionary that has no counterpart in the IBM DB2 database results in a missing file or index message.

The utility checks to make sure that each table and index defined in the Lawson dictionary has a matching table and index in the IBM DB2 database. It does not check to make sure that all objects defined in the IBM DB2 database have counterparts in the Lawson dictionary. Any tables or indexes placed in the IBM DB2 database from a non-Lawson source do not show up in a report.

- **Mismatched column definitions**

  When the columns defined in a Lawson table or index are not matched by the IBM DB2 database definitions of those columns, the utility returns a mismatched file or index definition message. Mismatches include differences in data types, lengths, and so on. All fields are assumed to not allow nulls only those that do allow nulls have output noting this fact.

  When **verifyibm** compares definitions, it matches each column in the Lawson table or index with the IBM DB2 database column at the same position in the table or index. If a column is missing in the SQL definition, the missing column appears in the report as a mismatch in the succeeding columns.

While the utility finds the differences between the Lawson dictionary and the IBM DB2 database definitions, it does not fix the mismatches or alter the definitions in any way.

If you run **verifyibm** without options, it prints out a line that notifies you as it checks each table and index in the specified product line, system code, or file. If you run the utility in quiet mode, it prints out only the messages that identify the differences between the definitions, followed by a summary.

Verifying the Dictionary

This procedure guides you in running the utility to ensure that the Lawson database dictionary and the IBM DB2 database definition match.

To verify the dictionary

1 Run the utility on your product line or data area, and direct the output to a file. Type:

verifyibm [-qcsV] [-u *DbLoginName*`]` **[-p** *Password*`]` *productline*`|`*dataarea* `[`*systemcode filename*`]`

where the following parameters apply.

This parameter	Indicates
-q	Quiet mode.
-c	Check views.
-s	Strict mode.
-u *DbLoginName*	The database login name to use.
-p *Password*	The password for the database login name.

This parameter	Indicates
-V	Print the utility version number.

2 Open the output in a text editor. Use the information it contains to diagnose any mismatches between the Lawson repository and your IBM DB2 database.

What Is the Build Data Definition Language Utility?

The **bldibmddl** (Build IBM DB2 Data Definition Language) utility generates SQL statements for creating or deleting selected tables, indexes, and stored procedures. The utility can operate on specific tables, all tables in a system code, or all tables in a product line or data area. The utility creates Data Definition Language (DDL) text according to the highest possible version with the given compatibility. The definitions that generate the SQL statements reside in the Lawson dictionary and are maintained by Lawson utilities.

The **bldibmddl** utility can be used to:

- Generate DDL for the Lawson tables and indexes.
- Repair problems that are identified by the **verifyibm** utility.
- Drop and recreate tables or indexes that are out of sync with the Lawson data dictionary or are somehow missing, or to conserve space and enhance performance (**-IR**).
- Generate DDL statements to a file for examination or tweaking.
- Drop indexes before you load data, then rebuild the indexes after the data is loaded (**-DI** and **-I**).
- Truncate a table by dropping and recreating it (**-R**).

If a table or index needs rebuilding or modification without going through the reorganization process, or if you want to make changes to the definition statements issued by Lawson to IBM DB2, use the **bldibmddl** utility supplied by Lawson.

To ensure that no one else uses the file containing the generated DDL, execute this utility from a secured directory or change the permissions on the file. After you have used the file, delete it.

Using the Build Data Definition Language Utility to Fix Dictionary Mismatches

This procedure guides you in using the **bldibmddl** to fix any dictionary mismatches you encounter.

Caution: The **bldibmddl** utility does not save, copy, or back up any of the data in a table before it drops the table. Therefore, use the utility carefully. If you have questions about the result of a combination of options, Lawson recommends that you first run the utility without the **-U** option and direct the output file to a text file. Review the generated DDL to ensure that the utility is doing what you expect.

To Fix Mismatches

1 Identify any objects that need repair.

2 Issue **bldibmddl** commands to generate DDL to fix the problems you have identified in step 1.

Important: Send the DDL to one or more files do not issue it directly to the database.

Use the following syntax and commands:

bldibmddl [-UqaoRDTIEKGBWXSV] [-u *DbLoginName*] **[-p** *Password*] *productline*|*dataarea* [*systemcode*] [*filename*]

-U	Update and create objects in the database. If you use **-U**, the generated SQL statements update the IBM DB2 database definition. Otherwise, SQL statements are generated but not executed. If you are working with critical data, do not use the update mode without first examining the DDL. Instead, the preferred method is to direct the output to a file that you can examine, edit if necessary, and issue the command to the database through the SQL utility.
-q	Quiet mode. Do not echo DDL to **stdout**.
-a	Positioning mode. Start at: *dataarea* [*systemcode*] [*filename*] If you use the **-a** option, you can start the utility at a specific system code or file.
-o	Offline mode. The command is run without connecting to the database.
-R	Rebuild; drop objects before creating. If you do not specify either the **-R** or **-D** option, output is generated to create tables and indexes.
-D	Drop objects only. If you do not specify either the **-R** or **-D** option, output is generated to create tables and indexes.

-T	Affect table objects only. If you do not specify the **-T** or **-I** option, output is generated for both tables and indexes.
-I	Affect index objects only. If you do not specify the **-T** or **-I** option, output is generated for both tables and indexes. If you do not specify either the **-R** or **-D** option, output is generated to create tables and indexes.
-E	Affect text index objects only. Use with the **-I** option.
-S	Generate DDL into files by system codes: *dbname*.*systemcode*.**ddl**
-V	Print the utility version.
-G	Affect trigger objects only.
-B	Affect non-native objects only. Cannot be used with the **-U** option, but requires the **-T** and/or **-I** options. The default parameter combination is **-BTI**.
-W	Affect native view objects only. Cannot be used with **-U** or any other object type option.
-X	Affect non-native view objects only. Cannot be used with **-U** or any other object type option.

3 Inspect the DDL in the output files and edit it if necessary.

4 If you need to save any data before you drop a table for rebuilding, use the **dbdump** utility to unload the data.

Caution: Dumping data, changing the data definition, and reloading the same data are not data safe operations and could corrupt data. This procedure should be performed only by someone with expert knowledge of the database and Lawson data.

5 Use a SQL tool to issue the DDL to the database.

6 Run the **verifyibm** utility to verify that your changes have fixed the previous mismatches.

7 If necessary, reload any data that was lost during the rebuild.

Managing Lawson Data

5

This chapter contains the following topics:

Lawson Tools for Data Management

Use Lawson tools to access and manage Lawson data. This ensures that your data remains accurate and accessible by Lawson programs and utilities.

Warning: While Lawson Software provides some instructions for accessing the database via non-Lawson tools, using third-party products correctly is the responsibility of the user. Lawson cannot assume any responsibility for the results (damage of data and/or structure) of improperly using non-Lawson tools. Refer to the third-party tools documentation when attempting to access Lawson data via any non-Lawson tool.

Where to Find More Information

This guide provides instructions for using a number of tools designed to help you move, copy, or update Lawson data. The guide does not provide a complete reference for every utility or process. For more information on tools and processes, consider the following resources:

- The “Product Line Maintenance” chapter in the *Lawson Administration: Server Setup and Maintenance* guide provides instructions on copying a full product line, copying jobs, and related tasks such as copying a product line's programs and compiling associated source code.
- The Lawson Knowledge Base, on http://support.lawson.com, offers a wealth of support notes and other information related to some of these tasks.

Dumping and Loading Data Files

The intended use for the following utilities is to dump the database from one Environment and load it into another:

- Dump Database File (**dbdump**)

  This utility dumps database files to standard output; or you can save the data to a file for import.

- Load Database file (**dbload**)

  This utility loads the database from a file created by **dbdump**.

Important: To have the system automatically dump and load the attachment files, use the **expsysdb** and **impexp** utilities. For instructions, see the *Lawson Administration: Server Setup and Maintenance* guide.

Considerations

- Do not use the **dbload** utility to import data into a Lawson database from a non-Lawson application system.
- Unless you use the **`dbdump -d`** option, the utility truncates fields larger than 80 columns (or 132 columns, with the **`-p`** option). To dump files containing fields with more than 80 columns, use **`dbdump -d`** or **`rngdbdump`** with either the **`-d`** or **`-c`** option.
- If your data files have attachments, dump both the header and detail attachment files. They reside in the same data area as the data to which they are attached, and they are named as follows:

  `L_H`*`Prefix`* **`L_D`***`Prefix`*

  where **`Prefix`** is the prefix of the data file the attachments are related to (for example **`L_HEMP`**, if the file name is EMPLOYEE), **`L_H`** indicates a header attachment file (the first 512 bytes of the attachment), and **`L_D`** indicates the detail attachment file (the remaining bytes, up to 32,000, of the attachment).

To dump from a Lawson database file to a flat file

- At a command prompt, type:

  dbdump [-dp] [-i *index***]** *dataarea***|***dataID filename* **[>** *flatfile***]**

  where the following program options apply.

Program Option	Description
-d	Dump to a flat file. Use with the **dbload** utility. You can also use this option to dump files containing fields larger than 80 columns.
-p	Printer format (132 columns). Writes the output in 132-column format. The default is to write the output in 80-column format for online display.
-i	Use an index. This option dumps the database using an index sequence. The index name must follow **-i** (for example, **-iGLMSET1**).

To load data from a dbdump-formatted flat file into a Lawson database file

- At a command prompt, type:

  dbload [-anrf] *dataarea***|***dataID filename* **[***flatfilename***]**

  where the following program options apply.

Program Option	Description
-a	Overwrite all duplicate records and replace existing database records with duplicate records in the flat file. If the flat file has multiple occurrences of a record, the last occurrence is stored in the database.
-n	Do not overwrite duplicate records. It does not replace existing database records with duplicate records from the flat file and ignores duplicate records in the flat file.
-r	Report on duplicate records that print duplicate records between the flat file and the database. It does not replace the existing database records.
-f	Do not check for existing records. This option is required for SQL databases. It writes whatever records exist in the flat file, completely overwriting the records in the database.

Copying, Updating, and Replicating Data

Using several Lawson utilities (**dbcopy**, **dbupdate**, and **cmpdict**), you can effectively copy, upgrade, or replicate the data in one data area (the *SourceDataArea*) to another (the *DestinationDataArea*).

The following examples illustrate the potential use of these utilities:

- Compare dictionaries

  Compare the source data structures with those in the destination using **cmpdict**. Load the output into a spreadsheet to distribute and analyze before taking action on the data.

- Deploy from a test environment to production

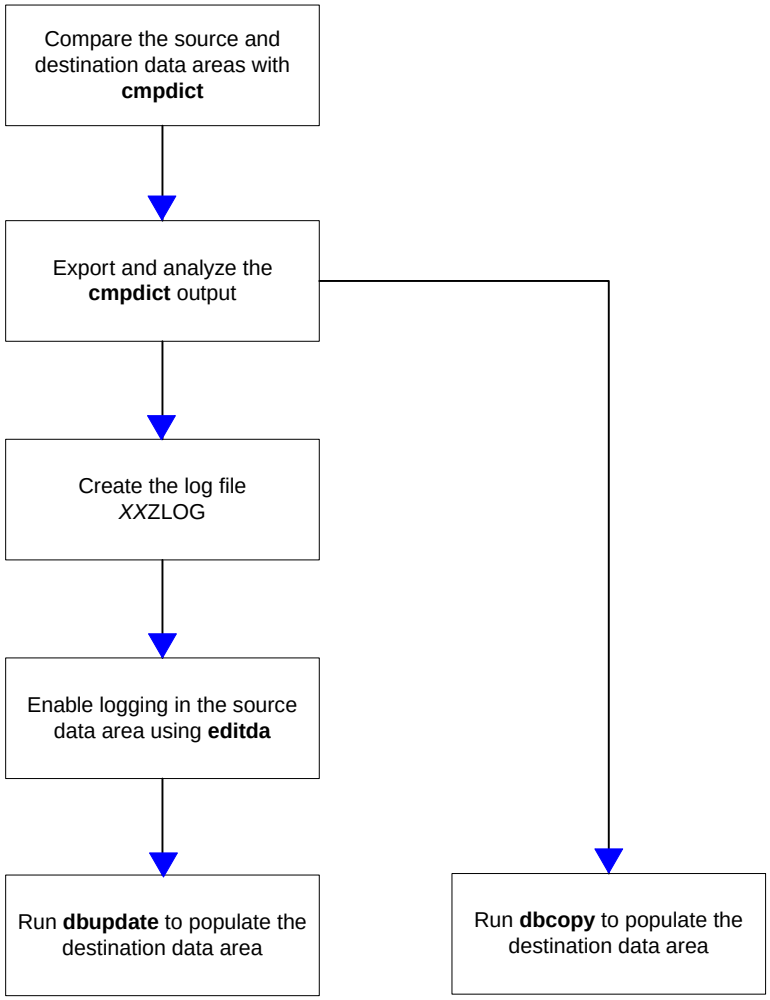
  If you use live input data (for example, Employee Self-Service applications) to test a new feature or release of Lawson software, use the **dbcopy** utility to update the production data area with data from the test data area.

- Upgrade data

  Some data tables are updated on a regular basis to other systems. The **dbupdate** utility reads an update log to update tables in the destination data area with only data that has changed. Rather than drop the destination tables and copy the entire set of history tables from source to destination, **dbupdate** is a relatively fast alternative.

The following illustration shows how you might use these utilities together to accomplish these goals.

Figure 9. Process flow: Copy or update a data area



Comparing Lawson Data Dictionaries

Use the Dictionary Comparison utility (**cmpdict**) to compare the data structures of two product line or data area dictionaries. This utility can help identify table, field, index, condition, and relationship definition changes made between release levels of an application, product line, or data area.

The utility reports on the types of changes outlined in the following table.

Database object	Changes reported
File	• System code • File name • Action • View • Database type • Data Location • Index Location • Initial record count • Record count increment value
Index	• System code • File name • Action • Name • Unique
File field	• System code • File name • Action • Position • Reference field name • Field name • Occurrence • Field type • Precision

Database object	Changes reported
Index field	• System code • Index name • Action • Position • Reference field name • Field name • Direction
Index condition	• System code • Index name • Action • Condition name
Database file condition	• System code • File name • Action • Condition name
Database file relationship	• System code • File name • Action • Relation name

To compare dictionaries

1 At a command prompt, type

`cmpdict [-d]` *`SourceDataArea DestinationDataArea`*

where

Parameter	Description
`-d`	Produce detailed output.

Parameter	Description
SourceDataArea	The first data area that you want to compare.
DestinationDataArea	The second data area that you want to compare.

Tip: The output of a **cmpdict** command can be redirected to an ASCII file and then imported into a spreadsheet.

2 Review the output. As necessary, refer to "cmpdict Output" on page 92.

cmpdict Output

The output of a **cmpdict** command can be redirected to an ASCII file and then imported into a spreadsheet. The format of the **cmpdict** command output depends on which type of database object is being compared. See these sections for more details:

- "File (FILE)" on page 92
- "File Field (FILEFIELD)" on page 94
- "File Condition (CONDITION)" on page 95
- "File Relationship (RELATION)" on page 96
- "Index (INDEX)" on page 97
- "Index Field (INDEXFIELD)" on page 98
- "Index Condition (INDEXCOND)" on page 99

File (FILE)

The following example shows the columns written by **cmpdict** when the database object being compared is a file in the database.

Note: The first row shown in this example is not written by the utility; it is provided for the sake of explaining the output more clearly.

```
SysCode  FileName  Subject  Action        View    DbType  DatLoc  IdxLoc  InitRecs  IncRecs
AR       ARISGBLK  FILE     DELETE
SY       DATECOLS  FILE     NEW
AR       ARISGBLK  FILE     CHANGE        ISVIEW
HR       L_HEMP    FILE     NONE
HR       EMPLOYEE  FILE     DBTYPE-CHANGE         ORACLE
```

```
HR      EMPLOYEE  FILE    LOCATION-CHANGE             datloc  idxloc
HR      EMPLOYEE  FILE    SIZE-CHANGE                         initRecs  incRecs
```

Column	Description	Explanation
1	SystemCode	The system code in which the file exists.
2	FileName	The name of the file being compared between data areas.
3	Subject	The type of database object being compared. In this case, a database file (table).
4	Action	The type of difference in the database object between the source and destination data areas: • NEW The database object is new in destination data area. • DELETE The database object exists in the source data area, but not in the destination data area. • CHANGE The database object is different in destination data area. • DBTYPE-CHANGE The database type was changed. • LOCATION-CHANGE Either the data or index storage location was changed. • SIZE-CHANGE Either the initial record count or the number of records to increment by was changed. • NONE The database object is identical in both data areas.
5	View	If the file is a view, the phrase IsView will appear.
6	Database type	The database type, as specified in Database Spaces (Database Definition).
7	Data Location	The data storage location, as specified in Database Spaces.
8	Index Location	The index storage location, as specified in Database Spaces.
9	Initial record count	The initial record count for the file, as specified in Database Definition.
10	Record count increment value	The number of records to increment the file's record count by, as specified in Database Definition.

File Field (FILEFIELD)

The following example shows the columns written by **`cmpdict`** when the database object being compared is a field in a database file.

Note: The first row shown in this example is not written by the utility; it is provided for the sake of explaining the output more clearly.

```
Sys  FileName  Subject    Action           Position  Ref-Field-Name   Name       Occ  Type           Prec  Scale
FT   FILE1     FILEFIELD  NEW               BEGIN     UNIQUE-I-D
FT   FILE1     FILEFIELD  NEW               AFTER     FIELD2          VERSION-I-D
FT   FILE2     FILEFIELD  CHANGE                      FIELD3                1   Alpha          5     0
FT   FILE2     FILEFIELD  NEW               AFTER     FIELD3          FIELD4
FT   FILE7     FILEFIELD  CHANGE                      FIELD1                1   Alpha (Varchar)  2     0
FT   FILE7     FILEFIELD  NEW               AFTER     FIELD1          VERSION-I-D
FT   FILE8     FILEFIELD  VALUELIST-CHANGE            FIELD1
```

Column	Description	Explanation
1	SystemCode	The system code in which the field exists.
2	FileName	The name of the file being compared between data areas.
3	Subject	The type of database object being compared. In this case, a field in a file.
4	Action	The type of difference in the database object between the source and destination data areas: • NEW Indicates that the database object is new in destination data area. • DELETE Indicates that the database object exists in the source data area, but not in the destination data area. • CHANGE Indicates that the database object is different in destination data area. • VALUELIST-NEW Indicates that the field's value list is new in the destination data area. • VALUELIST-CHANGE Indicates that the field's value list has been changed in the destination data area. • VALUELIST-DELETE Indicates that the field's value list has been deleted from the destination data area.

Column	Description	Explanation
5	Position	The location of the field in the file, when the field is new or the change is that the field moved within the file: • BEGIN Indicates that the field is at the beginning of the file in the destination data area. • END Indicates that the field is at the end of the file in the destination data area. • AFTER Indicates that the field follows the Ref-Field-Name field in the destination data area.
7	Ref-Field-Name	A field used to indicate the position of another field in a file.
8	Name	The name of the field being compared.
9	Occ	The number of times the field occurs.
10	Type	The data type of the field listed in column 8.
11	Prec	The precision of the field listed in column 8.
12	Scale	The scale of the field listed in column 8.

File Condition (CONDITION)

The following example shows the columns written by **`cmpdict`** when the database object being compared is a database file condition.

Note: The first row shown in this example is not written by the utility; it is provided for the sake of explaining the output more clearly.

```
SystemCode  FileName  Subject    Action  Name
FT          FILE8     CONDITION  NEW     DUMMY
```

Column	Description	Explanation
1	SystemCode	The system code in which the field exists.
2	FileName	The name of the file being compared between data areas.
3	Subject	The type of database object being compared. In this case, a condition in a file.

Column	Description	Explanation
4	Action	The type of difference in the database object between the source and destination data areas: • NEW Indicates that the database object is new in destination data area. • DELETE Indicates that the database object exists in the source data area, but not in the destination data area. • CHANGE Indicates that the database object is different in destination data area.
5	Name	The name of the condition in the file.

File Relationship (RELATION)

The following example shows the columns written by **`cmpdict`** when the database object being compared is a database file relationship.

Note: The first row shown in this example is not written by the utility; it is provided for the sake of explaining the output more clearly.

```
SystemCode  TableName  Subject   Action  Name
FT          FILE8      RELATION  CHANGE  FILE8-REL
```

Column	Description	Explanation
1	SystemCode	The system code in which the file exists.
2	FileName	The name of the file being compared between data areas.
3	Subject	The type of database object being compared. In this case, a relationship in a file.

Column	Description	Explanation
4	Action	The type of difference in the database object between the source and destination data areas: • NEW Indicates that the database object is new in destination data area. • DELETE Indicates that the database object exists in the source data area, but not in the destination data area. • CHANGE Indicates that the database object is different in destination data area.
5	Name	The name of the relationship.

Index (INDEX)

The following example shows the columns written by **`cmpdict`** when the database object being compared is an index.

Note: The first row shown in this example is not written by the utility; it is provided for the sake of explaining the output more clearly.

```
SysCode  FileName    Subject  Action   Name      Unique
AR       AROITEMS    INDEX    DELETE   AROSET10
PR       TIMERECORD  INDEX    NEW      TRDSET10
PR       TIMERECORD  INDEX    CHANGE   TRDSET11
PR       TIMERECORD  INDEX    PRIMARY  TRDSET12
```

Column	Description	Explanation
1	SystemCode	The system code in which the file exists.
2	FileName	The name of the file being compared between data areas.
3	Subject	The type of database object being compared. In this case, an index in a file.

Column	Description	Explanation
4	Action	The type of difference in the database object between the source and destination data areas: • NEW Indicates that the database object is new in destination data area. • DELETE Indicates that the database object exists in the source data area, but not in the destination data area. • CHANGE Indicates that the database object is different in destination data area. • PRIMARY Indicates that the database object became the primary index in the destination data area.
5	Unique	Whether the index is unique or not.

Index Field (INDEXFIELD)

The following example shows the columns written by **`cmpdict`** when the database object being compared is an index field.

Note: The first row shown in this example is not written by the utility; it is provided for the sake of explaining the output more clearly.

```
SysCode  IdxName  Subject     Action  Position  Ref-Field-Name  Name      Direction
PR       TRDSET6  INDEXFIELD  MOVE    AFTER     PROCESS-GRP     TIME-SEQ
AR       AROSET1  INDEXFIELD  NEW     AFTER     TRANS-TYPE      INVOICE
```

Column	Description	Explanation
1	SysCode	The system code in which the index field exists.
2	IdxName	The name of the index being compared between data areas.
3	Subject	The type of database object being compared. In this case, a field in an index.

Column	Description	Explanation
4	Action	The type of difference in the database object between the source and destination data areas: • NEW Indicates that the database object is new in destination data area. • DELETE Indicates that the database object exists in the source data area, but not in the destination data area. • CHANGE Indicates that the database object is different in destination data area.
5	Position	The location of the field in the file, when the field is new or the change is that the field moved within the file: • BEGIN Indicates that the field is at the beginning of the file in the destination data area. • END Indicates that the field is at the end of the file in the destination data area. • AFTER Indicates that the field follows the Ref-Field-Name field in the destination data area.
7	Ref-Field-Name	A field used to indicate another field in the index.
8	Name	The name of the index field being compared.
9	Direction	Whether the index field has changed from ascending to descending, or vice versa.

Index Condition (INDEXCOND)

The following example shows the columns written by **`cmpdict`** when the database object being compared is an index condition.

Note: The first row shown in this example is not written by the utility; it is provided for the sake of explaining the output more clearly.

```
SystemCode  IndexName  Subject    Action  Name
PR          TRDSET4    INDEXCOND  CHANGE  SOMECONDNAME
```

Column	Description	Explanation
1	SystemCode	The system code in which the index condition exists.
2	IndexName	The name of the index where the condition exists.
3	Subject	The type of database object being compared. In this case, a condition in an index.
4	Action	The type of difference in the database object between the source and destination data areas: • NEW Indicates that the database object is new in destination data area. • DELETE Indicates that the database object exists in the source data area, but not in the destination data area. • CHANGE Indicates that the database object is different in destination data area.
5	Name	The name of the condition being compared.

Copying Lawson Data with dbcopy

Use the **dbcopy** command to copy data from one data area (the *SourceDataArea*) to another (the *DestinationDataArea*).

To copy data

- At a command prompt, type:

  dbcopy [-cdpfiRt] [-I *FieldDataFile***] [-S** *SourceDirectory***] [-D** *DestinationDirectory***]** *SourceDataArea DestinationDataArea* **[***FileName***|***SystemCode...***]**

  where the following program options may be used.

Program Option	Description
c	Convert data where the two files do not exactly match. Columns in both files are copied and columns only in the destination file are set to the default value. If the files are the same, no conversion is done.
d	Delete the file and indexes from the destination data area database (if they exist), create the file in the database, copy the data, and then create the indexes after copying.
p	Precreate indexes before copying. Only use with **-d**.
f	Fix records while copying.
i	Ignore errors while copying.
R	Reject records with bad keys while copying.
t	Truncate the file in the destination data area database prior to copying the data.
I *FieldDataFile*	Read data to initialize destination file fields. The format for the data in *FieldDataFile* is *FileName FieldName FieldOccurrence FieldValue*. The data in *FieldDataFile* may be space, tab, or comma-delimited. Each entry (row) is delimited by a newline character. If *FieldName* in *FieldDataFile* exists in both source and destination data areas, then the corresponding *FieldValue* overrides the value read in from the source database.
FileName	The file in the destination data area that this row contains a default value for.
FieldName	The name of a field in *FileName* that this row contains a default value for.
FieldOccurrence	The field's occurrence. Set this option to **1** (one) or higher.
FieldValue	The default value you want to populate *FieldName*.
S	Read source data from files in the *SourceDirectory* directory.
D	Write destination data to files in the *DestinationDirectory* directory.
V	Print the version of **dbcopy**.

Configuring Database Change Logs for dbupdate

It is possible to configure your system so that any change to files in a data area (*SourceDataArea*) are logged in a file. This file can be used by the **dbupdate** utility to upgrade a new data area (*DestinationDataArea*) to match the source data area.

When you enable logging for a file, then insert, update, and delete SQL triggers are created by the database driver. Any time a trigger is fired, a row is added or replaced in a log file that contains the following fields.

Field	Description
LogFileName	A 10-character alphanumeric field that contains the name of the file being changed.
PrimaryKey	A 199-character alphanumeric field that contains the primary key values of the row in the file being changed. **Note:** Changes to files with primary keys longer than 200 bytes cannot be logged.

Since the logging is performed using triggers, all log files and all files being logged must be of the same database type.

Enabling database change logging

Note: For more information, see "Edit Data Area Utility (editda)" on page 59.

1 Create a directives file with the following content:

```
D SourceDataArea
F FileName LogChanges=N|Y|U|A
```

where the following parameters apply.

This parameter	Indicates
SourceDataArea	The data area that you want to log changes for.
FileName	The file in **SourceDataArea** that you want to log changes for.

LogChanges=N (No logging) — or — **LogChanges=Y** (Per file logging) — or — **LogChanges=U** (Upgrade logging) — or — **LogChanges=A** (Both Resource Management and Upgrade logging)	Logging is disabled when **N** is entered for the LogChanges parameter. Logging is enabled when **Y, U**, or **A** is entered for the LogChanges parameter. • **Y** indicates that logging triggers are activated for the file in question. It is written to the file RMLOG (defined under the LA system code). • **U** indicates that logging triggers are activated for the file in question. It is written to the UGZLOG file, defined under the UG system code. • **A** indicates that logging triggers are activated for the file in question. It is written to the RMLOG and UGZLOG files.

2 Run the **editda** utility. Type:

```
editda -c SourceDataArea DirectivesFile
```

3 Rebuild the dictionary. Type:

```
blddbdict -p SourceDataArea
```

4 Reorganize the database. Type:

```
dbreorg -p SourceDataArea
```

Disable database change logging

Note: For more information, see "Edit Data Area Utility (editda)" on page 59.

1 Create a directives file with the following content:

```
D SourceDataArea
F FileName LogChanges=N
```

where the following parameters apply.

This parameter	Indicates
SourceDataArea	The data area that you want to log changes for.
FileName	The file in **SourceDataArea** that you want to log changes for.
LogChanges=N	Logging is disabled for the specified file.

2 Run the **editda** utility. Type:

editda -c *SourceDataArea DirectivesFile*

3 Rebuild the dictionary. Type:

blddbdict -p *SourceDataArea*

4 Reorganize the database. Type:

dbreorg -p *SourceDataArea*

Updating Lawson Data Between Data Areas with dbupdate

The **dbupdate** command retrieves data records from the source data area file using the saved primary key in the log file. It then converts, inserts, or updates records in the destination data area file to match the source data area file. If a record exists in the destination data area but is not found in the source data area, the record is deleted from the destination data area.

For **dbupdate** to process a file, the number, names, and order of fields in the primary keys for the specified files in the source and destination data areas must match. Field types and sizes do not need to be identical, as long as the field values can be converted.

To update data

- At a command prompt, type:

  dbupdate [-fiR] [-I *FieldDataFile] SourceDataArea DestinationDataArea LogFile file ...*

  where the following program options may be used.

Program Option	Description
-f	If needed, fix records while copying.
-i	Ignore errors while copying.
-R	Reject records with bad keys while copying.

Program Option	Description
-I *FieldDataFile*	Read in data to initialize specified fields in the destination file. Each row represents one field to be initialized with the value specified. The format for the data in *FieldDataFile* is: *FileName FieldName FieldOccurrence FieldValue* The data in *FieldDataFile* may be space, tab, or comma-delimited. Each entry (row) is delimited by a new line character. **Note:** If *FieldName* in *FieldDataFile* exists in both source and destination data areas, then the corresponding *FieldValue* overrides the value read in from the source database.
FileName	The file in the destination data area that this row contains a default value for.
FieldName	The name of a field in *FileName* that this row contains a default value for.
FieldOccurrence	The number of times *FieldName* occurs. Set this option to **1** (one) or higher.
FieldValue	The default value you want to populate *FieldName*. The value must be conform to the standard format for each field data type. Valid Lawson data types are described in the *Lawson Doc for Developers: Application Development Workbench*.
SourceDataArea	The data area containing the file whose values you want to use to update the file in the destination data area.
DestinationDataArea	The data area containing the file that you want to update.
LogFile	The file that recorded changes to the database. For more information, see "Configuring Database Change Logs for dbupdate" on page 102.
file ...	The name(s) of the file(s) that you want to update.
-V	Prints the version of **dbupdate**.

Copying Data Using SQL Code

The **sqldbcopy** utility executes a SQL code that upgrades one database file from one data area to another. The utility converts data for columns with different data types between source and destination, minimum and maximum date and time values, and variant subset index switch column values.

The SQL code can enhance the performance of database copy tasks nder the conditions described in this documentation.

The utility is designed to determine as much of the required database connection information as it can, based on the source and target data areas. The generated SQL code implements a single INSERT INTO/SELECT FROM statement. This engages the database engine to perform the complete copy of the data within the engine itself without sending data across DB Connections.

The generated SQL code is sent to the database engine using JDBC to upgrade a database file from one data area to another. These data areas must exist within the same database type.

Lawson files are categorized based on their data locations or tablespaces, as defined in the destination data area's dbspaces in dbdef. Lawson files in a data location will be handled by one thread. If files are categorized into 10 data locations, for example, 10 threads process 10 sets of files, that is, one thread handle per set. The number of simultaneous threads that can be run depends on the DEGREEOFPARALLELISM setting in the properties file. Moreover, indexes are created in parallel for each processed Lawson file.

Before you start Set the environment variable IBMJDBC to the directory path where the DB2 JDBC driver is located.

`%IBMJDBC%/db2java.zip`

To copy data using a generated SQL code

- At a command prompt, type:

  `sqldbcopy [-H] -P` *`propertiesFile`* **`[-D`** *`directivesFile`***`]`**

  Consider the following options:

Program Option	Description
`-H`	Print detailed usage.
`-P`	Input file name and path for setting properties.
`-D`	Directives file defining which Lawson files or data locations to include or exclude in the data copy

Properties File

Consider the following options:

Attribute	Description
SRCSCHEMA	This attribute is the source schema. It overrides the value set in the cap file when OVERRIDE is TRUE.
DSTSCHEMA	This attribute is the destination schema. It overrides the value set in the cap file when OVERRIDE is TRUE.
SRCDATAAREA	This attribute is the source data area.
DSTDATAAREA	This attribute is the destination data area.
DBTYPE	This attribute is the database type: msf, msf2008, ora11, or ibm.
LOGGINGDIRECTORY	This attribute is the directory name and path where message and log files are written to. For each Lawson file that failed in the copy, warnings and errors are logged in a message file LawsonFile.msg. For each run, results are logged in sqldbcopy.log. timestamp.
DIRECTIVESFILE	This attribute is the input file name and path. This value is overridden when the –D option is specified at the command line.
FIELDDATFILE	This attribute is the input file name and path for initializing destination file fields.
DEGREEOFPARALLELISM	This attribute refers to the degree of parallelism.
RECREATETABLE	This attribute is set to TRUE if you want to drop and recreate the table before the copy. The default value is FALSE. If the value is set to TRUE, indexes are created only at the end of the copying process.
DONOTEXECUTE	This attribute is for not executing in the database. This attribute can have a value of either TRUE or FALSE.
OVERRIDE	This attribute allows the values for srcschema and dstschema to override the actual schema values in the source and destination database configuration cap files. it can have a value of either true or false.

Attribute	Description
TRACE	This attribute prints SQL commands to LOGGINGDIRECTORY/sqldbcopy.sql. This attribute can have a value of TRUE or FALSE. The default value is FALSE.

Important: Property values that include back slashes must be escaped with a back slash. For example, **`C:/This/is/an/example`** should be **`C://This//is//an//example`**.

Directives File

This attribute is an input file name and path of directives defining which Lawson files or data locations should be copied. The directives or entries in the input file are defined as follows:

```
A [AllFiles] <Y | N>
D <Include | Exclude> DatLocation
F <Include | Exclude> LawsonFileName
```

Important: The A directive is mandatory and specifies whether all Lawson files in the data area are to be processed.

Example: Process all files

The directives input file has the following entries:

```
A Y
```

Example: Process all files except PRDISTRIB and PREMPLOYEE

The directives input file has the following entries:

```
A Y
F exclude prdistrib
F exclude premployee
```

Populating the Directives File

When **sqldbcopy**is run, one or more tables might fail to process. Rather than manually determining the tables that have not completed processing, you can run the **populatedirectives**utility before rerunning **sqldbcopy**. The **populatedirectives**utility uses the log file and the old directives file to populate a new directives file that contains the tables that must complete processing. The new directives file can then be referenced when you rerun **sqldbcopy**.

To populate the directives file

- At a command prompt, type:

  `populatedirectives` *`dataarea logFile oldDirectivesFile`*

  Consider the following options:

Program Option	Description
`dataarea`	Data area to process
`logFile`	Log file that resulted from previously running sqldbcopy
`oldDirectivesFile`	Directives file previously used by sqldbcopy

Implementing Lawson Security

6

This chapter contains the following concepts and procedures:

What Is the Build Security Utility?

The **bldibmsec** utility will duplicate Lawson table level security in the IBM DB2 database.

Security constraints and privileges placed on database tables are enforced when users attempt to use Lawson applications to access those tables. When non-Lawson applications or tools access Lawson data stored in the IBM DB2 database, they do not encounter any Lawson security restrictions or privileges.

When you make changes to any of your table-level security constraints, run **bldibmsec** to implement the changes in the IBM DB2 database. When you drop and recreate any tables either by using the utility or by reorganizing the Lawson database, you also drop all security constraints on those tables in the IBM DB2 database. Run **bldibmsec** again to reinsert your table-level constraints.

For more general information on security planning and implementation, see

- Administration guides provided by your database vendor
- *Lawson Security and Resource Management Administration Guide*

Using the Utility with Lawson Security and Resource Management

In Lawson Security, data level security and user information may be stored in LDAP. The **bldibmsec** utility uses Lawson Security APIs to retrieve security data from LDAP, based on security rules and the specified database service. Grant and revoke SQL statements are generated from this information.

Database drivers use database services to connect to the RDBMS. If your system uses Lawson Security, you must specify a database service for this utility using the *DatabaseService* parameter. RMIds and their permissions on tables in a product line are determined, and access to the database service is validated. If the RMId has access to the service, **bldibmsec** resolves the database login and generates the grant or revoke SQL statement.

Note: The **checkLS** flag must be set to ON for a user (RMid) in order for grant or revoke SQL statements to be generated for that user.

Important: The **bldibmsec** utility only resolves designated database logins for RMIds. It does not retrieve privileged database logins. This means that the service specified by the *DatabaseServices* argument must have a USE_USER_ID or USE_USER_AND_PRIVILEGED_ID LoginProcedure assigned to it.

Database Time Stamp Considerations

The **bldibmsec** utility checks the table version stamp, with the following results:

- If the table version stamp is missing, the utility returns an error.
- If the utility finds the table version stamp, it checks the Lawson repository to see which security classes have access to which tables in the given data area.
- If a security class has full access to a table, the utility grants the matching IBM DB2 role permission to select, insert, update, and delete rows in the table.
- If a security class has read-only access to a table, the utility grants the matching IBM DB2 role permission only to select rows in the table.
- If a security class has no access to a table, it revokes all permissions on the table from the matching IBM DB2 role.

Implementing Lawson Security Constraints in the Database

Use this procedure to duplicate Lawson table level security in the IBM DB2 database.

When you make changes to any of your Lawson table-level security constraints, run **bldibmsec** to implement the changes in the IBM DB2 database.

When you drop and recreate any tables either by using the **bldibmsec** utility or by reorganizing the database, you also drop all security constraints on those tables in the IBM DB2 database. Run **bldibmsec** again to reinsert your table-level constraints.

To implement security constraints in the database

- At a command prompt, type

  bldibmsec [-UDGVq] [-u *DbLoginName***] [-p** *Password***]** *DataArea DatabaseService*

  Consider the following options:

Option	Description
-u *DbLoginName*	The database user login.
-p *Password*	The password for the database user.
-U	Update mode. Issue query directly to the database.
-D	Drop-only processing; revoke all permissions from all Lawson tables for all Lawson users. Once Lawson table-level security exists in the IBM DB2 database, generate SQL statements to remove it by running the **bldibmsec** utility with the drop (**-D**) flag.
DataArea	The data area that you want to apply security to.
DatabaseService	Lawson Security only. The name of the database service defined for your system. For each RMId, **bldibmsec** validates its access to the *DatabaseService*. If the RMId has access to the service, **bldibmsec** resolves the RMId's database login and generates the grant/revoke SQL statement.
V	Print the version.
-q	Do not print the commands to **stdout**.

Example: SQL Output File (Lawson Security and Resource Management)

For this example, Lawson users and database services are configured as shown in the following table:

User/service	Configuration	
User1	RMId	**rmidUser1**
User2	RMId	**rmidUser2**
DatabaseService1	Name	**use_priv_svc**
	LoginProcedure	**USE_PRIVILEGED_ID**
	Privileged database login	**lawson**
DatabaseService2	Name	**use_user_svc**
	LoginProcedure	**USE_USER_ID**
	rmidUser1 database login	**dbuser1**

User/service	Configuration	
	rmidUser2 database login	**dbuser2**

With this configuration, the following command produces no results (no SQL is generated):

bldibmsec *productline* **use_priv_svc**

bldora11sec *productline* **use_priv_svc**

However, the following syntax will result in generated SQL code:

bldibmsec *productline* **use_user_svc**

bldora11sec *productline* **use_user_svc**

The resulting SQL grants select, delete, and update on employee to dbuser1, select, delete, update on employee to dbuser2, etc.

Error Processing

If an error occurs that the **bldibmsec** utility cannot handle, it stops immediately. The commands that have executed up until that point remain in effect, but no further commands are executed. To revoke the permissions granted up to that point, run the utility again with the same set of options plus the **-D** option. For example, if when the error occurred you were running the following:

bldibmsec -U *dataarea*

then run:

bldibmsec -DU *dataarea*

Improving Performance

7

This chapter contains the following concepts and procedures:

Estimating Physical Disk Usage

This section contains the following topics:

What Is the Disk Usage Utility?

Note: The **ibmdu** utility generates an error if the table version stamp is missing.

When you initially implement Lawson applications with an IBM DB2 database, one of the most challenging tasks is calculating the necessary disk space. The IBM DB2 Disk Usage (**ibmdu**) utility can help you calculate disk usage by providing an estimate (plus or minus ten percent) of the amount of disk space required for the table and index data in a given product line. Use this utility during initial setup or when migrating from another database.

Define the product line in **dbdef** and create the dictionary before you run **ibmdu**. If you make changes to the file or index definitions, rebuild the dictionary before you run the utility. The **ibmdu** utility may be run against the current, new, or old dictionary files.

The **ibmdu** utility uses the information found in the Lawson dictionary and both **law_dba_table** and **law_dba_index** to estimate the file and index object sizes. The utility calculates the space needed for an initial number of records by gathering information from **dbdef** or from the **INITIAL_EXTENT** parameters defined in **law_dba_table**.

Two modes of operation are available—online and offline. In either mode, you must tell the utility what block size to use. Specify the block size by choosing the **-b***x* option, with *x* representing the block size variable. For example, **ibmdu -ob8** *dataarea* runs offline with a block size of 8K.

Using the Disk Usage Utility in Online Mode

If you select the online (default) mode, all other conditions for usage must have been met and IBM DB2 must be installed and running before you use the utility. If you are running **ibmdu** on the active dictionary, the product line must currently be in IBM DB2. If you are running **ibmdu** on either a new or old dictionary, the product line must have been in IBM DB2 when that dictionary was built. When running in online mode, the utility queries the IBM DB2 database for information on the existence and size of database spaces and other parameters necessary for determining required disk space. When run in online mode, **ibmdu** also checks the IBM DB2 database to determine how much space you have available in your tablespaces.

ibmdu -[c][w]b*x dataarea* **[***tablespace...***]**

In online mode, the **ibmdu** utility run has the following options:

-c	Show information about the existing IBM DB2 tablespaces.
-w	Show detailed warning messages for tables and indexes.
-b*x*	Use a block size of **x**K. This must be the last specified option in the command line.

Using the Disk Usage Utility in Offline Mode

Note: You do not need to install IBM DB2 to use **ibmdu** in offline mode.

If you select the **-o** option, **ibmdu** operates in offline mode. The utility assigns default values to the parameters required to make the estimate. In offline mode, **ibmdu** reads the Lawson dictionary to find the initial and incremental record counts for each table and index listed in the product line's dictionary. From the dictionary, **ibmdu** reads in the size of a row in each table and index. After it has the record counts and row sizes, it uses the same formulas as **bldibmddl** to estimate the actual physical size required to hold the extents for Lawson tables.

Keep in mind the following when you use the **ibmdu** utility in offline mode:

- The accuracy of the **ibmdu** estimate depends entirely on the accuracy of the row count estimate placed in File Size Definition.
- The row counts in the dictionary are set by the File Size Definition screen in **dbdef**, followed by **blddbdict**. You can change these values (as well as tablespaces) and rerun **ibmdu** without rebuilding the dictionary. To get good estimates, set these values according to the row counts determined by your analysts.
- The estimates are indeed estimates, not exact calculations. You can generally expect them to be off by less than ten percent in either direction, but there are cases where larger margins can be encountered. These cases are indicated by a set of parentheses in the output, and are summarized if you use the warnings (**-w**) option.

  ibmdu -o[w]b*x dataarea* **[***tablespace...***]**

The **ibmdu** utility run in offline mode has the following options:

-o	Run offline.
-s	Show tablespace summary.
-n*x*	Use record count of *x*.
-w	Show detailed warning messages for tables and indexes.
-b*x***K**	Use a block size of *x***K**. This must be the last specified option in the command line.

Estimating Space Requirements Using the Disk Usage Utility

This procedure will help you determine the space requirements for a table in your product line.

Need More Details? Check out the following concepts:

- "What Is the Disk Usage Utility?" on page 114
- "Using the Disk Usage Utility in Online Mode" on page 115
- "Using the Disk Usage Utility in Offline Mode" on page 115

To determine space requirements

1 Run the **ibmdu** utility in either online or offline mode to determine your current space requirements.

Note: The tables you change will depend on your site requirements.

2 Change the values of the initial number of records for your tables by using the File Size Definition form in **dbdef**.

3 Run the **blddbdict** utility to generate a new +dictionary.

   Tip: Running **ibmdu** in online mode (rather than offline mode) here will be more accurate because it uses the actual extent and page sizes instead of the default values.

4 Run the **ibmdu** utility again using the +dictionary as input to determine your new space requirements.

5 Repeat steps 2-4 until your sizing is complete.

6 Type the following to implement your file size changes. At the command, type

   dbreorg -g [*filename...***]**

What is the Build Lawson Tables Script?

The **%GENDIR%/ibm/build_law_strings.sql** script creates the law_dba_strings table.

What is the law_dba_strings table?

The law_dba_strings table is used to create table partitions when tables are created. This table is used for adding SQL strings to SQL creation commands that are used by Lawson.

Important: You must have sufficient understanding of DDL syntax when using the LAW_DBA_STRINGS table.

The following example shows the syntax of the **law_dba_strings** table:

```
create table law_dba_strings (
object_name varchar (128) not null,
object_type varchar(128) not null,
string_type varchar(128) not null,
position int not null,
string_col varchr(2000)) not null);

create unique index law_dba_strset1
on law_dba_strings (object_name,
object_type, string_type, position);
```

The following table provides descriptions of the options in the **law_dba_strings** table:

Column	Description
object_name	This column must contain the name of a user table, index, or tablespace in all uppercase letters with no qualifications. Examples: **GLMSET1**(valid) **glmset1** (invalid) **LAWPROD.GLMSET1**(invalid)
object_type	This column must contain the object type of the data in the object_name column. This column can have a value of either TABLE or INDEX in all uppercase letters.
string_type	This column must contain the name of the type of string to be added to the object . The following are the valid values for this column. These values must be typed in uppercase letters: OPTIONS - Data in the string_col column is added to the end of the DDL for the object_name or object_type. This value is valid for TABLE or INDEX object types.
position	Use this column if the data in the string_col is too long to fit in one record. When multiple identical records are created with different position values, the string_col values are concatenated and used.
string_col	If the string_type value is OPTIONS, this column contains the DDL string to be added to the end of the create DDL for the object_name/object_type.

Example: Adding data to the law_dba_strings table

```
insert into law_dba_strings
values (‘GLMASTER’, ‘TABLE’,
‘OPTIONS’, 1, ‘compress yes’)
```

Analyzing Performance

This section contains the following topics:

What Are Timed Statistics?

The timed statistics feature gives you information on the amount of processing time used for each function performed on a specific table for each user and program combination. This information is useful when you want to understand performance tuning issues.

You can choose the timed statistics feature to monitor statistics for specific user, specific program, or specific combinations of user and program. You can also set this feature to monitor all users and programs, but if there are many users accessing the database, timed statistics might produce a large amount of output.

Each time the statistics are generated, the output goes to a directory specified by the DIR parameter of the database server configuration file (**ladb.cfg**) or set by the **dbadmin** utility. You can print or view the output files using a text editor. The output file has the name:

/tmp/*programname.dataarea****.stats.****pid.yyyyyy*

where *yyyyyy* uniquely identifies the statistics session.

The summary section lists a count of the total number of times a function was called, as well as the total amount of time in milliseconds (thousandths of a second) used by the calls to the function. A breakdown by table follows the summary. This breakdown shows the number of calls and amount of time for each function that accesses a specific table. It also shows a listing of calls to any indexes on the table.

The summary information is useful for identifying user actions. For example, if two users are running the same program and one user complains about slow performance while the other does not, the information provided in the summary section shows which actions are being performed most frequently by each user. Checking the timed statistics for each user might show that the user who is experiencing slow performance is doing a large number of page forwards and page backwards on the form, as indicated by a high count for the FindNxt and FindPrv functions.

The table breakdowns are useful for determining which tables are being accessed most frequently by different users and programs. They are also useful for determining whether or not queries are efficiently using indexes. If a large number of calls are being made to a table without using any available indexes, you might find that the indexes or queries are not constructed very well. The report lists only the number of calls made to an index; it does not list how many of those calls successfully used the index. If you accidentally drop an index, the timed statistics output does not reflect it.

The meanings of the tracked calls are summarized in the table below.

Type of call	Meaning
nonInterface	Time in application, that is, time not in the database APIs.
BeginTrans	Begin a transaction state.
EndTrans	End transaction state and commit any inserts, updates, and deletes.
Inserts	Create a new row.
Stores	Rewrite or write a new row. A rewrite must follow a MODIFY call. Similarly, a write must follow a CREATE or RECREATE call.

Type of call	Meaning
Deletes	Delete a single row.
FullDelete	Delete a single row, following the defined rules.
FullDeleteRng	Delete a range of rows, following the defined rules.
ReadDBData	Retrieve data from a table.
Mods	Retrieve a single row and ready it for updating or deletion.
Updates	Rewrite an existing row with the values in the columns.
Inserts	Insert a single row with the values in the columns.
Counts	Retrieve a count of rows in the table.
Open Table	Open the table for data access.
Close Table	Close the table for data access.
Find	Retrieve a single row.
FindNlt	Retrieve the first row equal to or greater than the specified key(s).
FindNxt	Retrieve the row following the current row for the specified index. Performed after a successful FindNlt call.
FindPrv	Retrieve the row before the current row specified by the key(s) via a specified index.
FindBegRng	Retrieve the first row in a range specified by the key(s).
FindSubRng	Retrieve the first row in a range specified by the key(s) and having an end value specified on its last key.
DelRng	Delete a range of rows.
DelSubRng	Delete a range of rows that has an end value specified for its last key.
FilterDelRng	Delete a range of rows, using a filter.
FilterDelSubRng	Delete a range of rows that has an end value specified for its last key, using a filter.
FindNextRng	Retrieve the next row in a range. Must be preceded by a FindBegRng.
FindPrevRng	Retrieve the previous row in a range.
AggSubRng	Retrieve aggregate results.
Total Calls on Index 00	The number of API calls for a specified index.

What Is the Timed Statistics Log?

The timed statistics log tracks each generation of a timed statistics output file. You can turn logging on and off by setting the TIMESTATSLOG flag in the Database Server Configuration File.

%LAWDIR%/system/ladb.cfg

When set to TRUE (on), the timed statistics log output is sent to the file:

%LAWDIR%/system/ibmdb.log

How Can I Analyze Timed Statistics?

A script (analyze_stats.pl) is available to analyze timed statistics output. The script creates a configuration file for the tables analyzed by the report. The generated configuration file shows tables that you could configure for caching and the recommended number of records to be cached. The generated configuration file can be used as is. You could also modify the settings or use the recommendations to create a new configuration file or update an existing one. If you already have a configuration file for a table that has been analyzed through the analyze_stats.pl script, you can merge the recommendations into your existing configuration file.

After you run a process with **timestats** turned on, you can run analyze_stats.pl. The script does the following:

- Create a report of the **timestats** data with output organized in meaningful ways.

  By default, the report is sent to the screen (standard output). Send to a file if you want to be able to print the output.

  The report can be disabled with the -q option. If the -q option is used, only the configuration file is created.

- Generate a configuration file with recommendations about how caching could improve processing of the tables accessed by the program:

  /location/*productline*|*dataarea.programname***.cfg**

  where *location* is the directory where the timestats data is located, and *productline*|*dataarea* and *programname* are those specified in the **timestats** data.

  If you run the run the analyze_stats.pl script using the -d (data area) option, all programs in the data area are analyzed and a configuration for each is created. The generation of the configuration file can be disabled with the -r option. If the -r option is used, only the report is created.

Changing the Status of Timed Statistics Collection

This procedure provides instructions for turning timed statistics collection on and off. You can edit the database server configuration file (**ladb.cfg**), or you can use the **dbadmin** utility to set the TIMESTATS parameter.

Other parameters can be set to further configure timed statistics collection. For details on these parameters, see "The Lawson Database Server Configuration File (ladb.cfg)" on page 44.

To change the status of timed statistics collection using the database server configuration file

1 Use a text editor to edit the database server configuration file:

%LAWDIR%/system/ladb.cfg

2 Add or modify the line that contains the TIMESTATS variable.

Set the variable to TRUE to turn on timed statistics collection. Set it to FALSE to turn off timed statistics collection.

3 Save and exit the Lawson database server configuration file.

To change the status of timed statistics collection using the dbadmin utility

- At a command line type

dbadmin -set TIMESTATS ON

- or -

dbadmin -set TIMESTATS OFF

Configuring Record Caching

This section contains the following topics:

Record Caching

Caching records on a file-by-file basis involves these tasks:

- Update the configuration file for an individual batch program.
- Specify the number of records from a certain file to hold in memory while the program runs.

When the batch program needs access to a record from a file that has cached records, the program first checks the cache to see if the record is there. If it is, the program retrieves the record from the cache, bypassing the file on disk. If the record is not in the cache, the program reads the record from the disk and places it in the cache for future use, if there is room.

If a program adds, deletes, or updates records for a file that has cached records, changes are made in the cache, if necessary, and then in the file on disk.

Because it is faster to retrieve a record from memory (logical I/O) than disk (physical I/O), caching improves performance when used with batch programs that repeatedly read the same unchanging records. However, if you use caching with programs that perform many adds, updates, or deletes, performance is degraded by the need to change not only the records in the file but also in the cache. Therefore, when choosing files for caching, use only those files that are accessed frequently for read-only purposes on the same set of static records.

Array Fetch Buffers

Array fetching at the database level and record caching allows the database driver to make better use of existing shared memory. This reduces the number of context switches (inter-process communications) and improves performance in when Lawson 4GL database APIs are used to fetch data, or when Lawson processes such as DrillAround, **dbreorg**, or OLEDB retrieve data.

An end user experiences this feature via better throughput on bulk data retrieval operations when the data is held in a SQL database. Inter-process communications (context switches) are costly. For every 10 records retrieved, 22 context switches take place. Implementation of array fetch capability make better use of existing shared memory.

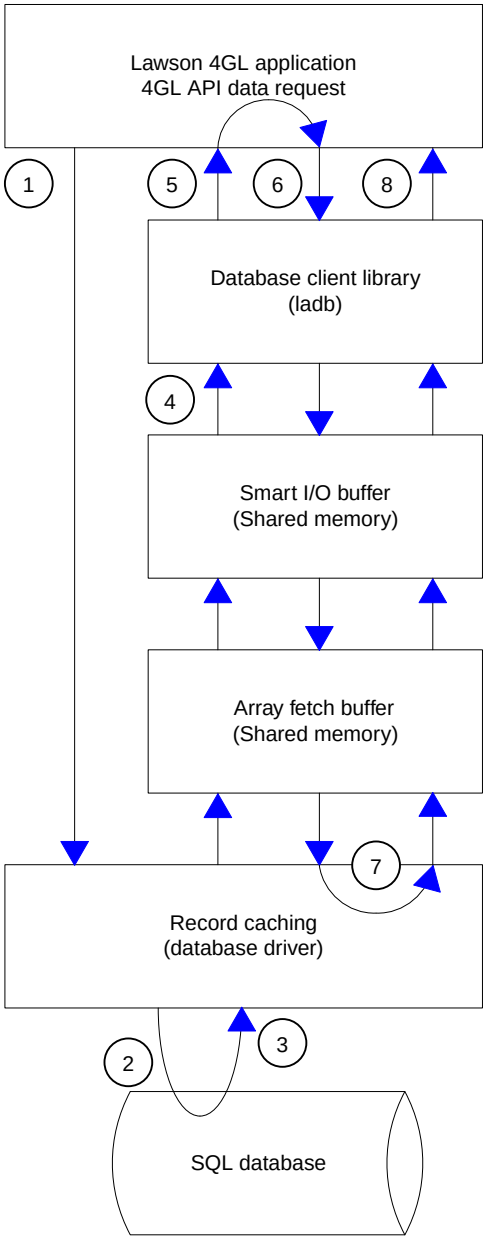
The types of programs realizing performance benefits from array fetch implementation when retrieving data are:

- Lawson 4GL programs using Lawson 4GL database APIs such as 850-FIND-BEGRNG or 870-MODIFY-PREV for set retrieval operations
- The OLEDB provider
- Lawson database utilities (for example, **dbdump**)

When **ladb** requests a set of data, the database driver (**ibmdb**) retrieves enough records to satisfy the full set API request, and caches them at the driver level. The database driver transfers as many records as possible to shared memory, depending on the size of the records being retrieved and the available shared memory. The cache created by **ladb** is cleared whenever an EndDBWork event occurs, such as a NOT FOUND reply.

The following process is initiated when the Lawson 4GL program PO20 needs to retrieve line items for a purchase order.

Figure 10. Process flow: Data retrieval with array fetching



1 PO20 sends an 850-FIND-BEGRNG API request, resulting in an in-process call to the database client API, which in turn sends a message (inter-process communication) to the database driver.

2 The database driver converts the request into a SQL SELECT statement, and sends a message (inter-process communication) to the SQL database server requesting up to 10 records.

3 The SQL database server replies with a message containing 10 records which are cached at the database driver.

4 The database driver sends the first record to the database client library.

5 The database client library gives the record to PO20 by populating working storage.

6 PO20 requests the next line item by sending an 860-FIND-NXTRNG API request, resulting in an in-process call to the database client API, which in turn sends a message (inter-process communication) to the database driver.

7 Instead of sending a message to the SQL database server, the database driver satisfies the request from the cache set up by the request in step 3, and returns one more record back to the database client library.

8 The database client library gives the next record over to PO20 by populating working storage.

The rest of the records are retrieved in the same manner. If more than 10 line items exist, the database driver cache is exhausted before a NOT FOUND is returned. This causes the next FindNextRngDBRec message to trigger another message to the SQL database server requesting up to 10 more records for its cache.

Tracing data retrieval paths with Lawson debugging tools

When debugging is enabled with the DEBUG=TRUE statement in the ladb.cfg file, the database driver logs the number of records returned for each set API request. When application debugging is enabled by the existence of the APPDEBUG file (**`%LAWDIR%/system/APPDEBUG`**), **ladb** logs whenever data request is satisfied from the cache instead of direct retrieval from the database. For instructions on enabling debugging, see the *Lawson Administration: Server Setup and Maintenance* guide.

Control Fetch and Insert Size for a Program

If you determine, through the Analyze Timed Statistics feature (analyze_stats.pl) or some other method, that a particular program is spending more time retrieving data than it should, the problem might be that the ARRAYBUFSIZE variable has been set inappropriately for the program.

The ARRAYBUFSIZE variable in the IBM database driver configuration file lets you control the number of records that are retrieved when a program performs a FindNlt, FindBegRng, or FndSubRng call to the database. The setting you make in the database driver configuration file applies to all programs in a data area. The default for this setting is 10.

However, depending on how a particular program functions, the setting in the database driver configuration file might not be appropriate. The most likely reason for this is that the program is retrieving data in a way that is not typical, and therefore the number of records to be retrieved at one time should also be different. You can create a configuration file for the program (*program*.cfg) that contains a setting for ARRAYBUFSIZE. This configuration file setting overrides the setting in the database driver configuration file for the specific program only. If a configuration file has already been created for the program, add the setting for ARRAYBUFSIZE to this file. For information about program configuration files, see *Lawson Administration: Server Setup and Maintenance*.

If you have determined that you need to change ARRAYBUFSIZE for a particular program, set the variable in the program configuration file to be the opposite of what it is in the database driver configuration file. For example, if the database driver configuration file setting is 10, trying setting it to 50 in the program configuration file.

The INSERTBUFSIZE variable can also be set for a specific program, although the situation where this might be needed is much less common. The same general guidelines apply for INSERTBUFSIZE

as for ARRAYSIZE. Use the Analyze Timed Statistics feature to determine if a program is spending an inordinate amount of time inserting rows. If it is, create (or update) a program configuration file with a setting for INSERTBUFSIZE that is the opposite of the setting in the database driver configuration file. Bear in mind that using a larger value for INSERTBUFSIZE increases the memory used for the buffer.

Example of program-specific buffer settings

The following configuration file entries create a default array buffer size of 10, with override values of 1 for GL292 and 100 for HR11:

- **%LAWDIR%/***dataarea***/IBM**

```
ARRAYBUFSIZE=10
```

- **%LAWDIR%/***dataarea***/glsrc/GL292.cfg**

```
ARRAYBUFSIZE=1
```

- **%LAWDIR%/***dataarea***/hrsrc/HR11.cfg**

```
ARRAYBUFSIZE=100
```

Caching Records for an Individual Program and Files

Caching records can improve performance when used with batch programs that repeatedly read the same unchanging records. This procedure describes how to turn caching on and off, and how to monitor caching.

To turn record caching on

1. Locate the program configuration file (*programcode*.cfg), which resides in %LAWDIR%/*productline/systemcode*/src or the %LAWDIR%/*productline*/obj directory. Create the file if necessary.
2. Add the following parameter to the file, specifying the file name and the number of records to cache:

   CACHE *FileName NumberOfRecords*

To specify all records, type **-1** for *NumberOfRecords*.

3 Save the configuration file.

Example

To turn caching on for the Accrual Code file (Apaccrcode) when running the AP200 program, make the following entry in the AP200.cfg configuration file:

```
CACHE       Apaccrcode    20
```

The number 20 in the entry specifies that up to twenty records are held in the cache for this file-program combination. Each time the program initially reads a record, it places the record in the cache until 20 records are cached. After that, the cache takes on no more records, unless a record it holds is deleted from the file or the program performs a delete range call (DELETERNG).

To turn caching off

- Remove the CACHE entry from the configuration file.

  - or -

  Set the number of records to cache to zero (0).

To monitor caching

- Add the following entries to the same configuration file that contains the CACHE entry.

  STATSFILE *FileName* **CACHESTATS TRUE**

  For *FileName*, use either a full path name or a relative path name for the output file, but do not use an environment variable. If you do not add the STATSFILE entry, the system puts the output in the job log.

  The system generates one report for each program. If a program is used more than once, each instance of the program generates a set of statistics appended to the end of the output file.

Example

The output file shows statistics for each file named in the program configuration file. The following example shows statistics generated by a program accessing the Apvenmast file.

```
Cache Statistics for file APVENMAST

   Configured to cache 600 records
   600 Records added to cache
   0 Records changed in cache
   0 Records deleted from cache
   0 Attempts to exceed cache size
   0 Cache Cleared

   Index 0 statistics
      Index was updated from the beginning
      11623 Attempted Finds
      11023 Successful Finds

   Index 1 statistics
      Unused

   Index 2 statistics
```

```
Unused
  .
  .
  .
```

The number of records to cache for this file is set to -1, meaning that all records are cached. In the example, the cache adds all 600 records from the table as they are read, and then proceeds to read from the cache rather than the file on disk. The example shows no changes being made to the records, and no records being deleted. The fact that reads are being repetitively performed on unchanging records indicates that caching is appropriate in this instance.

To turn monitoring off

- Remove the CACHESTATS entry from the configuration file.

  - or -

  Set the entry to FALSE.

Enabling Database Joins

This section contains the following topics:

Database Joins Overview

Careful implementation of database joins at the database driver level, combined with data caching strategies, can enhance the ability to quickly retrieve records from the database. The use of joins reduces the number of context switches that occur when records are retrieved from the database. Taking direct advantage of SQL database capabilities improves performance by putting more of the data retrieval work at SQL database server, instead of the application client.

Lawson 4GL database APIs have been enabled for joins at the database level, allowing any Lawson application using data retrieval APIs to be enabled for joins. This provides opportunity for performance benefits, based on your data processing needs, by implementing joins for targeted programs.

The implementation of joins within the database driver is transparent to Lawson applications—they function correctly whether a join is created or not. To use the generation of join statements by the database drivers, your database configuration must meet the criteria shown in the following list. The database driver checks for these conditions, and does not create invalid joins.

- The tables being joined must reside in the same SQL database.

- Joins are supported only for one-to-one relationships and self-joins are not allowed.

  Note: This limitation is due to the current Lawson caching implementation.

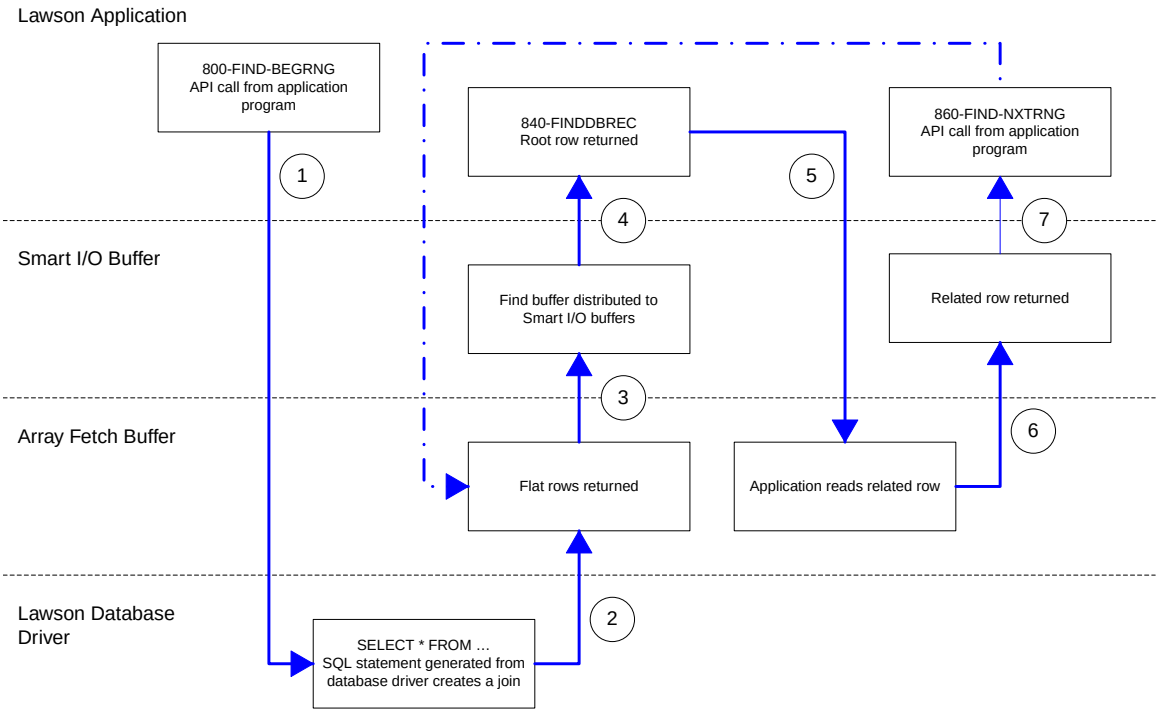
- The internal size of the shared memory buffer is the limitation of maximum join record length.
- Returned values must be less than 16K.
- A maximum of 10 relations can be evaluated in a join.
- Add a statement to a configuration file. For more information, see "Enabling Generation of Joins for an Application" on page 130.

Database Joins Process Flow

Lawson applications navigate data based on predefined database relationships within Lawson metadata. Lawson database drivers generate SQL statements using the Lawson prefix as the table aliases. Predefined metadata definitions (data relationships) are used by Lawson APIs to retrieve data, rather than relying on ad hoc queries. This allows the database driver to understand which joins are good candidates and how they should be implemented.

The database driver retrieves data from the database when an application initiates a request. The retrieved data is managed by array fetching buffers and Smart I/O caches to dramatically reduce the number of inter-process communications (context switches). The following diagram shows how the system handles the data request using joined tables.

Figure 11. Process flow: Data retrieval with database joins implemented



1. A Lawson application sends an **`800-FIND-BEGRNG`** API request that includes related records from separate tables.

   This initiates a call to the Lawson database driver (ibmdb), which in turn creates a SQL **`SELECT`** statement with a left join to obtain relevant data from the related tables.
2. The data is retrieved from the database and flat rows are returned to the array fetch buffer.
3. Data from the array fetch buffer is distributed to Smart I/O buffers.
4. The initial record sought by the application (the "root row") is returned.
5. The application reads the related row to obtain the data from the Smart I/O buffer.
6. The application requests another set of data with an **`860-FIND-NXTRNG`** API.
7. Array Fetch buffer is read and rows are distributed to Smart I/O buffers.
8. The application requests more data, and the process iterates from step 3 through step 8.

Enabling Generation of Joins for an Application

Joins are enabled by placing a statement in a program-specific configuration file.

To enable a join

1 Open the program-specific configuration file for editing:

%LAWDIR%/*productline*|*dataarea*/*xx***src/***xxnnn***.cfg**

where the following parameters apply.

Parameter	Description
productline|*dataarea*	The product line or data area that contains the program you want to enable joins for.
xxnnn	The program code, where *xx* is the two-letter system code abbreviation, and *nnn* is the numeric portion of the program code.

2 Add the following line to the file:

JOIN=*FileName.IndexName(RelationName***.0)[; (***RelationName***.0) ...]**

where the following parameters apply.

Parameter	Description
FileName	The table you want to enable joins for, as defined in dbdef.
IndexName	The index that is used to access data from the primary file (*FileName*).
RelationName	The relationship that is used to access data from the joined file as defined in dbdef.

Example

This entry enables use of joins when retrieving data from the GLSYSTEM table, using index GLSSET1 and the CHART-OF-ACCTS relation:

```
JOIN=GLSYSTEM.GLSSET1(CHART-OF-ACCTS.0)
```

The SQL generated by this enabled join is shown below:

```
select GLS.COMPANY, GLS.R_NAME, GLS.BS_ACCT, ... ,
  GLS.FB_UPDATING, GLS.ML_UPDATING, GCH.CHART_NAME, GCH.DESCRIPTION,
   ... , GCH.COMT_SUB_ACCT
from EMP8124K.GLSYSTEM GLS
LEFT JOIN EMP8124K.GLCHART GCH ON GCH.CHART_NAME = GLS.CHART_NAME
where GLS.COMPANY = 1
order by
GLS.COMPANY
```

Diagnosing Errors

A

This appendix contains the following topics:

Where Do I Look for Error Information?

There are many places from which an error can originate: the operating system, the database, an application, the Lawson environment, or the presentation layer itself. Fortunately, the IBM DB2 database has an extensive set of error checking and interpreting routines designed to help identify where the problem may originate. When the IBM DB2 database encounters such an error, it passes the error back to the **ibmdb** database driver, which either relays the message back to the screen, out to a log file, or both.

The best place to look for information regarding an error is in the log file:

%LAWDIR%/*system*/ladb.log

This file can be examined using a text editor.

Lawson also supplies a utility, named vwdblog, that opens up the ladb.log file in the Lawson editor and moves to the end of the file. You can use this utility by typing **vwdblog** at the command prompt.

How Do I Interpret the Error Information?

The ladb.log file contains useful information that might help you resolve errors. The ladb.log file includes:

- A brief error message from the IBM DB2 server.
- The last SQL statement executed.
- The API being processed.
- The row being used.
- The index keys being used.

What Must I Know Before Calling S&D?

If you have a problem for which you require assistance from the Lawson Support & Delivery (S&D), gather some information before you call. Certain pieces of information can be very helpful to the S&D staff when they attempt to diagnose or fix a problem. Along with a thorough description of the problem as it is exhibited, the main things they want to know are:

Required Information for S&D	Tips for Finding Information
What platform/OS are you on?	Type: **`uname -a`**
What IBM DB2 version and fixpack level are you running?	Type: **`db2level`**
What version of the Lawson database driver are you running?	At the command line, type: **`univver ibmdb`**
What version of Lawson applications are you running?	Record the release number associated with the application. At the command prompt, type: **`dbdef`** *`dataarea`*
What version of Environment binaries are you running?	To see a utility version, type: **`univver`** *`utilityname`* For example **`univver ladb`** You can also use any core program such as **lapm** or **lajs** instead of **ladb** to obtain the version.
What parameters do you have set in your Lawson IBM configuration file?	S&D might want a copy of this file. The file resides in: **%LAWDIR%/***dataarea***/IBM**
What error message(s) are you receiving in the ladb.log file? Are there any other messages in the log that might be related?	S&D might want a copy of the relevant portions of the log. This file resides in: **%LAWDIR%/***system***/ladb.log**
What error message(s) are you receiving in the IBM DB2 error logs?	S&D might want a copy of the relevant portions of the log.

SQL Conventions

B

This appendix contains the following topics:

Accessing Data

Warning: Lawson cannot assume any responsibility for the results (damage of data and/or structure) of improperly using non-Lawson tools. Please refer to the third-party tools documentation when attempting to access Lawson data via any non-Lawson tool.

Lawson recommends that you use the tools provided by Lawson to access Lawson data. This ensures that your data remains accurate and accessible by Lawson programs and utilities. On rare occasions, you might use non-Lawson tools to view or report on your Lawson data. Never update Lawson data using non-Lawson tools. Lawson Software does provide some instructions for accessing the database via non-Lawson tools; however, using third-party products correctly is the responsibility of the user.

Lawson data appears in three different formats, depending on the data access tool you choose:

- Using text editors to view Lawson program source files, you see your data files and fields represented with Lawson COBOL naming conventions.
- Using Lawson Environment utilities such as Database Definition (**dbdef**) or import/export tools (**importdb**), you see your data files and fields represented in the Lawson 4GL naming conventions.

The following section provides information to help you understand how Lawson has handled naming conventions between these three paradigms. For more information, see "Lawson Implementation of SQL Conventions" on page 135.

Lawson Implementation of SQL Conventions

Lawson has implemented several conventions to manage your database systems.

Naming Columns

Lawson products use the hyphen (-) to separate words in column names (for example, **MONTH-PTS**). IBM DB2 does not allow hyphens, so the database driver automatically replaces the hyphens with underscores when it communicates with the database. For example, the column **MONTH-PTS** changes to **MONTH_PTS**.

Reserved Words

IBM DB2 reserves some words that cannot be used for table, column, or index names. Lawson products do not have this restriction. To avoid problems between the two naming conventions, the database driver prefixes **R_** to table, column, and index names that are reserved by IBM DB2. For example, if a table has a column called **VALUES**, the database driver changes the name to **R_VALUES** for the IBM DB2 server. For a complete list of reserved words, see the *IBM DB2 Guide to SQL*.

Additional Columns

IBM DB2 supports only some Lawson database features. To make these features work with the IBM DB2 server, the database driver adds columns to the rows that make up the data tables in the database.

Subsets

Lawson databases use subsets. A subset is an index that has a condition or conditions attached to it. A subset index finds only those rows that satisfy the condition.

To make subsets work in IBM DB2, the database driver does two things. First, it adds a subset switch column to tables with subsets (that is, indexes that have conditions). Second, it makes sure the index allows duplicates. The database driver creates one subset switch column of type **CHAR(01)** for each index with a condition. If a row satisfies the condition, the database driver puts a **Y** in this column.

All rows that satisfy this condition are unique. If a row does not satisfy the condition, the database driver puts **N** in the column. The rows that do not satisfy the condition might be duplicates. When a condition exists, the subset column is the first column in the IBM DB2 index. The name of this additional column is the index name followed by **_SS_SW** (SubSet SWitch).

For example, if the Lawson GLMASTER table has an index GLMSET4 with a condition, the database driver adds a column named **GLMSET4_SS_SW** to the GLMASTER table in IBM DB2. This column has **Y** and **N** values identifying the rows that do and do not satisfy the condition, respectively.

The subset switch columns attach to the table immediately after the last normal column. If there are multiple subset indexes, the subset columns attach to the table in index-number order. These columns must have **Y** and **N** values in them for the database driver to process the rows correctly. Therefore, update the values of these columns every time you update the row with a non-Lawson program.

You also must assign the correct values to the columns when you create the row using a non-Lawson program. Use IBM DB2 triggers to automatically set the correct value.

Date Types

The use of *YYMMDD* fields (6-digit dates) in indexes is not supported. If a date field needs to be indexed, use an 8-digit *YYYYMMDD* field. If you use a *YYMMDD* field in an index, dates may not be returned in the expected order.

Occurring Columns

Lawson database design allows occurring columns (one-dimensional arrays). IBM DB2 does not handle such columns directly, but the database driver can store occurring columns in IBM DB2 by breaking each occurrence of the array into a separate column.

When you store each occurrence of a column in a separate column, each occurrence stores with the correct data type. The name of each occurrence is the column name followed by *_n*, where *n* is the number of the occurrence.

For example, if the occurring column name is **MONTH-PTS**, the first occurrence in IBM DB2 is **MONTH_PTS_01**, the second is **MONTH_PTS_02**, and so on.

Groups

Lawson applications no longer use occurring groups (two-dimensional arrays); however, some older Lawson applications might still have them. The database driver treats occurring groups like occurring columns in IBM DB2. For example, a table in the Lawson dictionary has an occurring group as follows.

```
Field Name     Occurs    Type        Size
----------     ------    ----        ----
Date-Group     5         GROUP
Begin-Date               ccyymmdd    8
End-Date                 ccyymmdd    8
```

The database driver changes the table information to the following format for IBM DB2.

```
BEGIN_DATE_GROUP      CHAR(40)
END_DATE_GRP          CHAR(40)
```

The database driver stores data in these columns as if they were occurring columns. Every option that applies to occurring columns also applies to occurring groups. Only a logical connection between the columns preserves the group concept.

Column Order

The database driver creates columns in the order in which they are defined. The database driver puts:

- Long strings (**_0**, **_1**, and so on) and occurring columns (**_n**) in order of their definition.
- Subset switch columns (**_SS_SW**) immediately after the last normal column.

Index

/

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V

W