Oracle® Database Lite

Oracle Lite Client Guide

Release 10.3

E12548-02

February 2010

ORACLE®

Oracle Database Lite Oracle Lite Client Guide Release 10.3

E12548-02

Contents

13 Using Stored Procedures and Triggers

Preface

This preface introduces you to the *Oracle Database Lite Client Guide* discussing the intended audience, documentation accessibility, and structure of this document.

Audience

This manual is intended for application developers as the primary audience and for database administrators who are interested in application development as the secondary audience.

Documentation Accessibility

Send Us Your Comments

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: olitedoc_us@oracle.com
- FAX: (650) 506-7355. Attn: Oracle Database Lite
- Postal service:

  Oracle Corporation
  Oracle Database Lite Documentation
  500 Oracle Parkway, Mailstop 4op2
  Redwood Shores, CA 94065
  U.S.A.

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

1 The Oracle Database Lite RDBMS

The following sections describe the Oracle Database Lite Relational Database Management System (RDBMS):

- Section 1.1, "Oracle Lite Database Overview"
- Section 1.2, "Execution Models for Applications that Use the Oracle Lite Database"

1.1 Oracle Lite Database Overview

The Oracle Lite database is compliant to the SQL92 standard and compatible to Oracle databases. In addition, it is compliant to the ACID requirements for transaction support. Because it is a small database specifically designed for a client device, it has a small footprint and is easy to administer. The Oracle Lite database can be installed on the following platforms: Linux, UNIX, Windows (Win32) and WinCE platforms.

You can use the Oracle Lite database that contains client data in a file with an ODB extension; any BLOB objects—either binary or character—and the indexes are stored in a file with an OBS extension. The Oracle Lite database exists solely to store and retrieve the user data specific to this device. It is not a replication of the entire Oracle database.

Because BLOB data and indexes are stored in an OBS file, there is no limit for BLOB data or indexes. The limitation for BLOB data and indexes is the space limitations of the operating system or 16 terabytes. There still exists a 4 GB limitation for the ODB file; however, this is not as much of an issue now that BLOB data can be stored in the OBS files.

Note: If you have been using the Oracle Lite database prior to the 10.3 release, you can upgrade your database to remove all BLOB data from within it and transfer the BLOB objects to OBS files by using the `defragdb` utility, which is documented in Section C.7, "DefragDB to Defragment and Reduce Size of the Oracle Lite Database".

The Oracle Lite database is an RDBMS that supports ODBC, JDBC, ADO.NET and SODA interfaces. SODA is an Oracle Database Lite specific C++ object API created to access the Oracle Lite database. SODA provides access to SQL as well as object-oriented functionality. See Chapter 8, "Oracle Database Lite Data Access APIs" for more information on each language.

1.2 Execution Models for Applications that Use the Oracle Lite Database

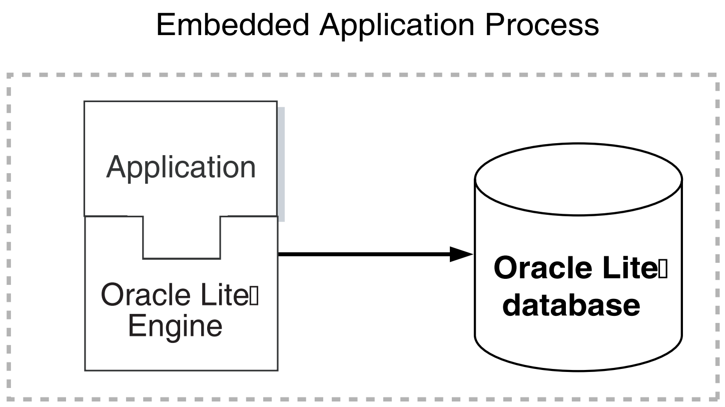
When building an application that only uses the Oracle Lite database, you can build it for one of the following purposes:

- For an application that uses the Oracle Lite database to store data solely for a single application, use the embedded application option. See Section 1.2.1, "Embedded Application in Single Process" for more information.
- You can design an application where the data is stored in a back-end Oracle database, but only the data that the user needs to see or update is downloaded to the client device. When either side modifies the data, it is synchronized between the client device and the back-end database. See Section 1.2.2, "Mobile Option for a Client in a Single Process" for more information.
- For multiple applications accessing the same database, use the client/server option. See Section 1.2.3, "Multiple Processes Accessing the Same Database" for more information.
- For any application that accesses the database remotely, use the client/server option. See Section 1.2.4, "Multiple Embedded Application Clients Accessing Remote Database" for more information.
- For an application that must access the database remotely and include the ability to synchronize the client data using Branch Office. See Section 1.2.5, "Multiple Clients Accessing Remote Database" for more information.

1.2.1 Embedded Application in Single Process

Figure 1–1 demonstrates an application that embeds the Oracle Lite database within the application. When the application is launched, the Oracle Lite database libraries are loaded into the same process as the application.

Figure 1–1 Embedded Application With ODB Libraries in Single Process

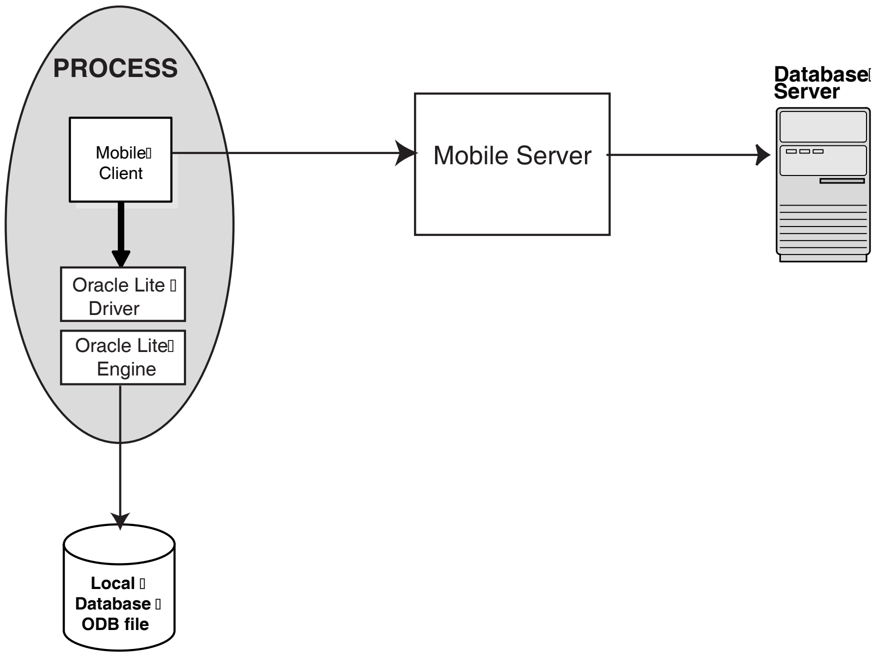


See Chapter 4, "Building an Embedded Application" for more information on how to embed an Oracle Lite database into an application.

1.2.2 Mobile Option for a Client in a Single Process

If you chose to install the Mobile client and synchronized your user on a single device, then when you launch your application, the Oracle Lite database libraries are loaded into the same process as your application. This scenario is demonstrated in Figure 1–2.

Figure 1–2 Diagram of Mobile Client and ODB Libraries in SIngle Process

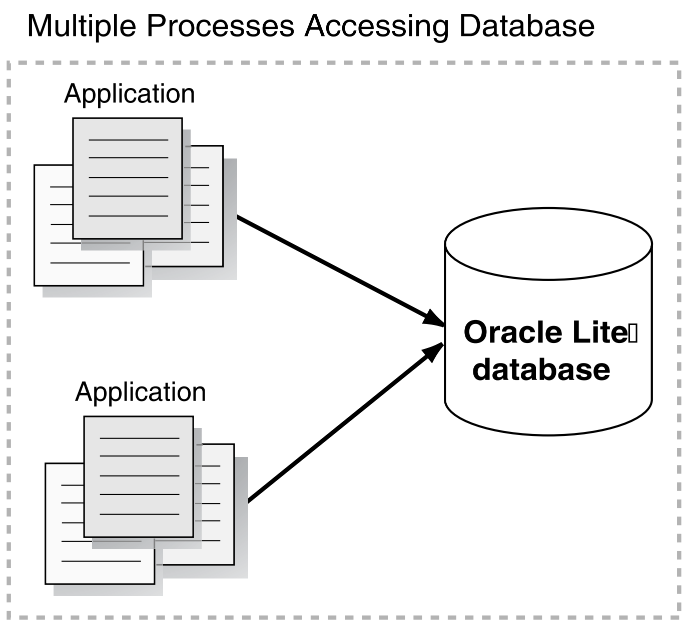


For details of how to create a Mobile application using the Oracle Lite database, see Section 2.2, "Creating and Managing the Database for a Mobile Client" in the *Oracle Database Lite Administration and Deployment Guide*.

1.2.3 Multiple Processes Accessing the Same Database

Figure 1–3 shows how you can configure multiple application processes to share the same database on the same machine. Thus, when each application is launched, each application exists in its own process and can access the same database independently. In this scenario, Oracle Database Lite libraries use shared memory to coordinate locking between both processes.

Figure 1–3 Applications in Multiple Processes Accessing Single Database



For more details, see Chapter 4, "Building an Embedded Application".

Note: This scenario is not available on WinCE.

1.2.4 Multiple Embedded Application Clients Accessing Remote Database

If you are embedding a database into your application software, but you want the applications to be located on clients that are remote from the data within the Oracle Lite database, then use the client/server embedded approach with the multi-user service, as described in Chapter 5, "Building a Client/Server Environment".

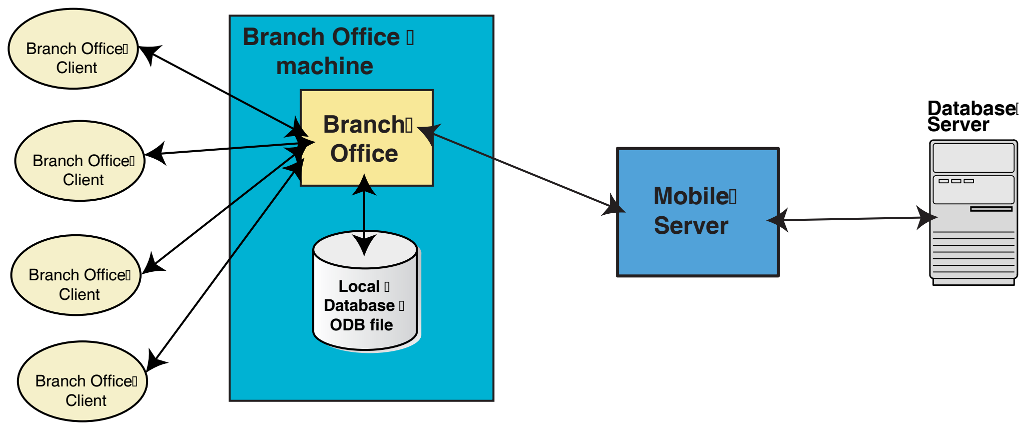
1.2.5 Multiple Clients Accessing Remote Database

If you have several remote clients accessing the same data, you can use Branch Office to facilitate the remote applications. Figure 1–4 demonstrates how multiple remote Branch Office clients access the data through the Branch Office machine to the Mobile Server and finally accessing the back-end Oracle database.

The Branch Office machine contains the Branch Office executables and the local Oracle Lite database, which all clients access for their information. When a synchronization is requested, information is communicated between the Branch Office and the back-end database through the Mobile Server.

Note: Oracle Database Lite is not identical to the Oracle database; thus, it is not designed for large amounts of transferred data or a large number of concurrent transactions.

Figure 1–4 Using Branch Office for Managing Multiple Clients that Access a Remote Database



See Chapter 9, "Manage Your Branch Office" in the *Oracle Database Lite Administration and Deployment Guide* for more information.

2 System Requirements for the Oracle Lite Database as the Mobile Client

Before you install, you must check to see that you have the correct hardware and software necessary for using Oracle Lite as your Mobile client on your device. The requirements for the Oracle Lite client are detailed in the following sections:

- Section 2.1, "System Requirements for the Oracle Lite Database as the Mobile Client on Windows"
- Section 2.2, "System Requirements for the Oracle Lite Database as the Mobile Client on Linux"

2.1 System Requirements for the Oracle Lite Database as the Mobile Client on Windows

Before you install, you must check to see that you have the correct hardware and software necessary for Windows Mobile clients. The requirements for both are detailed in the following sections:

- Section 2.1.1, "Hardware Requirements for the Oracle Lite Database as the Mobile Client on Windows"
- Section 2.1.2, "Software Requirements for the Oracle Lite Database as the Mobile Client on Windows"

2.1.1 Hardware Requirements for the Oracle Lite Database as the Mobile Client on Windows

The hardware requirements for Windows Mobile clients are described in the following table:

Table 2–1 Hardware Requirements for Mobile Clients

Component	Hardware Requirements for this Component
Mobile Client for Win32	CPU: Pentium 4, 1 GHz Disk Space: 30 MB RAM: 256 MB
Mobile Client for Web-to-Go	CPU: Pentium 4, 1 GHz Disk Space: 70 MB RAM: 512 MB

Table 2–1 (Cont.) Hardware Requirements for Mobile Clients

Component	Hardware Requirements for this Component
Mobile Client for Windows CE/Windows Mobile	CPU: ARM-based processor or Emulator Storage Space: 8 MB Free program memory: 16 MB It matters what external memory media and file system you use. See Section 2.1.1.1, "What File System and External Memory Media Should You Use for Windows CE?" for more information.
Branch Office	CPU: Pentium 4, 1 GHz Disk Space: 70 MB RAM: 512 MB

2.1.1.1 What File System and External Memory Media Should You Use for Windows CE?

When you are using a WinCE device, you will probably use some form of external memory media, such as Compact Flash or SD cards. Of these two, the SD card is more reliable in its method of connecting into the device. We have seen some issues of database corruption when using Compact Flash cards, since the card can be removed in the middle of a write without notice.

When you do choose a media, you can also decide on the type of file system you use. We strongly recommend that you use the Transaction-Safe FAT (TFAT) file system over the FAT system, which is the default and more widely used. FAT has some unreliability that was not noticeable in a laptop or desktop situation. However, this unreliability in its ability to flush its buffers fully when writing out to the removable memory does not handle well when the media is removed in the middle of the transaction. Thus, the shortcomings of the FAT file system is being seen more in the WinCE environment with removable memory. The TFAT design provides transaction-safety for data storage. That is, the data that is being written out to the removable media is either completely committed or rolled-back. Thus, the TFAT file system is highly recommended for any removable data—especially for the Oracle Lite database. There is a performance consideration for using the TFAT file system. It is slower than FAT, but also more reliable.

You need to decide whether performance or reliability is your priority. The following is the order of reliability with 1 being the least reliable and 3 being the most reliable:

1. Compact Flash media with the FAT file system.
2. SD card media with the FAT file system.
3. Compact Flash or SD card media with the TFAT file system.

Note: See the Microsoft Web site for information on how to create the TFAT file system.

2.1.2 Software Requirements for the Oracle Lite Database as the Mobile Client on Windows

The software requirements for Windows Mobile clients are described in the following sections:

- Section 2.1.2.1, "Certified Operating Systems and Other Software Requirements"

- Section 2.1.2.2, "Supported and Certified Technologies for Windows Mobile Clients"
- Section 2.1.2.3, "Supported Platforms for Oracle Database Lite WinCE"
- Section 2.1.2.4, "Windows Mobile Client Notes"

2.1.2.1 Certified Operating Systems and Other Software Requirements

Table 2–2 Software Requirements for Windows Mobile Clients

Mobile Client	Certified Operating System	Other Software Requirements
Mobile Client for Win32	Windows Vista Ultimate, Windows XP Professional Edition with Service Pack 2, or Windows 2003	If using any Java APIs, including synchronization or JDBC, use JRE 5.0 If implementing any .NET applications, use Compact Framework .NET 1.1 or 2.0
Mobile Client for Web-to-Go	Windows Vista Ultimate, Windows XP Professional Edition with Service Pack 2, or Windows 2003	
Mobile Client for Windows CE	Windows CE 5.0 See Section 2.1.2.3, "Supported Platforms for Oracle Database Lite WinCE" for full details.	If using JDBC, use either IBMJ9 or the CrEme JDK version 4.1 from `NSIcom.com`. ActiveSync version 3.8 or higher. Microsoft .NET Compact Framework 1.0
Mobile Client for Windows Mobile	Windows Mobile 2003 Windows Pocket PC 2003 Windows Mobile 2003, 2nd edition Windows Pocket PC 2003, 2nd edition See Section 2.1.2.3, "Supported Platforms for Oracle Database Lite WinCE" for full details.	ActiveSync version 3.8 or higher. Microsoft .NET Compact Framework 1.1 If using JDBC, use either IBMJ9 or the CrEme JDK version 4.1 from `NSIcom.com`.
	■ Windows Mobile 5 ■ Windows Mobile 5 for Pocket PC ■ Windows Mobile 5 for Pocket PC Phone Edition ■ Windows Mobile 5 AKU2	ActiveSync version 4.1 or higher. Microsoft .NET Compact Framework 1.1 or 2.0 If using JDBC, use either IBMJ9 or the CrEme JDK version 4.1 from `NSIcom.com`.
	■ Windows Mobile 6 ■ Windows Mobile 6 Classic ■ Windows Mobile 6 Professional	ActiveSync version 4.5 or higher. Microsoft .NET Compact Framework 1.1 or 2.0 If using JDBC, use either IBMJ9 or the CrEme JDK version 4.1 from `NSIcom.com`.
OC4J	Windows Vista Ultimate, Windows XP Professional Edition with Service Pack 2, or Windows 2003	
Branch Office	Windows XP Professional Edition with Service Pack 2, or Windows 2003	

You should install all of the patches required for the JDK for the Windows operating system. This is constantly under review and published on the JDK download page on the Sun Microsystems Web site.

2.1.2.2 Supported and Certified Technologies for Windows Mobile Clients

The following are the supported and certified technologies for Mobile clients:

Note: Ensure that after you install the required software, that they the appropriate directories are included in the `PATH`. For example, after you install the JDK, ensure that the `JAVA_HOME` is included in the `PATH`.

Table 2–3 Supported and Certified Technologies for Windows Mobile Clients

Mobile Client	Supported Technologies	Certified Technologies
Mobile Client for Win32	▪ Sun Microsystems Java Runtime Edition 5.0 ▪ JDBC 1.2 ▪ ADO.Net 1.1 – requires Microsoft .Net Framework 1.1 or 2.0 ▪ ADO.Net 2.0 – requires Microsoft .Net Framework 2.0 ▪ ODBC 2.0 and 3.5 ▪ SQL92	
Mobile Client for Web-to-Go	▪ Sun Microsystems Java Runtime Edition 5.0 ▪ Java Servlets 2.2 ▪ JDBC 1.2 ▪ Oracle Java Server Pages Version 9.0.2.0.0 ▪ Oracle UIX version 2.1.7 ▪ Oracle XML Parser 9.0.3.0.0	Struts version 1.1 is certified for use with Oracle Database Lite Web-to-Go.

Table 2–3 (Cont.) Supported and Certified Technologies for Windows Mobile Clients

Mobile Client	Supported Technologies	Certified Technologies
Mobile Client for Windows CE	■ ODBC 2.0 and 3.5 ■ JDBC 1.2 ■ ADO.Net 1.1 (Requires Microsoft Compact .Net Framework 1.0 + Service Pack 2) or 2.0 ■ ADO.Net 2.0 – requires Microsoft .Net Compact Framework 2.0 ■ Microsoft ActiveSync version 3.8 or for Windows CE 5.0, use Microsoft ActiveSync version 4.1 or higher.	Oracle Database Lite is certified with the following JVMs on Windows Mobile 2003 Second Edition: ■ IBM J9 Websphere Everyplace Micro Environment for Windows Mobile 2003 ARM Personal Profile ■ Creme JVM 4.1, which can be obtained at `http://www.nsicom.com` NOTE: Java Stored Procedures are not supported on Windows CE.
OC4J	■ Sun Microsystems Java Runtime Edition 5.0 ■ Java Servlets 2.4 ■ JDBC 1.2 ■ Oracle Java Server Pages Version 10.1.3.0.0 ■ Oracle UIX version 2.2.24 ■ Oracle XML Parser 10.1.3.0.0	■ Struts version 1.1 ■ JDeveloper 10.1.3
Branch Office		Struts version 1.1

2.1.2.3 Supported Platforms for Oracle Database Lite WinCE

Table 2–4 provides the full list of supported platforms for Pocket PC and Windows Mobile:

Table 2–4 Pocket PC and Windows Mobile Supported Platforms

Product Name	WinCE Version	Chipsets	Oracle Lite Client CAB file download from Setup page
Pocket PC 2003 Windows Mobile 2003	4.20.1081	ARMV4	Oracle Lite PPC2003 ARMV4, which uses the `<language>\ppc2003\armv4\olite.cab`
Windows Mobile 2003 2nd Edition	4.21.1088	ARMV4	Oracle Lite PPC2003 ARMV4, which uses the `<language>\ppc2003\armv4\olite.cab`
Windows Mobile 5 and Windows Mobile 5 AKU2	5.0 and 5.1.465	ARMV4I	Oracle Lite PPC50 ARMV4I, which uses the `<language>\ppc50\armv4i\olite.cab`
Windows Mobile 6	5.2.1236	ARMV4I	Oracle Lite PPC60 ARMV4I, which uses the `<language>\ppc60\armv4i\olite.cab`

If you wish to use Java, mSQL and utilities, you must also install the Tools CAB files after installation of the `olite.cab` file, which is described in Section 3.2.3, "Installing Tools CAB Files for Java, MSQL, and Utility Support".

2.1.2.4 Windows Mobile Client Notes

For Mobile Client Web-to-Go, Win32, Branch Office and BC4J:

- Internet Explorer 6.0 is required when using SSL to synchronize with the Mobile Server.

- The product requires the Microsoft C Runtime Library 7.1 (`msvcrt71.dll`), which you can download off the Microsoft site or other sites on the Web.

2.2 System Requirements for the Oracle Lite Database as the Mobile Client on Linux

The Mobile Server installation includes the following Mobile Clients for Linux:

- Mobile Client for Linux x86
- Mobile Client for Linux Web-to-Go

Note: The Device Manager agent (`DMagent`) must be running to successfully uninstall the Linux Client.

Before you install, you must check to see that you have the requirements necessary for Linux Mobile clients. The requirements for both are detailed in the following sections:

- Section 2.2.1, "Certified Platforms and Supported Technologies for Linux Mobile Clients"
- Section 2.2.2, "Software and Hardware Requirements for the Oracle Lite Database as the Mobile Client on Linux"
- Section 2.2.3, "Setting Environment Variables Before Installing the Linux Mobile Client"

2.2.1 Certified Platforms and Supported Technologies for Linux Mobile Clients

Table 2–5 provides the full list of certified and supported platforms for Linux Mobile clients:

Table 2–5 Certified Platforms and Supported Technologies for Linux Mobile Clients

Mobile Client	Certified Platforms	Supported Technologies
Oracle Lite Linux Web-to-Go	RedHat Enterprise Linux AS release 4	■ JavaSoft Java Runtime Edition 1.4.2 ■ Java Servlets 2.2 ■ JDBC 1.2 ■ Oracle Java Server Pages Version 9.0.2.0.0 ■ Oracle UIX version 2.1.7 ■ Oracle XML Parser 9.0.3.0.0
Oracle Lite Linux x86	RedHat Enterprise Linux AS release 4	■ JavaSoft Java Runtime Edition 1.4.2 ■ JDBC 1.2 ■ ODBC 2.0 ■ SQL92

Note for Oracle Lite Linux WEB: Mozilla version 1.7.x is the preferred internet browser on Linux.

2.2.2 Software and Hardware Requirements for the Oracle Lite Database as the Mobile Client on Linux

Table 2–6 provides the software and hardware requirements for Linux Mobile clients:

Table 2–6 Software and Hardware Requirements for Linux Mobile Clients

Mobile Client	Hardware Requirement	Software Requirements
Mobile Client for Linux Web-to-Go	CPU: Pentium III 360 MHz Disk Space: 40 MB RAM: 128 MB	Redhat Enterprise Linux AS Release 4 JDK 1.4.2 must be installed
Mobile Client for Linux x86	CPU: Pentium III 360 MHz Disk Space: 30 MB RAM: 128 MB	Redhat Enterprise Linux AS Release 4 JDK 1.4.2 must be installed

2.2.3 Setting Environment Variables Before Installing the Linux Mobile Client

peSet the following environment variables:

- Set `OLITE_HOME` to where Oracle Database Lite is installed, such as `/home/<user>/olite`
- Set `JAVA_HOME` to the Java installation directory
- Add the following to the `LD_LIBRARY_PATH`

  ```
  $JAVA_HOME/jre/lib/i386$JAVA_HOME/jre/lib/i386/server
  $OLITE_HOME/bin
  ```

- Add `$OLITE_HOME/bin` to the `PATH`

3 Installing the Oracle Lite Database

One of the benefits of Oracle Database Lite is that you can have an application downloaded onto a device, where data can be synchronized between the device and the back-end Oracle database.

In general, you can install the Oracle Lite database on the following Mobile client platforms:

- Windows clients (such as OC4J, Web-to-Go, Branch Office, and BC4J): The applications built for these clients use a Java browser.
- Linux, Win32, WinCE, and Windows Mobile clients: These applications are client/server applications. Thus, start the application as you would start any application on these platforms.

Note: You can configure only one device for a particular user. For example, it is not possible to have two devices both executing a Mobile client for the user `JOHN`.

When you install the Mobile client, Oracle Database Lite installs an Oracle Lite database within ODB and OBS files with an automatic name and assigns a data source name (DSN). For details on the client Oracle Lite database, see Section 2.1, "Oracle Database Lite Overview" in the *Oracle Database Lite Administration and Deployment Guide*.

The following sections detail how to install Mobile Client software on your client machine:

- Section 3.1, "Preparing the Device for a Mobile Application"
- Section 3.2, "Installing the Oracle Lite Mobile Client Software"
- Section 3.3, "Configuring for Default Sync When Installing the Client"
- Section 3.4, "Configuring the Client for Secure Socket Layer (SSL)"
- Section 3.5, "Specifying Whether the Client Uses a Static or Dynamic (DHCP) IP Address"
- Section 3.6, "Using Offline Instantiation to Distribute Multiple Mobile Clients"

See Chapter 6, "Managing Your Oracle Lite Mobile Client" for instructions on how to perform certain functions on the client. See Chapter 1, "Oracle Database Lite Management With the Mobile Server" in the *Oracle Database Lite Administration and Deployment Guide* for information on how to manage functionality from the Mobile Server.

3.1 Preparing the Device for a Mobile Application

In order for a device to execute Mobile applications, you must do the following:

Note: Install the Mobile client for any application after the application is published.

1. Install the Oracle Lite Mobile client software that is appropriate to the client platform on your client machine. For example, install either the Mobile Client for Win32, Mobile client for OC4J or Web-to-Go on a Windows 32 client machine.

   See Section 3.2, "Installing the Oracle Lite Mobile Client Software" for a full description.

2. Download the user applications and its associated data.

   Synchronize the Mobile client for the first time. Sign in with the username/password of the Mobile user who owns the Mobile applications. The data for each application is retrieved.

   Notes: For the restrictions on creating the username and password, see Section 5.3.1.2.1, "Defining Username and Password" in the *Oracle Database Lite Administration and Deployment Guide*.

   For more information about synchronization, see Chapter 6, "Managing Synchronization" in the *Oracle Database Lite Administration and Deployment Guide*. .

3. You can now launch your applications from your client machine or from your Mobile device.

3.2 Installing the Oracle Lite Mobile Client Software

Before you install the Oracle Lite Mobile client on your device, make sure that there is 1 MB of space available to download the `setup.exe`.

We do not support the following configuration scenarios:

- An Oracle Lite Mobile client and the Mobile Development Kit (MDK) cannot be installed on a single system.
- A client user cannot have more than one device.

Note: If you are installing a Mobile Client for Web-to-Go, follow the instructions provided in Section 3.2.1, "Installing Web-to-Go on Linux" before downloading the `setup.exe`.

To install the Mobile client software, perform the following tasks.

Note: Any developer can modify how the client is installed before the installation with the INF file. For details on how to customize your Mobile client, see Section 7.1, "Customize the Mobile Client Software Installation for Your Mobile Device" in the *Oracle Database Lite Administration and Deployment Guide*.

1. On the client, open a browser to point to the Mobile Server using the following URL.

   ```
   http://<mobile_server>:<port>/webtogo/setup
   ```

   Note: Substitute `https` if using HTTP over SSL.

   Figure 3–1 displays the Mobile Client setup page, which contains links to install Mobile Client software for multiple platforms and languages. You can select another language than English on the Language pulldown.

   For viewing platforms, you can choose to see all available platforms for the indicated language, or only those platforms for Windows or Windows CE with the Platform pull-down menu.

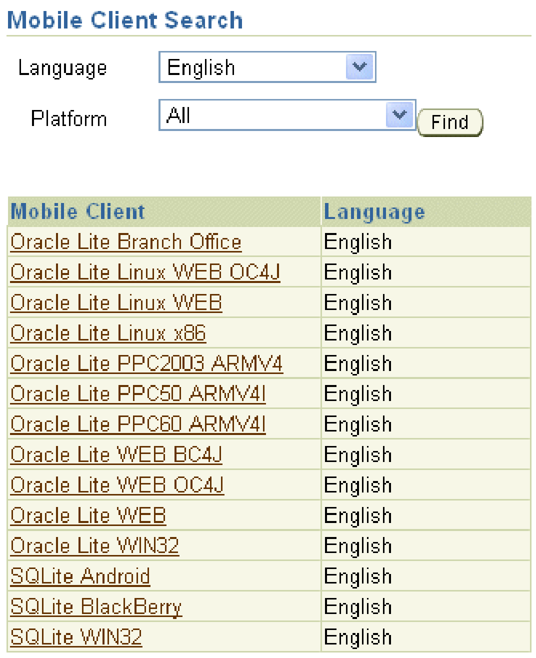
   For the Windows and WinCE platforms, the Pocket PC 2003 (PPC2003), Pocket PC SDK 5.0 (PPC50), and Windows Mobile 6 Professional SDK (PPC60) client platforms are provided in the Mobile client setup page. In addition, these client CAB files are optimized for size to minimize the footprint on your device; thus, they exclude Java, `msql` or utility support. Thus, you may have to perform one or more of the additional steps:

   - If you are using a client with the Standard SDK for WinCE 5.0 platforms for Windows Mobile 5 (WCESTDSDK), then use the appropriate CAB files that are provided in the MDK install. For information on how to install the WCESTDSDK CAB files, see Section 3.2.2, "Installing Standard SDK WinCE 5.0 CAB Files for Your Mobile Client".
   - If you want the Java, `msql`, or utility support on any Windows or WinCE Mobile client; then install the tools CAB file for that platform as described in Section 3.2.3, "Installing Tools CAB Files for Java, MSQL, and Utility Support".

   When you select the language, the collation sequence for the Oracle Lite database is also preconfigured for you. You can also specify the collation sequence with the `NLS_SORT` parameter. You can only perform a linguistic sort on Oracle Lite databases that have the collation sequence of FRENCH, GERMAN, CZECH, OR XCZECH. You cannot do a linguistic sort on a BINARY collation sequence, which is used with all languages, except the three previously listed. For more details on linguistic sort, see Section 4.4, "Support for Linguistic Sort" in the *Oracle Database Lite Client Guide*.

   Note: Only the Mobile Development Kit has the full National Language Support for (Traditional and Simplified) Chinese, Japanese, and Korean (CJK). All other components, including Mobile clients, support CJK without the Traditional Chinese language. However, the Simplified Chinese language is supported.

Figure 3–1 Mobile Client Setup Page



Mobile Client Search

Language: English

Platform: All

Mobile Client	Language
Oracle Lite Branch Office	English
Oracle Lite Linux WEB OC4J	English
Oracle Lite Linux WEB	English
Oracle Lite Linux x86	English
Oracle Lite PPC2003 ARMV4	English
Oracle Lite PPC50 ARMV4I	English
Oracle Lite PPC60 ARMV4I	English
Oracle Lite WEB BC4J	English
Oracle Lite WEB OC4J	English
Oracle Lite WEB	English
Oracle Lite WIN32	English
SQLite Android	English
SQLite BlackBerry	English
SQLite WIN32	English

Note: Available clients may differ from what is shown above.

---

2. Click the Mobile client for your language and client platform.

Note: There are two client versions for the Web-to-Go model. The Oracle Lite WEB and the Oracle Lite WEB OC4J. Use Oracle Lite WEB OC4J when you need full J2EE 1.3 compliance. The original Oracle Lite Web client uses the Oracle Lite Servlet engine, which does not support all features for J2EE 1.3. The Oracle Lite WEB OC4J uses the OC4J stack within OracleAS; thus, you have full J2EE 1.3 support. However, you also must create the client according to OC4J/J2EE specifications.

---

3. The Save As dialog box appears. The file name field displays the setup executable file for the selected platform as an `.exe` file type. Save the executable file to a directory on the client machine.

Note: For WinCE, you install any of the Oracle Lite Windows Mobile platforms to ActiveSync. Then, when the device is put into the cradle, ActiveSync installs the Oracle Database Lite onto the device when it synchronizes.

---

4. Install the Mobile client. For all platforms, except installing WinCE on ActiveSync, go to the directory where you saved the setup executable file. Double-click the file to execute it.
5. Enter the username and password for the Mobile user.

Note: For the restrictions on creating the username and password, see Section 5.3.1.2.1, "Defining Username and Password" in the *Oracle Database Lite Administration and Deployment Guide*.

6. You may be required to select the type of privilege under which to install the Mobile client. This may already be designated by the administrator in the INF file before installation or the current user may have a privilege that defaults to a certain privilege for the installation.

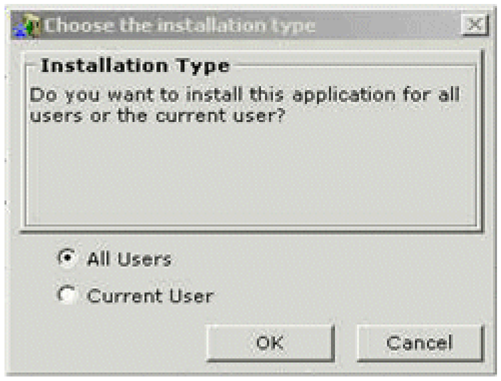
   - All Users—The user installing this Mobile client has administrator privileges and can install the Mobile client for itself as well as any members that may be associated with the user.

     Note: After installing a user with administrative privileges, you can add member users to allow multiple users on a Mobile client. The member user has its own username/password, but executes any synchronization and modifications under the user's authority. For information on how to install and use member users, see Section 5.3.3, "Adding New Members and Associating Them With Their User" in the *Oracle Database Lite Administration and Deployment Guide*.

   - Current User—Selecting this option designates that the user does not have administrator privileges, but can install and use the Mobile client as a single user.

     Note: For details on how to designate the user privilege and for more information on user installation types, see Section 7.1, "Customize the Mobile Client Software Installation for Your Mobile Device" in the *Oracle Database Lite Administration and Deployment Guide*.

Figure 3–2 Select Installation Privileges


Choose the installation type

Installation Type

Do you want to install this application for all users or the current user?

[✓ All Users] Current User

7. Provide the client directory name where to install the Mobile client.
8. Once installed, synchronize the Mobile client for the first time. During the first synchronization, all applications and data for this user is brought down and installed on your Mobile client.

9. Each platform has further steps. See Table 3–1 for a description of the steps for each platform.

Note: See Section 3.3, "Configuring for Default Sync When Installing the Client" for directions on how to enable a default synchronization after any client installation on your device.

Table 3–1 Initializing the First Synchronization for Each Mobile Client Platform

Oracle Mobile Client	Initial Synchronization Details
Oracle Lite WEB or Oracle Lite WEB OC4J, both for Web-to-Go support	The synchronization step takes place when you click **Next**, after executing the `setup.exe`. This prompts you to login to the Mobile client for OC4J or Web-to-Go. If you want to synchronize at another time, do the following: 1. Open a browser to the Mobile client. For example, if you install a Web client with port 8080, point the browser to `http://localhost:8080/webtogo`. 2. Log in with the username/password for the Mobile user. 3. Click **Sync** on the tabs in the upper right corner.
Oracle Lite PocketPC for WinCE devices	Perform the following steps. 1. If you install the PocketPC platform to ActiveSync, insert the WinCE device in the cradle. ActiveSync performs a synchronization to install Oracle Database Lite on the device. 2. After Oracle Database Lite is installed on the device, then start the Device Manager Agent on the device by either selecting Oracle DM in the programs group or by executing `dmagent.exe`, which is in the `orace` directory. 3. Enter the username, password and Mobile Server URL. You can either enter the complete URL of the Mobile Server, the IP address or the hostname of the Mobile Server. If left off, the prefix "`http://`" is added automatically. Only use the hostname if the device is properly configured to use DNS name resolution. Otherwise, enter the IP address. The device is now registered with the Mobile Server and ready to be used.
All other platforms	Perform the following steps. 1. Locate the directories where you installed the runtime libraries, and launch the Mobile Sync application. 2. The `mSync` dialog appears. Enter the user name and password of the Mobile user. If you do not know your user name and password, ask your system administrator, who creates users and assigns passwords to each user. In the **Server** field, enter the URL for your Mobile Server. Click **Apply** and click **Sync**.

3.2.1 Installing Web-to-Go on Linux

Perform the following to install and run Web-to-Go on Linux.

WARNING: If you are testing the Oracle Database Lite on Suse Linux, you must do the following before installation:

```
ln -s /usr/lib/libssl.so.0.9.7 /usr/lib/libssl.so.4
ln -s /usr/lib/libcrypto.so.0.9.7 /usr/lib/libcrypto.so.4
```

Once the installation is complete, perform your tests and then remove the soft links, as these may cause problems with other programs you have installed on your machine. This instruction is only for testing and should not be a permanent option.

---

1. Download the Web-to-Go setup executable by clicking the "Oracle Lite Linux WEB" link on the Mobile Server setup page.
2. After the download is complete, set execution permissions on the setup executable with `chmod 755 setup`.
3. Execute the setup command, as follows:

   ```
   ./setup
   ```

4. To start Web-to-Go in the debug mode, do the following:

   ```
   cd $OLITE_HOME/bin
   ./webtogo -d0
   ```

   To start Web-to-Go in the daemon mode, do the following:

   ```
   cd $OLITE_HOME/bin
   ./webtogo
   ```

   To kill Web-to-Go, which is in the daemon mode, do the following:

   ```
   cd $OLITE_HOME/bin
   ./webtogo -k
   ```

To uninstall Web-to-Go and delete the database files, perform the following:

```
cd $OLITE_HOME
./uninst
```

The `dmagent` is automatically launched in a daemon mode when setup is executed. However if you want to restart it, first kill the current process and then perform the following:

```
cd $OLITE_HOME/bin
./dmagent
```

3.2.2 Installing Standard SDK WinCE 5.0 CAB Files for Your Mobile Client

By default, the Windows 2003, Windows Mobile 5 and Mobile 6 CAB files are installed with the Mobile Server and thus, are displayed as options on the Mobile client setup page. These CAB files are registered with the Mobile Server. However, if your Mobile client is a Standard SDK WinCE 5.0 platform, use one of the WCESTDSDK CAB files contained in the MDK install.

Note: All CAB files on either the Mobile Server or MDK do not include Java, mSQL or utility support. For information on how to install these utilities, see Section 3.2.3, "Installing Tools CAB Files for Java, MSQL, and Utility Support".

An Oracle Database Lite client platform consists of a CAB file, an Installation Configuration File (INF file) that describes how to install the files, and an INI file that specifies the platform.

The following steps describe how to install the Standard SDK WinCE platform:

1. The WCESTDSDK CAB file must be copied from the MDK install to either the Mobile Server or the Mobile client, as described below:

   - Option One: Register the WCESTDSDK CAB file on the Mobile Server. The Mobile Server client setup page displays the predefined client platforms that you can download and install on your Mobile client device.

     If you want the Standard SDK WinCE CAB files to be displayed on the Mobile Server client setup page, then register the desired platform in the Mobile Server. After registration, the Mobile client can download the SDK CAB file from the Client setup page.

     Find the unregistered CAB file for the desired platform and language in the MDK installation in the following directory:

     ```
     <ORACLE_HOME>\Mobile\SDK\wince\<platform>\cabfiles
     ```

     Copy and rename the CAB file. The CAB files are named `olite.<language>.<platform>.<chipset>.CAB`. Rename the CAB file to `olite.cab` and copy it into a subdirectories according to language and client platform type relative to the *<ORACLE_HOME>*`\mobile_oc4j\j2ee\mobileserver\applications\mobileserver\setup\` directory. Take note of the directory path as you will provide the location of the CAB file in the INF file. in the INF file.

   - Option Two: Copy the desired WCESTDSDK CAB file directly to the Mobile client from the *<ORACLE_HOME>*`\Mobile\SDK\wince\`*<platform>*`\cabfiles\` directory. Each CAB file is named `olite.<language>.<platform>.<chipset>.cab`.

2. On the Mobile Server, create an INF file and place it in the appropriate subdirectory according to the language and platform type in the *ORACLE_HOME*`\mobile_oc4j\j2ee\mobileserver\applications\mobileserver\setup\dmc` directory. The INF file provides the instructions for installing the CAB file on the client platform. You can copy one of the existing INF files, such as the `std500.inf` file. If you want to add additional instructions, copy the file and make sure the INI file refers to the new INF file.

   If you have to modify it for the new platform, make sure that you give it a new name to avoid changing an existing platform. Provide the location of the CAB file—which you found in step 1—in the `<file><item><src>` and `<des>` tags, which are described in Section 7.10, "Installation Configuration (INF) File"in the *Oracle Database Lite Administration and Deployment Guide*.

   The following demonstrates how to specify a CAB file located in the `WINCE/<language>/stdsdk500/<cpu>` directory, which is relative to the `setup` directory, and the destination for the CAB file.

```
<file>
  <item type='WINCE'>
    <src>/$OS_LANG$/stdsdk500/$CPU$/olite.cab</src>
    <des>$APP_DIR$\olite.cab</des>
  </item>
</file>
```

3. On the Mobile Server, create an INI file that refers to the INF file for this platform. See Section 3.2.2.1, "Defining the INI File" for details.
4. On the Mobile Server, register the new platform with the device manager resource loader, which uses the INI script to create a new Platform.

   ```
   ORACLE_HOME\mobile\server\admin\dmloader
           <repository_owner>/<repository_password>@jdbc_url <ini_filename>
   ```

   For example, to load the `std500.ini` file as shown in step 3, perform the following:

   ```
   ORACLE_HOME\mobile\server\admin\dmloader
           <repository_owner>/<repository_password>@jdbc_url std500.ini
   ```

   Note: if you supply a RAC URL as the JDBC URL, then enclose it within two double-quotes as the operating system treats the equal sign (=) as a delimiter, which truncates the RAC URL and throws the `syntax error: unexpected token '('.` error

5. On the Mobile Server, copy the `setup_<language>.exe` files to the following directory on the Mobile Server:

   ```
   <ORACLE_HOME>\mobile_oc4j\j2ee\mobileserver\applications\mobileserver
   \setup\dmc\wince\<platform>\<chipset>\
   ```

   For example, registering the `wcestd500_sdk` CAB file, the setup files should be copied to the following directory:

   ```
   ORACLE_HOME\mobile_oc4j\j2ee\mobileserver\applications\mobileserver
   \setup\dmc\wince\ppcstd500\armv4i
   ```

6. Restart the Mobile Server to see the newly registered platform in the setup GUI.
7. On the client, open a new browser that points to the setup page to select the newly registered platform with the SDK CAB file.

3.2.2.1 Defining the INI File

Create an INI file that refers to the INF file, as well as other attributes. The following shows how the INI file is organized:

```
# List platforms to be created in the [Platform] section
#
# Format: platform_name;language
[PLATFORM]
# Provide string to be displayed in the setup UI
PLATFORM1;LANGUAGE
#
# Platform details. One entry for each platform listed in the
#[PLATFORM] Section. Provide the same info but prepend with "PLATFORM."
[PLATFORM.PLATFORM1;LANGUAGE]
TYPE=OS_CPU_LANGUAGE_NAME
```

```
INF=file.inf
BOOTSTRAP=dmcommand
ATTRIBUTES=attribute1=value1&attribute2=value2
```

Where the tags define the following:

- `PLATFORM`: Provide the platform type and language separated by a semi-colon.
- `TYPE`: Provide a name for the platform that is a concatenation of the operating system, CPU, language, and name—where each are separated by an underscore—such as `WINCE_ARMV4I_US_OLITE_STD500`.
- `INF`: Provide the name of the INF file.
- `BOOTSTRAP`: You can find a list of the bootstrap commands in a pull-down in the Mobile Devices page.
- `ATTRIBUTES`: The attributes are separated by an ampersand (`&`). These are the same attributes that are discussed in Section 7.4.3.2, "Create a Custom Platform By Extending an Existing Platform" in the *Oracle Database Lite Administration and Deployment Guide* and are as follows:
  - Can the device be updated: `update=true|false`
  - Is the platform enabled: `enabled=true|false`
  - Can applications on the device be updated: `app_upgrade=true|false`
  - Should the device manager on the client be started automatically: `dmc=auto`

For example, the following is an INI file that describes the WinCE Standard SDK 5.00 for ARMV4I:

```
# Platforms
#
[PLATFORM]
# Windows CE Standard SDK 5.00 - ARMV4I
# Provide string to be displayed in the setup UI
Oracle Lite WCESTD500 ARMV4I;US
#
# Windows CE Standard SDK 5.00 ARM V4i
[PLATFORM.Oracle Lite WCESTD500 ARMV4I;US]
TYPE=WINCE_ARMV4I_US_OLITE_STD500
INF=std500.inf
BOOTSTRAP=DeviceInfo
ATTRIBUTES=dmc=auto&update=true&enabled=true
```

3.2.3 Installing Tools CAB Files for Java, MSQL, and Utility Support

The Tools CAB file provides Java, `msql`, and utility support on the client devices. If you want to use tools on your Mobile client, then install the tools CAB file. This CAB file is installed as an application on your Mobile client.

The following sections describe the steps to create and publish the tools CAB file as an application for the Mobile client:

- Section 3.2.3.1, "Defining the Tools CAB as an Application in Packaging Wizard"
- Section 3.2.3.2, "Assigning the Tools CAB to the User"

3.2.3.1 Defining the Tools CAB as an Application in Packaging Wizard

Perform the following to install the Tools CAB, if must be published as an application:

1. Start the Packaging Wizard.

2. Select **Create New Application Definition**. Click **OK**
3. Select the platform for the specific CAB—such as `Oracle Lite PPC60 ARMV4I: US`. Click **Next**.
4. Fill in the CAB file details. The following example provides CAB file details for the `Oracle Lite PPC60 ARMV4I: US` platform.
   - Application Name: `tools cab ppc60 armv4i`
   - Virtual Path: `/toolscab_ppc60_armv4i`
   - Description: `ppc60 armv4i tools cab`
   - Local Application Directory: Click **Browse**. Select the path where the tools CAB file exists.

   Click **Next**. The files are now listed under the specified directory.
5. Delete the file entries other than those for the `tools` CAB.
6. Click **Next** until you reach the DDL panel.
7. Click **Finish**.
8. Publish the application through one of the following options:
   - Select **Create Files** option to create a JAR file.
   - Select **Publish the Current Application** option on the server.

3.2.3.2 Assigning the Tools CAB to the User

Once published, assign the application containing the Tools CAB to the Mobile client user:

1. Select the Applications tab in the Mobile Manager.
2. Click on the Tools CAB file description. In our previous example, this was `tools cab ppc60.armv4i`.
3. Select the Access tab.
4. Select the user or group to which this application should be available for download.
5. Click **Save** to save the preferences.

3.3 Configuring for Default Sync When Installing the Client

In the default configuration, all Mobile clients do not automatically synchronize after you install the client. However, you can modify your configuration to automatically sync each client after it is installed, as follows:

1. Logon to the Mobile Server as an Administrator and launch the Mobile Manager tool.
2. Click on Mobile Devices, followed by Administration.
3. Click on Command Management.
4. Edit the Command Device Info (Retrieve device information).
5. Insert 'Synchronize' as a Selected Command and click **Apply** to accept the changes.

See Section 7.6, "Sending Commands to Your Mobile Devices" in the *Oracle Database Lite Administration and Deployment Guide* for more details on sending commands to your Mobile device.

3.4 Configuring the Client for Secure Socket Layer (SSL)

As the end user, you can configure the Mobile client for OC4J or Web-to-Go to establish an SSL connection between the Mobile client and the Mobile Server. A complete description of how to configure your Mobile client to use SSL is described in Section 11.4.6, "Client-Side Configuration for Secure Socket Layer (SSL)" in the *Oracle Database Lite Administration and Deployment Guide*.

3.5 Specifying Whether the Client Uses a Static or Dynamic (DHCP) IP Address

Use the `IP_CONFIG` parameter in the server `webtogo.ora` file to specify the method the client uses to retrieve its IP address. Your client device can use either a static IP address or a dynamic (DHCP) method in retrieving an IP address. If you are using DHCP, then you need to set this parameter to `DYNAMIC`; the default is `STATIC`.

If you are using DHCP, then the underlying code needs to know to not use the IP address that was used for the previous connection/synchronization. If you are using DHCP and have set this parameter to `STATIC`, your synchronization may never occur, since it is probably trying to synchronize to an IP address that is no longer valid for this device.

You set this parameter in the server `webtogo.ora` file, so that the Mobile Server knows if the client is DHCP, then may have a different IP address each time.

For more information, see Section A.2, "WEBTOGO" in the *Oracle Database Lite Administration and Deployment Guide*.

3.6 Using Offline Instantiation to Distribute Multiple Mobile Clients

You can enable your users to install their client using a distribution method, such as a CD, through the network, or email. To install the Mobile client and perform the first synchronization to retrieve the applications (with the initial data) can be a performance issue. In this case, the administrator pre-creates the Mobile binaries with the user ODB files (includes the applications and data for the user) to the client. The download of this package is faster than having each user perform the first synchronization on their device. Thus, this procedure helps users avoid an expensive performance hit when creating and synchronizing the Mobile client for the first time.

Offline instantiation is a tool that enables an administrator to gather and package the Mobile client binaries and the user applications and data into a single directory. Offline instantiation is part of the Mobile Development Kit, which can be installed only on a Windows platform. Thus, you create all of your user distribution files on a Windows machine and you can only create multiple user distribution files for OC4J, Web-to-Go, Branch Office, Win32, and WinCE Mobile clients. We recommend that you use the same Windows environment where a Mobile server exists to create your distribution files.

See Chapter 10, "Offline Instantiation" in the *Oracle Database Lite Administration and Deployment Guide* for full instructions on how to use the Offline Instantiation engine to create and deploy multiple clients.

4

Building an Embedded Application

Perform the following when building an application with an embedded Oracle Lite database:

- Create the Oracle Lite database.
- Create the users.
- Package your application.

The following sections describe how to perform these tasks:

- Section 4.1, "Creating the Oracle Lite Database"
- Section 4.2, "Creating Users for the Oracle Lite Database"
- Section 4.3, "Packaging Your Embedded Application With the Oracle Database Lite Runtime"
- Section 4.4, "Connecting to the Oracle Lite Database"
- Section 4.5, "Using Oracle Database Lite Samples"

4.1 Creating the Oracle Lite Database

To create the Oracle Lite database, you must first create a data source name (DSN) for the database and then create the database. This is described in the following sections:

- Section 4.1.1, "Creating a Data Source Name with ODBC Administrator"
- Section 4.1.2, "Creating a New Oracle Lite Database"

4.1.1 Creating a Data Source Name with ODBC Administrator

The data source name (DSN) points to the physical location of the ODB file. The DSN is used when creating the Oracle Lite database (ODB) file. How you create the DSN is platform-dependent, as described in the following sections:

- Section 4.1.1.1, "Creating DSN on a Windows System"
- Section 4.1.1.2, "Creating DSN on a LINUX System"

4.1.1.1 Creating DSN on a Windows System

Create the DSN on a Windows system through the Microsoft ODBC Administrator, which is a tool that manages the `ODBC.INI` file and associated registry entries in Windows 2003/XP. Within this tool, add the data source name for your ODB file and specify the database file you want to dedicate as the default for the data source name.

The `ODBC.INI` file is available in Windows under `%WINDIR%`. For more information on DSN properties, see Table 4–1 and Table 4–2.

Note: The name of the ODB file is used in the next step: Section 4.1.2, "Creating a New Oracle Lite Database". For more information on the ODBC Administrator, and for instructions on creating a data source name using the tool, refer to Appendix C.8, "ODBC Administrator and the Oracle Database Lite ODBC Driver".

4.1.1.2 Creating DSN on a LINUX System

In order to create a DSN on a LINUX platform, add the DSN in the `ODBC.INI` file. In this file, add the DSN in its own section, where the section name is the DSN name. The `ODBC.INI` file is available in Linux under `$OLITE_HOME/bin`. For the Linux platform, you must have write permissions on the directory where this is located to be able to modify them.

For example, the following `ODBC.INI` example contains two DSN configurations:

- The `Polite` DSN configuration is for a single Oracle Lite database installed on the client.
- The `Politecl` DSN configuration describes a multi-user service DSN, as shown with the `ServerHostName` and `ServerPortNumber` elements. This service is described further in Chapter 5, "Building a Client/Server Environment".

```
[Polite]
Description=Oracle Lite 40 Data Source
Data_Directory=/home/olite
Database=polite
IsolationLevel=Read Committed
Autocommit=Off
CursorType=Forward Only

[Politecl]
Description=Oracle Lite 40 Data Source
Data_Directory=/home/olite
ServerHostName=localhost
ServerPortNumber=1160
Database=polite
IsolationLevel=Read Committed
Autocommit=Off
CursorType=Static
```

The default port number is 1160.

The parameters that you can use are listed in Table 4–1:

Table 4–1 ODBC.INI DSN Parameters

DSN Parameter	Description
`Description`	An optional description for the data source. Use only for Windows environment.
`Data Directory`	The path to the data directory where the database resides. This is an existing path.
`Database`	Oracle Database Lite database name to be created. Do not include the `.ODB` extension.

Table 4–1 (Cont.) ODBC.INI DSN Parameters

DSN Parameter	Description
`Default Isolation Level`	Determines the degree to which operations in different transactions are visible to each other. For more information on the supported isolation levels, refer to Section 15.2, "What Are the Transaction Isolation Levels?". The default level is `Read Committed`. Other options are `Repeatable Read`, `Single User`, and `Serializable`.
`Autocommit`	Commits every database update operation in a transaction when that operation is performed. Auto-commit values are Off and On. The default value is Off. ■ On: DML and DDLs are automatically committed. ■ Off: An application has to explicitly issue the transaction commit or rollback commands. **Note**: In the Microsoft ODBC SDK, the ODBC driver defaults to auto-commit mode. However, the default for Oracle Database Lite is manual-commit mode. In this environment, if you execute SQLEndTrans / SQLTransact call with SQL_COMMIT option using the ODBC driver, you receive a SQL_SUCCESS, because ODBC believes that auto-commit is on. However, no commit actually occurs, because ODBC transfers the transaction to Oracle Database Lite, whose default is manual-commit. You must configure the Microsoft ODBC Driver Manager to transfer control of the SQLEndTrans / SQLTransact API call to Oracle Database Lite by explicitly setting autocommit to OFF in ODBC. When you do this, ODBC does not try to autocommit, but gives control of the transaction to Oracle Database Lite. To set auto-commit to off, execute either the `SQLSetConnectAtrr` or `SQLSetConnectOption` method with `SQL_AUTOCOMMIT_OFF` as the value of the `SQL_AUTOCOMMIT` option. Then, the `SQLEndTrans / SQLTransact` calls will commit as defaulted within Oracle Database Lite. Thus, if you want auto-commit on, turn it on only within Oracle Database Lite.
`Cursor Type`	■ *Forward Only*: Default. A non-scrollable cursor which only moves forward but not backward through the result set. As a result, the cursor cannot go back to previously fetched rows. ■ *Dynamic*: Capable of detecting changes to the membership, order, or values of a result set after the cursor is opened. If a dynamic cursor fetches rows that are subsequently deleted or updated by another application, it detects those changes when it fetches those rows again. ■ *Keyset Driven*: Does not detect change to the membership or order of a result set, but detects changes to the values of rows in the result set. ■ *Static*: Does not detect changes to the membership, order or values of a result set after the cursor is opened. If a static cursor fetches a row that is subsequently updated by another application, it does not detect the changes even if it fetches the row again. See Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types" for details on the restrictions when combining cursor types and isolation levels.

If your DSN connects to a multi-user service—see Chapter 5, "Building a Client/Server Environment"—then the DSN entries have the following additional parameters:

Table 4–2 DSN Configuration Parameters for Multi-User Service on LINUX

Parameter	Description
`ServerHostName`	Provide the server machine hostname or IP address where the database service is running.
`ServerPortNumber`	The port number where the database service is listening for incoming requests. The default port number is 1160.
`ServerDSN`	The server-side DSN. Thus, the client DSN name on the client machine can be different from the DSN on the server mahcine. This is required only if the client and server machines are not the same and the `Database Directory` and `Database` parameters are not required.

4.1.2 Creating a New Oracle Lite Database

To create a new Oracle Lite database, use the `CREATEDB` command-line utility providing the DSN name, the database name, and the system user password, as follows:

```
CREATEDB myDSN myDBname sysPwd
```

For example, if the name of the DSN is `POLITE`, the ODB name is `myDB`, and the system user password is `MANAGER`:

```
CREATEDB polite mydb manager
```

See Appendix C.2, "CREATEDB" for more information.

The new database file is located in the `<`*`ORACLE_HOME`*`>\Mobile\Sdk\oldb40` directory. For ease of maintenance, it is recommended that you use one database directory for all of your Oracle Lite databases.

4.2 Creating Users for the Oracle Lite Database

When you create a user, Oracle Database Lite creates a schema with the same name and automatically assigns it to that user as the default schema. Thus, the user can access database objects in its schema without prefixing them with the schema name. Users with the appropriate privileges can create additional schemas with the `CREATE SCHEMA` command.

You connect to the database with the username. All schemas are owned by the user who created them. If the schema name is different from the username, you must provide the schema name prefix in order to access objects in that schema.

When you create a database using the `CREATEDB` utility or the `CREATE DATABASE` command, Oracle Database Lite creates a special user called `SYSTEM`, which has all database privileges.

Note: For more information on the `CREATEDB` utility, see Appendix C.2, "CREATEDB".

To access data and perform operations in another user schema, a user must grant you DBA or ADMIN privileges. The `SYSTEM` user can access all data, as it automatically holds DBA and ADMIN privileges.

You can create multiple users in your Oracle Lite database for your embedded application with the `CREATE USER` command. See the *Oracle Database Lite SQL Reference* for information on how to manage your user through SQL commands.

Note: Both username and passwords are limited to a maximum of 28 characters.

While most information you need to understand about SQL and your Oracle Lite database can be gathered from the Oracle Database manuals and the *Oracle Database Lite SQL Reference*, the following sections help you understand concepts related specifically to the Oracle Lite database.

- Section 4.2.1, "Pre-Defined Roles"
- Section 4.2.2, "Building and Populating Demo Tables"

4.2.1 Pre-Defined Roles

Oracle Database Lite combines some privileges into pre-defined roles for convenience. In many cases it is easier to grant a user a pre-defined role than to grant specific privileges in another schema. Oracle Database Lite does not support creating or dropping roles. Following is a list of Oracle Database Lite pre-defined roles:

Table 4–3 Pre-Defined Roles

Role Name	Privileges Granted To Role
`ADMIN`	Enables the user to create other users and grant privileges other than `DBA` and `ADMIN` on any object in the schema: `CREATE SCHEMA`, `CREATE USER`, `ALTER USER`, `DROP USER`, `DROP SCHEMA`, `GRANT`, `REVOKE`
`DBA`	Enables the user to issue the following DDL statements which otherwise can only be issued by `SYSTEM`: All ADMIN privileges, `CREATE TABLE`, `CREATE ANY TABLE`, `CREATE VIEW`, `CREATE ANY VIEW`, `CREATE INDEX`, `CREATE ANY INDEX`, `ALTER TABLE`, `ALTER VIEW`, `DROP TABLE`, `DROP VIEW`, and `DROP INDEX`.
`RESOURCE`	The `RESOURCE` role grants the same level of control as the `DBA` role, but only over the user's own schema. The user can execute any of the following commands in a SQL statement: `CREATE TABLE`, `CREATE VIEW`, `CREATE INDEX`, `CREATE CONSTRAINT`, `ALTER TABLE`, `ALTER VIEW`, `ALTER INDEX`, `ALTER CONSTRAINT`, `DROP TABLE`, `DROP VIEW`, `DROP INDEX`, `DROP CONSTRAINT`, and `GRANT` or `REVOKE` privileges on any object under a user's own schema.

General Note: Unlike the Oracle database server, Oracle Database Lite does not commit data definition language (DDL) commands until you explicitly issue the `COMMIT` command.

4.2.2 Building and Populating Demo Tables

Oracle Database Lite comes with a script called `POLDEMO.SQL`, which enables you to build the same tables that are in your Oracle Lite default starter database (`POLITE.ODB`).

You can use SQL scripts to create tables and schema, and to insert data into tables. A SQL script is a text file, generally with a `.SQL` extension, that contains SQL commands. You can run the following SQL script from the Mobile SQL prompt.

```
SQL> @<ORACLE_HOME>Mobile\DBS\Poldemo.sql
```

You can also enter:

```
SQL> START Poldemo.sql
```

Note: You do not need to include the `.SQL` file extension when running the script.

4.3 Packaging Your Embedded Application With the Oracle Database Lite Runtime

In order to use the Oracle Lite database and embed it into your application, you must include not only the Oracle Lite database, but certain libraries in your application.

The following sections describe what libraries to include for each operating system platform:

- Section 4.3.1, "Packaging an Embedded Application on Windows"
- Section 4.3.2, "Packaging an Embedded Application on Linux"

4.3.1 Packaging an Embedded Application on Windows

To package an embedded application on Windows, perform the following:

1. Copy the following files from the Mobile Development Kit library, which is located in *`ORACLE_HOME`*`/Mobile/Sdk`, into the directory in your PATH where your application DLLs are located.
   - `olite40.msb`: Oracle Database Lite message file
   - `oljdbc40.dll`: JDBC JNI library
   - `olobj40.dll`: Oracle Database Lite object kernel
   - `olod2040.dll`: Oracle Database Lite ODBC driver
   - `olsql40.dll`: Oracle Database Lite SQL runtime library
   - `olstddll.dll`: Oracle Lite Common library
2. If you are using the Multi-User Service, copy `olsv2040.exe`, `olcl2040.dll`, and `olsvmsg.dll` into your PATH where your application DLLs are located.
3. To use any Java program with Oracle Database Lite, make sure that the `olite40.jar` file, which is installed in `OLITE_HOME/bin`, is in the application `CLASSPATH`. If the Java program uses the multi-user service, also place this JAR file in the SYSTEM `CLASSPATH`. This JAR file contains the JDBC driver for Oracle Database Lite. Your environment must provide a Java Runtime Environment from Sun, version JDK 1.4.2 version or higher.
4. If you want to support the mSQL command-line tool for querying and managing the Oracle Lite database, then you must place the following files in the `PATH`:
   - `msql.dll`
   - `msql.exe`

- msql.jar

5. Manage ODBC for creating the DSN and registering the ODBC driver. On Linux, modify the ODBC.INI file. On Windows, perform the following:

   a. To use Microsoft ODBC for the ODBC environment—including DSN creation support—or to create and manage DSN names programmatically, place the olad2040.dll, olAES.dll, olCast5.dll, olil125x.dll, olr24US.dll, mfc71.dll, msvcp71.dll, and msvcr71.dll in the PATH. The olad2040.dll provides a plug-in to programmatically access the ODBC administration tool—odbcad32—that is used to create DSNs.

   b. Register the ODBC driver for the product in the Windows Registry, as follows:

   ```
   KEY:HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI\Oracle Lite 40 ODBC
   Driver
   VALUE:32Bit = 1
   VALUE:ApiLevel = 0
   VALUE:ConnectFunctions = YYN
   VALUE:Driver = <path to olod2040.dll>
   VALUE:DriverODBCVer = 02.00
   VALUE:SQLLevel = 0
   VALUE:Setup = <path_to_olad2040.dll>
   KEY:HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI\ODBC DRIVERS
   VALUE:Oracle Lite 40 ODBC Driver = Installed
   ```

6. Configure the POLITE.INI file and place it in the system Windows directory, such as c:\winnt, as follows:

   ```
   [All Databases]
   NLS_LANGUAGE=ENGLISH
   NLS_LOCALE=ENGLISH
   DB_CHAR_ENCODING=Native
   DATA_DIRECTORY=<default_directory_to_create_database_files>
   ```

   Note: See Appendix A, "POLITE.INI Parameters for the Oracle Lite Database" for more information on how to configure the POLITE.INI file.

4.3.2 Packaging an Embedded Application on Linux

To package an embedded application on Linux, perform the following:

1. Define *OLITE_HOME* as the installation directory where your application resides. All Oracle Database Lite files are located in the *OLITE_HOME*/bin directory. Include the *OLITE_HOME*/bin directrory in the PATH variable as well as in the LD_LIBRARY_PATH variable.

2. Copy the following files from the Mobile Development Kit library located in *ORACLE_HOME*/Mobile/Sdk into *OLITE_HOME*/bin.

   ```
   olite40.msb
   liboljdbc40.so
   libokapi.so
   libolobj40.so
   libolodbc.so
   libolsql.so
   libolstd.so
   libolutil.so
   libolaes.so
   ```

```
libolcast5.so
libolil125x.so
```

3. If you are using the multi-user service, copy the following into the *OLITE_HOME*/bin directory:

```
olsv
oldaemon
libolsv2040.so
```

4. To use any Java program with Oracle Database Lite, include the `olite40.jar` file, which is installed in *OLITE_HOME*/bin, in the application CLASSPATH. If the Java program uses the multi-user service, also place this JAR file in the CLASSPATH. The `olite40.jar` file contains the JDBC driver for Oracle Database Lite. Your environment must provide a Java Runtime Environment from Sun, version 1.4.2 or higher.
5. To provide the mSQL command-line tool for querying and managing the Oracle Lite database, place the following files in the *OLITE_HOME*/bin:

```
libmsql.so
msql
msql.jar
```

6. Create the DSN and register the ODBC driver by modifying the *OLITE_HOME*/bin/odbc.ini file.
7. Configure the POLITE.INI file and place it in *OLITE_HOME*/bin as follows:

```
[All Databases]
NLS_LANGUAGE=ENGLISH
NLS_LOCALE=ENGLISH
DB_CHAR_ENCODING=Native
DATA_DIRECTORY=<default_directory_to_create_database_files>
```

Note: See Appendix A, "POLITE.INI Parameters for the Oracle Lite Database" for more information on how to configure the POLITE.INI file.

4.4 Connecting to the Oracle Lite Database

Connect to the file-based Oracle Lite database using your application or mSQL, which is a command line interface. See Appendix C.1, "The mSQL Tool" for full details.

When connecting to the database from an application, use the DSN name that you created in Section 4.1.1, "Creating a Data Source Name with ODBC Administrator". and the database name (ODB name) that you defined in Section 4.1.2, "Creating a New Oracle Lite Database".

To connect to a database with the POLITE database (ODB) name, SYSTEM user, MANAGER password, and the mydsn data source name, perform the following:

```
C:>msql system/manager@jdbc:polite:mydsn
```

Note: On WinCE, the mSQL utility is a GUI installed on your platform.

You can replace `mydsn` with a previously defined ODBC data source name. To connect to the default DSN `POLITE`, the mSQL statement would be as follows:

```
C:>msql system/manager@jdbc:polite:polite
```

Note: Review the *Oracle Database Lite SQL Reference* before using the starter database to understand the SQL used to manage information in Oracle Database Lite.

The Oracle Lite database has its own JDBC driver, which you can use to connect to the client Oracle Lite database.

- **Local Connection**—Use the following URL syntax to initiate a local connection to a client database:

  ```
  jdbc:polite:<DSN>
  ```

  The following example connects to a client Oracle Lite database with the default DSN: `polite`:

  ```
  jdbc:polite:polite
  ```

- **Remote Connection**—Use the following JDBC URL to initiate a remote connection to the client Oracle Lite database:

  ```
  jdbc:polite[@<hostname>]:[<port>]:<DSN>
  ```

  The hostname and port are optional. The following example connects to the local machine on port 1000 to the `polite` database.

  ```
  jdbc:polite@:1000:polite
  ```

These URLs default to using the type 2 JDBC driver. You can specify that the connection uses a type 4 JDBC driver. For full details on both drivers and all options for connection, see Section 10.3, "JDBC Drivers to Use When Connecting to the Oracle Lite Database".

4.5 Using Oracle Database Lite Samples

After you perform a complete installation of Oracle Database Lite, the samples are available in your `<ORACLE_HOME>\Mobile\Sdk` directory. The tools, locations for samples, and descriptions are listed in Table 4–4.

Table 4–4 Sample File Directory

Tool	Location of Sample Applications	Description
Java	`<ORACLE_HOME>\Mobile\Sdk\samples\jdbc`	Demonstrates programming with JDBC. See Chapter 10, "JDBC Programming" for more information.
ODBC	`<ORACLE_HOME>\Mobile\Sdk\samples\odbc\win32\c_samples`	Provides ODBC programs written in C See Section 9.2, "Executing the ODBC Examples".
Visual Basic	`<ORACLE_HOME>\Mobile\Sdk\samples\odbc\win32\update`	Demonstrates the ease of querying tables in Oracle Database Lite with Visual Basic application. See Section 4.5.1, "Executing the Visual Basic Sample Application" for more information.

Table 4–4 (Cont.) Sample File Directory

Tool	Location of Sample Applications	Description
Multiple Field Service	`<ORACLE_HOME>\Mobile\Sdk\samples\odbc\win32\mfs`	The Multiple Field Service sample uses ODBC to access the Oracle Lite database.

Note: Most examples use the data source name (DSN) `POLITE`. If you need to drop and recreate, use the `REMOVEDB` and `CREATEDB` utilities, which are documented in Appendix B.2, "CREATEDB" or Appendix B.3, "REMOVEDB" in the *Oracle Database Lite Client Guide*.

4.5.1 Executing the Visual Basic Sample Application

The Visual Basic Sample application example uses Visual Basic 2005 to demonstrate how to develop a Visual Basic application with Oracle Database Lite. It uses the ODBC DSN, `POLITE`. To use the `AddNew`, `Update`, and `Delete` macros, you need a unique `EMPNO` column of the `EMP` table. This is the default condition when you connect to the default database.

The following instructions for installing and running the Visual Basic sample application assume that you have already installed Oracle Database Lite and Visual Basic.

1. Section 4.5.1.1, "Open the Sample Application"
2. Section 4.5.1.2, "View and Manipulate the Data in the EMP Table"

4.5.1.1 Open the Sample Application

1. To open the sample application, select **Open Project** from the File menu of Visual Studio 2005.
2. In the dialog box, navigate to the `<ORACLE_HOME>\Mobile\Sdk\samples\odbc\win32\update` directory.
3. Select `update.vbproj`, and click **Open**.
4. Follow the instructions in `readMe.txt` in the same location to execute the sample.

4.5.1.2 View and Manipulate the Data in the EMP Table

1. To view data in the `EMP` table:
   - Click **Show** to show the `EMP` table data.
   - Click **Next** to show the next record.
   - Click **Previous** to show the previous record.
2. To manipulate data in the `EMP` table, use the Add, Update, and Delete features.

5 Building a Client/Server Environment

If you want multiple clients to access an Oracle Lite database, you need to configure a client/server environment. The following sections describe how to configure this environment with the multi-user service:

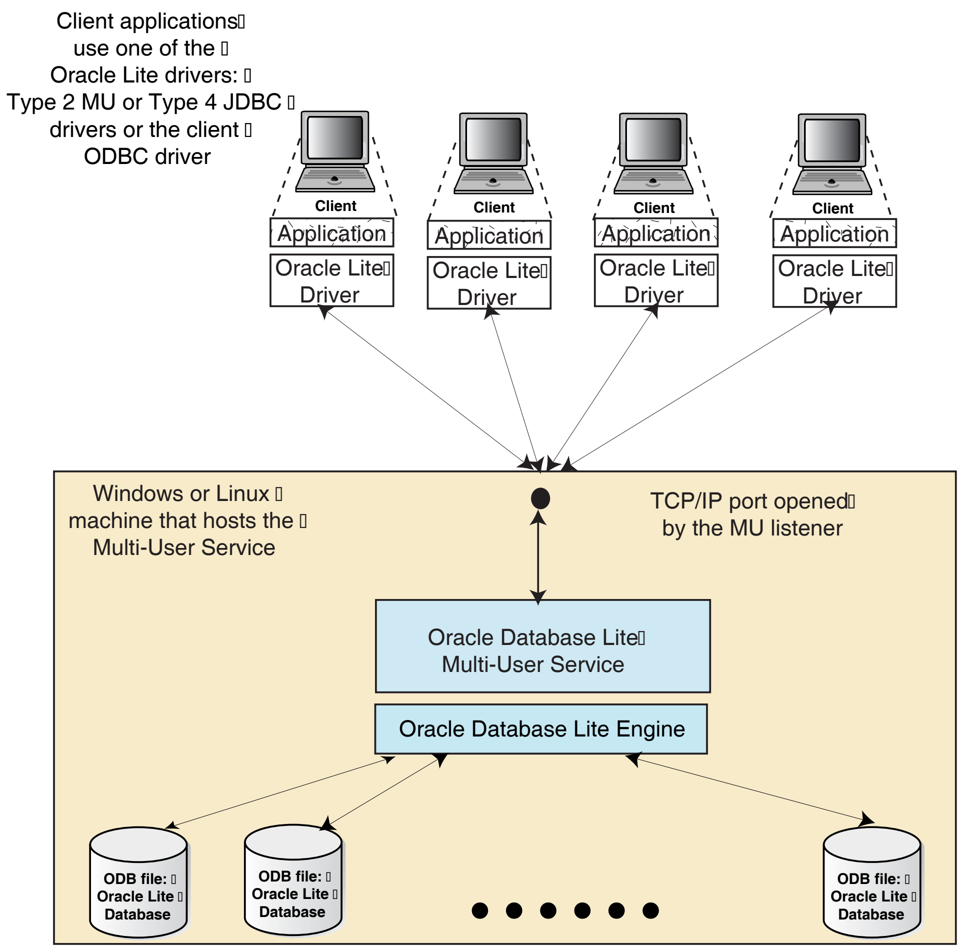
- Section 5.1, "Overview of the Multi-User Service"
- Section 5.2, "Administration for the Multi-User Service on the Windows Platform"
- Section 5.3, "Administration for the Multi-User Service on the Linux Platform"
- Section 5.4, "Debugging the Multi-User Service"
- Section 5.5, "Creating DSNs"
- Section 5.6, "Accessing the Database"
- Section 5.7, "Verifying the Connection Using mSQL"

5.1 Overview of the Multi-User Service

Multiple clients can execute an application that accesses the same database. You can protect all of your data on a centralized machine and allow clients to remotely access the information within the database.

Figure 5–1 demonstrates how to centralize multiple Oracle Lite databases (ODB files) by installing them on a Windows or Linux host machine. The Oracle Database Lite multi-user service facilitates the communication between the remote application clients by starting the multi-user service, which opens the designated ports, and then translates the DSNs to the appropriate database.

Figure 5–1 Diagram of Multi-User Service


Client applications use one of the Oracle Lite drivers: Type 2 MU or Type 4 JDBC drivers or the client ODBC driver
Client
Application
Oracle Lite Driver
Client
Application
Oracle Lite Driver
Client
Application
Oracle Lite Driver
Client
Application
Oracle Lite Driver
Windows or Linux machine that hosts the Multi-User Service
TCP/IP port opened by the MU listener
Oracle Database Lite Multi-User Service
Oracle Database Lite Engine
ODB file: Oracle Lite Database
ODB file: Oracle Lite Database
ODB file: Oracle Lite Database

The multi-user service enables you to use up to sixty-four concurrent client connections, each of which can connect to up to five `ODB` database files on the remote machine. All clients and server must install the same binary with the same NLS. The server machine with the multi-user service and each of the clients can be installed on either Windows or Linux platforms.

When you implement the client/server data access for your application, the flow of events is as follows:

1. Remote client sends the connect request to the multi-user service.

   The client connection can use the ODBC 2.0 client driver, the JDBC Type 2 MU driver, or the JDBC Type 4 driver. In addition, the client provides the following in the connection string: remote host and port where the multi-user service is listening for incoming calls and the DSN of the Oracle Lite database (ODB file) where the data is stored. The default port number is 1160.

2. Multi-user service receives incoming call with the DSN from the remote client.
3. The multi-user service parses the DSN name and completes the request by connecting to the Oracle Lite database that maps to the DSN name.

You can provide a client/server environment for multiple users accessing a single entry point for the Oracle Lite database with the multi-user service. For the multi-user service, you first create the application and the database in the same manner as the embedded application, as described in Chapter 4, "Building an Embedded

Application". Then, configure for a client/server environment and set up a listener to receive requests from each of these clients. The difference between an embedded application and the client/server options is that you will configure two DSNs: one for the client and one for the server. The DSN for the server is the same as the one configured for the embedded application.

5.2 Administration for the Multi-User Service on the Windows Platform

The following sections describe how to install, start, stop and query the multi-user service on Windows:

- Section 5.2.1, "Installation and Configuration on Windows"
- Section 5.2.2, "Starting the Multi-User Service on Windows"
- Section 5.2.3, "Stopping the Multi-User Service on Windows"
- Section 5.2.4, "Querying the Multi-User Service on Windows"

5.2.1 Installation and Configuration on Windows

To install and configure the Oracle Database Lite multi-user service, perform the following steps:

1. Ensure that you install the `olsv2040.exe` in the following directory.

   <*OLITE_HOME*>`\Mobile\Sdk\bin`

   Note: This directory must also be included in your system `PATH`.

   If not already available, re-install the MDK to retrieve the component. A sample <*OLITE_HOME*> location is `C:\Olite`.

2. To install the service, start the Command Prompt and enter the following command.

   ```
   olsv2040.exe /install [/account=AccountName][/password=ValidPassword]
       [/wdir=WorkingDirectory] [/port=ServicePort]
   ```

   where the optional parameters can be as follows:

   - `AccountName`: Provide either the `DomainName\UserName` or `.\UserName`.
   - `ValidPassword`: If you specify an account, but don't want to give out the password in command prompt during service installation. You can provide the password to the "Log On" page of "Oracle Lite Multiuser Service Properties" dialog box which can be found in Services.
   - `WorkingDirectory`: If you use '`.`' in SQL scripts that load Java classes, you must specify a working directory.
   - `ServicePort`: The default port number is 1160.

3. If you are using Java Stored Procedures, then perform the following to set up the environment for Java Stored Procedures:

   a. If you have the JDK installed on your PC, ensure that the system `PATH` variable includes the following:

   ```
   <JDK_HOME>\bin
   <JDK_HOME>\jre\bin
   <JDK_HOME>\jre\bin\hotspot
   ```

b. If you have *JRE* installed on your PC, ensure that the system `PATH` variable includes the following:

```
<JRE_HOME>\bin
<JRE_HOME>\bin\hotspot
```

Note: *JRE* does not include the Java compiler. Therefore, attempts to load a Java source into the database with the `CREATE JAVA SOURCE` command and the `loadjava` utility will fail without the Java compiler.

---

c. Ensure that your system `CLASSPATH` variable includes the following:

```
<OLITE_HOME>\bin\Olite40.jar and '.'
```

4. You may change the startup type from the Windows service console. Highlight the Oracle Database Lite multi-user service and select **Properties**. When required, change the startup type to manual. The property also contains startup parameters, but has not been tested.
5. Reboot your PC.

5.2.2 Starting the Multi-User Service on Windows

The Oracle Database Lite multi-user service can be started in many ways. By default, the service property "Startup Type" is automatic; thus, the service is started every time you reboot the machine. If you modify the "Startup Type" to "Manual", then you start Oracle Lite multi-user service by entering any one of the following startup commands from the Command Prompt:

- `olsv2040.exe /start`
- `net start "Oracle Lite Multiuser Service"`

5.2.3 Stopping the Multi-User Service on Windows

To stop the multi-user service, use one of the following commands:

- `olsv2040.exe /stop`
- `net stop "Oracle Lite Multiuser Service"`

5.2.4 Querying the Multi-User Service on Windows

You can query the multi-user status for the following details:

- current status
- current startup parameters
- configuration
- installed startup parameters

Issue the following command to see these details:

```
olsv2040.exe /query
```

The results of this command are as follows:

```
OliteService reports the following status:
  The service is running...
     port= 1160
     wdir = C:\WINDOWS\SYSTEM32

The current status of Oracle Lite Multiuser Service:
  Current State              : SERVICE_RUNNING
  Acceptable Control Code    : (0x1) SERVICE_ACCEPT_STOP

The configuration of Oracle Lite Multiuser Service:
  Service Type    : (0x10) SERVICE_WIN32_OWN_PROCESS
  Start Type      : SERVICE_AUTO_START
  Error Control   : SERVICE_ERROR_NORMAL
  Binary Path     : C:\Oracle\product\Mobile\Sdk\bin\olsv2040.exe
  Display Name    : Oracle Lite Multiuser Service
  Start Name      : LocalSystem

Installed Service startup parameters
  port = 1160
  wdir = \%WINDIR%\SYSTEM32
```

where the port number is 1160 and the working directory is `C:\WINDOWS\SYSTEM32`. The service is installed under the `LocalSystem` account, where the startup parameters for installation are port = 1160 and working directory = `\%WINDIR%\SYSTEM32`. In addition, the Start Type is `SERVICE_AUTO_START` and the binary path as `C:\Oracle\product\Mobile\Sdk\bin\olsv2040.exe`, which is where you installed the Oracle Database Lite multi-user service.

You can also set the default for the service port and working directory by modifying the `SERVICE_PORT` and `SERVICE_WDIR` parameters in the `polite.ini` file. However, if you do so, then it overrides any of the command-line options for port and working directory.

Note: When you execute the multi-user service with the `/debug` option, then the result of the current status from the `/query` shows that the service is stopped. Since the `/debug` option is executed in a console, the Service Control Manager does not know that the service is running.

5.3 Administration for the Multi-User Service on the Linux Platform

The following sections describe how to start, stop and query the multi-user service on Linux:

Note: There is no need to install this service on Linux.

- Section 5.3.1, "Starting and Stopping the Multi-User Service on Linux"
- Section 5.3.2, "Querying the Multi-User Service on Linux"

5.3.1 Starting and Stopping the Multi-User Service on Linux

The Oracle Database Lite multi-user service can be started or stopped with the `olsv` executable. To see all of the options, execute `olsv -help`, which displays the following:

```
$ olsv -h
Usage: olsv [option]
Options are:
-start start the server as daemon
-stop | -s stop the server
-debug | -d start the server as console app for debugging
-query | -q get the server status
-kill | -k equivalent to kill -s SIGKILL PID
-port | -p PORT execute the server on the specified port——default port is 10000
-wdir | -w DIR run/debug the server in the specified dir
-help | -h display this message
if no option specified, -start by default
```

To stop the multi-user service, use the following command:

```
olsv -stop
```

5.3.2 Querying the Multi-User Service on Linux

You can query the multi-user status with `olsv -query` which provides the following information:

- process id (PID)
- environment variables
- current status

Issue the following command to see these details:

```
olsv -query
```

The results of this command are as follows:

```
Oracle Lite Multiuser Server is running, PID = 2619
----------------------------------------------------------------------
Environment:
TERM=xterm
SHELL=/bin/csh
OLITE_HOME=/scratch/myuser/oracle/OraHome/mobile/sdk
JDKDIR=/usr/local/packages/jdk14
MOBILE_CLIENT=/scratch/myuser/mobileclient
USER=myuser
LD_LIBRARY_
PATH=/usr/local/packages/jdk14/jre/lib/i386:/usr/local/packages/jdk14/jre/lib/i386
/server:/scratch/myuser/olite/lib
HOSTTYPE=i386-linux
JAVA_HOME=/usr/local/packages/jdk14
LANG=en_US.UTF-8
HOME=/home/myuser
OSTYPE=linux
LOCAL_PACK=/usr/local/packages/icc_remote
JAVA_HOME=/usr/local/packages/jdk14
VENDOR=intel
MACHTYPE=i386
ORACLE_HOME=/scratch/user/oracle/OraHome
CLASSPATH=.:/scratch
----------------------------------------------------------------------
Status Summary:
Database Version: 10.3.0
Time Started: 11-27-2006 17:58:15
Listening Port: 10000
```

```
Total Connections: 0
Current Connections: 0
No errors encountered
```

You can also set the default for the service port and working directory by modifying the `SERVICE_PORT` and `SERVICE_WDIR` parameters in the `polite.ini` file. However, if you do so, then it overrides any of the command-line options for port and working directory.

5.4 Debugging the Multi-User Service

If the service the does not start, debug the service using the following method:

1. Edit the `POLITE.INI` file, which is available in Windows under `%WINDIR%\POLITE.INI` and in Linux under `$OLITE_HOME/bin`, to add the following entries in the `[ALL_DATABASES]` section:
   - `OLITE_SERVER_TRACE=TRUE`
   - `OLITE_SERVER_LOG=<filename>`. This is used for the LINUX platform only.

   Note: For more details on these parameters, see Section A.2, "All Databases Section".

2. Should the service fail, the multi-user service generates the `olsv.log` file in the current working directory. Ensure that the `PATH` and `CLASSPATH` variables are accurate and that the `PATH` includes the directory that contains `jvm.dll`.
3. Correct the cause and retry.

5.5 Creating DSNs

To access the database using an ODBC or Visual Basic application, you must create the DSN enabled from the embedded connection. When you add a DSN using the ODBC Administration tool, choose the **Oracle Lite 40 ODBC Driver(Client)**, which creates a client DSN. If you are executing the service on the same machine where the client application is running, leave the Database Host Name, Database Port Number, and Database Host DSN value empty. The remaining values must be included in the same manner as the 'Oracle Lite ODBC Driver' DSN. If you start the service on a port other than 1160, then you must specify the Database Port Number.

5.6 Accessing the Database

To access the database, you need not make any changes to the ODBC or VisualBasic application. The DSN automatically routes the request to the service through the ODBC driver `olcl2040.dll`. For a JDBC application, change the URL for the connect string, which is similar to the one used while connecting to the database using mSQL.

5.7 Verifying the Connection Using mSQL

Using the Command Prompt, verify the connection to the multi-user service in the following ways:

Connect to `A-DSN` on a Windows local host on port 1160.

```
msql system/passwd@jdbc:polite@::a-dsn
```

Connect to `A-DSN` on the local host on port 1000.

```
msql system/passwd@jdbc:polite@:1000:a-dsn
```

Connect to `A-DSN` on a Windows local host on port 1160 using the Type4 JDBC driver.

```
msql system/passwd@jdbc:polite4@::a-dsn
```

Note: Oracle Database Lite supports Type2 and Type4 JDBC drivers. Type4 is a pure Java JDBC driver that communicates with the service in the Oracle Database Lite network protocol.

For more information on Type 2 and Type 4 JDBC drivers connecting to Oracle Database Lite, see Chapter 10, "JDBC Programming". For details on mSQL, see Chapter C.1, "The mSQL Tool".

6

Managing Your Oracle Lite Mobile Client

One of the benefits of Oracle Database Lite is that you can have an application downloaded onto a device, where data can be synchronized between the device and the back-end Oracle database.

In general, the types of Oracle Lite Mobile clients are as follows:

- Web clients (such as Windows Web-to-Go over OC4J, Linux Web-to-Go, Windows Web-to-Go, Branch Office, and BC4J): The application built for these clients uses a Java browser.
- Linux, Win32, and WinCE clients: These applications are client/server applications. Thus, start the application as you would start any application on these platforms.

The following sections detail how to set up your Mobile client device and how to use the Oracle Database Lite technology on that device:

- Section 6.1, "Start the Mobile Client"
- Section 6.2, "Log on to Mobile Client Workspace"
- Section 6.3, "Synchronize or Execute Applications on the Mobile Client"
- Section 6.4, "Manage the Mobile Client"

6.1 Start the Mobile Client

The following details how to start the Oracle Lite Mobile client:

- Web clients (such as Windows Web-to-Go over OC4J, Windows Web-to-Go, Linux Web-to-Go, Branch Office, and BC4J): The application built for these clients uses a Java browser. You can initiate the application or synchronization using the Mobile Workspace GUI.
- Linux, Win32, and WinCE clients: These applications are client/server applications. Thus, start the application as you would start any application on these platforms. You can initiate synchronization either by implementing it within your application using the synchronization APIs (see Chapter 2, "Synchronization" and Chapter 3, "Invoking Synchronization APIs from Applications" in the *Oracle Database Lite Developer's Guide* for more information) or by executing the `msync` executable, described in Section 6.4.2, "Use the mSync GUI to Initiate Synchronization of Your Linux, WinCE, and Win32 Applications".

For Win32 and WinCE devices, you do not have to perform anything extra to start the Mobile client. However, for the Linux and Windows-based clients (OC4J and Web-to-Go), you may have to perform an extra step.

When you installed the Mobile client on Linux or Windows, it configured that the Mobile client should always be started automatically when the device is initiated. So, most of the time, you do not have to do anything to start the Mobile client. However, if you have a failure, you can manually start the Mobile client, as follows:

- Web Mobile client: If using the OC4J container for your Web applications, then start the Web OC4J Mobile client and its OC4J container by executing `runmobileclient`. If you want to use the Oracle Database Lite servlet container that was used prior to Release 10.3, then start the Web-to-Go Mobile client by executing the `webtogo` executable.

  Both executables are located in the `<mobile_client>/bin` directory.
- Linux Mobile client: Start the Linux Mobile client by executing the `webtogo.sh` file in the `<mobile_client>/bin`.

6.2 Log on to Mobile Client Workspace

If you are using a Windows or Linux Web Mobile client, then you must have installed the appropriate Mobile client on your client machine. Upon machine startup, the Mobile client is automatically started. Then, you can connect to the Mobile Client Workspace with a browser by connecting to one of the following:

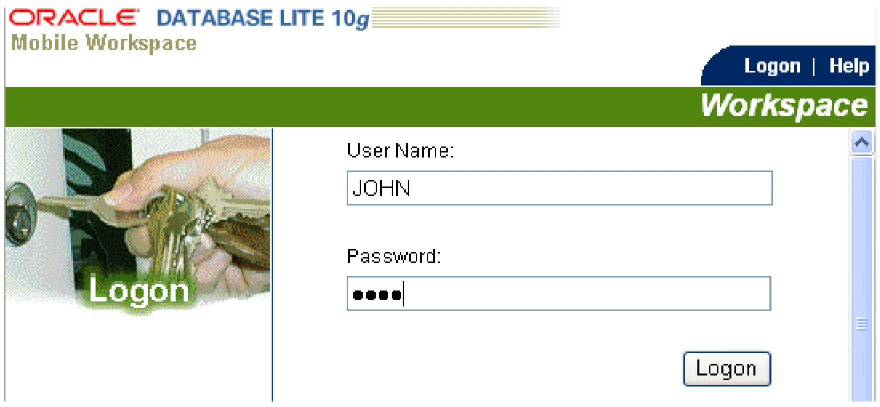
- `http://<client_machine>/webtogo`
- `http://localhost/webtogo`.

If the Windows Mobile client is unexpectedly terminated, you can start the client by double-clicking on the following icon in the corner of your Windows desktop:

For all other scenarios, launch the Mobile client by accessing your Oracle Database Lite program group and choosing the Mobile client.

Figure 6–1 displays the Mobile Client Workspace Logon page.

Figure 6–1 The Mobile Workspace Logon Page



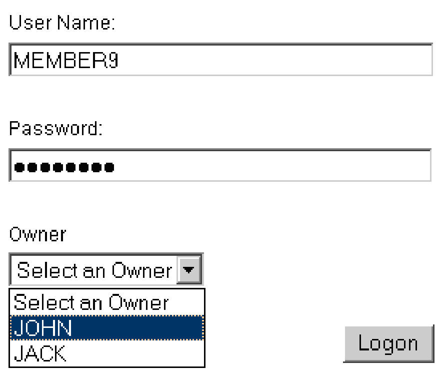
User Name: JOHN

Password: ••••

Enter the username and password for the Mobile user and click **Logon**.

However, if a member logs on who is attached to more than one user, the member must identify the user under whose context the member is acting. As shown in Figure 6–2, the member provides its name, password and selects the user under whose context he/she is acting. The user owns the data and the application, so this is required. Click **Logon**.

Figure 6–2 Member Logon Page


User Name: MEMBER9

Password: ●●●●●●●●●

Owner: Select an Owner

6.3 Synchronize or Execute Applications on the Mobile Client

The following details how to synchronize each Mobile client stack type:

- Web clients (such as OC4J, Windows Web-to-Go, Linux Web-to-Go, Branch Office, and BC4J): Synchronization can be configured to be automatic or manual. For manual synchronization, use the Sync Tab in the Mobile Workspace GUI, as described in Section 6.4.1.1.5, "Initiate Manual Synchronization".
- Linux Web-to Go clients: Synchronization can be configured to be automatic or manual. For manual synchronization, use one of the following methods:
    - Use the Sync Tab in the Mobile Workspace GUI, as described in Section 6.4.1.1.5, "Initiate Manual Synchronization"
    - Execute the `msync` executable, described in Section 6.4.2, "Use the mSync GUI to Initiate Synchronization of Your Linux, WinCE, and Win32 Applications".
    - Implement synchronization within your application using the synchronization APIs (see Chapter 2, "Synchronization" and Chapter 3, "Invoking Synchronization APIs from Applications" in the *Oracle Database Lite Developer's Guide* for more information)
- Linux, Win32, and WinCE clients: Initiate synchronization through one of the following methods:
    - Execute the `msync` executable, described in Section 6.4.2, "Use the mSync GUI to Initiate Synchronization of Your Linux, WinCE, and Win32 Applications".
    - Implement synchronization within your application using the synchronization APIs (see Chapter 2, "Synchronization" and Chapter 3, "Invoking Synchronization APIs from Applications" in the *Oracle Database Lite Developer's Guide* for more information)

Note: The Mobile client device clock must be accurate for the time zone set on the device before attempting to synchronize. An inaccurate time may result in the following exception during synchronization: `CNS: 9026 "Wrong username or password. Please enter correct value and reSync."`

When you initiate a synchronization from the client, either manually or by scheduling a job, the synchronization cannot occur if there is an active connection with an uncommitted transaction opened from another source. This could be from scheduling two jobs to synchronize at the same time, from mSync, mSQL, Web-to-Go or the client synchronization APIs.

The first synchronization for the Mobile client creates several Oracle Lite database (ODB) files on your client device for storing either Oracle Database Lite information or your application information. These ODB files are stored in the `<ORACLE_HOME>\Mobile\SDK\Oldb40` directory, as follows:

```
consroot.odb
webtogo.odb
<username>\acl.odb
          \conscli.odb
          \A<number>.odb
```

The following describes the purpose and access for these ODB files:

- The `consroot`, `acl`, and `conscli` ODB files are used for Oracle Database Lite data. The `webtogo.odb` file is only present when using a Web-to-Go application. If you want to access the information within these ODB files, use `SYSTEM/MANAGER` as the username/password.
- The snapshot data for the application is contained within the `A<number>.odb` file, where `<number>` is a randomly-generated number. You can specify the name of this ODB file when you create your publication; see Chapter 5, "Using Mobile Database Workbench to Create Publications" in the *Oracle Database Lite Developer's Guide* for details.

  If you want to access the snapshot data directly using `msql` or any other SQL tool, use `SYSTEM` as the username and provide the password for the user that owns the snapshot. For example, if the user `John/Foo` initiated the synchronization, to access the `A<number>.odb` file for this snapshot, you would use `SYSTEM/Foo` as the username/password.

6.4 Manage the Mobile Client

There are several tools that you can use on the client to manage functionality. The following sections describe other tools that you can use in each platform:

- Section 6.4.1, "Manage Your Clients Locally With the Mobile Client Workspace"
- Section 6.4.2, "Use the mSync GUI to Initiate Synchronization of Your Linux, WinCE, and Win32 Applications"
- Section 6.4.3, "Reset the Mobile User Password"
- Section 6.4.4, "Use the Device Manager Client GUI to Manage the Client-Side Device"
- Section 6.4.5, "Initiate Updates for the Oracle Lite Client"
- Section 6.4.6, "Configure JAVA_HOME for Web-to-Go Clients"
- Section 6.4.7, "Defragmentation and Reducing Size of the Client Application Databases"
- Section 6.4.8, "Communicate Between the Internet and Intranet Through a Reverse Proxy"

6.4.1 Manage Your Clients Locally With the Mobile Client Workspace

Start the Mobile Client Workspace GUI tool for managing OC4J, Windows Web-to-Go, Linux Web-to-Go, BC4J, or Branch Office Mobile clients. The Mobile Client Workspace displays only the relevant functionality for the user that logs in. Use the Mobile Client Workspace to manage the application or to initiate synchronization on the client-side device.

- Section 6.4.1.1, "Instructions for Using the Mobile Client Workspace"
- Section 6.4.1.2, "Execute Mobile Applications Installed on Your Mobile Client"
- Section 6.4.1.3, "Customize the Mobile Client Workspace"
- Section 6.4.1.4, "Schedule Data Synchronization Jobs"

6.4.1.1 Instructions for Using the Mobile Client Workspace

The Mobile Client Workspace provides you access to your Mobile applications through hyperlinks in a Web browser. The following tabs are available when you use the Mobile Client Workspace for a Mobile client:

- Section 6.4.1.1.1, "Display Installed Applications"
- Section 6.4.1.1.2, "Configure the Mobile Client"
- Section 6.4.1.1.3, "Enable Remote Access for Mobile Client"
- Section 6.4.1.1.4, "Configure Application Synchronization Default"
- Section 6.4.1.1.5, "Initiate Manual Synchronization"
- Section 6.4.1.1.6, "Log Off"

6.4.1.1.1 Display Installed Applications The Applications tab displays the list of applications that have been installed on the client, which you can execute. To access a Mobile application, click the icon or application name. The Mobile Client Workspace allows you to execute multiple applications concurrently in separate browsers.

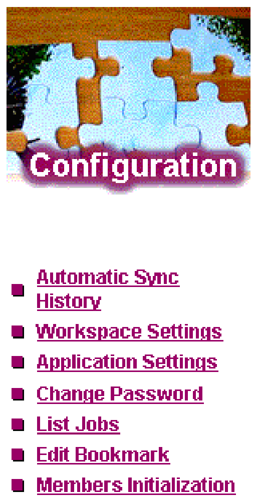
Note: When you log on as an Administrator, the Mobile Manager is listed as the available application.

6.4.1.1.2 Configure the Mobile Client To modify your Workspace configuration, select the Configuration tab.

Note: When you select the Configuration tab, maximize the window to your display. If you do not, you might not be able to see all of the configuration options on the left.

The configuration options appear on the left as shown in Figure 6–3:

Figure 6–3 Configuration Options for Client Workspace



Note: For member users, only the Change Password option is shown.

These options are described in the following sections:

- Automatic Sync History
- Workspace Settings
- Application Settings
- Change Password
- List Jobs
- Member Initialization

Automatic Sync History

This screen shows you the history for all automatic synchronization events that occurred for your Mobile client. The following information is displayed about each event:

Table 6–1 History of Automatic Synchronization Events for Mobile Client

Label	Description
Event code	An internal code that supplies more information to Oracle Support, if called.
Event type	An icon showing the outcome of the synchronization, as follows: ■ An `X` demonstrates there was an error. ■ An exclamation point indicates a warning. ■ An `i` indicates the synchronization was successful.
Timestamp	Date and time of the automatic synchronization.
Event Description	A message that describes the outcome of the automatic synchronization between your Mobile client and the back-end Oracle database.

Table 6–1 (Cont.) History of Automatic Synchronization Events for Mobile Client

Label	Description
Delete	Check the delete checkbox and click **Save** if you no longer wish to see the history for this event.

In addition, you can perform the following:

- If you have more than a single page of events listed, click Previous and Next to view all events.
- To delete one or more events, select the Delete checkbox of these items and then click **Save**.
- To delete all events on this screen, click **Select All** and then click **Save**.
- To deselect all Delete checkboxes, click **Clear All**.

This screen displays what automatic synchronization events took place in the past and the results of these events.

Note: To configure an application to use the default setting for automatic synchronization, see Section 6.4.1.1.4, "Configure Application Synchronization Default".

Workspace Settings

Configure your Mobile Client Workspace settings, such as display options and client options. Table 6–2 discusses the Web client settings:

Table 6–2 Workspace Configuration Options

Label	Description
Display icons?	Enables you to display Web application icons. To display Web application icons, select **Yes**. If you do not want to display Web application icons, select **No**.
Display description?	Enables you to display Web application descriptions. To display Web application descriptions, select **Yes**. If you do not want to display Web application descriptions, select **No**.
Applications per row	Enables you to specify the number of applications arranged in a horizontal row in the Workspace.
Use default settings for synchronization?	If you select this option, all applications (including new applications) are synchronized at once. If you de-select this option, then you can choose which applications synchronize—for manual synchronization only. Using this list, you can reduce sychronization time by selecting which applications you want to synchronize. In this mode, you are notified when new applications become available.
Mobile Client for OC4J or Web-to-Go Mode	Selecting **Always Offline** enables you to work continuously. If you want to use the backwards compatible option of offline/online, select the **Online/Offline** option.
Ask before ugrading the Mobile Client for OC4J or Web-to-Go?	Selecting this option instructs the Mobile client to ask you before downloading newer software versions for the Mobile client. If you do not select this option, your Mobile client is automatically upgraded the next time you synchronize if a new version is available.

Table 6–2 (Cont.) Workspace Configuration Options

Label	Description
Enable Automatic Sync?	By default, this is enabled. If checked, then automatic sync is automatically enabled. Uncheck if you want to manually synchronize this client yourself.
Number of Automatic Sync History Messages to be displayed per page.	This defines the number of history records to display on the Automatic Sync History page. By default, this value is 25.
Enable File-Based Sync?	By default, this is disabled. If checked, then file-based sync is enabled. Uncheck if you want to synchronize this client over the network. For details about file-based synchronization, see Section 5.8, "Synchronizing to a File with File-Based Sync" in the *Oracle Database Lite Administration and Deployment Guide*.

Application Settings

Select the applications that you want to synchronize by default. In the default synchronization mode, the Mobile Server synchronizes your client applications automatically with the Mobile Server. See Section 6.4.1.1.4, "Configure Application Synchronization Default" for full details.

Change Password

The password is stored on both the client and the Mobile server. To ensure that the password is modified on both the client and the Mobile server, only change the password using the Client Workspace when you have a connection to the Mobile Server. See Section 6.4.3, "Reset the Mobile User Password" for more details.

List Jobs

You can schedule a time when synchronization is to be initiated on the client. This is configured as a job initiated by the client. There is a small job engine on the client, so when you set up a job to execute at a certain time and interval, it initiates the application at the specified time. The only job that you can schedule on the client is for synchronization. You should not schedule multiple client synchronization jobs for the same time on the client. In fact, you should make sure that all other connections to the database, such as mSQL, JDBC, and so on, are closed before starting the synchronization. For more information on scheduling jobs on the server-side, see Chapter 6, "Managing Jobs with the Job Scheduler" in the *Oracle Database Lite Administration and Deployment Guide*.

Scheduled replication jobs will not run on client if there are any pending transactions that have not been committed on that client.

Member Initialization

After the Mobile client is installed and the user completes the first synchronization, you can use the member initialization screen to perform a batch initialization of any selected members. Each of these members belong the the user that logged into the client Workspace. This can be performed at any time after the first synchronization assuming that there is an available network connection to the Mobile Server.

Note: The other method for member initialization is automatically performed when the member requests a manual synchronization.

After the successful authentication of each user, the schema for that user will be created in the database using the password provided by the user. Once the schema is created, the member may login to the system without requiring any connectivity to the Mobile Server.

This is useful where connectivity to the Mobile Server is not always readily available; thus, this option enables you to initialize all desired members at the most opportune time.

To initialize one or more members of this user, perform the following:

1. Select the checkbox for all members you want to initialize.
2. Enter the password for all selected members, where the password of the member should be same as the one set during creation of the member.
3. Click **Save**.

If any of the members fail to initialize, the confirmation page displays the members that failed to initialize. The only reason that this can occur is if the incorrect password is provided. Provide the correct password and try again.

If the user, the owner of these members, wishes to disable any member, then this user should perform the following:

1. Select the Disable checkbox next to each member that you want to disable. Alternatively, if you want to enable any disabled member, unclick the Disable checkbox next to the desired members.
2. Click **Save**.

The Reset link discards any options selected on the Members Initialization page and reload the first page.

6.4.1.1.3 Enable Remote Access for Mobile Client To block a remote machine from getting access to the Mobile Client for Web, set the `DISABLE_REMOTE_ACCESS` parameter in the client-side `webtogo.ora` file to `YES`. Once this parameter is set to `YES` and the Mobile client is restarted, only the request coming from the local machine is served by the Mobile client listener. Any other request is blocked and not served.

For Mobile Client for OC4J, this parameter will not turn off remote access, as the access is controlled by the OC4J layer, not the Oracle Database Lite layer. For Branch Office clients, this parameter must be set to `NO`, as all clients must have remote access. For more information on the `DISABLE_REMOTE_ACCESS` parameter, see Section A.2, "[WEBTOGO]" in the *Oracle Database Lite Administration and Deployment Guide*.

Note: The `DISABLE_REMOTE_ACCESS` parameter only works for Web-to-Go and Branch office clients.

However, if you want to disable the remote access for the Mobile client for OC4J, then perform the following in the OC4J configuration file on the Mobile client:

Open the `<mobile_client_home>\mobile_client_oc4j\j2ee\mobileclient\application-deployments\mobileclient\webtogo\orion-web.xml` file.

Add the following lines after the the `<orion-web-app>` section:

```
<access-mask default="deny">
     <ip-access ip=<ip_address> mode="allow"/>
     <host-access domain="localhost" mode="allow"/>
```

```
</access-mask>
```

6.4.1.1.4 Configure Application Synchronization Default

The Application Settings option allows you to select the applications that you want to assign to the default synchronization setting. In the default mode, the Mobile client synchronizes your client application with the Mobile Server when the user requests synchronization. Selecting fewer applications decreases the amount of data to download and speeds up the synchronization process.

To select the applications for synchronization, click **Application Settings**. The application settings page appears in the right frame of the Mobile Workspace and displays the following information:

Table 6–3 describes the Application Settings page.

Table 6–3 Application Settings Page Description

Label	Description
Synchronize	If selected, the Mobile client synchronizes your client application with the Mobile Server when the user initiates synchronization.

Select the Synchronize check box next to the application and click **Save**. If you make an error, click **Reset** to return to the previous settings.

6.4.1.1.5 Initiate Manual Synchronization

The Sync tab enables you to synchronize data with the Oracle database. By default, when you click Sync, the Mobile Client for Web-to-Go is connected to the Mobile Server and synchronizes data between them. In addition, application updates and new applications accessible to the user are downloaded from Mobile Server.

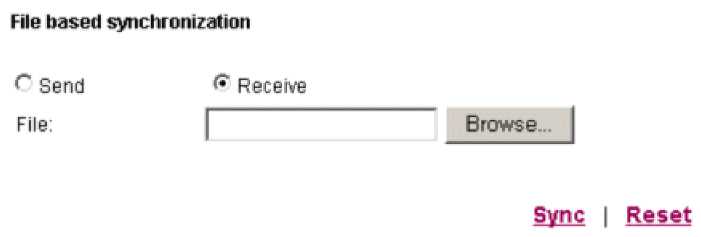
In this mode, you can work continuously whether you have a connection or not. You only need a connection to the Mobile Server when you synchronize your data and applications.

However, if you do not have connectivity, you can perform a file-based synchronization. That is, if you are not able to synchronize because of a lack of network availability, you can save your transactions in an encrypted file, which can be manually copied over to the Mobile Server. Then, when the Mobile Server has updates for the client, these updates are also saved to an encrypted file which are manually uploaded by you.

To perform a file-based sync, do the following:

1. Enable File Based Sync on the Configuration->Workspace Settings page.
2. On the Sync page, click **Sync**. This causes the screen shown in Figure 6–4 to appear.

Figure 6–4 Provide Filename for File-Based Sync


File based synchronization

Send [✓ Receive]

File:

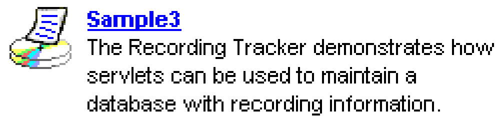
3. To send or receive synchronization data between the client and the Mobile Server, perform the following:
   - To download synchronization transactions to an encrypted file, select the Send radio button and enter the directory and filename for the destination file
   - To upload the synchronization file from the Mobile Server, select the Receive radio button and browse for the encrypted file that was copied to the client.

For information on setting synchronization options, see Section 6.4.1.1.2, "Configure the Mobile Client".

6.4.1.1.6 Log Off The Log Off tab automatically closes all running applications and returns you to the logon page.

6.4.1.2 Execute Mobile Applications Installed on Your Mobile Client

Web-to-Go and OC4J applications appear in the Mobile Workspace with an icon, name, and description.



Sample3
The Recording Tracker demonstrates how servlets can be used to maintain a database with recording information.

The icon and application name are both hyperlinks. To execute a Web-to-Go or OC4J application, click either the icon or application name.

The Web Mobile clients, such as OC4J or Web-to-Go, enable you to work disconnected from the Mobile Server. You need a connection to the Mobile Server only when you choose to synchronize any data changes from the client with the Oracle database.

The Web Mobile client propagates the tables that your applications use on the Mobile Server to the Mobile client as database snapshots. When you define your snapshot, you can use the `SQL WHERE` clause to specify a parameterized SQL query, where only the row data that your application uses is downloaded to the client. Thus, you can define what is downloaded to the client: the entire contents of the table or the subset of information that is relevant to the specific user. Most applications specify a particular subset of data that is relevant only to the user to be downloaded.

You can work continuously with Web Mobile clients storing your data changes in the Oracle Lite database. When you click the Sync tab, the Web Mobile client updates the Oracle database with any data changes you made on your client. The Mobile Server downloads any new applications, application changes, or data changes to your client.

6.4.1.3 Customize the Mobile Client Workspace

You can customize the Mobile Client Workspace. See Section 4.5.2.10, "Customizing the Workspace Application" in the *Oracle Database Lite Developer's Guide*.

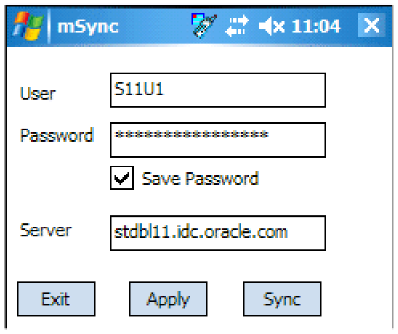
6.4.1.4 Schedule Data Synchronization Jobs

The Mobile Workspace enables you to create data synchronization jobs for your site from the OC4J or Web-to-Go Mobile client. This synchronization job automatically triggers synchronization with the Mobile Server at the start date and at the specified time for this job that you set using the Mobile Workspace. See Section 6.4.1.1.2, "Configure the Mobile Client" for more information.

6.4.2 Use the mSync GUI to Initiate Synchronization of Your Linux, WinCE, and Win32 Applications

You can initiate synchronization of the client using the mSync GUI for Linux, WinCE and Win32 clients, as shown in Figure 6–5.

Figure 6–5 Using the mSync GUI to Initiate Synchronization



User S11U1

Password ****************

[✓] Save Password

Server stdbl11.idc.oracle.com

To bring up the mSync GUI, execute `msync.exe` on WinCE and Win32 or `msync` on Linux, which is located in the `/bin` subdirectory under the directory where you installed the Mobile client. Modify the following supplied values, if incorrect:

- Username and password for the user that is starting the synchronization.

> **Note:** See Section 4.3.1.2.1, "Define Username and Password" in the *Oracle Database Lite Administration and Deployment Guide* for conventions for creating the username or password.

- Check if you want the password saved for future requests.
- Host name where the Mobile Server is installed.

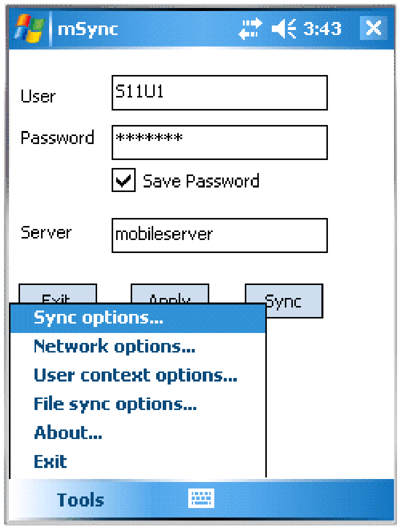
Click **Sync** to start the Synchronization. Click **Apply** to save any modifications you made to the entries. Click **Exit** to leave the tool.

If there are software updates that are waiting to be downloaded to the client, then the update tool is automatically executed after the end of the synchronization process. See Section 6.4.5, "Initiate Updates for the Oracle Lite Client" for more information.

> **Note:** The only time that the client does not check for software updates is if you are using Branch Office or the Synchronization APIs. If you want to launch the update UI, then enter `update` on the command line.

You can also modify the tool options by selecting the Tools Selection at the bottom of the UI, as shown in Figure 6–6.

Figure 6–6 The mSync Tools Selection



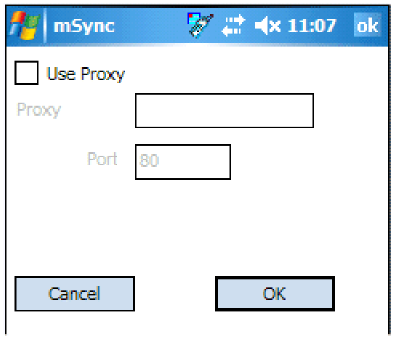
The following sections describe the Tools options:

- Section 6.4.2.1, "Network Options for MSync Tool"
- Section 6.4.2.2, "Sync Options for MSync Tool"
- Section 6.4.2.3, "Set User Context for Member"
- Section 6.4.2.4, "Sync to a File Using File-Based Sync"
- Section 6.4.2.5, "Use Mobile Client Tools on Linux"

6.4.2.1 Network Options for MSync Tool

Figure 6–7 displays the Network options screen where you can specify a proxy if your network provider requires that you use a proxy server to access the internet. . Click **Use Proxy** to use a proxy and then enter the proxy server and port number.

Figure 6–7 The mSync Network Options Selection



6.4.2.2 Sync Options for MSync Tool

Figure 6–8 displays the Sync Options screen where you can specify the following:

- Mobile User Password—Modify the existing password. The Mobile user password is stored on both the client and the Mobile Server. To ensure that both are modified, only change the password when connected to the Mobile Server. See Section 6.4.3, "Reset the Mobile User Password" for details.

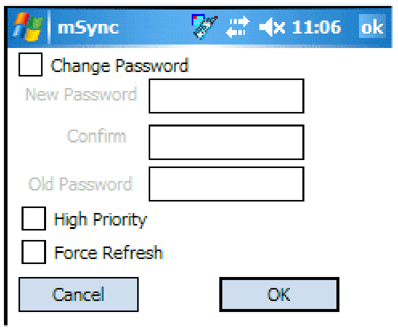
- High Priority—Select this checkbox to specify synchronizing only High Priority data. This specifies under what conditions the different priority records are synchronized. By default, the value is LOW, which is synchronized last. If you have a very low network bandwidth and a high ping delay, you may only want to synchronize your HIGH priority data.

  When you select this checkbox, you are enabling pre-defined high priority records to be synchronized first. This only for those publication items that have specified a restricting predicate. See Section 1.2.10, "Priority-Based Replication" in the *Oracle Database Lite Troubleshooting and Tuning Guide* for more information.

- Force Refresh—The force refresh option is an emergency only synchronization option. Check this option when a client is corrupt or malfunctioning, so that you decide to replace the Mobile client data with a fresh copy of data from the enterprise data store with the forced refresh. When this option is selected, any data transactions that have been made on the client are lost.

  When a force refresh is initiated all data on the client is removed. The client then brings down an accurate copy of the client data from the enterprise database to start fresh with exactly what is currently stored in the enterprise data store.

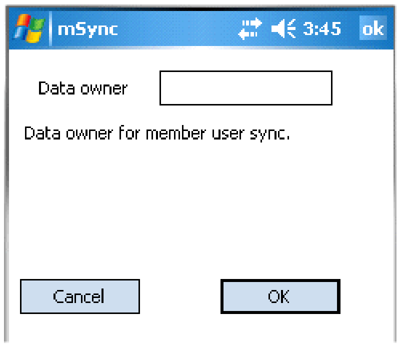
Figure 6–8 The mSync Options Selection



6.4.2.3 Set User Context for Member

If a member is attached to more than one user, then you must specify the user, also known as the data owner, under which the member is synchronizing. Select the appropriate user as the data owner under which to perform the synchronization.

Figure 6–9 Setting the User Context



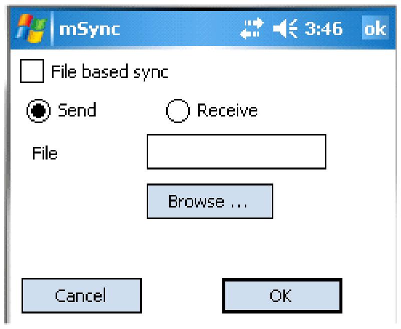
6.4.2.4 Sync to a File Using File-Based Sync

Once you select File Based Sync off the Tools menu, the screen shown in Figure 6–10 is displayed. To synchronize to a file, click on the File based sync checkbox and perform the following:

- If you select the send radio button, then browse for a directory where you want the client to save the upload data file from the Mobile client for the Mobile Server.
- If you select the receive radio button, then provide the location for the download data file from the Mobile Server.

For full details on File-Based Sync, see Section 5.8, "Synchronizing to a File with File-Based Sync" in the *Oracle Database Lite Administration and Deployment Guide*.

Figure 6–10 File Sync Options



6.4.2.5 Use Mobile Client Tools on Linux

The Mobile Client for Linux supports the `msync`, `dmagent` and `update` tools. To use the UI-based tools, use the following executables: `msync`, `dmagent`, or `update`.

To synchronize on a Linux client with the command line tool, use the `msync` executable for synchronization, as follows:

```
./msync username/password@server[:port][@proxy:port]
```

For example,

```
./msync john/john@testserver:8000
```

The other `msync` options, such as `-save`, `-a`, `-password` and `-force` currently will not result in a successful sync. This is a limitation only for the `msync` executable in the MDK installation on Linux.

6.4.3 Reset the Mobile User Password

Because the Mobile user password is stored on both the client and the Mobile Server, modify the password as follows:

- Modify the password on the client using either mSync UI or Client Workspace. Only modify the password using these tools if you are connected to the Mobile Server to ensure that the user password change is propagated to the Mobile repository.
- Modify the Mobile user password in the Mobile Manager in the User Properties page. If you simply want to invalidate the Mobile user, then you only have to modify the password on this screen; however, if you want to reset the password on both the Mobile Server and the Mobile user, then also send a Reset Password

command from the Device Management section in the Mobile Manager to the Mobile client.

After sending the Reset Password command, you need to perform a synchronization on the client with the new password. Then, you will be able to connect to the client database using the new password.

Note: If you modify the password on the server and do not send the Reset Password, then the client cannot synchronize. In this case, either send the Reset Password or return the password back to its original value on the server before retrying the synchronization.

See Section 11.2, "Which Password is Which" in the *Oracle Database Lite Administration and Deployment Guide* for details on all Oracle Lite Database passwords.

6.4.4 Use the Device Manager Client GUI to Manage the Client-Side Device

On any client, you can manage the Mobile device client software using the Oracle Lite Device Manager. See Section 7.8, "Using the Device Manager Agent (dmagent) on the Client" in the *Oracle Database Lite Administration and Deployment Guide* for a full description.

6.4.5 Initiate Updates for the Oracle Lite Client

You can initiate a request for software updates from the Mobile Server by executing the Oracle Database Lite Update tool. For details, see Section 7.7.3, "Initiate Updates of Oracle Database Lite Software for Mobile Clients" in the *Oracle Database Lite Administration and Deployment Guide*.

6.4.6 Configure JAVA_HOME for Web-to-Go Clients

Web-to-Go clients can execute against the Java version you specify, either J2SDK or J2RE. You can specify the `JAVA_HOME` that points to the desired JVM or JRE installation using one of the following methods:

- You can pre-configure a default `JAVA_HOME` value for all users by editing the `WEBTOGO.INF` file on the server before the Web-to-Go client is installed. When you invoke the `setup.exe` file on the Mobile client device, the entries in the pre-configured `WEBTOGO.INF` file are placed in the `webtogo.ora` file, which is installed on the Mobile client.

  You can only specify the location of the Java environment on the Web-to-Go Mobile clients if you know the install location of Java on the device where the client will be installed. This would apply to devices that are pre-imaged specifically for Web-to-Go Mobile clients. For full details on how to pre-configure the `JAVA_HOME` value, see Section 7.2, "Configuring Mobile Clients Before Installation" in the *Oracle Database Lite Administration and Deployment Guide*.

- For Web-to-Go clients that have already been installed on a Win32 platform, you can set the `JAVA_HOME` parameter in the `webtogo.ora` file on the Mobile client to point to a specific Java environment with the `setjavahome.bat` utility.

  For example, the following sets the `JAVA_HOME` parameter in the `webtogo.ora` file to point to the `C:\jdk1.5` directory:

  ```
  setjavahome.bat <JAVA_HOME>
  ```

The following examples show how the `setjavahome.bat` utility sets the `JAVA_HOME` parameter on the Web-to-Go Mobile client to `C:\jdk1.5` and to `C:\program files\jdk1.5`. The second execution shows that when you have a path name that includes white spaces, you must wrap the directory path within double quotes.

```
setjavahome.bat C:\jdk1.5
setjavahome.bat "C:\program files\jdk1.5"
```

6.4.7 Defragmentation and Reducing Size of the Client Application Databases

On each client device, an Oracle Lite database stores the application data—either as an embedded database that exists solely for the application or as a repository for data for a specific user that is synchronized with a back-end Oracle database.

You can use the DefragDB utility on your database to perform the following optimizations:

- Reduce size of Oracle Lite databases by defragmenting the Oracle Lite database.
- Remove any BLOB data from the Oracle Lite database. All BLOB data—both binary and character— and indexes are stored in separate files with the extension of `.obs` for Oracle Blob Store. This changes the size limit on your device to either the operating system file size limitations or 16 terabytes.

Before executing this tool, you must stop ALL applications, as the database is erased during this process. This includes the Oracle Lite applications, such as the Sync Agent, Web-to-Go, and so on. To stop the Sync Agent, see Section 5.4.2, "Start, Stop, or Get Status for Automatic Synchronization" in the *Oracle Database Lite Administration and Deployment Guide*.

For full details on the DefragDB tool, see Section C.7, "DefragDB to Defragment and Reduce Size of the Oracle Lite Database" in the *Oracle Database Lite Client Guide*.

6.4.8 Communicate Between the Internet and Intranet Through a Reverse Proxy

If your Mobile client is on either side of the firewall, you can set up a proxy or reverse proxy to facilitate communication between the Mobile client and Mobile Server. See Section 11.6, "Using a Firewall Proxy or Reverse Proxy" in the *Oracle Database Lite Administration and Deployment Guide*.

7

Managing the Oracle Lite Database

The following sections describe how to manage the Oracle Lite Relational Database Management System (RDBMS):

- Section 7.1, "Moving Your Client Data Between an Oracle Lite Database and an External File"
- Section 7.2, "Backing Up an Oracle Lite Database"
- Section 7.3, "Encrypting a Database"
- Section 7.4, "Support for Linguistic Sort"
- Section 7.5, "Discovering Oracle Lite Database Version Number"
- Section 7.6, "Row Sort Limitations of the Oracle Lite Database"
- Section 7.7, "Troubleshooting the Source of a Checksum Error Against Database"
- Section 7.8, "Enable Tracing for the Oracle Lite Database"

7.1 Moving Your Client Data Between an Oracle Lite Database and an External File

You can move data between an Oracle Lite database and an external file either through programmatic APIs or the Load Utility (`OLLOAD`). The following sections describe both methods:

- Section 7.1.1, "Move Data Between an Oracle Lite Database and an External File Using Programmatic APIs"
- Section 7.1.2, "Oracle Database Lite Load Utility (OLLOAD)"

7.1.1 Move Data Between an Oracle Lite Database and an External File Using Programmatic APIs

Using the Oracle Database Lite Load APIs, you can develop applications to load data from an external file into a table in Oracle Database Lite, or to unload (dump) data from a table in Oracle Database Lite to an external file. The details of the APIs and file formats are provided in Appendix C.11.2, "Oracle Database Lite Load Application Programming Interfaces (APIs)".

7.1.2 Oracle Database Lite Load Utility (OLLOAD)

The Oracle Database Lite Load Utility enables you to load data from an external file into a table in Oracle Database Lite, or to unload (dump) data from a table in Oracle

Database Lite to an external file. For more information on the OLLOAD utility, see Appendix C.11.1, "OLLOAD".

7.2 Backing Up an Oracle Lite Database

You can backup the Oracle Lite database either by using the backupdb utility or by copying the files to another location.

Oracle Database Lite uses the ODB and OBS files with dependent log files that can be backed up by copying to another location. Before any files can be copied, disconnect all applications that access the database and shut down the multi-user service, if running. Once that has been accomplished, execute the backupdb utility, which copies the *.odb, *.obs, and *.opw files to the filename of your choice to make a backup of the database.

```
backupdb DSN|NONE DBName backup_filename [DB_password]
```

For full details, see Appendix C.6, "BACKUPDB".

7.3 Encrypting a Database

You can encrypt the Oracle Lite database. Once encrypted, the data stored in the database files cannot be interpreted by examining the files. A password is used to derive a 128-bit encryption key. Oracle Database Lite uses the Advanced Encryption Standard (AES) encryption.

For information on encrypting the database used by the client, see Appendix C.4, "ENCRYPDB".

If you do not want to use AES encryption, then you can insert your own encryption module to supplant AES; see Section 14.3, "Providing Your Own Encryption Module for the Client Oracle Lite Database" for complete details.

7.4 Support for Linguistic Sort

Linguistic sort is a feature for the ASCII version of the Oracle Lite database. It produces culturally acceptable order of strings for a specified language or collation sequence. The ASCII version supports several code pages defined by single-byte 8-bit encoding schemes. Each of these code pages is a super set of 7-bit ASCII, and the additional accented characters necessary to support certain European languages are included in the upper 128 bytes.

A new string comparison mechanism is provided that produces strings in a linguistically correct order by mapping each collation element of a string to the corresponding 8-bit value of the supported code page.

The only supported languages for linguistic sort are French, German, Czech and XCzech. The collation sequence for these Oracle Lite databases can be specified with the NLS_SORT parameter.

All other languages use the BINARY collation sequence, which does not enable linguistic sort.

7.4.1 Creating Linguistic Sort Enabled Databases

The linguistic sort capability must be enabled when the database is created using the CREATEDB command line utility with the <collation_sequence> enabled.

Note: For more information on the `CREATEDB` utility, see Section C.2, "CREATEDB".

The behavior of the `ORDER_BY` clause and the `WHERE` condition are determined by how the `NLS_SORT` parameter is implemented. Binary sorting is the default setting, and is used unless the `<collation_sequence>` parameter is set to use the linguistic sort ordering rules.

`NLSRT` is not supported in the current version of Oracle Database Lite. Therefore, `NCHAR` data type is not yet available.

7.4.2 How Collation Works

Collation refers to ordering of strings into a culturally acceptable sequence. A collation sequence is a sequence of all collation elements from an alphabet from smallest collation order to the largest. Once a collation sequence is given, orders of all strings from the same alphabet are fixed. As such, the collation sequence encodes the linguistic requirements on collation. A collation element is the smallest sub-string that can be used by the comparison function to determine the order of two strings.

7.4.3 Collation Element Examples

Normally, a collation element is just one character. In binary sorting, only one property, the code value that represents a character, is used. But in linguistic sorting, usually three properties. The primary level of difference is the base character. The secondary level of difference is for diacritical marks on a given base character. The tertiary level of difference is for the case of a given character. Punctuation can function as a fourth level of difference, but comparisons for punctuation occur last and are made at the binary rather than the linguistic level. These are used for each collation element. The following sections contain examples that demonstrate sorting priorities.

7.4.3.1 Sorting Normal Characters

This section lists a set of examples that describe how to sort normal characters.

Example 1

`'a' < 'b'`. There is a primary difference between them on the character level.

Example 2

`'À' > 'a'`. This difference occurs on the secondary level. Note that `'À'`and 'a' are considered "equal" on the primary level.

Example 3

`'À' < 'à'` in FRENCH but 'À' > 'à' in GERMAN. This difference on the tertiary level. Note that 'À' and 'à' are considered being "equal" on the primary and secondary level. Also note that the case convention may be different for different language.

Example 4

'às' < 'at'. This is a difference on the primary level. This example shows the role of difference levels: the lower level differences are ignored if there is a primary level difference anywhere in the strings.

Example 5

`'+data' < '-data' <'data' <'data-'`. If strings are compared and present no difference on the primary, secondary, or tertiary levels, they are compared for punctuation.

7.4.3.2 Reverse Sorting of French Accents

Some languages, particularly French, require words to be ordered on the secondary level according to the last accent difference. This behavior is known as French secondary sorting or French accent ordering.

Example

'côte' < 'coté' in FRENCH but 'coté' < 'côte' in GERMAN. Note that the secondary difference of 'e' and 'é' occurred later than those of 'ô' and 'o'.

7.4.3.3 Sorting Contracting Characters

There are some special cases where two or more characters in a group can function as a single collation element. These types of collation elements are called 'contracting characters' or 'group characters'. In these cases each of these characters properties are assigned appropriate values.

Example

'h' < 'ch' < 'i' in XCZECH. Here 'ch' is assigned a primary property value which differentiates it from 'h' and 'i', such that 'h' < 'ch' < 'i'. Note that 'ch' is treated as a single character.

7.4.3.4 Sorting Expanding Characters

If a letter sorts as if it were a sequence of more than one letter, it is called an 'expanding character'. For example, in German the sharp s (ß) is treated as if it were a string of two characters 'ss' when comparing with other letters.

7.4.3.5 Sorting Numeric Characters

Only sorting of single digit characters from '0' to '9' is currently supported. For the supported European languages a digit character is always sorted as greater than any alphabetic character. For other languages this may be not the same. Other numeric characters such as Roman numeric characters and counting sequences, such as "one", "two", "three", are not supported at this time.

Example

'1' > 'z' in any European language, '1' < 'a' in LATVIAN. Note that this difference occurs on the primary level.

7.5 Discovering Oracle Lite Database Version Number

Use the `ODBINFO` utility to discover the version number and volume identifier of the Oracle Lite database. See Section C.9, "ODBINFO" for full details.

7.6 Row Sort Limitations of the Oracle Lite Database

Currently, the Oracle Database Lite engine cannot sort any row that exceeds 4040 bytes in length. If the selected columns exceed this length, then the database engine issues an error. Therefore, you cannot recover queries that use the `UNION` operation where both select clauses sort intermediate results, where the returned results are long rows with size greater than 4040 bytes.

7.7 Troubleshooting the Source of a Checksum Error Against Database

You can perform diagnostics if you experience database corruption due to file system write errors, I/O errors, or a media device problem. If you receive a POL-3207 error, you may wish to execute the validatedb tool to see if it is a checksum error. Then, setting `OLITE_WRITE_VERIFY` to `TRUE` generates error reporting if a checksum error occurs on the device for the Mobile client.

For more information, see Section A.2.16, "OLITE_WRITE_VERIFY".

7.8 Enable Tracing for the Oracle Lite Database

When an unexpected error is reported, users need to identify the location and cause of the error. Errors can be caused due to problems in the application code, Oracle tools—such as forms, SQLJ—or in the Oracle Lite database. Errors also occur in simple environments where a user application talks directly to the Oracle Lite database through JDBC or ODBC drivers. It may not be obvious which component is at fault—whether it is the user application, JDBC or ODBC drivers, or the core database runtime system.

If the optimizer spends too much time evaluating alternative plans or collecting index statistics, a query may take a long time for compilation. If the execution plan selected by the optimizer is not optimal, the query may also take a long time during execution. Based on these criteria, the tracing facility provides the compilation time and the execution plan.

The following sections describe how to set and use tracing.

- Section 7.8.1, "Enabling Trace Output"
- Section 7.8.2, "Description of Trace Information"

7.8.1 Enabling Trace Output

By setting the parameter `OLITE_SQL_TRACE = YES` in the `polite.ini` or `polite.txt` file on the client device, Oracle Database Lite generates a trace file named `oldb_trc.txt` that shows the following:

Note: Any value other than `YES` disables the tracing feature. The parameter value is checked once during database startup. Hence, users must set this value before connecting to the database.

- The order tables are accessed by a query.
- The table scan access method used.
- The value of any bind variables utilized by the query.
- The time it takes for the first record to be retrieved.

These are the main items reported that you can use to tune the majority of SQL queries.

If the trace file identifies that a full table scan is occurring, the most common way to get better performance from the query is to add an index that accommodates that query.

When you enable tracing, the trace information is dumped to a file named `oldb_trc.txt` in the current working directory of the database process. If the file already exists, then the trace output is appended to the end. If it does not exist, then a new file is automatically created. For a database service on Windows or the Oracle Lite database daemon for a Linux platform, the current working directory is specified by the `wdir` parameter during startup of the database service or daemon.

Note: To implement the tracing feature, the database process must contain permissions to create the trace file in the current working directory.

7.8.2 Description of Trace Information

The following trace information is provided:

Table 7–1 Trace Output

Trace Output	**Description**
Statement Text	Each time a SQL statement is prepared, its text is dumped into the trace file. The SQL statement itself is output without any formatting. If a SQL statement contains a new line character, it is also included in the SQL statement output.
Compilation Time	After the SQL statement is compiled, the compilation time is printed.
Execution Plan	If there are no errors, the execution plan is printed when available. Only statements that contain a `WHERE` clause generate an execution plan. The printed plan contains the execution order of tables for each sub-select.
Bind Value	If a SQL statement contains markers, then the bind value is printed for every line. Each value for the marker or bind variable is printed on a separate line in the following format: `Marker [<number>]: <Value>` Where, `<number>` is the number of the marker and `<value>` denotes the value of the marker before execution.
Temporary Table Created	Each time a temporary table is created, its name is dumped into the trace file.
Table Access	Each time a table is accessed, the following information is dumped into the trace file: ▪ Table Name: The name of the table been accessed is dumped into the trace file. ▪ Access Method: The access method used by the database is dumped into the trace file. For a description of how this information is presented, see Section 7.8.2.1, "Table Name Output".
Temporary Table Sorted	Each time a temporary table is sorted, its name and sorting time (in milliseconds) are dumped into the trace file.
First Fetch Time	If the SQL statement is a SELECT statement, the time spent on fetching the first row is dumped into the trace file.
Tid	The thread ID is dumped into the trace file in front of some of the dumped information. The thread is displayed in the following format: `Tid: <thread id>`

7.8.2.1 Table Name Output

The name of the table that is currently being accessed and the method used to access the table are printed in the following formats.

- If the table is accessed sequentially, the format is:

  Table Name: `<table name>`

  Access Method: `Sequential`

  Where `<table name>` is the name of the table being accessed.

- If indices are used, the format is:

  Table Name: `<table name>`

  Access Method:
  ```
  Term[<number>], Index No: <index number>,
           IndexName: <index name>
  ```

  `<table name>` is the name of the table being accessed.

  `Term[<number>]` is the internal representation of the conjunct search conditions in the WHERE clause.

  `<index number>` is the index number. Each index has an unique number in the database.

  `<index name>` is the name of the index if any.

8 Oracle Database Lite Data Access APIs

To access the data within the ODB file from your application through one of the following APIs:

- For relational database development:
  - ODBC—See Section 8.1, "ODBC" for more information.
  - JDBC—See Section 8.2, "JDBC" for more information.
  - ADO.NET—See Section 8.3, "ADO.NET" for more information.

    Any interface that supports ODBC or JDBC data sources, such as ADO.Net, can also be used to access Oracle Database Lite. The interfaces can be used either independently or in combination.
- For object and relational database development:
  - Simple Object Data Access (SODA)—See Section 8.4, "SODA" for more information.

The following sections describe the different development interfaces that you can use to store and retrieve data from the file-based Oracle Lite database:

- Section 8.1, "ODBC"
- Section 8.2, "JDBC"
- Section 8.3, "ADO.NET"
- Section 8.4, "SODA"

8.1 ODBC

The Microsoft Open Database Connectivity (ODBC) interface is a procedural, call-level interface for accessing any SQL database, and is supported by most database vendors. It specifies a set of functions that allow applications to connect to the database, prepare and execute SQL statements at runtime, and retrieve query results.

You can call ODBC from within C or C++ applications.

Oracle Database Lite supports Level 3 compliant ODBC 2.0 and the ODBC 3.5 drivers through Oracle Database Lite ODBC drivers with some restrictions. The ODBC 2.0 driver is installed by default. The ODBC 3.5 driver should be used solely for the standalone application that uses an embedded Oracle Lite database.

To use the ODBC 3.5 driver, you need to configure the `ODBC.INI` file. On Windows, you can modify the ODBC driver either with the `odbcad32` command-line tool or by executing the ODBC Administrator GUI tool by clicking Control Panel -> Administrative Tools -> Data Sources (ODBC).

Note: You cannot use ODBC 3.5 for any multi-user listener application.

For a full description of using the ODBC drivers on the client, see Chapter 9, "ODBC Drivers". For an example, see the Oracle Database Lite ODBC sample application, as described in Section 9.2, "Executing the ODBC Examples".

8.2 JDBC

The Java Database Connectivity (JDBC) interface specifies a set of Java classes that provide an ODBC-like interface to SQL databases for Java applications. JDBC, part of the JDK core, provides an object interface to relational databases. Oracle Database Lite conforms to the JDBC 1.2 API specification standard. You can use JDBC to access data from within your Java applications.

Oracle Database Lite supports JDBC through an Oracle Database Lite Type 2 and Type 4 JDBC drivers that interpret the JDBC calls and pass them to Oracle Database Lite. The Type 4 JDBC driver can only be used for the multi-user service, as described in Chapter 5, "Building a Client/Server Environment".

For your applications, you must include the correct binaries when you package the application, as described in Section 4.3, "Packaging Your Embedded Application With the Oracle Database Lite Runtime".

See Chapter 10, "JDBC Programming" for more information on using JDBC.

8.3 ADO.NET

The Oracle Database Lite ADO.NET Provider implements the Microsoft ADO.NET specification. Use this programming interface to access data in .NET applications. The Oracle Database Lite ADO.NET data provider supports both .NET and Compact .NET frameworks. You can access data within your database using ADO.Net from within C# applications.

The Oracle Database Lite ADO.NET provider resides in the `Oracle.DataAccess.Lite` namespace. The ADO.Net classes that enable you to connect to the Oracle Lite database, to manage transactions, create commands, manage performance and manage BLOB objects are described in Chapter 11, "Oracle Database Lite ADO.NET Provider".

8.4 SODA

SODA is an interface for Oracle Database Lite development using C++. It provides object-oriented data access using method calls, relational access using SQL and object-relational mapping to bridge the gap between the two.

Object functionality is approximately three times faster than ODBC for simple operations. It allows rich datatypes—such as arrays, object pointers, and standard SQL columns. A programmer can store any data structure in the database and not worry about relational design or performing joins.

A C++ developer can use the interface for executing SQL statements. The resulting code is shorter and clearer than ODBC code. SQL queries can return objects, which can be examined and modified directly through the object-oriented layer without calling any additional SQL statements.

Finally, object-relational mapping enables the application to access relational data as if it was an object hierarchy. This is essential for replicating rich data types or object pointers to the Oracle database server.

For more information, see Chapter 12, "Using Simple Object Data Access (SODA)".

9 ODBC Drivers

The following sections describe the support for ODBC and samples:

- Section 9.1, "Supported ODBC Drivers for Oracle Database Lite"
- Section 9.2, "Executing the ODBC Examples"

Note: A sample for using ODBC to access the Oracle Lite database is in the <*ORACLE_HOME*>`\Mobile\Sdk\samples\odbc\win32\c_samples` directory.

9.1 Supported ODBC Drivers for Oracle Database Lite

The Microsoft Open Database Connectivity (ODBC) interface is a procedural, call-level interface for accessing any SQL database, and is supported by most database vendors. It specifies a set of functions that allow applications to connect to the database, prepare and execute SQL statements at runtime, and retrieve query results.

Oracle Database Lite supports Level 3 compliant ODBC 2.0 and the ODBC 3.5 drivers through Oracle Database Lite ODBC drivers with the following restrictions:

- ODBC 2.0 driver: The default driver for all Oracle Database Lite components. This will be installed by default, unless otherwise configured.
- ODBC 3.5 driver : If you want to use the ODBC 3.5 driver, you must configure the `ODBC.INI` file to use the `olod3540.dll` as the ODBC driver. Configure the `ODBC.INI` file by executing the ODBC administrator. On Windows, you can modify the ODBC driver either with the `odbcad32` command-line tool or by executing the ODBC Administrator GUI tool by clicking Control Panel -> Administrative Tools -> Data Sources (ODBC).

  If your application uses the ODBC 3.5 driver, link with the `olod3540.lib`, which is the ODBC 3.5 driver library. The data sources that use the ODBC 3.5 driver for each connection must specify the correct library in the `Driver32` field.

  If you want to use the Visual Studio 2005 built-in features to design database applications, then you must use the ODBC 3.5 driver.

For more information on ODBC, see the following:

- Microsoft ODBC documentation.
- Using ODBC within a stored procedure, as described in Section 13.6.2, "Using Stored Procedures to Return Multiple Rows".

9.2 Executing the ODBC Examples

The ODBC examples are located in <*ORACLE_HOME*>`\Mobile\Sdk\Samples` and must be compiled using a C++ complier. To build them, use `nmake`.

There are five ODBC examples: `odbctbl`, `odbcview`, `odbcfunc`, `odbctype`, and `long`. You use the `POLITE` DSN to execute these examples. The `POLITE` DSN is automatically created during the Mobile Development Kit installation.

The first four examples have their own output windows listing the activity log. Closing the current example window causes the next example to be run. The output displayed in the example windows is also printed in the following log files: `odbctbl.log`, `odbcview.log`, `odbcfunc.log`, `odbctype.log`. The `long` example output is collected in the output file: `long.out`.

The following sections describe the functionality of the samples:

- Section 9.2.1, "ODBCTBL"
- Section 9.2.2, "ODBCVIEW"
- Section 9.2.3, "ODBCFUNC"
- Section 9.2.4, "ODBCTYPE"
- Section 9.2.5, "LONG"

9.2.1 ODBCTBL

This is an ODBC SQL table example, which shows how to manipulate tables using the ODBC API. It creates the `EMP` table with columns ID, `NAME`, `START_DATE`, `SALARY`. After creation, it populates this table with data, performs an update on the salary column, selectively deletes some rows, then selects from the resulting table and shows the results of the fetch operation. At the end, the `EMP` table is dropped.

9.2.2 ODBCVIEW

This is an ODBC SQL view example, which demonstrates how to manipulate views using the ODBC API. It creates the `EMP` table and the `HIGH_PAID_EMP` view, selecting the full name (using the `CONCAT` scalar function), `HIRE_DATE` and `SALARY` from the `EMP` table. Then, the example populates the `EMP` table and selects from the `HIGH_PAID_EMP` view to show the populated data. The salary column of `EMP` is updated, some rows are delete, and a select from `HIGH_PAID_EMP` is issued to demonstrate how the changes are reflected in the view. Finally, the view and the table are dropped.

9.2.3 ODBCFUNC

This is an ODBC SQL scalar functions example, which shows you how to use scalar functions in the ODBC API. It creates table `EMP`, populates it with the data, then performs a select on `ID`, `FULL_NAME` from `EMP`. When it calculates the full name, it uses the ODBC scalar function `CONCAT`—with last and first names as arguments. The example updates the table, converting the last name to uppercase and first name to lowercase for IDs less than three using ODBC scalar functions `UCASE` and `LCASE`. The new data is selected and displayed again. Finally, the table `EMP` is dropped.

9.2.4 ODBCTYPE

This is ODBC SQL types example, which shows you how to manipulate different data types using the ODBC API. This test creates the `EMP` table, populates it with data, selects all the rows and displays the result. However, the columns are bound differently from the previous tests. First, it calls `SQLNumResultCols` to find the number of result columns. Then, for each result column, it calls `SQLDescribeCol` to retrieve all of the information about that column, such as column name, column name length, column type, column length, column scale, and so on. This information is used to bind the column. Thus, you can see how you can retrieve the type information from the database using the ODBC API.

9.2.5 LONG

This example exercises the basic read/write functions of `SQL LONG VARCHAR`. It first drops, then creates the `LONG_DATA` table with one `LONG VARCHAR` column and inserts the data into the table. For each row the data is put in frames, where each frame represents a buffer of long varchar data (of length 4096). The example uses `SQLParamData` and `SQLPutData` to send each frame to populate the row. Then, issues a select to fetch the rows and read long varchar data from the table. For each row, the data is also read in frames, using `SQLGetData` until `SQL_NO_DATA_FOUND` is returned. These actions are logged into the `long.out` file.

10 JDBC Programming

The following sections describe the Oracle Database Lite support for JDBC programming:

Note: A sample for using JDBC to access the Oracle Lite database is in the <*ORACLE_HOME*>`\Mobile\Sdk\samples\jdbc` directory.

10.1 JDBC Compliance

Oracle Database Lite provides a native JDBC driver that allows Java applications to communicate directly with the Oracle Database Lite object relational database engine. The Oracle Database Lite implementation of JDBC complies with JDBC 1.2. In addition, Oracle Database Lite provides certain extensions specified by JDBC 2.0, which are compatible with the Oracle database JDBC implementation. For a complete JDBC reference, see the Sun Microsystems Web site.

10.2 JDBC Environment Setup

For your Java applications using the client/server model, include the `olite40.jar`, which is located in *`OLITE_HOME`*`/bin`, in the system `CLASSPATH` on the server machine and in the user `CLASSPATH` on the client machine.

When using the Oracle Lite JDBC driver in your OC4J application, use the default classloader instead of a per-application classloader, which many J2EE containers use. Ensure that the `olite40.jar` is in the OC4J `CLASSPATH` when OC4J initiates and not in the `/lib` subdirectory of your application WAR file.

Note: For more information on how to start the Multiuser Oracle Database Lite Database Service, see Chapter 5, "Building a Client/Server Environment".

10.3 JDBC Drivers to Use When Connecting to the Oracle Lite Database

Note: JDK 1.4.2 or 5.0 is required to connect to the Oracle Lite database.

Oracle Database Lite supports Type 2 and Type 4 drivers.

- The Type 2 driver uses native code on the client side through which it interfaces with the Oracle Database Lite ODBC driver.
- The Type 4 JDBC driver is a pure Java driver and uses the Oracle Database Lite network protocol to communicate with the Oracle Database Lite service. Before using this driver, ensure that you start Oracle Database Lite. Any Java applet can use the Type 4 JDBC driver.

The supported Type 2 and Type 4 drivers are described in the following sections:

- Section 10.3.1, "Type 2 Driver"
- Section 10.3.2, "Type 4 (Pure Java) Driver Connection URL Syntax"

10.3.1 Type 2 Driver

For most applications, use the type 2 driver for connecting to the database. You can use the type 2 driver to connect either to the local Oracle Lite database or to the server where a Multi-User Service is managing the Oracle Lite databases.

- To connect to the local Oracle Lite database, use the following URL syntax:

  ```
  jdbc:polite[:uid / pwd]:localDSN[;key=value]*
  ```

  where the `localDSN` is the DSN name for the local Oracle Lite database (the ODB file on the local machine) and the optional key=value pairs are listed in Table 10–1.

  The following example retrieves a connection to the local Oracle Lite database, where the DSN name is `polite`:

  ```
  DriverManager.getConnection("jdbc:polite:polite","system","admin");
  ```

- To access the Oracle Lite database on a remote host where a Multi-User Service or Branch Office is located, use the following URL syntax:

  ```
  jdbc:polite[:uid / pwd]@[host]:[port]:serverDSN [;key=value]*
  ```

  where the host, port, and serverDSN identify the host, port and DSN of the remote host where the Oracle Lite database (the ODB file on the local machine) and the multi-user service is located. The optional key=value pairs are listed in Table 10–1.

  For more information on how to install and start the Multiuser Oracle Database Lite Service, refer to Chapter 5, "Building a Client/Server Environment".

You can provide additional configuration information in the JDBC driver URL within key-value pairs, as specified in Table 10–1, each of which are separated by a semi-colon. The information specified within the key-value pairs always overrides the information that is specified in the URL.

Table 10–1 Key/Value Pairs for JDBC Connect URL

Argument	Description
`jdbc`	Identifies the protocol as JDBC.
`polite`	Identifies the subprotocol as `polite`.
uid / *pwd*	The optional user ID and password for Oracle Database Lite, each of which are limited to 28 characters. If specified, this overrides the specification of a username and password defined in the UID and PWD arguments. If the database is encrypted, you must include the password in the key-value pair.
host	The name of the machine that hosts the Multi-User Service or Branch Office and on which the Oracle Database Lite service `olsv2040.exe` runs. This host name is optional. If omitted, it defaults to the local machine on which the JDBC application runs.
port	The port number at which the Multi-User or Branch Office service listens. The port number is optional. If omitted, the port number defaults to port 1160.
dsn	Identifies the data source name (DSN) entry in the `odbc.ini` file. This entry contains all the necessary information to complete the connection to the server. **Note**: For a JDBC program, you need not create a DSN if you have supplied all the necessary values for the data directory and database through key=value pairs. On the windows platform, you can use the ODBC administrator to create a DSN. For more information, see Section 5.5, "Creating DSNs".
Data_Directory=	Directory in which the `.odb` file resides.
Database=	Name of database as given during its creation.
IsolationLevel=	Transaction isolation level: READ COMMITTED, REPEATABLE READ, SERIALIZABLE or SINGLE USER. For more information on isolation levels, see Section 15.2, "What Are the Transaction Isolation Levels?". **Note**: If you are retrieving a large object, such as a BLOB, within a READ COMMITTED transaction, see Section 4.3.46.9, "Select Statement Behavior When Retrieving BLOBs in a READ COMMMITTED transaction" section in the *Oracle Database Lite SQL Reference*.
Autocommit=	Commit behavior, either `ON` or `OFF`.
CursorType=	Cursor behavior: DYNAMIC, FORWARD ONLY, KEYSET DRIVEN or STATIC. For more information on cursor types, see Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types".
UID=	User name
PWD=	Password

Example of Using JDBC Type 2 Connection for Local Oracle Lite Database

```
String ConnectMe=("jdbc:polite:SCOTT/tiger:polite;
Data_Directory=<ORACLE_HOME>;Database=polite;IsolationLevel=SINGLE USER;
Autocommit=ON;CursorType=DYNAMIC")

try
   {Class.forName("oracle.lite.poljdbc.POLJDBCDriver")
   Connection conn = DriverManager.getConnection(ConnectMe)
   }
catch (SQLException e)
{
...
}
```

Example of Using Type 2 in Multi-User Service Situation

An example of this type of connection is given below.

```
try {
Connection conn = DriverManager.getConnection(
   "jdbc:polite@yourhostname
        ;Data_Directory=<ORACLE_HOME>
        ;Database=polite
        ;IsolationLevel=SINGLE USER
        ;Autocommit=ON
        ;CursorType=DYNAMIC", "Scott", "tiger")
}
catch (SQLException e)
{ }
```

10.3.2 Type 4 (Pure Java) Driver Connection URL Syntax

Use the JDBC Type 4 driver for any pure Java application that uses the Multi-User or Branch office services. The URL syntax for the type 4 driver as follows:

```
jdbc:polite4[:uid/pwd]@[host]:[port]:serverDSN[;key=value]*
```

The parameter `polite4` indicates that the JDBC type 4 driver is being used. For the rest of the parameters, see the definitions of those parameters for the type 2 driver, as described in Table 10–1.

Note: The URL works with the Oracle Database Lite service only.

10.4 DataSource Connection

In JDBC 2.0, the `DataSource` object is an alternative to the `DriverManager` facility. The `DataSource` object is the preferred method for retrieving a connection and is typically registered with a naming service based on the JNDI API. A driver that is accessed through a `DataSource` object does not register itself with the `DriverManager`.

The Oracle Database Lite JDBC driver contains the basic `DataSource` implementation to produce a standard `Connection` object. The retrieved connection is identical to a connection obtained through the `DriverManager`.

Oracle Database Lite implements `javax.sql.DataSource` interface with the `POLJDBCDataSource` class in the `oracle.lite.poljdbc` package.

As with any class that implements the `DataSource` interface, the `POLJDBCDataSource` object defines properties for connecting to a specific database. In addition to the standard DataSource properties, the `POLJDBCDataSource` class has one additional property, which is a `String` to define the URL of the database connection string, as described in Section 10.3.1, "Type 2 Driver" and Section 10.3.2, "Type 4 (Pure Java) Driver Connection URL Syntax".

- The URL can include username and password, which then overrides any previous individual property settings. The following URL specifies a username and password of `system/manager`:

  ```
  jdbc:polite4:system/manager@::polite
  ```

- You must have a username and password defined either in the URL or in the `getConnection` method. If defined in both places, then the username and password in the getConnection takes precedence over the URL definitions.

See the JDBC example—`JDBCEXJSR169`—on the CD for how to use the `POLJDBCDataSource` object.

10.5 Java Datatypes and JDBC Extensions

The Oracle Database Lite JDBC driver supports JDBC 1.2 and provides extensions that support certain features defined in JDBC 2.0. The extensions include support for BLOB (large binary object) and CLOB (large character object) datatypes and scrollable result sets. The Oracle Database Lite JDBC extensions are compatible with the Oracle database JDBC implementation. However, Oracle Database Lite does not support the following Oracle database JDBC datatype extensions: `Array`, `Struct`, or `REF`.

The following sections list and describe the Oracle Database Lite datatypes and data access extensions. For details regarding function syntax and call parameters, see the Sun Microsystems Java 2 specification at the Sun Microsystems Web site.

- Section 10.5.1, "Mapping Datatypes Between Java and Oracle"
- Section 10.5.2, "Datatype Extensions"
- Section 10.5.3, "Data Access Extensions"

10.5.1 Mapping Datatypes Between Java and Oracle

Oracle Database Lite performs type conversions between Java and Oracle datatypes as indicated by the following table. Table 10–2 lists the Java datatypes and the corresponding SQL datatypes that result from the type conversion.

Table 10–2 Datatype Conversions

Java Datatype	SQL Datatype
`byte[], byte[][], Byte[]`	`BINARY, RAW, VARBINARY, BLOB`
`boolean, Boolean`	`BIT`
`String, String[]`	`CHAR, VARCHAR, VARCHAR2, CLOB`
`short, short[], short[][], Short, Short[]`	`SMALLINT`
`int,int[], int[][], Integer, Integer[]`	`INT`
`float, float[], float[][], Float, Float[]`	`REAL`
`double, double[], double[][], Double, Double[]`	`DOUBLE, NUMBER` (without precision)
`BigDecimal, BigDecimal[]`	`NUMBER(n)`
`java.sql.Date, java.sql.Date[]`	`DATE`
`java.sql.Time, java.sql.Time[]`	`TIME`
`java.sql.Timestamp, java.sql.Timestamp[]`	`TIMESTAMP`

Table 10–2 (Cont.) Datatype Conversions

Java Datatype	SQL Datatype
`java.sql.Connection`	Default JDBC connection to database

10.5.2 Datatype Extensions

BLOBs and CLOBs store data items that are too large to store directly in a database table. Rather than storing the data, the database table stores a locator that points to the location of the actual data. BLOBs contain a large amount of unstructured binary data items and CLOBs contain a large amount of fixed-width character data items (characters that require a fixed number of bytes per character).

You can select a BLOB or CLOB locator from the database using a standard SELECT statement. When you select a BLOB or CLOB locator using SELECT, you acquire only the locator for the large object, not the data itself. Once you have the locator, however, you can read data from or write data to the large object using access functions.

Note: If you are retrieving a large object, such as a BLOB, within a READ COMMITTED transaction, see Section 4.3.46.9, "Select Statement Behavior When Retrieving BLOBs in a READ COMMMITTED transaction" section in the *Oracle Database Lite SQL Reference*.

Table 10–3 lists the methods included in the Oracle Database Lite BLOB class and their descriptions:

Table 10–3 Methods in the Oracle Database Lite BLOB Class

Function	Description
`length`	Returns the length of a BLOB in bytes.
`getBinaryOutputStream`	Returns BLOB data.
`getBinaryStream`	Returns a BLOB instance as a stream of bytes.
getBytes	Reads BLOB data, starting at a specified point, into a buffer.
`getConnection`	Returns the current connection.
`isConvertibleTo`	Determines if a BLOB can be converted to a particular class.
`putBytes`	Writes bytes to a specified point in the BLOB data.
`makeJdbcArray`	Returns the JDBC array representation of a BLOB.
`toJdbc`	Converts a BLOB to a JDBC class.
`trim`	Trims to length.

Table 10–4 lists the methods included in the Oracle Database Lite CLOB class and their descriptions.

Table 10–4 Methods in the Oracle Database Lite CLOB Class

Function	Description
`length`	Returns the length of a CLOB in bytes.
`getSubString`	Retrieves a substring from a specified point in the CLOB data.
`getCharacterStream`	Returns CLOB data as a stream of Unicode characters.
`getAsciiStream`	Returns a CLOB instance as an ASCII stream.

Table 10–4 (Cont.) Methods in the Oracle Database Lite CLOB Class

Function	Description
`getChars`	Retrieves characters from a specified point in the CLOB data into a character array.
`getCharacterOutputStream`	Writes CLOB data from a Unicode stream.
`getAsciiOutputStream`	Writes CLOB data from an ASCII stream.
`getConnection`	Returns the current connection.
`putChars`	Writes characters from a character array to a specified point in the CLOB data.
`putString`	Writes a string to a specified point in the CLOB data.
`toJdbc`	Converts a CLOB to a JDBC class.
`isConvertibleTo`	Determines if a CLOB can be converted to a particular class.
`makeJdbcArray`	Returns a JDBC array representation of a CLOB.
`trim`	Trims to length.

10.5.3 Data Access Extensions

Oracle Database Lite provides access functions to set and return values of the CLOB and BLOB datatypes. In addition, stream classes provide functions that enable stream-format access to large objects.

The large object access functions are located in the `OraclePreparedStatement`, the `OracleCallableStatement`, and the `OracleResultSet` class.

Table 10–5 lists the data access functions included in the `OracleResultSet` class.

Table 10–5 Data Access Functions in the OracleResultSet Class

Function	Description
`getBLOB`	Returns a locator to BLOB data.
`getCLOB`	Returns a locator to CLOB data.

The stream format access classes are `POLLobInputStream`, `POLLobOutputStream`, `POLClobReader`, and `POLClobWriter`.

The `POLLobInputStream` class includes the following data access function.

Function	Description
`read`	Reads from a large object into an array.

The `POLLobOutputStream` class includes this data access function.

Function	Description
`write`	Writes from an output stream into a large object.

The `POLClobReader` class extends the class `java.io.reader`. It includes these data access functions.

Function	Description
`read`	Reads characters from a CLOB into a portion of an array.
`ready`	Indicates whether a stream is ready to read.
`close`	Closes a stream.
`markSupported`	Indicates whether the stream supports the `mark` operation.
`mark`	Marks the current position in the stream. Subsequent calls to the `reset` function reposition the stream to the marked location.
`reset`	Resets the current position in the stream to the marked location. If the stream has not been marked, this function attempts to reset the stream in a way appropriate to the particular stream, such as by repositioning it at its starting point.
`skip`	Skips characters in the stream.

The `POLClobWriter` class extends the class `java.io.writer`. It includes these data access functions:

Function	Description
`write`	Writes an array of characters to the output stream.
`flush`	Writes any characters in a buffer to their intended destination.
`close`	Flushes and closes the stream.

10.5.3.1 Reading from a BLOB Sample Program

The following sample uses the `getBinaryStream` method to read BLOB data into a byte stream. It then reads the byte stream into a byte array, and returns the number of bytes read.

```
// Read BLOB data from BLOB locator.
InputStream byte_stream = my_blob.getBinaryStream();
byte [] byte_array = new byte [10];
int bytes_read = byte_stream.read(byte_array);
```

10.5.3.2 Writing to a CLOB Sample Program

The following sample reads data into a character array, then uses the `getCharacterOutputStream` method to write the array of characters to a CLOB.

```
java.io.Writer writer;
char[] data = {'0','1','2','3','4','5','6','7','8','9'};

// write the array of character data to a CLOB
writer = ((CLOB)my_clob).getCharacterOutputStream();
writer.write(data);
writer.flush();
writer.close();
```

10.6 Limitations

If data truncation occurs during a write, a SQL data truncation exception is thrown. A SQL data truncation warning results if data truncation occurs during a read.

The Oracle Database Lite JDBC classes and the JDBC 2.0 classes use the same name for certain datatypes (for example, `oracle.sql.Blob` and `java.sql.Blob`). If your program imports both `oracle.sql.*` and `java.sql.*`, attempts to access the overlapping classes without fully qualifying their names may result in compiler errors. To avoid this problem, use one of the following steps:

1. Use fully qualified names for BLOB, CLOB, and data classes.
2. Import the class explicitly (for example, import `oracle.sql.Blob`).

Class files always contain fully qualified class names, so the overlapping datatype names do not cause conflicts at runtime.

10.7 New JDBC 2.0 Features

This section describes JDBC 2.0 methods or interfaces that are supported by the Oracle Database Lite JDBC driver. Topics include:

- Section 10.7.1, "Interface Connection"
- Section 10.7.2, "Interface Statement"
- Section 10.7.3, "Interface ResultSet"
- Section 10.7.4, "Interface Database MetaData"
- Section 10.7.5, "Interface ResultMetaData"
- Section 10.7.6, "Interface PreparedStatement"

10.7.1 Interface Connection

This section describes the JDBC 2.0 Interface methods that are implemented by the Oracle Database Lite JDBC driver.

10.7.1.1 Methods

Statement createStatement(int resultSetType, int resultSetConcurrency)

Creates a statement object that generates ResultSet objects with the given type and concurrency.

Map getTypeMap()

Gets the TypeMap object associated with this connection.

CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)

Creates a CallableStatement object that generates ResultSet objects with the given type and concurrency.

PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)

Creates a `PreparedStatement` object that generates `ResultSet` objects with the given type and concurrency.

void setTypeMap(Map map)

Installs the given type map as the type map for this connection.

10.7.2 Interface Statement

This section describes the JDBC 2.0 Interface Statement methods that are implemented by the Oracle Database Lite JDBC driver.

Connection getConnection()

Returns the Connection object that produced this Statement object.

int getFetchDirection()

Retrieves the direction for fetching rows from database tables that is the default for result sets generated from this Statement object. Only FETCH_FORWARD is supported for now.

int getFetchSize()

Retrieves the number of result set rows that is the default fetch size for result sets generated from this Statement object. Only fetch size = 1 is supported for now.

int getResultSetConcurrency()

Retrieves the result set concurrency. Only CONCUR_READ_ONLY is supported for now.

int getResultSetType()

Determine the result set type. Only TYPE_FORWARD_ONLY and TYPE_SCROLL_INSENSITIVE are supported for now.

void setFetchDirection(int direction)

Gives the driver a hint as to the direction in which the rows in a result set will be processed.

void setFetchSize(int rows)

Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed.

10.7.3 Interface ResultSet

This section describes the JDBC 2.0 Interface ResultSet methods that are implemented by the Oracle Database Lite JDBC driver.

10.7.3.1 Fields

The following fields can be used to implement the Interface ResultSet feature.

static int CONCUR_READ_ONLY

The concurrency mode for a ResultSet object that may NOT be updated.

static int CONCUR_UPDATABLE

The concurrency mode for a ResultSet object that may be updated. Not supported for now.

static int FETCH_FORWARD

The rows in a result set will be processed in a forward direction; first-to-last.

static int FETCH_REVERSE

The rows in a result set will be processed in a reverse direction; last-to-first. Not supported for now.

static int FETCH_UNKNOWN

The order in which rows in a result set will be processed is unknown.

static int TYPE_FORWARD_ONLY

The type for a ResultSet object whose cursor may move only forward.

static int TYPE_SCROLL_INSENSITIVE

The type for a ResultSet object that is scrollable but generally not sensitive to changes made by others.

static int TYPE_SCROLL_SENSITIVE

The type for a ResultSet object that is scrollable and generally sensitive to changes made by others. Not supported for now.

10.7.3.2 Methods

This section describes the JDBC 2.0 ResultSet method implemented by the Oracle Database Lite JDBC driver.

boolean absolute(int row)

Moves the cursor to the given row number in the result set.

void afterLast()

Moves the cursor to the end of the result set, just after the last row.

void beforeFirst()

Moves the cursor to the front of the result set, just before the first row.

boolean first()

Moves the cursor to the first row in the result set.

Array getArray(String colName)

Gets an SQL ARRAY value in the current row of this ResultSet object.

BigDecimal getBigDecimal(int columnIndex)

Gets the value of a column in the current row as a java.math.BigDecimal object with full precision.

BigDecimal getBigDecimal(String columnName)

Gets the value of a column in the current row as a java.math.BigDecimal object with full precision.

int getConcurrency()

Returns the concurrency mode of this result set.

Date getDate(int columnIndex, Calendar cal)

Gets the value of a column in the current row as a java.sql.Date object.

int getFetchDirection()

Returns the fetch direction for this result set.

int getFetchSize()

Returns the fetch size for this result set.

int getRow()

Retrieves the current row number.

Statement getStatement()

Returns the Statement that produced this ResultSet object.

int getType()

Returns the type of this result set.

boolean isAfterLast()

boolean isBeforeFirst()

boolean isFirst()

boolean isLast()

boolean last()

Moves the cursor to the last row in the result set.

boolean previous()

Moves the cursor to the previous row in the result set.

void refreshRow()

Refreshes the current row with its most recent value in the database. Currently does nothing.

boolean relative(int rows)

Moves the cursor a relative number of rows, either positive or negative.

10.7.3.3 Methods that Return False

The following three methods always return false because this release does not support deletes, inserts, or updates.

boolean rowDeleted()

Indicates whether a row has been deleted.

boolean rowInserted()

Indicates whether the current row has had an insertion.

boolean rowUpdated()

Indicates whether the current row has been updated.

void setFetchDirection(int direction)

Gives a hint as to the direction in which the rows in this result set will be processed.

void setFetchSize(int rows)

Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed for this result set.

10.7.4 Interface Database MetaData

This section describes the JDBC 2.0 Interface Database MetaData methods that are implemented by the Oracle Database Lite JDBC driver.

10.7.4.1 Methods

The following methods can be used to implement the Interface Database MetaData feature.

Connection getConnection()

Retrieves the connection that produced this metadata object.

boolean upportsResultSetConcurrecny(int type, int concurrency)

Supports the concurrency type in combination with the given result set type.

boolean supportsResultSetType(int Type)

Supports the given result set type.

10.7.4.2 Methods that Return False

The following methods return false, because this release does not support deletes or updates.

boolean deletesAreDetected(int Type)

Indicates whether or not a visible row delete can be detected by calling ResultSet.rowDeleted().

boolean insertsAreDetected(int Type)

Indicates whether or not a visible row insert can be detected by calling ResultSet.rowInserted().

boolean othersDeletesAreVisible(int Type)

Indicates whether deletes made by others are visible.

boolean othersInsertsAreVisible(int Type)

Indicates whether inserts made by others are visible.

boolean othersUpdatesAreVisible(int Type)

Indicates whether updates made by others are visible.

boolean ownDeletesAreVisible(int Type)

Indicates whether a result set's own deletes are visible.

boolean ownInsertsAreVisible(int Type)

Indicates whether a result set's own inserts are visible.

boolean ownUpdatesAreVisisble(int Type)

Indicates whether a result set's own updates are visible.

boolean updatesAreDetected(int Type)

Indicates whether or not a visible row update can be detected by calling the method ResultSet.rowUpdated.

10.7.5 Interface ResultMetaData

This section lists methods that can be implemented using the Interface ResultMetaData feature.

10.7.5.1 Methods

The following method can be used to implement the Interface ResultMetaData feature.

String getColumnClassName(int column)

Returns the fully-qualified name of the Java class whose instances are manufactured if the method ResultSet.getObject is called to retrieve a value from the column.

10.7.6 Interface PreparedStatement

This section describes methods that can be implemented using the Interface PreparedStatement feature.

10.7.6.1 Methods

The following methods can be used to implement the Interface PreparedStatement feature.

Result SetMetaDatagetMetaData()

Gets the number, types and properties of a ResultSet's columns.

void setDate(int parameter Index, Date x, Calendar cal)

Sets the designated parameter to a `java.sql.Date` value, using the given Calendar object.

void setTime(int parameterIndex, Time x, Calendar cal)

Sets the designated parameter to a `java.sql.Time` value, using the given Calendar object.

void setTimestamp(int parameter Index, Timestamp x, Calendar cal)

Sets the designated parameter to a `java.sql.Timestamp` value, using the given Calendar object.

10.7.6.1.1 Limitation currently, the option `setQueryTimeOut` is not supported.

10.8 J2ME Support

The following sections describe what Oracle Database Lite supports for J2ME:

- Section 10.8.1, "JDBC Drivers for J2ME CDC and CLDC"
- Section 10.8.2, "J2ME Support for Windows CE"

10.8.1 JDBC Drivers for J2ME CDC and CLDC

Oracle Database Lite JDBC drivers for J2ME are supported in a limited capacity. The following sections describe what are supported in Oracle Database Lite JDBC/J2ME drivers:

- Section 10.8.1.1, "JDBC Driver for J2ME CDC"
- Section 10.8.1.2, "JDBC Driver for J2ME CLDC"

10.8.1.1 JDBC Driver for J2ME CDC

You can use the `olite40.jar` file for JDBC J2ME CDC application development. However, the Oracle Database Lite JDBC driver for J2ME CDC does not implement all classes and methods of the Sun Microsystems "JSR-000169 JDBC Optional Package for CDC/Foundation Profile".

The JDBC definition classes (`java.sql.*`) are an optional package for CDC/Foundation profile based JVMs, as defined by the Javasoft JSR 169 specification. Obtain these classes from your JVM vendor. If your JVM vendor does not supply these classes, then use the sample implementation provided with Oracle Database Lite, which can be found in the *<OLITE_HOME>*`\Mobile\Sdk\samples\j2me` directory.

See Section 10.1, "JDBC Compliance" and Section 10.5, "Java Datatypes and JDBC Extensions" for what is supported in this JAR file for the JDBC driver on J2ME CDC.

10.8.1.2 JDBC Driver for J2ME CLDC

Oracle Database Lite provides the JDBC driver for J2ME CLDC application development. Only a subset of the JDBC APIs are available. The JDBC APIs for J2ME CLDC are provided in the `oracle.lite.jdbc` package, which is included in the `olitejdbccldc.jar` file. You must include this JAR file in your application to use JDBC for J2ME CLDC. The interface is a subset of features available in the Oracle Lite JDBC driver.

See Section 10.1, "JDBC Compliance" and Section 10.5, "Java Datatypes and JDBC Extensions" for what is supported in this JAR file for the JDBC driver on J2ME CDC.

The following lists what is NOT currently implemented for the JDBC API:

- You cannot use any methods requiring floating point data types, such as `ResultSet.getDouble`.
- You cannot use any methods requiring `java.sql.Date`, `java.sql.Time` or `java.sql.Timestamp` data types. When working with SQL date, time and timestamp data, consider using one of the following methods instead: `ResultSet.getString`, `PreparedStatement.setString` and other string getter and setter methods.
- You cannot use any `DatabaseMetaData` objects.
- The JDBC driver has no `finalize()` methods. Applications must explicitly close database objects when done.

- Since the `java.math` library is omitted from MIDP, you may not use the `BigDecimals` object. Use the appropriate `java.lang.String` getter or setter method.

Table 10–6 details the classes and exceptions available in this package. For more information on these classes, such as the supported methods, see the Javadoc in the *Oracle Database Lite API Specification*.

Table 10–6 J2ME CLDC Class and Exception Summary

Classes and Exceptions	Description
`BLOB`	The representation (mapping) in the Java programming language of an SQL BLOB value.
`CLOB`	The representation (mapping) in the Java programming language of an SQL CLOB value.
`Connection`	Connection represents a JDBC connection to an Oracle Lite database.
`PreparedStatement`	A `PreparedStatement` contains a pre-compiled SQL statement which may have parameter markers.
`ResultSet`	Represents a result set which is usually generated by executing a statement that queries the database.
`ResultsSetMetaData`	The `ResultSetMetaData` class can be used to get information about the types and properties of the columns in a `ResultSet` object.
`OliteDataSource`	`OLiteDataSource` partially implements the `javax.sql.DataSource` interface. It is for retrieving a `Connection` object.
`OracleConnection`	Provides the same functional support as the `Connection` object and also provides support for `BLOB/CLOB` objects.
`OraclePreparedStatement`	Provides the same functional support as the `PreparedStatement` object and also provides support for `BLOB/CLOB` objects.
`OracleResultSet`	Provides the same functional support as the `ResultSet` object and also provides support for `BLOB/CLOB` objects.
`Statement`	A `Statement` object is used for executing a static SQL statement and obtaining the results produced by it.
`Types`	The class that defines the constants that are used to identify generic SQL types, called JDBC types. This class is never instantiated.
`DataTuncation`	An exception that reports a `DataTruncation` warning (on reads) or throws a `DataTruncation` exception (on writes) when JDBC unexpectedly truncates a data value.
`SQLException`	An exception that provides information on a database access error or other errors.
`SQLWarning`	An exception that provides information on database access warnings.

For the JDBC/J2ME/CLDC driver, only the JDBC Type 4 connection URL syntax is supported. The following example demonstrates how to create a `Connection` object using the `OLiteDataSource` object.

```
import oracle.lite.jdbc.OLiteDataSource;
import oracle.lite.jdbc.Connection;
import oracle.lite.jdbc.Statement;
```

```
…

public class TestOLiteDataSource implements Runnable
{
  …
  public void run()
  {
    String strUrl = "jdbc:polite4:system/manager@::polite";
    try {
      OLiteDataSource olds = new OLiteDataSource();
      olds.setUrl(strUrl);
      Connection conn = olds.getConnection();
      conn.setAutoCommit(true);
      Statement stmt = conn.createStatement();
      stmt.execute("create table t1(c1 int) ");
    }
    catch (SQLException e) {
  …
    }
  }
}
```

Note: For a full description of what is supported for each object, see the *Oracle Database Lite API Specification* for the Javadoc on these objects.

10.8.2 J2ME Support for Windows CE

Oracle Database Lite is certified with the following JVMs on Windows Mobile 2003 Second Edition:

- IBM J9 Websphere Everyplace Micro Environment for Windows Mobile 2003 ARM Personal Profile
- Creme JVM, which can be obtained at `http://www.nsicom.com`

The following sections describe which class to use in connecting to an Oracle Lite database for each JVM type:

- Section 10.8.2.1, "Using IBM J9"
- Section 10.8.2.2, "Using Creme 4.1"

10.8.2.1 Using IBM J9

When using IBM J9, use the `DataSource` class to connect to an Oracle Lite database, as shown below:

```
POLJDBCDataSource dsPolite = new POLJDBCDataSource();
dsPolite.setUrl(DSN);
dsPolite.setUser(UserName);
dsPolite.setPassword(Password)
politeConnection = dsPolite.getConnection();
```

Perform the following to execute the `ExampleClass` sample class, which is part of the `ExamplePackage.jar`:

```
j9 -jcl:ppro10 "-Xbootclasspath/p:path\classes.zip;path\jdbcjsr169.jar"
 -classpath "path\jdbcjsr169.jar;\Orace\olite40.jar;path\ExamplePackage.jar"
 ExampleClass
```

Note: The `jdbcjsr169.jar` can be obtained from the SDK installation in the `<`*`ORACLE_HOME`*`>\Mobile\Sdk\samples\j2me` directory.

Where the `jdbcjsr169.jar` file contains the optional JDBC definitions. For more details, refer to Section 10.8.1.1, "JDBC Driver for J2ME CDC".

Note: Ensure that you replace the path with the correct path to the required JAR and ZIP files.

10.8.2.2 Using Creme 4.1

When using the Creme 4.1 JVM, use the `DriverManager` class to connect to an Oracle Lite database, as shown below:

```
politeConnection = DriverManager.getConnection(DSN,UserName,Password);
```

Perform the following command at the Creme Command prompt to execute the `ExampleClass` sample class, which is part of the `ExamplePackage.jar` file:

```
Creme -Of -classpath '<path>\jdbcjsr169.jar;\Oracle\olite40.jar;
        <path>\ExamplePackage.jar' ExampleClass <command_line_arguments>
```

11 Oracle Database Lite ADO.NET Provider

The following sections discuss the Oracle Database Lite ADO.NET provider for Microsoft .NET and Microsoft .NET Compact Framework. The Oracle Database Lite ADO.NET provider resides in the `Oracle.DataAccess.Lite` namespace.

- Section 11.1, "Discussion of the Classes That Support the ADO.NET Provider"
- Section 11.2, "Limitations for the ADO.NET Provider"

11.1 Discussion of the Classes That Support the ADO.NET Provider

The Oracle Database Lite ADO.NET driver is implemented in the following DLLs:

- Windows—`Oracle.DataAccess.Lite.dll`
- Windows CE—`Oracle.DataAccess.Lite_wce.dll`

If you are building an application that uses the ADO.Net driver, you must package the driver with the application files. The assembly DLL must be located in the same directory as your application executable (`*.exe` file). Alternatively, you can add the driver to the Windows global assembly cache. See the Microsoft documentation on how to add this DLL to the Global Assembly Cache.

The following sections describe classes for the Oracle Database Lite ADO.NET provider:

- Section 11.1.1, "Establish Connections With the OracleConnection Class"
- Section 11.1.2, "Transaction Management"
- Section 11.1.3, "Create Commands With the OracleCommand Class"
- Section 11.1.4, "Maximize Performance Using Prepared Statements With the OracleParameter Class"
- Section 11.1.5, "Large Object Support With the OracleBlob Class"

11.1.1 Establish Connections With the OracleConnection Class

The `OracleConnection` interface establishes connections to Oracle Database Lite. This class implements the `System.data.IDBConnection` interface. When constructing an instance of the `OracleConnection` class, implement one of the following to open a connection to the back-end database:

- Pass in a full connection string as described in the Microsoft ODBC documentation for the `SQLDriverConnect` API, which is shown below:

```
OracleConnection conn = new OracleConnection
      ("Data_Directory=\\orace;Database=polite;DSN=*;uid=system;pwd=manager");
```

```
conn.Open();
```

- Construct an empty connection object and set the `ConnectionString` property later.

With an embedded database, we recommended that you open the connection at the initiation and leave it open for the life of the program. When you close the connection, all of the `IDataReader` cursors that use the connection are also closed.

11.1.2 Transaction Management

By default, Oracle Database Lite connection uses the autocommit mode. If you do not want the autocommit to be on, then you can start a transaction with the `BeginTransaction` method in the `OracleConnection` object. The `BeginTransaction` method returns a reference to the `IDbTransaction` object. Then, when finished, execute either the `Commit` or `Rollback` methods on the returned `IDbTransaction`, which either commits or rolls back the transaction. Once the transaction is completed, the database is returned to autocommit mode.

Whenever you rollback a transaction, it rolls back all the operations that you have performed before. Sometimes you need to undo the transaction to a certain point. For this scenario, you can use the Save Points functionality. Save Points allows you to rollback a transaction to a certain point. Oracle Lite supports Save Points. Within a transaction, you can set up, remove or undo any number of Save Points using SQL statements. Using save points gives you better granular control over the transaction. Within the transaction, use SQL syntax to set up, remove and undo savepoints.

For WinCE devices, Oracle Database Lite supports only one process to access a given database. When a process tries to connect to a database that is already in use, the `OracleConnectionOpen` method throws an `OracleException`. To avoid this exception being thrown, close a connection to allow another process to connect.

The following is an example in turning off autocommit for a C# application:

```
OracleConnection conn = new OracleConnection ("DSN=consroot;uid=system");
conn.Open();
IDbTransaction trans = conn.BeginTransaction(); // Turn off AUTOCOMMIT
OracleCommand cmd = (OracleCommand)conn.CreateCommand();
cmd.CommandText = "create table TEST1 (c0 number)";
cmd.ExecuteNonQuery();
trans.Commit(); // AutoCommit is 'ON'
cmd.Dispose();
conn.Close();
```

The following is an example in turning off autocommit for a VB.NET application:

```
conn = New Oracle.DataAccess.Lite.OracleConnection(("DSN=consroot;uid=system")
conn.Open()
IDbTransaction trans = conn.BeginTransaction()
OracleCommand cmd = New OracleCommand (conn)
cmd.CommandText = "create table TEST1 (c0 number)"
cmd.ExecuteNonQuery()
trans.Commit()
cmd.Dispose()
conn.Close()
```

11.1.3 Create Commands With the OracleCommand Class

The `OracleCommand` class implements the `System.Data.IDBCommand` interface. Create any commands through the `CreateCommand` method of the

`OracleConnection` class. The `OracleCommand` has constructors recommended by the ADO.NET manual, such as `OracleCommand(OracleConnection conn, string cmd)`.

However, if you use the `OracleCommand` constructors, it is difficult to port the code to other platforms, such as the ODBC provider on Windows 32. Instead, create the connection and then use interface methods to derive other objects. With this model, you can either change the provider at compile time or use the reflection API at runtime.

11.1.4 Maximize Performance Using Prepared Statements With the OracleParameter Class

Parsing a new SQL statement can take significant time; thus, use prepared statements for any performance-critical operations. Although, `IDbCommand` has an explicit `Prepare` method, this method always prepares a statement on the first use. You can reuse the object repeatedly without needing to call `Dispose` or change the `CommandText` property.

11.1.4.1 SQL String Parameter Syntax

Oracle Database Lite uses ODBC-style parameters in the SQL string, such as the `?` character. Parameter names and data types are ignored by the driver and are only for the programmer's use.

For example, assume the following table:

```
create table t1(c1 int, c2 varchar(80), c3 data)
```

You can use the following parameters in the context of this table:

```
IDbCommand cmd = conn.CreateCommand();
cmd.CommandText  = "insert into t1 values(?,?,?);"
cmd.Parameters.Add("param1", 5);
cmd.Parameters.Add("param2", "Hello");
cmd.Parameters.Add("param3", DateTime.Now);
cmd.ExecuteNonQuery();
```

Note: The relevant class names are `OracleParameter` and `OracleParameterCollection`.

11.1.5 Large Object Support With the OracleBlob Class

The `OracleBlob` class supports large objects. Create a new `OracleBlob` object to instantiate or insert a new BLOB object in the database, as follows:

```
OracleBlob blob = new OracleBlob(conn);
```

Since the BLOB is created on a connection, you can use the `Connection` property of `OracleBlob` to retrieve the current `OracleConnection`.

Functions that you can perform with a BLOB are as follows:

- Section 11.1.5.1, "Using BLOB Objects in Parameterized SQL Statements"
- Section 11.1.5.2, "Query Tables With BLOB Columns"
- Section 11.1.5.3, "Read and Write Data to BLOB Objects"

11.1.5.1 Using BLOB Objects in Parameterized SQL Statements

You can use the BLOB object in parameterized SQL statements, as follows:

```
OracleCommand cmd = (OracleCommand)conn.CreateCommand();
cmd.CommandText = "create table LOBTEST(X int, Y BLOB)";
cmd.ExecuteNonQuery();
cmd.CommandText = "insert into LOBTEST values(1, ?)";
cmd.Parameters.Add(new OracleParameter("Blob", blob));
cmd.ExecuteNonQuery();
```

11.1.5.2 Query Tables With BLOB Columns

You can retrieve the `OracleBlob` object using the data reader to query a table with a BLOB column, as follows:

```
cmd.CommandText = "select * from LOBTEST";
IDataReader rd = cmd.ExecuteReader();
rd.read();
OracleBlob b = (Blob)rd["Y"];
```

Or you can write the last line of code, as follows:

```
OracleBlob b = (OracleBlob)rd.getvalue(1);
```

11.1.5.3 Read and Write Data to BLOB Objects

The `OracleBlob` class supports reading and writing to the underlying BLOB, and retrieving and modifying the BLOB size. Use the `Length` property of `OracleBlob` to get or to set the size. Use the following functions to read and write to the BLOB, as follows:

```
public long GetBytes(long blobPos, byte [] buf, int bufOffset, int len);
public byte [] GetBytes(long blobPos, int len);
public void SetBytes(long blobPos, byte [] buf, int bufOffset, int len);
public void SetBytes(long blobPos, byte [] buf);
```

For example, the following appends data to a BLOB and retrieves the bytes from position five forward:

```
byte [] data = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
 blob.SetBytes(0, data); //append data to the blob
byte [] d = blob.GetBytes(5, (int)blob.Length - 5); //get bytes from position 5 up
to the end
blob.Length = 0; //truncate the blob completely
```

Use the `GetBytes` method of the data reader to read the BLOB sequentially, but without accessing it as a `OracleBlob` object. You should not, however, use the `GetBytes` method of the reader and retrieve it as a `OracleBlob` object at the same time.

11.2 Limitations for the ADO.NET Provider

The following are limitations to the Oracel Database Lite ADO.NET provider:

- Section 11.2.1, "Partial Data Returned with GetSchemaTable"
- Section 11.2.2, "Creating Multiple DataReader Objects Can Invalidate Each Other"
- Section 11.2.3, "Calling DataReader.GetString Twice Results in a DbNull Object"
- Section 11.2.4, "Thread Safety"

11.2.1 Partial Data Returned with GetSchemaTable

The Oracle Database Lite ADO.NET provider method—`GetSchemaTable`—only returns partial data. For example, it claims that all of the columns are primary key, does not report unique constraints, and returns null for `BaseTableName`, `BaseSchemaName` and `BaseColumnName`. Instead, to retrieve Oracle Database Lite meta information, use `ALL_TABLES` and `ALL_TAB_COLUMNS` instead of this call to get Oracle Database Lite meta information.

11.2.2 Creating Multiple DataReader Objects Can Invalidate Each Other

The Oracle Database Lite ADO.NET provider does not support multiple concurrent `DataReader` objects created from a single `OracleCommand` object. If you need more than one active `DataReader` objects at the same time, create them using separate `OracleCommand` objects.

The following example shows how if you create multiple `DataReader` objects from a single `OracleCommand` object, then the creation of `reader2` invalidates the `reader1` object.

```
OracleCommand cmd = (OracleCommand)conn.CreateCommand();
cmd.CommandText = "SELECT table_name FROM all_tables";
cmd.Prepare();
IDataReader reader1 = cmd.ExecuteReader();
IDataReader reader2 = cmd.ExecuteReader();
```

11.2.3 Calling DataReader.GetString Twice Results in a DbNull Object

Calling the `GetString` method of `DataReader` twice on the same column and for the same row results in a `DbNull` object. The following example demonstrates this in that the second invocation of `GetString` results in a `DbNull` object.

```
 IDataReader dr = cmd.ExecuteReader();
        String st = null;
        while(dr.Read())
        {
                st = dr.GetString (1);
                st = dr.GetString (1);
        }
```

11.2.4 Thread Safety

To build a thread-safe program, make sure that different threads use separate `IDbCommand` and `IDataReader` objects. The `OracleConnection` and `IDbTransaction` methods can be called concurrently, except for when used to open and close the connection.

12
Using Simple Object Data Access (SODA)

SODA is an interface used in Oracle Database Lite C++ development that provides object-oriented data access using method calls, relational access using SQL and object-relational mapping to bridge the gap between the two. Object functionality is roughly three times faster than ODBC for simple operations. It enables rich datatypes—such as arrays and object pointers—as well as standard SQL columns. A programmer can store any data structure in the database and not think about relational design or performing joins.

A C++ developer can also use an interface for executing SQL statements. The resulting code is shorter and cleaner than ODBC. SQL queries can return objects to be examined and modified directly through the object-oriented layer, without calling any additional SQL statements.

Object-relational mapping enables the application to access relational data as if it was object hierarchy. Thus, your application can replicate rich data types or object pointers to the Oracle database server.

The SODA API method calls are documented in the *SODA: Simple Object Data Access API Reference*, which is located off the `<ORACLE_HOME>/Mobile/index.htm` page. The full sample code that is demonstrated in this chapter is located in the `<ORACLE_HOME>/Mobile/doc/soda/sodadoc/html/sodasimple_8cpp-source.html` file.

- Section 12.1, "Getting Started With SODA"
- Section 12.2, "Using SQL Queries in SODA Code for PocketPC Platforms"
- Section 12.3, "Virtual Columns and Object-Relational Mapping"
- Section 12.4, "Behavior of Reference-Counted and Copy-By-Assignment Objects"
- Section 12.5, "Another Library for Exceptions (ALE)"

12.1 Getting Started With SODA

In order to get started with SODA quickly, the following sections discuss the most frequently used classes:

- Section 12.1.1, "Overview of the SODA Classes"
- Section 12.1.2, "Demonstrating Frequently-Used SODA Classes"

12.1.1 Overview of the SODA Classes

When developing your C++ application, you would use the following classes the most:

- `DBSession` connects to the database and find and create classes.
- `DBClass` creates new database objects.
- `DBObject` modifies existing objects.
- `DBData` wraps an attribute value and is used for type conversion.
- `DBQueryExpr` builds single-table queries supported by SODA.
- `DBString (olString)` is a C string wrapper used by SODA.
- `DBList (olList)` is a template to store lists of values.
- `DBColList` and `DBDataList` instantiate these objects.

For example, implement the `DBObject` method, as follows:

```
DBObject obj;
...
obj["NAME"] = "Jack"
```

For full documentation for the SODA API method calls, see the SODA APIs, which you can find off the `<ORACLE_HOME>/Mobile/index.htm` page.

12.1.2 Demonstrating Frequently-Used SODA Classes

The following example demonstrates most of the SODA object-oriented functionality, as discussed in Section 12.1.1, "Overview of the SODA Classes".

```
void helloSODA() {
  puts("Hello SODA");
  try {
   DBSession sess("POLITE"); // Connect to the DSN, creating it if necessary

   // Find or create our class
   DBClass cls;
   try {
     cls = sess["PEOPLE"];
   } catch(DBException e) {
   cls = sess.createClass("PEOPLE", DBAttrList() <<
   DBAttr("ID", DB_INT) << DBAttr("NAME", DB_STRING));
  }
  // Create several objects. We can identify columns by name or positions
  DBObject o = cls.create("ID", 10, "NAME", "Alice");

  // The previous syntax of col1, val1, col2, val2 ... works for up to
// 32 columns. This version works for any number of columns
   cls.create(DBSetList() << "ID" << 10 << "NAME" << "Alice");
   cls.create(0, 20, 1, "Bob");

   // Note the automatic type conversion
   cls.create("ID", "314", 1, 3.14159265358);

   // Execute a query (will return two objects)
   DBCursor c = cls.createCursor(DBColumn("ID") == 10 ||
          DBColumn("NAME") == "Bob");

  DBObject ob;

  while (ob = ++c) {
   DBString s = ob["NAME"];
   puts(s);
   o["ID"] = (int)o["ID"]+1;
  }

  // Delete an object
  o.remove();

  // Clean up so that create class is successful next time
  sess.rollback();
 } catch(DBException e) {
 DBString s = e.getMessage();
 printf("Error: %s\n", (const char *)s);
 }
}
```

12.2 Using SQL Queries in SODA Code for PocketPC Platforms

To add SQL queries to your SODA code for PocketPC platforms, do the following:

1. Include `sodasql.h`, instead of `soda.h`.
2. Link your program with `sodasql.lib`.
3. Install `sodasql.dll` at runtime.
4. Create a `DBSqlSession` object instead of the `DBSession` object. Execute relational queries and other SQL statements with the help of `DBSqlStmt` and `DBSqlCursor` classes. Query results can be returned either as column values or as `DBObjects` for matching rows.

The following is a sample that uses the SODA relational interface:

```
void helloSQL() {
 try {
  puts("Hello SQL");
  DBSqlSession sess("POLITE");
  sess.execute("create table odtest(c1 int, c2 varchar(80))");
  DBSqlStmt stmt = sess.prepare("insert into odtest values(?,?)");

  // The values stand in for two ?'s in the statement above
  stmt.execute(5, "John");
  stmt.execute(10, "Mike");
  stmt.execute(15, "Alice");

  // Execute a single-table query that will return DBObject's for matching
  // rows. Use a standard SODA interface to access the objects
  DBSqlCursor c = sess.execute("odtest", "c1 < 15");
  while (++c) {
    DBObject o = c.getObject();
    DBString s = o["c2"];
    int val = o["c1"];
    printf("%d %s\n", val, (const char *)s);
    o["c1"] = val+1; // Can modify objects in addition to just reading them
  }

  // Execute a usual relational query
  c = sess.execute("select * from odtest");
  while (++c) {
    DBString s = c["c2"];
    printf("%d %s\n", (int)c["c1"], (const char *)s);
  }
 } catch(DBException e) {
 DBString s = e.getMessage();
 printf("Error: %s\n", (const char *)s);
 }
}
```

12.3 Virtual Columns and Object-Relational Mapping

A programmer does not view the data as column values that are stored in the database. For example, a master-detail relationship might be expressed as matching values in two tables, but for a program it is more natural to access a column in the master object, which contains an array of pointers to details.

The `DBVirtualCol` class enables the translation between the conceptual view of the data and the actual data in the tables. You can create a column that is completely under programmer's control through the `get`, `set` and `remove` methods and adding it to a class at runtime.

In fact, SODA contains a specialized `DBValueRel` class that extends the `DBVirtualCol` class to map master-detail relationships to object pointers. The following sample builds a binary search tree in the database using object-relational mapping:

```
struct HelloVirtual {
 int lpos, rpos, vpos;
 void visit(DBObject o);
 HelloVirtual();
};

void HelloVirtual :: visit(DBObject o) {
 while (o) {
 visit(o[lpos]);
 printf("%d\n", (int)o[vpos]);
 o = o[rpos];
 }
}

HelloVirtual :: HelloVirtual() {
 try {
   DBSession sess("POLITE");
   // TreeNode represent a binary tree with left and right pointers
   // that point back to parent's id column
   DBClass cls = sess.createClass("TreeNode",
   DBAttrList() << DBAttr("val", DB_INT) << DBAttr("id", DB_INT)
     << DBAttr("lchild", DB_INT) << DBAttr("rchild", DB_INT));
   // Create an index on id to speed up search
   cls.createIndex("i1", DBColList() << "id", true);
   // Set a sequence as a default value of id, so that we don't have to set it
   // explicitely
   DBSequence seq = sess.createSequence("s1");
   cls.defaultVal("id", seq);
   // Create the virtual columns
   DBValueRel lref("left", DBSrcCol(cls, "lchild") -> DBDstCol(cls, "id"),
        DB_UPD_DETAIL);
   DBValueRel rref("right", DBSrcCol(cls, "rchild") -> DBDstCol(cls, "id"),
        DB_UPD_DETAIL);
   // Cache column positions for frequent access
   lpos = cls["left"], rpos = cls["right"], vpos = cls["val"];
   // Root of the binary tree
   DBObject root;
   // Insert some random numbers into our binary search tree
   for (int i = 0; i < 50; i++) {
    int v = rand();
    DBObject o = cls.create(vpos, v); // Note automatically generated ids
    DBObject par; int dir;
    for(DBObject cur = root; cur; par = cur, cur = cur[dir])
    dir = (int)cur[vpos] >= v ? lpos : rpos;
    if (par) par[dir] = o; else root = o;
   }
   // Do in-order traversal of the tree, printing out numbers in sorted order
   visit(root);
  } catch(DBException e) {
  DBString s = e.getMessage();
  puts(s);
 }
}
```

12.4 Behavior of Reference-Counted and Copy-By-Assignment Objects

Most C++ classes in SODA are reference-counted, which means that assigning one variable of one type to another cannot copy an object, but creates another way to refer to the same object. For example, the `DBData` class represents values that can be stored in persistent objects.

The following example demonstrates reference-counting:

```
DBData d = 5; // Create a new object containing value 5
                 // and make a reference to it
DBData d2 = d; // Both reference the same object
d << 20; // Add another value to existing object.
         // Both d and d2 reference the new array
d2 = 10; // d references the array, d2 is reassigned to the new data.
d.clear(); // Clear the last reference to the array of 5 and 20
           //and free the array
```

The programmer does not need to free objects when they are no longer used. However, this method is relatively expensive and not practical for objects that are created and destroyed often, such as when new lists of values, such as `DBSetList`, are allocated for each SODA call. Various lists in SODA, such as `DBSetList`, `DBDataList` and so on, are copy-on-assignment rather than reference-counted objects.

The following example demonstrates copy-by-assignment:

```
DBSetList ls << "cost" << 1000;
DBSetList ls2 = ls; // Created a copy of ls
ls << "value" << "priceless" // Only ls is changed
ls2.clear(); // Just set this copy to size 0
```

Note: SODA includes non-database classes and templates for your convenience, which have names that start with `ol` rather than `DB`—such as `olHash`. Avoid making unnecessary copies of these classes.

You can optimize your implementation by not creating an unnecessary copy by passing a reference or a `const` reference to the object, rather than an object, for both reference-counted and copy-on-assignment classes when calling a function. For example, `void func(const DBData &v)` avoids creating an unnecessary copy.

Note: The `clear` method only nullifies a particular reference to reference-counted objects. `DBSession` and `DBCursor` classes provide a `close` method that releases the underlying database resources, even while the objects are still referenced. Using anything that relies on a closed cursor or database connection throws a `DBException`.

12.5 Another Library for Exceptions (ALE)

Many embedded compilers, such as Visual C++ for PocketPC, do not support C++ exceptions. Oracle Database Lite includes ALE, which is a library that closely mimics C++ exceptions. The following sections describe how to use ALE:

- Section 12.5.1, "Decorating Classes With ALE"
- Section 12.5.2, "New Operator and ALE"
- Section 12.5.3, "Global Variables"
- Section 12.5.4, "Exceptions and Inheritance"
- Section 12.5.5, "Using ALE with PocketPC ARM Compilers"
- Section 12.5.6, "Troubleshooting ALE Runtime Errors"
- Section 12.5.7, "Compiling Your Program With ALE"
- Section 12.5.8, "ALE Code on Systems That Support Exceptions"

12.5.1 Decorating Classes With ALE

A decorated class is one that uses ALE for handling its exceptions. If your embedded compiler does not support exception handling, then use ALE, which relies on careful accounting of all objects that are already constructed or are being constructed. ALE supports stack unwinding and catching exceptions based on object type.

To use ALE, C++ source code needs to be modified where the try, catch and throw blocks are replaced with the ALE macros, which is known as decorating classes. The following is an example of a decorated class:

```
#include "ale.h"
struct Error {
    const char *msg;
    Error(const char *msg) :msg(msg) {}
};

aleTry {
    olArray<char> a(5);
    aleThrow(Error,Error("Adios"));// Or aleThrowObj(Error,("Adios"));
} aleCatch(Error,e) {
    puts(e.msg);
} aleCatch(aleBadAlloc,e) {
    puts("Tough!");
} aleCatchAll {
    puts("Some other exception happened\n");
    aleReThrow;
} aleEnd;
```

Table 12–1 ALE Macros for C++ Exceptions

Macro	Action
`aleTry`	Equivalent to C++ try. Use to enclose the code that might encounter any exception.
`aleCatch (type, varName)`	Catch exception of a given type and store it in the variable `varName`, which is local to the block. Unlike the regular C++ exceptions, the `type` name string must match the argument of the throw exactly.
`aleThrow (type, obj)`	Throw an exception contained in `obj` of `type`. ALE supports single inheritence of exceptions, as described in Section 12.5.4, "Exceptions and Inheritance".
`aleThrowObj (type, arg1, arg2, ....)`	Construct a new object of `type` with the specified arguments and throw it as an exception.
`aleCatchAll`	Catch any exceptions that are not handled explicitly.
`aleReThrow`	Rethrow the exception that is caught in the innermost `aleCatch` or `aleCatchAll` block.
`aleEnd`	Close the exception handling construct. Add a semi-colon `;` after `aleEnd`.

If your embedded compiler does not support exception handling, then use ALE, which relies on careful accounting of all objects that are already constructed or are being constructed. Your class is involved in exception handling if the class is on the stack when an exception is thrown, its constructor may throw an exception, or it has a decorated superclass and member—even if the class does not do any additional exception-related processing.

When you decorate one class with ALE, you must decorate all classes involved with this class. If you omit a decoration in one of the classes, then the program might fail. Therefore, it is best to decorate all your classes except for plain C-style structures that do not have any constructors or destructors.

If your class does not have any constructors with a body that throws exceptions (it is OK if the superclass or member constructor does), then you can use a simple form of class decoration by adding `ALELAST(ClassName)`, without a semicolumn, at the end of class declaration. `ALELAST` informs the library to register the class and clean-up if an error occurs. The following demonstrates how to use `ALELAST`:

```
template<class T> class PtrHolder {
   T *ptr;
public:
   ~PtrHolder() { delete ptr; }
   operator T *() { return ptr; }
   ALELAST(PtrHolder)
};
```

If any of the constructors throw exceptions, then you need to do the following:

1. Add `ALECLAST(ClassName)`, rather than `ALELAST` to the end of the class body.
2. Add `ALECONS(ClassName)` in the beginning of the body of each constructor.

This is demonstrated, as follows:

```
template<class T> class Array {
    T *a;
    size_t len;
public:
    Array(size_t len=0} : len(len) {
        ALECONS(Array);
        a = new T[len];
    }
    Array(const Array &arr) : len(arr.len) {
        ALECONS(Array);
        for (size_t i = 0; i < len; i++)
            a[i] = arr.a[i];
    }
    ...
    ALECLAST(Array)
};
```

In this example, the class contains an explicit copy constructor. If your class does not contain an explicit copy constructor and instances can be copied, then you need to explicitly write a copy constructor and add `ALECONS(ClassName);` rather than using a copy constructor that is generated by the compiler. If you need a more complicated initialization than a default or copy constructor, then use a global pointer—which can be initialized by another global object, rather than a global instance.

12.5.2 New Operator and ALE

Decorated classes can be safely used with the `new` and `delete` functions, including using `new` and `delete` for `array` and `placement new`. However, systems that do not support exceptions usually do not declare `std::bad_alloc` and `std::no_throw` types. Use `aleBadAlloc` and `aleNoThrow` instead of using `std::bad_alloc` and `std::no_throw` types.

One design decision is whether to decorate classes with constructors that only throw `bad::alloc` if they run out of memory using `ALELAST` or `ALECLAST`. If you are writing classes for a single application that does not allocate much memory, you might dispense with error checking and just use `ALELAST`. Your program might crash because of incorrect cleanup calls if it runs out of memory. If your class allocates a lot of memory or you are writing a highly-reusable framework, then it is best to use `ALECLAST` and decorate all the constructors.

12.5.3 Global Variables

If you need a global or static variable to be decorated with ALE, declare it using `aleGlobal` template, as follows:

```
aleGlobal<MyType> myGlobal; // Initialized with default constructor
aleGlobal<MyType> myGlobal1(MyType("Hello", 5")); // Initialized with copy
constructor.
...
MyType *t = myGlobal; // Declared variables behave as pointers
```

Declare all global or static decorated instances using `aleGlobal` or you may receive runtime errors.

12.5.4 Exceptions and Inheritance

Unlike regular C++ exception handling, ALE requires that class names in `aleThrow` and `aleCatch` match exactly. Typedef names, throwing a subclass, and catching a superclass will not work. To build a hierarchy of exceptions, add `ALEPARENT` declaration to the subclass, as follows:

```
class BaseE {
    ALELAST(BaseE)
};
class DerivedE : public BaseE {
    ALELAST(DerivedE)
    ALEPARENT(BaseE)
};
```

`DerivedE` can be caught as `BaseE`. If you use multiple inheritance, then the first base class must be declared as a parent.

12.5.5 Using ALE with PocketPC ARM Compilers

The Microsoft Embedded Visual C++ for ARM has a bug that is triggered when an ALE-decorated object (or any object with embedded pointers) is passed by value to a function or method. The affected code receives an ALE fatal error message at runtime. To avoid this problem, always pass SODA objects and instances of other classes that use ALE as a constant reference rather than value. For example, modify the following code:

```
void createName(DBClass cls, DBString name) {
cls.create("name", name);
}
```

to the following implementation:

```
void createName(const DBClass &cls, const DBString &name) {
cls.create("name", name);
```

12.5.6 Troubleshooting ALE Runtime Errors

If your classes are not decorated properly, then you will receive runtime errors. On PocketPC, ALE displays a message box explaining the problem, and then the program terminates. In addition, the error and the dump of the ALE stack is appended to `aleDump.txt`, which exists in the root directory of the device. In simple cases, the error message pinpoints the exact problem; for example `ALECONS is missing for class MyArray`. Usually, one of the classes found near the top of the ALE stack is not decorated properly. If you do not decorate a class and its superclasses or members are decorated, then you may receive a runtime error and see the superclasses/members on the stack.

12.5.7 Compiling Your Program With ALE

To build a program that uses ALE, include `ale.h` from the Oracle Database Lite SDK and link with the `olStdDll.lib` library. You need `olStdDll.dll` at runtime.

12.5.8 ALE Code on Systems That Support Exceptions

For systems that already support C++ exceptions, like Win32, Oracle Lite includes a dummy `ale.h` that defines the same macros, but uses regular C++ exceptions to implement them. If you are writing code that must execute on both Win32 and PocketPC, remember to test the code with the actual ALE library to ensure that all your classes are decorated correctly. `ALELAST`, `ALECLAST` or `ALECONS` have no effect on the Win32 platform.

13 Using Stored Procedures and Triggers

Oracle Database Lite enables you to use stored procedures written in Java, C, C++ and C#. The following sections describe how you can load and define these stored procedures in the Oracle Lite database:

13.1 Overview of Stored Procedures and Triggers

A stored procedure is a method that is stored in Oracle Database Lite. The procedure can be invoked by applications that access the database. Stored procedures can return a single value, a row, or multiple rows.

A trigger is a stored procedure that executes when a specific event occurs, such as a row update, insertion, or deletion. An update of a specific column can also fire a trigger. Triggers, however, cannot return a value. A trigger can operate at the statement-level or row-level.

- A statement-level trigger fires once per triggering statement, no matter how many rows are affected.
- A row-level trigger fires once for every row affected by the triggering statement.

Note: For more information on general knowledge of creation and management of stored procedures, see the *Oracle Database* documentation.

13.2 Using Java Stored Procedures in Oracle Database Lite

To create a Java stored procedure, perform the following:

1. Create the class that you want to store in Oracle Database Lite. You can use any Java IDE to write the procedure, or you can simply reuse an existing procedure that meets your needs.

   When creating the class, consider the following restrictions on calling Java stored procedures from SQL DML statements:

   - When called from an `INSERT`, `UPDATE`, or `DELETE` statement, the method cannot query or modify any database tables modified by that statement.
   - When called from a `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement, the method cannot execute SQL transaction control statements, such as `COMMIT` or `ROLLBACK`.

   Note: Any SQL statement in a stored procedure that violates a restriction produces an error at run time.

2. Provide your class with a unique name for its deployment environment, since only one Java Virtual Machine is loaded for each Oracle Database Lite application. If the application executes methods from multiple databases, then the Java classes from these databases are loaded into the same Java Virtual Machine. We recommend that you prefix the Java class name with the database name to ensure that the Java class names are unique across multiple databases.
3. If you are executing any DML statements in your Java stored procedure, then—in order for these statements to exist within the same transaction—you must pass an argument of type `java.sql.Connection` as the first argument in the method. You must have the `Connection` object in order to prepare and execute any statements. Oracle Database Lite supplies the appropriate argument value of the Oracle Lite database `Connection` object for you; the application executing the method does not need to provide a value for this parameter.

Once created, Java stored procedures must be loaded into any database—including the Oracle Lite database—with either the `loadjava` Java utility or with the SQL command, `CREATE JAVA`. In Oracle Database Lite, once the Java stored procedure is loaded, you define how the table can use the methods of the stored procedure in one of the following ways:

- The table can invoke the methods within the Java stored procedure directly when you attach it to the table with with the `ALTER TABLE ATTACH JAVA` SQL command. When you attach the Java stored procedure, then the table inherits all methods of the Java stored procedure class.

  When you attach the Java class to a table then the static methods in the class become table-level stored procedures of the table and the non-static (instance) methods become row-level stored procedures.

- Execute the `CREATE FUNCTION` or `CREATE PROCEDURE` SQL command defines a call specification which declares a Java method so that it can be called from SQL. The call specification tells Oracle which Java method to invoke when a call is made.
- Execute the `CREATE TRIGGER` SQL command which specifies that the Java stored procedure executes when a specific event occurs on the table, such as an insert, update, or delete.

13.2.1 Load and Define Java Stored Procedures in an Oracle Lite Database

When you want to use Java stored procedures in an Oracle Lite database, the administrator must manually load and define the stored procedures either through utilities or within a SQL script, as follows:

1. Load the Java class into the Oracle Lite database with either the `loadjava` command-line utility or the SQL statement `CREATE JAVA`.

   The `loadjava` utility automates the task of loading Java classes into the database. Using `loadjava`, you can load Java class, source, and resource files, individually or in archives.

2. Define the methods in the class that you want to call from SQL. As described in Section 13.2, "Using Java Stored Procedures in Oracle Database Lite", you can attach the entire stored procedure for all methods to be defined, or create the call specification with either the `CREATE FUNCTION` or `CREATE PROCEDURE` commands.

> **Note:** For more information, see Section 13.3.1, "Using Load and Define for Java Stored Procedures".

13.3 Creating Java Stored Procedures

Oracle Database Lite supports the following development models for creating Java stored procedures:

- Section 13.3.1, "Using Load and Define for Java Stored Procedures"
- Section 13.3.2, "Using Attach to Define the Java Stored Procedure"

In addition, see the following sections for additional information about managing your Java stored procedures:

- Section 13.3.3, "Calling Java Stored Procedures From a Multithreaded C or C++ Application"

13.3.1 Using Load and Define for Java Stored Procedures

You only need to define procedures that should be callable from SQL. Many stored procedures are only called by other stored procedures, and do not need to be defined. Only static methods are supported when you do load and define Java stored procedures.

As referred to in previous sections, perform the following to create Java stored procedures:

1. Develop a Java class that contains the methods you want to store. Make sure that the class compiles and executes without errors.

2. Load the Java class into Oracle Database Lite with either the `loadjava` utility or the `SQL CREATE JAVA` command.

3. Define any static methods in the Java class that you want to make accessible to SQL by creating call specifications for these methods. By defining a method, you associate a SQL name to the method. SQL applications use this name to invoke the method.

   This model is supported by Oracle database, which enables you to utilize skills and resources you have already developed in implementing Oracle database enterprise applications and data. There is the following difference:

- In Oracle Database Lite, you cannot define a method that is mapped to a `main` method.
- In the Oracle database, call specs that define `main` methods are permitted.

4. Invoke the stored procedure through a SQL DML statement.
5. If you no longer intend to use the stored procedure, you can drop it from the database.

Note: The load and define development model only supports Java static methods. To store static and non-static (instance) methods, you must attach the class to database tables, as described in Section 13.3.2, "Using Attach to Define the Java Stored Procedure".

The following sections describe in detail how to perform these functions:

- Section 13.3.1.1, "Loading Java Stored Procedure Classes Into the Oracle Lite Database"
- Section 13.3.1.2, "Defining Stored Procedures to SQL Using Create Function or Create Procedure"
- Section 13.3.1.3, "Calling Defined Stored Procedures"
- Section 13.3.1.4, "Dropping Defined Stored Procedures"
- Section 13.3.1.5, "Example Using the Load and Define Model"

13.3.1.1 Loading Java Stored Procedure Classes Into the Oracle Lite Database

To load Java classes into the Oracle Database Lite database, you can use one of the following:

- Section 13.3.1.1.1, "loadjava"—The `loadjava` database command-line utility automates the task of loading Java classes into Oracle Database Lite and Oracle databases.
- Section 13.3.1.1.2, "Using CREATE JAVA"—The SQL statement `CREATE JAVA` loads Java classes manually.

13.3.1.1.1 loadjava The `loadjava` command-line utility creates schema objects from files and loads them into the database. Schema objects can be created from Java source files, class files, and resource files. Resource files may be image files, resources, or anything else a procedure may need to access as data. You can pass files to `loadjava` individually, or as ZIP or JAR archive files.

Oracle Database Lite does not keep track of class dependencies. Make sure that you load into the database, or place in the `CLASSPATH`, all supporting classes and resource files required by a stored procedure. To query the classes that are loaded in the database, you can query the `okJavaObj` meta class.

Note: The table name and column names are case sensitive.

Syntax

```
loadjava {-user | -u} username/password[@database]
   [-option_name -option_name ...] filename filename ...
```

Arguments
This section discusses the `loadjava` arguments in detail.

User
The `user` argument specifies a username, password, and database directory in the following format:

```
<user>/<password>[@<database>]
```

For example:

```
scott/tiger@ ORACLE_HOME\Mobile\Sdk\OLDB40\Polite.odb
```

Options
Oracle Database Lite supports the following options that are listed and described in Table 13–1.

Table 13–1 Options

Option	Description
`-force	-f`	Forces files to be loaded, even if a schema object with the same name already exists in the database.
`-verbose	-v`	Directs `loadjava` to display detailed status messages while running.
`-meta	-m`	Creates the meta information in the database but does not load the classes. This is useful when the classes are in a .jar file and are not loaded into the database.

When specifying multiple options, you must separate the options with spaces. For example:

```
-force -verbose
```

The Oracle database supports additional options. If used with Oracle Database Lite, the additional options are recognized but not supported. Using them does not result in an error.

To view the options supported by Oracle database, see the `loadjava` help information using the following syntax.

```
loadjava {-help | -h}
```

Filenames
On the command line, you can specify as many class, source, JAR, ZIP, and resource files as you like, in any order. You must separate multiple file names with spaces, not commas. If passed a source file, `loadjava` invokes the Java compiler to compile the file before loading it into the database. If passed a JAR or ZIP file, `loadjava` processes each file in the JAR or ZIP. It does not create a schema object for the JAR or ZIP archive. The `loadjava` utility does not process a JAR or ZIP archive within another JAR or ZIP archive.

The best way to load files is to place them in a JAR or ZIP and then load the archive. Loading archives avoids the complications associated with resource schema object names. If you have a JAR or ZIP that works with the JDK, then you can be sure that loading it with `loadjava` also works, and you can avoid the complications associated with resource schema object naming.

As it loads files into the database, `loadjava` must create a name for the schema objects it creates for the files. The names of schema objects differ slightly from filenames, and different schema objects have different naming conventions. Class files are self-identifying, so `loadjava` can map their filenames to the names of schema objects automatically. Likewise, JAR and ZIP archives include the names of the files they contain.

However, resource files are not self-identifying; `loadjava` derives the names of Java resource schema objects from the literal names you enter on the command-line (or the literal names in a JAR or ZIP archive). Because classes use resource schema objects while executing, it is important that you specify the correct resource file names on the command line.

The best way to load individual resource files is to run `loadjava` from the top of the package tree, specifying resource file names relative to that directory. If you decide not to load resource files from the top of the package tree, you must be aware of how `loadjava` derives a name for your resource.

When you load a resource file, `loadjava` derives the name of the resource schema object from the file name that you enter on the command line. Suppose you type the following relative and absolute pathnames on the command line:

```
cd \scott\javastuff
loadjava options alpha\beta\x.properties
loadjava options  \scott\javastuff\alpha\beta\x.properties
```

Although you have specified the same file with a relative and an absolute pathname, `loadjava` creates *two* schema objects:

- `alpha\beta\x.properties`
- `\scott\javastuff\alpha\beta\x.properties`.

The `loadjava` utility generates the resource schema object's name from the file names *you enter*.

Classes can refer to resource files relatively (for example, `b.properties`) or absolutely (for example, `\a\b.properties`). To ensure that `loadjava` and the class loader use the same name for a schema object, pass `loadjava` *the name of the resource that the class passes to the* `java.lang.Object.getResource` *or* `java.lang.Class.`*`getResourceAsStream`* *method.*

Instead of remembering whether classes use relative or absolute resource names and changing directories so that you can enter the correct name on the command line, you can load resource files into a JAR file, as follows:

```
cd \scott\javastuff
jar -cf alpharesources.jar alpha\*.properties
loadjava options alpharesources.jar
```

Or, to simplify further, put both the class and resource files in a JAR, which makes the following invocations equivalent:

```
loadjava options alpha.jar
loadjava options \scott\javastuff\alpha.jar
```

Example

The following loads a class and resource file into Oracle Database Lite. It uses the `force` option; if the database already contains objects with the specified names, `loadjava` replaces them.

```
c:\> loadjava -u scott/tiger@c:\Olite\Mobile\Sdk\OLDB40\Polite.odb -f Agent.class\
images.dat
```

13.3.1.1.2 Using CREATE JAVA

To load Java classes manually, use the following syntax:

```
CREATE [OR REPLACE] [AND RESOLVE] [NOFORCE]
   JAVA {CLASS [SCHEMA <schema_name>] |
   RESOURCE NAMED [<schema_name>.]<primary_name>}
   [<invoker_rights_clause>]
   RESOLVER <resolver_spec>]
   USING BFILE ('<dir_path>', '<class_name>')
```

The following apply to the `CREATE JAVA` parameters:

- The `OR REPLACE` clause, if specified, recreates the function or procedure if one with the same name already exists in the database.
- For compatibility with the Oracle database, Oracle Database Lite recognizes but ignores the `<resolver_spec>` clause. Unlike the Oracle database, Oracle Database Lite does not resolve class dependencies. When loading classes manually, be sure to load all dependent classes.
- Oracle Database Lite recognizes, but ignores, `<invoker_rights_clause>`.

Example

The following demonstrates a `CREATE JAVA` statement. It loads a class named `Employee` into the database.

```
CREATE JAVA CLASS USING BFILE ('c:\myprojects\java',
   'Employee.class');
```

13.3.1.2 Defining Stored Procedures to SQL Using Create Function or Create Procedure

After loading the Java class into the Oracle Lite database using `loadjava` or `CREATE JAVA`, define any static method in the class that you want to call from SQL by creating a call specification for it. The call specification maps the Java method's name, parameter types, and return types to SQL counterparts.

> **Note:** Another method to define the Java stored procedure is to attach it, as described in Section 13.3.2, "Using Attach to Define the Java Stored Procedure". The attach method defines the entire stored procedure and you can store both class-level (static) methods and object-level (non-static) methods.

You do not need to define every stored procedure, only those that serve as entry points for your application. In a typical implementation, many stored procedures are called only by other stored procedures, not by SQL users.

To create a call spec, use the SQL commands `CREATE FUNCTION` for methods that return a value or `CREATE PROCEDURE` for methods that do not return a value. The `CREATE FUNCTION` and `CREATE PROCEDURE` statements have the following syntax:

```
CREATE [OR REPLACE]
   { PROCEDURE [<schema_name>.]<proc_name> [([<sql_parms>])] |
   FUNCTION [<schema_name>.]<func_name> [([<sql_parms>])]
   RETURN <sql_type> }
   <invoker_rights_clause>
   { IS | AS } LANGUAGE JAVA NAME
   '<java_fullname>  ([<java_parms>])
   [return <java_type_fullname>]';
   /
```

The following apply to this statement's keywords and parameters:

- `<sql_parms>` has the following format:

  ```
  <arg_name> [IN | OUT | IN OUT]
           <datatype>
  ```

- `<java_parms>` is the fully qualified name of the Java datatype.
- For compatibility with the Oracle database, Oracle Database Lite recognizes but ignores the `<invoker_rights_clause>` clause.
- `<java_fullname>` is the fully qualified name of a static Java method.
- `IS` and `AS` are synonymous.

For example, assume the following class has been loaded into the database:

```
import java.sql.*;
import java.io.*;

public class GenericDrop {
   public static void dropIt (Connection conn, String object_type,
                        String object_name) throws SQLException {
      // Build SQL statement
      String sql = "DROP " + object_type + " " + object_name;
      try {
         Statement stmt = conn.createStatement();
         stmt.executeUpdate(sql);
         stmt.close();
      } catch (SQLException e) {
         System.err.println(e.getMessage());}
   }  // dropIt
}  // GenericDrop
```

Class `GenericDrop` has one method named `dropIt`, which drops any kind of schema object. For example, if you pass the arguments "table" and "emp" to `dropIt`, the method drops the database table EMP from your schema.

The following call specification defines the method to SQL:

```
CREATE OR REPLACE PROCEDURE drop_it (
      obj_type VARCHAR2,
      obj_name VARCHAR2)
   AS LANGUAGE JAVA
   NAME 'GenericDrop.dropIt(java.sql.Connection,
      java.lang.String, java.lang.String)';
   /
```

Note: You must fully qualify the Java datatype parameters.

Given that you have a table named `TEMP` defined in your schema, you can execute the `drop_it` procedure from SQL Plus as follows.

```
Select drop_it('TABLE', 'TEMP') from dual;
```

You can also execute the `drop_it` procedure from within a ODBC application using an ODBC CALL statement. For more information, refer Section 13.3.3, "Calling Java Stored Procedures From a Multithreaded C or C++ Application".

13.3.1.3 Calling Defined Stored Procedures

After defining the stored procedure to SQL, call it with a SQL DML statement. For example, assume that this class is stored in the database:

```
public class Formatter {
   public static String formatEmp (String empName, String jobTitle) {
      empName = empName.substring(0,1).toUpperCase() +
         empName.substring(1).toLowerCase();
      jobTitle = jobTitle.trim().toLowerCase();
      if (jobTitle.equals("analyst"))
         return (new String(empName + " is an exempt analyst"));
      else
      return (new String(empName + " is a non-exempt " + jobTitle));
  }
}
```

Class `Formatter` has one method named `formatEmp`, which returns a formatted string containing an employee's name and job status. Create a call specification for `Formatter` as follows:

```
CREATE OR REPLACE FUNCTION format_emp (ename VARCHAR2, job VARCHAR2)
   RETURN VARCHAR2
   AS LANGUAGE JAVA
   NAME 'Formatter.formatEmp (java.lang.String, java.lang.String)
   return java.lang.String';
   /
```

The call specification defines the method `formatEmp` as `format_emp`. Invoke it as follows:

```
SELECT FORMAT_EMP(ENAME, JOB) AS "Employees" FROM EMP
   WHERE JOB NOT IN ('MANAGER', 'PRESIDENT') ORDER BY ENAME;
```

This statement produces the following output:

```
Employees
--------------------------------------------
Adams is a non-exempt clerk
Allen is a non-exempt salesman
Ford is an exempt analyst
James is a non-exempt clerk
Martin is a non-exempt salesman
Miller is a non-exempt clerk
Scott is an exempt analyst
Smith is a non-exempt clerk
Turner is a non-exempt salesman
Ward is a non-exempt salesman
```

Note: Oracle Database Lite does not support the Oracle database `SQL CALL` statement for invoking stored procedures.

For information on calling stored procedures from C and C++ applications, see Section 13.3.3, "Calling Java Stored Procedures From a Multithreaded C or C++ Application".

13.3.1.4 Dropping Defined Stored Procedures

Oracle Database Lite provides tools and SQL commands for dropping stored procedures. You should use caution when dropping procedures from the database, since Oracle Database Lite does not keep track of dependencies between classes. You must ensure that the stored procedure you drop is not referenced by other stored procedures. Dropping a class invalidates classes that depend on it directly or indirectly.

To remove Java stored procedure classes from Oracle Database Lite that were loaded, use either of the following:

- Section 13.3.1.4.1, "Using dropjava" for directions on how to use the `dropjava` utility
- Section 13.3.1.4.2, "Using SQL Commands" for directions on how to use the `SQL DROP JAVA` statement

To drop call specifications, use either `DROP FUNCTION` or `DROP PROCEDURE`.

13.3.1.4.1 Using dropjava The `dropjava` command-line utility automates the task of dropping Java classes from Oracle Database Lite and Oracle databases. `dropjava` converts file names into the names of schema objects and drops the schema objects. Use the following syntax to invoke `dropjava`:

```
dropjava {-user | -u} username/password[@database]
  [-option] filename filename ...
```

Arguments
This section describes the arguments to `dropjava`.

User
The `user` argument specifies a username, password, and absolute path to the database file in the following format:

```
<user>/<password>[@<database>]
```

For example:

```
scott/tiger@c:\Olite\Mobile\Sdk\OLDB40\Polite.odb
```

Option
By specifying the verbose option (`-verbose | -v`), you can direct `dropjava` to produce detailed status messages while running.

Oracle database supports additional options. If used with Oracle Database Lite, the additional options are recognized but not supported. Using them does not result in an error.

For a complete list of supported and recognized options, from the command prompt type:

```
dropjava -help
```

Filename
For the `filename` argument, you can specify any number of Java class, source, JAR, ZIP, and resource files, in any order. JAR and ZIP files must be uncompressed. `dropjava` interprets most file names the same way `loadjava` does:

- For class files, `dropjava` finds the class name in the file and drops the corresponding schema object.
- For source files, `dropjava` finds the first class name in the file and drops the corresponding schema object.
- For JAR and ZIP files, `dropjava` processes the archived file names as if they had been entered on the command line.

If a file name has an extension other than **.java**, **.class**, **.jar**, or **.zip**, or has no extension, then `dropjava` assumes that the file name is the name of a schema object, then drops all source, class, and resource schema objects with that name. If `dropjava` encounters a file name that does not match the name of any schema object, it displays an error message and then processes the remaining file names.

13.3.1.4.2 Using SQL Commands To drop a Java class from Oracle Database Lite manually, use the `DROP JAVA` statement, which has the following syntax:

```
DROP JAVA { CLASS | RESOURCE } [<schema-name> .]<object_name>
```

To drop a call specification, use the `DROP FUNCTION` or `DROP PROCEDURE` statement:

```
DROP { FUNCTION | PROCEDURE } [<schema-name>.]<object_name>
```

The schema name, if specified, is recognized but ignored by Oracle Database Lite.

13.3.1.5 Example Using the Load and Define Model

The following example creates a Java stored procedure using the load and define model.

In this example, you store the Java method `paySalary` in the Oracle Database Lite. `paySalary` computes the take-home salary for an employee.

This example covers the following steps.

- Step 1: Create the Java Class
- Step 2: Load the Java Class into the Database
- Step 3: Define the Function
- Step 4: Execute the Function

More examples of Java stored procedures are located in the `<`*`ORACLE_HOME`*`>\Mobile\SDK\samples\jdbc` directory.

Step 1: Create the Java Class

Create the Java class `Employee` in the file `Employee.java`. The `Employee` class implements the `paySalary` method:

```
import java.sql.*;
public class Employee {
   public static String paySalary(float sal, float fica, float sttax,
                            float ss_pct, float espp_pct)  {
      float deduct_pct;
      float net_sal;
        // compute take-home salary
      deduct_pct = fica + sttax + ss_pct + espp_pct;
      net_sal = sal * deduct_pct;
      String returnstmt = "Net salary is " + net_sal;
      return returnstmt;
   } // paySalary
}
```

Note: The keyword "public class" should not be used in a comment before the first public class statement.

Step 2: Load the Java Class into the Database

From mSQL, load the Java class using `CREATE JAVA`, as follows:

```
CREATE JAVA CLASS USING BFILE ('c:\myprojects\doc',
'Employee.class');
```

This command loads the Java class located in `c:\myprojects\doc` into the Oracle Database Lite.

Step 3: Define the Function

Create a call specification for the `paySalary` method. The following call specification defines the Java method `paySalary` as function `pay_salary`:

```
CREATE FUNCTION pay_salary (
sal float, fica float, sttax float, ss_pct float, espp_pct float)
RETURN VARCHAR2
AS LANGUAGE JAVA NAME
'Employee.paySalary(float, float, float, float, float)
return java.lang.String';
/
```

Step 4: Execute the Function

To execute `pay_salary` in mSQL:

```
SELECT pay_salary(6000.00, 0.2, 0.0565, 0.0606, 0.1)
FROM DUAL;
```

To execute `pay_salary` in ODBC:

```
SQLExecDirect(hstm,
   "SELECT pay_salary(6000.00,0.2,0.0565,0.0606,0.1)
   FROM DUAL);
```

Because the arguments to `pay_salary` are constants, the `FROM` clause specifies the dummy table `DUAL`. This `SELECT` statement produces the following output:

```
Net salary is 2502.6
```

13.3.2 Using Attach to Define the Java Stored Procedure

You can create Java stored procedures by attaching classes to a table and invoking methods in the class by name. Using this model, you can store both class-level (static) methods and object-level (non-static) methods.

For this model, follow these steps:

1. Develop a Java class with the methods you want to store. Make sure that the class compiles and executes without errors.
2. Attach the class to a table using the `SQL ALTER TABLE` command. Once the class is attached, then the methods in the class become table-level or row-level stored procedures of the table.
3. Invoke methods in the class directly from SQL. Identify the method with `table_name.method_name`.

This information is specific to Oracle Database Lite; you cannot attach classes to Oracle database tables as described here. The load and define model for developing stored procedures, described in Section 13.3.1, "Using Load and Define for Java Stored Procedures", only supports class (static) methods. By attaching classes to tables, however, you can store and call Java class and instance methods.

The following sections describe the details for attaching and using Java stored procedures:

- Section 13.3.2.1, "Attaching a Java Class to a Table"
- Section 13.3.2.2, "Table-Level Stored Procedures"
- Section 13.3.2.3, "Row-Level Stored Procedures"
- Section 13.3.2.4, "Calling Attached Stored Procedures"
- Section 13.3.2.5, "Dropping Attached Stored Procedures"
- Section 13.3.2.6, "Example of An Attached Java Stored Procedure"

13.3.2.1 Attaching a Java Class to a Table

To attach a Java class to a table, use the SQL command `ALTER TABLE`, which has the following syntax:

```
ALTER TABLE [schema.]table
   ATTACH JAVA {CLASS|SOURCE} "cls_or_src_name "
   IN {DATABASE|'cls_or_src_path '}
   [WITH CONSTRUCTOR ARGS (col_name_list )]
```

Note: You can attach either a source file or a class file. Source files are compiled by the Java compiler found in the system path.

Where:

- The `cls_or_src_name` variable specifies a fully qualified name of a class or source file. This includes the package name followed by class name, such as `Oracle.lite.Customer`. Do not include the file extension in the class or source file name. The name is case-sensitive. If you use lowercase letters, enclose the name in double quotes (" "). Make sure that the source or class is in the package specified by `cls_or_src_name`. For example, the source file of the example class `Customer` should contain the line "`package Oracle.lite;`". The class file is stored in the database in the same package. Oracle Database Lite creates the package if it does not already exist.
- If you have already attached the Java class to another table in the database, you can use the `IN DATABASE` clause. If the class has not yet been attached, specify the directory location of the class or source file in `cls_or_src_path`.
- Prior to executing a row-level stored procedure, Oracle Database Lite creates a Java object for the row, if one does not already exist. If the `ALTER TABLE` statement includes a `WITH CONSTRUCTOR` clause, then Oracle Database Lite creates the object using the class constructor that is the best match given the datatypes of the columns included in `col_name_list`. If the `ALTER TABLE` statement does not include a `WITH CONSTRUCTOR` clause, then Oracle Database Lite uses the default constructor.

You can use the ODBC functions `SQLProcedures` and `SQLProcedureColumns` to retrieve information about methods defined in a table.

13.3.2.2 Table-Level Stored Procedures

Table-level stored procedures are the static methods of the attached Java class. Therefore, when executing the method, Oracle Database Lite does not instantiate the class to which it belongs. In a call statement, you refer to table-level stored procedures as *table_name.method_name*.

Statement-level triggers and BEFORE INSERT and AFTER DELETE row-level triggers (see "Section 13.4.1, "Statement-Level vs. Row-Level Triggers") must be table-level stored procedures.

13.3.2.3 Row-Level Stored Procedures

Row-level stored procedures are the non-static methods in the attached Java class. To execute a row-level stored procedure, Oracle Database Lite instantiates the class to which the procedure belongs. The arguments to the class constructor determine which column values the constructor uses as parameters to create the class instances. In a call statement, you refer to row-level stored procedures as method_name (without the table qualifier). Row-level triggers can indirectly execute row-level stored procedures.

13.3.2.4 Calling Attached Stored Procedures

After attaching the class to a table using the ALTER TABLE statement, you can call it with a SELECT statement. Refer to table-level stored procedures as *table_name.method_name* and row-level procedures as *method_name*.

For example, to execute a table-level stored procedure:

```
SELECT table_name.proc_name[arg_list]
   FROM {DUAL|[schema.]table WHERE condition};
```

The proc_name is the name of the table-level stored procedure. Each argument in arg_list is either a constant or a reference to a column in the table. If all the arguments of arg_list are constants, the FROM clause should reference the dummy table DUAL.

Execute a row-level stored procedure as follows:

```
SELECT [schema.]proc_name[arg_list]
   FROM [schema.]table
   WHERE condition;
```

If you call a procedure in the form *table_name.method_name*, and a table or method with that name does not exist, Oracle Database Lite assumes that *table_name* refers to a schema name and *method_name* refers to a procedure name. If you reference *method_name* only, Oracle Database Lite assumes that the referenced method is a row-level procedure. If there is no such procedure defined, however, Oracle Database Lite assumes that *method_name* refers to a procedure in the current schema.

Note: Oracle Database Lite does not support the Oracle8*i* SQL CALL statement for invoking stored procedures.

You can use a callable statement to execute a procedure from ODBC or JDBC applications. For more information, see Chapter 10, "JDBC Programming" or Section 13.3.3, "Calling Java Stored Procedures From a Multithreaded C or C++ Application".

13.3.2.5 Dropping Attached Stored Procedures

Oracle Database Lite provides tools and SQL commands for dropping stored procedures. You should use caution when dropping procedures from the database, since Oracle Database Lite does not keep track of dependencies between classes. You must ensure that the stored procedure you drop is not referenced by other stored procedures. Dropping a class invalidates classes that depend on it directly or indirectly.

You use the `ALTER TABLE` command to drop stored procedures, which has the following syntax:

```
ALTER TABLE [schema.]table
   DETACH [AND DELETE] JAVA CLASS "class_name"
```

Note: You must enclose the class name in double quotes (`"  "`) if it contains lowercase letters.

Detaching the Java class does not delete it from the database. To delete the Java class file from the database, use the `DETACH AND DELETE` statement.

If you delete a Java class from the database after invoking it as a stored procedure or trigger, the class remains in the Java Virtual Machine attached to the application. To unload the class from the Java Virtual Machine, commit changes to the database, if necessary, and close all applications connected to the database. To replace a Java class, you must close all connections to the database and reload the class.

13.3.2.6 Example of An Attached Java Stored Procedure

The following example shows how to create a Java stored procedure in Oracle Database Lite. In this example, you attach the Java method `paySalary` to the table EMP. `paySalary` computes the take-home salary for an employee.

This example covers the following steps:

- Step 1: Create the Table
- Step 2: Create the Java Class
- Step 3: Attach the Java Class to the Table
- Step 4: Execute the Method

Step 1: Create the Table

Create the table using the following SQL command:

```
CREATE TABLE EMP(Col1 char(10));
```

Step 2: Create the Java Class

Create the Java class `Employee` in the file `Employee.java`. The `Employee` class implements the `paySalary` method:

```
import java.sql.*;
public class Employee {
   public static String paySalary(float sal, float fica, float sttax,
                         float ss_pct, float espp_pct)  {
      float deduct_pct;
      float net_sal;
        // compute take-home salary
      deduct_pct = fica + sttax + ss_pct + espp_pct;
      net_sal = sal * deduct_pct;
      String returnstmt = "Net salary is " + net_sal;
      return returnstmt;
   } // paySalary
}
```

Step 3: Attach the Java Class to the Table

From mSQL, attach the Java class using the `ALTER TABLE` command:

```
ALTER TABLE EMP ATTACH JAVA SOURCE "Employee" IN 'C:\tmp';
```

This command attaches the Java source file for the `Employee` class, which resides in the directory `C:\tmp`, to the EMP table.

Step 4: Execute the Method

To execute the `paySalary` method in mSQL, type the following statement:

```
SELECT EMP."paySalary"(6000.00,0.2,0.0565,0.0606,0.1)
   FROM DUAL;
```

To execute `paySalary` from ODBC, invoke `SQLExecDirect`:

```
SQLExecDirect(hstm,
   "SELECT EMP.\"paySalary\"(6000.00,0.2,0.0565,0.0606,0.1)
   FROM DUAL);
```

This statement produces the following result:

```
Net salary is 2502.6
```

13.3.3 Calling Java Stored Procedures From a Multithreaded C or C++ Application

When invoking a Java stored procedure from a multithreaded C or C++ application, you should load `jvm.dll` from the application's `main` function. This resolves a problem that occurs with the Java Virtual Machine's garbage collection when a C or C++ application creates multiple threads that invoke a stored procedure directly or indirectly. The Java Virtual Machine runs out of memory because the threads do not detach from the Java Virtual Machine before exiting. Since Oracle Database Lite cannot determine whether the Java Virtual Machine or the user application created the thread, it does not attempt to detach them.

The `main` function should load the library before taking any other action, as follows:

```
int main (int argc, char** argv)
{
   LoadLibrary("jvm.dll");
   ...
}
```

The library loads the Java Virtual Machine into the application's main thread. It attempts to detach any thread from the Java Virtual Machine if the thread detaches from the process. The `jvm.dll` behaves correctly even if the thread is not attached to a Java Virtual Machine.

13.4 Using Triggers With Java Stored Procedures

Triggers are stored procedures that execute, or "fire", when a specific event occurs. A trigger can fire when a column is updated, or when a row is added or deleted. The trigger can fire before or after the event.

Triggers are commonly used to enforce a database's business rules. For example, a trigger can verify input values and reject an illegal insert. Similarly, a trigger can ensure that all tables depending on a particular row are brought to a consistent state before the row is deleted.

- Section 13.4.1, "Statement-Level vs. Row-Level Triggers"
- Section 13.4.2, "Creating Triggers"

- Section 13.4.3, "Dropping Triggers"
- Section 13.4.4, "Trigger Example Using the Attach Method"
- Section 13.4.5, "Trigger Arguments"
- Section 13.4.6, "Trigger Arguments Example Using Create Procedure"

13.4.1 Statement-Level vs. Row-Level Triggers

There are two types of triggers: row-level and statement-level. A row-level trigger is fired once for each row affected by the change to the database. A statement-level trigger fires only once, even if multiple rows are affected by the change.

The `BEFORE INSERT` and `AFTER DELETE` triggers can only fire table-level stored procedures, since a row object cannot be instantiated to call the procedures. The `AFTER INSERT`, `BEFORE DELETE`, and `UPDATE` triggers may fire table-level or row-level stored procedures.

13.4.2 Creating Triggers

Use the `CREATE TRIGGER` statement to create a trigger. The `CREATE TRIGGER` statement has the following syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name {BEFORE | AFTER} [{INSERT | DELETE |
   UPDATE [OF column_list]} [OR ]] ON table_reference
   [FOR EACH ROW] procedure_ref
   (arg_list)
```

In the `CREATE TRIGGER` syntax:

- Use the `OR` clause to specify multiple triggering events.
- Use `FOR EACH ROW` to create a row-level trigger. For a table-level trigger, do not include this clause.
- Use `procedure_ref` to identify the stored procedure to execute.

You can create multiple triggers of the same kind for a table if each trigger has a unique name within a schema.

In the following example, assume that you have stored and defined a procedure as `PROCESS_NEW_HIRE`. The trigger `AIEMP` fires every time a row is inserted into the EMP table.

```
CREATE TRIGGER AIEMP AFTER INSERT ON EMP FOR EACH ROW
   PROCESS_NEW_HIRE(ENO);
```

`UPDATE` triggers that use the same stored procedure for different columns of a table are fired only once when a subset of the columns is modified within a statement. For example, the following statement creates a `BEFORE UPDATE` trigger on table T, which has columns C1, C2, and C3:

```
CREATE TRIGGER T_TRIGGER BEFORE UPDATE OF C1,C2,C3 ON T
   FOR EACH ROW trigg(old.C1,new.C1,old.C2,new.C2,
   old.C3,new.C3);
```

This update statement fires `T_TRIGGER` only once:

```
UPDATE T SET C1 = 10, C2 = 10 WHERE ...
```

13.4.2.1 Enabling and Disabling Triggers

When you create a trigger, it is automatically enabled. To disable triggers, use the `ALTER TABLE` or `ALTER TRIGGER` statement.

To enable or disable individual triggers, use the `ALTER TRIGGER` statement, which has the following syntax:

```
ALTER TRIGGER <trigger_name> {ENABLE | DISABLE}
```

To enable or disable all triggers attached to a table, use ALTER TABLE:

```
ALTER TABLE <table_name> {ENABLE | DISABLE} ALL TRIGGERS
```

13.4.3 Dropping Triggers

To drop a trigger, use the `DROP TRIGGER` statement, which has the following syntax:

```
DROP TRIGGER [schema.]trigger
```

13.4.4 Trigger Example Using the Attach Method

This example creates a trigger that is defined using the attach method, as described in Section 13.3.2, "Using Attach to Define the Java Stored Procedure". For an example of creating triggers using the `SQL CREATE FUNCTION` or `CREATE METHOD`, see Section 13.4.6, "Trigger Arguments Example Using Create Procedure". In the example, you first create a table and a Java class. Then you attach the class to the table. And finally, you create and fire the trigger.

The `SalaryTrigger` class contains the `check_sal_raise` method. The method prints a message if an employee gets a salary raise of more than ten percent. The trigger fires the method before updating a salary in the EMP table.

Since `check_sal_raise` writes a message to standard output, use mSQL to issue the mSQL commands in the example. To start mSQL, invoke the Command Prompt and enter the following.

```
msql username/password@connect_string
```

`connect_string` is JDBC URL syntax. For example, to connect to the default database as user SYSTEM, at the Command Prompt.

```
msql system/passwd@jdbc:polite:polite
```

At the mSQL command line, create and populate the EMP table as follows.

```
CREATE TABLE EMP(E# int, name char(10), salary real,
   Constraint E#_PK primary key (E#));

INSERT INTO EMP VALUES (123,'Smith',60000);
INSERT INTO EMP VALUES (234,'Jones',50000);
```

Place the following class in **SalaryTrigger.java**:

```
class SalaryTrigger {
   private int eno;
   public SalaryTrigger(int enum) {
      eno = enum;
   }
   public void check_sal_raise(float old_sal,
      float new_sal)
   {
      if (((new_sal - old_sal)/old_sal) > .10)
```

```
        {
           // raise too high  do something here
           System.out.println("Raise too high for employee " + eno);
        }
     }
}
```

The `SalaryTrigger` class constructor takes an integer, which it assigns to attribute `eno` (the employee number). An instance of `SalaryTrigger` is created for each row (that is, for each employee) in the table `EMP`.

The `check_sal_raise` method is a non-static method. To execute, it must be called by an object of its class. Whenever the salary column of a row in `EMP` is modified, an instance of `SalaryTrigger` corresponding to that row is created (if it does not already exist) with the employee number (*E#)* as the argument to the constructor. The trigger then calls the `check_sal_raise` method.

After creating the Java class, you attach it to the table, as follows:

```
ALTER TABLE EMP ATTACH JAVA SOURCE "SalaryTrigger" IN '.'
   WITH CONSTRUCTOR ARGS(E#);
```

This statement directs Oracle Database Lite to compile the Java source file `SalaryTrigger.java` found in the current directory, and attach the resulting class to the EMP table. The statement also specifies that, when instantiating the class, Oracle Database Lite should use the constructor that takes as an argument the value in the `E#` column.

After attaching the class to the table, create the trigger as follows:

```
CREATE TRIGGER CHECK_RAISE BEFORE UPDATE OF SALARY ON EMP FOR EACH ROW
   "check_sal_raise"(old.salary, new.salary);
/
```

This statement creates a trigger called `check_raise`, which fires the `check_sal_raise` method before any update to the salary column of any row in EMP. Oracle Database Lite passes the old value and the new value of the salary column as arguments to the method.

In the example, a row-level trigger fires a row-level procedure (a non-static method). A row-level trigger can also fire table-level procedures (static methods). However, because statement-level triggers are fired once for an entire statement and a statement may affect multiple rows, a statement-level trigger can only fire a table-level procedure.

The following command updates the salary and fires the trigger:

```
UPDATE EMP SET SALARY = SALARY + 6100  WHERE E# = 123;
```

This produces the following output:

```
Raise too high for employee 123
```

13.4.5 Trigger Arguments

If using attached stored procedures, as described in Section 13.3.2, "Using Attach to Define the Java Stored Procedure", row-level triggers do not support Java-to-SQL type conversion. Therefore, the Java datatype of a trigger argument must match the corresponding SQL datatype (shown in section Section 13.6, "Converting Datatypes Between Java and SQL For Stored Procedures") of the trigger column. However, if you are using the load and define model, Oracle Database Lite supports datatype casting.

Table 13–2 describes how trigger arguments work in each type of column.

Table 13–2 Trigger Arguments

Trigger Argument	New Column Access	Old Column Access
`insert`	Yes	No
`delete`	No	Yes
`update`	Yes	Yes

Note: Triggers that have a `java.sql.Connection` object as an argument may be used only with applications that use the relational model.

13.4.6 Trigger Arguments Example Using Create Procedure

The following example shows how to create triggers that use IN/OUT parameters.

1. First, create the Java class `EMPTrigg`.

```
import java.sql.*;

public class EMPTrigg {
   public static final String goodGuy = "Oleg";

   public static void NameUpdate(String oldName, String[] newName)
   {
      if (oldName.equals(goodGuy))
         newName[0] = oldName;
   }

   public static void SalaryUpdate(String name, int oldSalary,
          int newSalary[])
   {
      if (name.equals(goodGuy))
         newSalary[0] = Math.max(oldSalary, newSalary[0])*10;
   }

   public static void AfterDelete(Connection conn,
                  String name, int salary) {
      if (name.equals(goodGuy))
         try {
            Statement stmt = conn.createStatement();
            stmt.executeUpdate(
             "insert into employee values('" + name + "', " +
                     salary + ")");
            stmt.close();
         } catch(SQLException e) {}
     }
 }
```

2. Create a new table EMPLOYEE and populate it with values.

```
CREATE TABLE EMPLOYEE(NAME VARCHAR(32), SALARY INT);
INSERT INTO EMPLOYEE VALUES('Alice', 100);
INSERT INTO EMPLOYEE VALUES('Bob', 100);
INSERT INTO EMPLOYEE VALUES('Oleg', 100);
```

3. Next, load the class into Oracle Database Lite.

```
CREATE JAVA CLASS USING BFILE ('c:\myprojects', 'EMPTrigg.class');
```

4. Use the `CREATE PROCEDURE` statement to define the `EMPTrigg` methods that you want to call:

```
CREATE PROCEDURE NAME_UPDATE(
   OLD_NAME IN VARCHAR2, NEW_NAME IN OUT VARCHAR2)
   AS LANGUAGE JAVA NAME
   'EMPTrigger.NameUpdate(java.lang.String, java.lang.String[])';
   /

CREATE PROCEDURE SALARY_UPDATE(
   ENAME VARCHAR2, OLD_SALARY INT, NEW_SALARY IN OUT INT)
   AS LANGUAGE JAVA NAME
   'EMPTrigger.SalaryUpdate(java.lang.String, int, int[])';
   /

CREATE PROCEDURE AFTER_DELETE(
   ENAME VARCHAR2, SALARY INT)
   AS LANGUAGE JAVA NAME
   'EMPTrigger.AfterDelete(java.sql.Connection,
           java.lang.String, int)';
   /
```

5. Now, create a trigger for each procedure:

```
CREATE TRIGGER NU BEFORE UPDATE OF NAME ON EMPLOYEE FOR EACH ROW
   NAME_UPDATE(old.name, new.name);

CREATE TRIGGER SU BEFORE UPDATE OF SALARY ON EMPLOYEE FOR EACH ROW
   SALARY_UPDATE(name, old.salary, new.salary);

CREATE TRIGGER AD AFTER DELETE ON EMPLOYEE FOR EACH ROW
   AFTER_DELETE(name, salary);
```

6. Enter the following commands to fire the triggers and view the results:

```
SELECT * FROM EMPLOYEE;
UPDATE EMPLOYEE SET SALARY=0 WHERE NAME = 'Oleg';
SELECT * FROM EMPLOYEE;

DELETE FROM EMPLOYEE WHERE NAME = 'Oleg';
SELECT * FROM EMPLOYEE;

UPDATE EMPLOYEE SET NAME='TEMP' WHERE NAME = 'Oleg';
DELETE FROM EMPLOYEE WHERE NAME = 'TEMP';

SELECT * FROM EMPLOYEE;
```

13.5 Tutorial for a Java Stored Procedure Invoked By a Trigger

In this tutorial, you create a Java class `EMAIL`, load the class into Oracle Database Lite, define its method to SQL, and create a trigger for the method. The `EMAIL` class appears in the source file `EMAIL.java`, and is available in the Java examples directory at the following location.

<*ORACLE_HOME*>`\Mobile\Sdk\Samples\JDBC`

`EMAIL` has a method named `assignEMailAddress`, which generates an email address for an employee based on the first letter of the employee's first name and up to seven letters of the last name. If the address is already assigned, the method attempts to find a unique email address using combinations of letters in the first and last name.

After creating the class, you load it into Oracle Database Lite using mSQL. For this example you use the SQL statement `CREATE JAVA`. Alternatively, you can use the `loadjava` utility to load the class into Oracle Database Lite. After loading the class, you define the `assignEMailAddress` method to SQL.

Finally, you create a trigger that fires the `assignEMailAddress` method whenever a row is inserted into `T_EMP`, the table that contains the employee information.

As arguments, `assignEMailAddress` takes a JDBC connection object, the employee's identification number, first name, middle initial, and last name. Oracle Database Lite supplies the JDBC connection object argument. You do not need to provide a value for the connection object when you execute the method. `assignEMailAddress` uses the JDBC connection object to ensure that the generated e-mail address is unique.

- Section 13.5.1, "Start mSQL"
- Section 13.5.2, "Create a Table"
- Section 13.5.3, "Create a Java Class"
- Section 13.5.4, "Load the Java Class File"
- Section 13.5.5, "Define the Stored Procedure"
- Section 13.5.6, "Populate the Database"
- Section 13.5.7, "Execute the Procedure"
- Section 13.5.8, "Verify the Email Address"
- Section 13.5.9, "Create a Trigger"
- Section 13.5.10, "Commit or Roll Back"

13.5.1 Start mSQL

Start mSQL and connect to the default Oracle Database Lite. Since the Java application in this tutorial prints to standard output, use the DOS version of mSQL. From a DOS prompt, type:

```
msql system/mgr@jdbc:polite:polite
```

The SQL prompt should appear.

13.5.2 Create a Table

To create a table, type:

```
CREATE TABLE T_EMP(ENO INT PRIMARY KEY,
   FNAME VARCHAR(20),
   MI CHAR,
   LNAME VARCHAR(20),
   EMAIL VARCHAR(8));
```

13.5.3 Create a Java Class

Create and compile the Java class `EMAIL` in the file `EMAIL.java` in `C:\tmp`. `EMAIL.java` implements the `assignEMailAddress` method. The code sample given below lists the contents of this file. You can copy this file from the following location.

<*ORACLE_HOME*>\Mobile\Sdk\Samples\JDBC

```
import java.sql.*;

public class EMAIL {
   public static void assignEMailAddress(Connection conn,
            int eno, String fname,String lname)
            throws Exception
   {
      Statement stmt = null;
      ResultSet retset = null;
      String emailAddr;
      int i,j,fnLen, lnLen, rowCount;

      /* create a statement */
      try {
         stmt = conn.createStatement();
      }
      catch (SQLException e)
      {
         System.out.println("conn.createStatement failed: " +
         e.getMessage() + "\n");
         System.exit(0);
      }
      /* check fname and lname */
      fnLen = fname.length();
      if(fnLen > 8) fnLen = 8;
      if (fnLen == 0)
         throw new Exception("First name is required");
      lnLen = lname.length();
      if(lnLen > 8) lnLen = 8;
      if (lnLen == 0)
         throw new Exception("Last name is required");
      for (i=1; i <= fnLen; i++)
      {
         /* generate an e-mail address */
         j = (8-i) > lnLen? lnLen:8-i;
         emailAddr =
               new String(fname.substring(0,i).toLowerCase()+
               lname.substring(0,j).toLowerCase());
         /* check if this e-mail address is unique  */
         try {
            retset = stmt.executeQuery(
                   "SELECT * FROM T_EMP  WHERE email = '"+
                   emailAddr+"'");
            if(!retset.next()) {
               /* e-mail address is unique;
               * so update the email column */
               retset.close();
               rowCount = stmt.executeUpdate(
                   "UPDATE T_EMP SET EMAIL = '"
                   + emailAddr + "' WHERE ENO = "
                   + eno);
               if(rowCount == 0)
                  throw new Exception("Employee "+fname+ " " +
                          lname + " does not exist");
               else return;
            }
         }
         catch (SQLException e) {
            while(e != null) {
               System.out.println(e.getMessage());
               e = e.getNextException();
            }
         }
      }
      /* Can't find a unique name */
      emailAddr = new String(fname.substring(0,1).toLowerCase() +
           lname.substring(0,1).toLowerCase() + eno);
      rowCount = stmt.executeUpdate(
           "UPDATE T_EMP SET EMAIL = '"
           + emailAddr + "' WHERE ENO = "
           + eno);
      if(rowCount == 0)
         throw new Exception("Employee "+fname+ " " +
              lname + " does not exist");
      else return;
   }
}
```

13.5.4 Load the Java Class File

To load the `EMAIL` class file into Oracle Database Lite, type:

```
CREATE JAVA CLASS USING BFILE
   ('c:\tmp', 'EMAIL.class');
```

If you want to make changes to the class after loading it, you need to:

1. Drop the class from the database, using `dropjava` or `DROP JAVA CLASS`
2. Commit your work
3. Exit mSQL
4. Restart mSQL

This unloads the class from the Java Virtual Machine.

13.5.5 Define the Stored Procedure

You make the stored procedure callable from SQL by creating a call specification (call spec) for it. Since `assignEMailAddress` does not return a value, use the CREATE PROCEDURE command, as follows:

```
CREATE OR REPLACE PROCEDURE
   ASSIGN_EMAIL(E_NO INT, F_NAME VARCHAR2, L_NAME VARCHAR2)
   AS LANGUAGE JAVA NAME 'EMAIL.assignEMailAddress(java.sql.Connection,
int, java.lang.String,
     java.lang.String)';
```

13.5.6 Populate the Database

Insert a row into `T_EMP`:

```
INSERT INTO T_EMP VALUES(100,'John','E','Smith',null);
```

13.5.7 Execute the Procedure

To execute the procedure, type:

```
SELECT ASSIGN_EMAIL(100,'John','Smith')
  FROM dual
```

13.5.8 Verify the Email Address

To see the results of the ASSIGN_EMAIL procedure, type:

```
SELECT * FROM T_EMP;
```

This command produces the following output:

```
     ENO  FNAME                 M LNAME                EMAIL
     ---- ------------------   - -------------------- --------
     100  John                  E Smith                jsmith
```

13.5.9 Create a Trigger

To make ASSIGN_EMAIL execute whenever a row is inserted into T_EMP, create an AFTER INSERT trigger for it. Create the trigger as follows:

```
CREATE TRIGGER EMP_TRIGG AFTER INSERT ON T_EMP FOR EACH ROW
  ASSIGN_EMAIL(eno,fname,lname);
```

A trigger named EMP_TRIGG fires every time a row is inserted into T_EMP. The actual arguments for the procedure are the values of the columns eno, fname, and lname.

You do not need to specify a connection argument.

13.5.9.1 Testing the Trigger

Test the trigger by inserting a row into T_EMP:

```
INSERT INTO T_EMP VALUES(200,'James','A','Smith',null);
```

13.5.9.2 Verify the Email Address

Issue a SELECT statement to verify that the trigger has fired:

```
SELECT * FROM T_EMP;
   ENO FNAME                M LNAME                EMAIL
   --- -------------------- - -------------------- --------
   100 John                 E Smith                jsmith
   200 James                A Smith                jasmith
```

13.5.10 Commit or Roll Back

Finally, commit your changes to preserve your work, or roll back to cancel changes.

13.6 Converting Datatypes Between Java and SQL For Stored Procedures

Oracle Database Lite performs type conversion between Java and SQL datatypes according to standard SQL rules. For example, if you pass an integer to a stored procedure that takes a string, Oracle Database Lite converts the integer to a string. For information about row-level triggers arguments, see Section 13.4.5, "Trigger Arguments". For a complete list of Java to SQL datatype mappings, see Section 10.5.1, "Mapping Datatypes Between Java and Oracle".

Note: In Oracle database, `DATE` columns are created as `TIMESTAMP`. Also, note that `TIMESTAMP WITH TIME ZONE` data type is not supported.

You must specify trigger methods accordingly.

Java does not allow a method to change the value of its arguments outside the scope of the method. However, Oracle Database Lite supports `IN`, `OUT`, and `IN/OUT` parameters.

Many Java datatypes are immutable or do not support `NULL` values. To pass `NULL` values and use `IN/OUT` parameters for those datatypes, a stored procedure can use an array of that type or use the equivalent object type. Table 13–3 shows the Java integer datatypes you can use to enable an integer to be an `IN/OUT` parameter or carry a `NULL` value.

Table 13–3 The Java Integer Datatypes

Java Argument	Can Be IN/OUT	Can Be NULL
`int`	No	No
`int[]`	Yes	Yes
`Integer`	No	Yes
`Integer[]`	Yes	Yes
`int[][]`	Yes	Yes

You can use mutable Java datatypes, such as `Date`, to pass a `NULL` or an `IN/OUT` parameter. However, use a `Date` array if a stored procedure needs to change the `NULL` status of its argument.

Note: Passing a `NULL` when the corresponding Java argument cannot be `NULL` causes an error.

13.6.1 Declaring Parameters for Java Stored Procedures

The return value of a Java method is the OUT parameter of the procedure. A primitive type or immutable reference type can be an IN parameter. A mutable reference type or array type can be an IN/OUT parameter. Table 13–4 shows the Java type to use to make the corresponding Oracle Database Lite parameter an IN/OUT parameter.

Table 13–4 Java Types for Oracle Database Lite IN/OUT Parameters

For IN/OUT parameters of type...	Use...
Number	`Integer[]` or `int[]`
Binary	`byte[]` or `byte[][]`
String	`string[]`

If the stored procedure takes a `java.sql.Connection`, Oracle Database Lite automatically supplies the argument using the value of the current transaction or row. This argument is the first argument passed to the procedure.

13.6.2 Using Stored Procedures to Return Multiple Rows

You can use stored procedures to return multiple rows. You can invoke stored procedures that return multiple rows only from JDBC or ODBC applications, however. For a stored procedure to return multiple rows, its corresponding Java method must return a `java.sql.ResultSet` object. By executing a SELECT statement, the Java method obtains a `ResultSet` object to return. The column names of the `ResultSet` are specified in the SELECT statement. If you need to address the result columns by different names than those used in the table, the SELECT statement should use aliases for the result columns. For example:

```
SELECT emp.name Name,  dept.Name Dept
   FROM  emp, dept
   WHERE emp.dept# = dept.dept#;
```

Because the return type of a stored procedure that returns multiple rows must be `java.sql.ResultSet`, the signature of that stored procedure cannot be used to obtain the column names or types of the result. Consequently, you should design additional tables to track the column names or result types for the stored procedures. For example, if you embed the preceding SELECT statement in a Java method, the method return type should be `java.sql.ResultType`, not `char Name` and `char Dept`.

Note: You can only create Java stored procedures that return multiple rows using the attached stored procedure development model, described in Section 13.3.2, "Using Attach to Define the Java Stored Procedure".

13.6.2.1 Returning Multiple Rows in ODBC

To execute a stored procedure that returns multiple rows in an OBDC application, use the following CALL statement, in which P is the name of the stored procedure and a_1 through a_n are arguments to the stored procedure.

```
{CALL P(a1,...,an)}
```

You use a marker (`?`) for any argument that should be bound to a value before the statement executes. When the statement executes, the procedure runs and the cursor on the result set is stored in the statement handle. Subsequent fetches using this statement handle return the rows from the procedure.

After you execute the `CALL` statement, use `SQLNumResultCols` to find the number of columns in each row of the result. Use the `SQLDescribeCol` function to return the column name and datatype.

13.6.2.2 Example

The following example shows how to use ODBC to execute a stored procedure that returns multiple rows. This example does not use the `SQLNumResultCols` or `SQLDescribeCol` functions. It assumes that you have created a stored procedure, which you have defined to SQL as PROC. PROC takes an integer as an argument.

```
rc = SQLPrepare(StmtHdl, "{call PROC(?)}", SQL_NTS);
CHECK_STMT_ERR(StmtHdl, rc, "SQLPrepare");

rc = SQLBindParameter(StmtHdl, 1, SQL_PARAM_INPUT_OUTPUT,
   SQL_C_LONG,SQL_INTEGER, 0, 0, &InOutNum, 0, NULL);
CHECK_STMT_ERR(StmtHdl, rc, "SQLBindParameter");
```

```
rc = SQLExecute(StmtHdl);
CHECK_STMT_ERR(StmtHdl, rc, "SQLExecute");

/* you can use SQLNumResultCols and SQLDescribeCol here */

rc = SQLBindCol(StmtHdl, 1, SQL_C_CHAR, c1, 20, &pcbValue1);
CHECK_STMT_ERR(StmtHdl, rc, "SQLBindCol");

rc = SQLBindCol(StmtHdl, 2, SQL_C_CHAR, c2, 20, &pcbValue2);
CHECK_STMT_ERR(StmtHdl, rc, "SQLBindCol");

while ((rc = SQLFetch(StmtHdl)) != SQL_NO_DATA_FOUND) {
   CHECK_STMT_ERR(StmtHdl, rc, "SQLFetch");
   printf("%s, %s\n", c1, c2);
}
```

13.7 Executing Java Stored Procedures from JDBC

After creating a Java stored procedures, you can execute the procedure from a JDBC application by performing one of the following:

- Pass a `SQL SELECT` string, which executes the stored procedure, to the `Statement.executeQuery` method.
- Use a JDBC `CallableStatement`.

The `executeQuery` method executes table-level and row-level stored procedures. `CallableStatement` currently only supports execution of table-level stored procedures.

13.7.1 Using the executeQuery Method

To call a stored procedure using the `executeQuery` method, perform the following:

1. Create a `Statement` object and assign the value returned by the `createStatement` method with the current connection object.
2. Execute the `Statement.executeQuery` method, passing the `SQL SELECT` string that invokes the Java stored procedure.

The following example executes a row-level procedure `SHIP` on a table named `INVENTORY` with the argument value stored in the variable *q*. The variable `p` contains the product ID for the product (row) for which you want to execute the stored procedure.

```
int res = 0;
Statement s = conn.createStatement();
ResultSet r = s.executeQuery("SELECT SHIP(" + q + ")" +
   "FROM INVENTORY WHERE PID = " + p);
if(r.next()) res = r.getInt(1);
r.close();
s.close();
return res;
```

If you need to execute a procedure repeatedly with varying parameters, use `PreparedStatement` instead of `Statement`. Because the SQL statements in a `PreparedStatement` are pre-compiled, a `PreparedStatement` executes more efficiently. Additionally, a `PreparedStatement` can accept `IN` parameters, represented in the statement with a question mark `(?)`. However, if the `PreparedStatement` takes a `long` type parameter, such as `LONG` or `LONG RAW`, you must bind the parameter using the `setAsciiStream`, `setUnicodeStream`, or `setBinaryStream` methods.

In the preceding example, if the `SHIP` procedure updates the database and the isolation of the transaction that issues the above query is `READ COMMITTED`, then you must append the `FOR UPDATE` clause to the `SELECT` statement, as follows:

```
"SELECT SHIP(" + q + ")" +
   FROM INVENTORY WHERE PID = " +
   p + "FOR UPDATE");
```

13.7.2 Using a Callable Statement

To execute the stored procedure using a callable statement, create a `CallableStatement` object and register its parameters, as follows:

```
CallableStatement cstmt = conn.prepareCall(
   "{?=call tablename.methodname() }");
cstmt.registerOutParameter(1, ...);
cstmt.executeUpdate();
cstmt.get..(1);
cstmt.close();
```

The following restrictions apply to JDBC callable statements:

- JDBC callable statements can only execute table-level stored procedures.
- Both IN and OUT parameters are supported. However, not all Java datatypes can be used as OUT parameters. For more information, see Section 13.6, "Converting Datatypes Between Java and SQL For Stored Procedures".
- Procedure names correspond to the Java method names, and are case-sensitive.
- As with prepared statements, if the callable statement has a "`long`" type, such as: `LONG`, `LONG VARBINARY`, `LONG VARCHAR`, `LONG VARCHAR2`, or `LONG RAW`, you must bind the parameter using the `setAsciiStream`, `setUnicodeStream`, or `setBinaryStream` methods.

Note: When no longer needed, you should reclaim system resources by closing JDBC objects, such as `Resultset` and `Statement` objects.

13.8 Using C++ Stored Procedures

A C++ stored procedure is a C++ procedure or function that exists in a DLL outside of Oracle Database Lite. The procedure can be invoked by applications that access the database. C++ stored procedures can return a single value, a row, or multiple rows.

The following sections describe how to create, build, and define a C++ stored procedure:

- Section 13.8.1, "Creating C++ Stored Procedures"
- Section 13.8.2, "Building Your C++ Stored Procedures"
- Section 13.8.3, "Define Your C++ Stored Procedure"
- Section 13.8.4, "C++ Stored Procedure Example"

13.8.1 Creating C++ Stored Procedures

When you are creating a C++ stored procedure, you use SODA APIs to access the database and transaction objects. This section demonstrates how to develop your C++ stored procedures.

- Section 13.8.1.1, "C++ Stored Procedure Include File and Procedure Definition"
- Section 13.8.1.2, "Access SODA Objects Within Your C++ Stored Procedure"

13.8.1.1 C++ Stored Procedure Include File and Procedure Definition

When you create the C++ source file, remember to do the following:

- Include the `olcsp.h` include file.
- The following defines the stored procedure or function prototypes. For each, you can have up to 32 parameters.
  - C++ stored procedure prototype:

    ```
    OL_CSP_CALL void cproc (const DBData &d1, const DBData &d2, ... , DBData
    &dN)
    ```

  - C++ stored function prototype:

    ```
    OL_CSP_CALL DBData cproc (const DBData &d1, const DBData &d2, ... , DBData
    &dN)
    ```

- Use `OL_CSP_CALL` before all of your procedures and functions, as it defines these as `extern "C" __declspec(dllexport)`. This enables the procedures and functions to be called from outside the DLL. For example, the following `sum` procedure uses this declaration:

  ```
  OL_CSP_CALL void sum (const DBData &a, const DBData &b, DBData &r)
  ```

- You can use the `DBData` object to represent almost any database type. In addition, it is easily cast to the correct datatype. For input parameters in the procedures, you can use `const DBData &`. For input/output parameters, use `DBData &` as the definition of the parameter.

  Once inside the procedure, cast the parameters as shown below:

  ```
  OL_CSP_CALL void sum (const DBData &a, const DBData &b, DBData &r)
  {  r = (int)a + (int)b; }
  ```

13.8.1.2 Access SODA Objects Within Your C++ Stored Procedure

We use SODA, instead of ODBC, to provide a reliable access to the database and transaction objects. To access the SODA API objects, use the methods defined in the `olcsp.h` include file, as follows:

> **Note:** For details on the SODA API, see Chapter 12, "Using Simple Object Data Access (SODA)".

- After you retrieve the session object, do not close the connection in side the procedure.
- Retrieve the SODA API `DBSession` object with the `olCSPGetSession()` method, as follows:

  ```
  DBSession &sess = olCSPGetSession();
  ```

- Retrieve the SODA API `DBSqlSession` object, which is used in preparing and executing SQL statements, with the `olCSPGetSqlSession` method. Once you retrieve the `DBSqlSession` object, you can prepare the SQL statement within a `DBSqlStmt` object. The returned `DBSqlSession` object is created based on the existing ODBC handle. The following example retrieves the SQL Session, prepares and executes a statement:

  ```
  DBSqlSession sess = olCSPGetSqlSession();
  DBSqlStmt stmt = sess.prepare("insert into testsql values(?)");
  stmt.execute(DBDataList() << v);
  ```

- If this procedure was executed within the trigger, you can retrieve the object on which the trigger was invoked with the `olCSPGetObject` method. This returns a `DBObject` of the trigger object. This will not work for BEFORE CREATE or AFTER DELETE triggers.
- If you want to use ODBC instead of SODA, you can retrieve the ODBC connection handle with the `olCSPGetODBCHdl` method.
- All C++ stored procedures can throw `DBException`. This exception is automatically thrown if any SODA/SODASQL operation fails inside your stored procedure. If you are using a WinCE device, then you must use the ALE library for exceptions. See the ALE documentation for more information.

13.8.2 Building Your C++ Stored Procedures

You can either build your stored procedure manually or by using the `olsp.mak` makefile. The following describes both processes:

- Section 13.8.2.1, "Linking in Appropriate Libraries"
- Section 13.8.2.2, "Automatically Build Your Stored Procedure"
- Section 13.8.2.3, "Manually Building Your Stored Procedure"

13.8.2.1 Linking in Appropriate Libraries

When you build your procedure, link in one or more of the following libraries:

- For all builds, link in `olobj40.lib`, which exists in `<`*`Mobile_ Server`*`>Mobile/SDK/lib`.
- If you are using SODASQL in your stored procedure, then link in `sodasql.lib`.

13.8.2.2 Automatically Build Your Stored Procedure

If you have only a single source file, then you can use the `olsp.mak` makefile to build. The resulting DLL is named the same as the source file. This makefile supports building procedures for both the desktop and Windows CE platforms. Set up the following within the makefile before you execute to ensure a proper build:

1. Place the `olsp.mak` makefile in the same location as your source file.
2. If you build C++ stored procedures for a Windows CE device, then before you execute the makefile, you need to run the batch file from embedded visual C++. This sets the appropriate build environment for your windows CE platform. The following example shows execution of the `wcearmv4.bat` batch file:

   ```
   C:\EVC4.0\EVC\wce420\bin\WCEARMV4.BAT
   ```

3. Set the `MOBILESDK` environment variable—which defines the Oracle Lite Mobile SDK directory. For example,

   ```
   MOBILESDK=C:\oracle\ora90\mobile\sdk
   ```

4. Within the makefile, define `CDEFINE` (compiler defines) and `LFL` (linker flags) macros for your Windows CE platform.
5. If you are building .Net procedures, then set the `NETFRKDIR` environment variable in the makefile to point to your .Net Framework directory
6. If building for the Compact Framework, then define the `CFK` macro and set the `CFSDKDIR` environment variable in the makefile to point to your Compact Framework SDK directory.
7. Execute the makefile, as follows:

   ```
   nmake -f olsp.mak MyProc.dll [macros] [options]
   ```

   The macros that you can use are as follows:

   - For debug mode, use the macro `DEBUG=/DDEBUG`.
   - For compact framework, use the macro `CFK=1`.
8. Copy the new DLL, such as `MyProc.dll`, into a directory on the system path. If your platform is a WinCE device, copy this DLL into the `\Windows` directory.

For example, if your source file is `MyProc.cpp` or `MyProc.cs`, then this makefile builds `MyProc.dll` for you.

13.8.2.3 Manually Building Your Stored Procedure

If you have more than one source file for the stored procedure, then you must manually build. Keep in mind the following:

1. Because you are using the `export "C"` declaration on the stored procedure, which supports the procedure being able to throw exceptions, use the `/EHc-` compiler flag when building the stored procedure.
2. If you are using the `SODASQL` in the procedure, then link with either the `olobj40.lib` and `sodasql.lib`.

13.8.3 Define Your C++ Stored Procedure

Define the methods in the class that you want to call from SQL with a call specification, which is created with either the `CREATE FUNCTION` or `CREATE PROCEDURE` commands.

Perform the following to define C++ stored procedures:

1. Define any methods in the C++ class that you want to make accessible to SQL by creating call specifications for these methods. By defining a method, you associate a SQL name to the method. SQL applications use this name to invoke the method.
2. Invoke the stored procedure through a SQL DML statement.

Define any static method in the class that you want to call from SQL by creating a call specification for it. The call specification maps the method's name, parameter types, and return types to SQL counterparts.

To create a call spec, use the SQL commands `CREATE FUNCTION` for methods that return a value or `CREATE PROCEDURE` for methods that do not return a value. The `CREATE FUNCTION` and `CREATE PROCEDURE` statements have the following syntax:

```
CREATE [OR REPLACE]
   { PROCEDURE <proc_name> [([<sql_parms>])] |
   FUNCTION <func_name> [([<sql_parms>])]
   RETURN <datatype> }
   AS LANGUAGE CPLUSPLUS NAME
   '<lib_name>::<func_name>)';
   /
```

Where:

- `<proc_name>` is a SQL procedure name; `<func_name>` is the name of the function in the DLL used for this procedure.
- `<sql_parms>` can be a maximum of 32 arguments. All arguments passed to the procedures are given as DBData values to the function, which must cast the arguments to the appropriate data type. The syntax has the following format:

  ```
  <arg_name> [IN | OUT | IN OUT] <datatype>
  ```

- `<datatype>` is the datatype.
- `<lib_name>` is the name of the DLL where the function is delcared, without the `.dll` extension.

For example:

The following call specification defines the method to SQL:

```
CREATE PROCEDURE bu1 (
    oc1 int,
    nc1 int,
    oc2 int,
    nc2 int)
  AS LANGUAGE CPLUSPLUS
  NAME 'CSPLib::bu1';
   /
```

13.8.4 C++ Stored Procedure Example

The following examples show how to create, build and define the stored procedures.

- Section 13.8.4.1, "C++ Stored Procedure and Trigger Example One"
- Section 13.8.4.2, "C++ Stored Procedure and Trigger Example Two"
- Section 13.8.4.3, "JDBC Calling a C++ Stored Procedure Example"

13.8.4.1 C++ Stored Procedure and Trigger Example One

The following example does the following:

1. Creates the t1 table.
2. Creates the call specification of `bu1` for the C++ stored procedure `bu1` in the `CSPLib.dll`.
3. Creates a `BEFORE UPDATE` trigger, `foo`, which calls the `bu1` C++ stored procedure before the values of `c1` and `c2` in the table are updated.

```
create table t1(c1 int, c2 int);

create procedure bu1(oc1 int, nc1 int, oc2 int, nc2 int)
   as language cplusplus name 'CSPLib::bu1';

create trigger foo before update of c1,c2 on t1
```

```
  for each row bu1(old.c1,new.c1,old.c2,new.c2);
```

The following demonstrates how the trigger is executed, which in turn invokes the C++ stored procedure:

```
insert into t1 values(1,2);
insert into t1 values(10,2);

--trigger fired here
update t1 set c1 = 10, c2 = 20 where c1 = 1;
update t1 set c1 = 100 where c1 = 10;
```

13.8.4.2 C++ Stored Procedure and Trigger Example Two

The following example does the same as Example 1, but with a more complicated trigger. The trigger and procedure are dropped at the end of this example.

```
create table t3(c1 int, c2 int);

create procedure bc2(tabref varchar, tranid int, opseq int, c1 int, c2 int) as
language cplusplus name 'CSPLib::bc2';

--special trigger columns here
create trigger foo2 before insert on t3 for each row bc2(OL__TABLEREF, OL__
TRANSID, OL__OPSEQ, new.c1,new.c2);

--trigger fired here
insert into t3 values(1,2);
insert into t3 values(10,20);
insert into t3(c1) values(100);
insert into t3(c2) values(100);

drop trigger foo2;
drop procedure bc2;
```

13.8.4.3 JDBC Calling a C++ Stored Procedure Example

The following example shows JDBC invoking a C++ stored procedure through the `CallableStatement`.

```
//The following statement creates the procedure TESTINOUT1 with in out parameters
stmt.execute("CREATE OR REPLACE PROCEDURE TESTINOUT1(A IN OUT INT,
     B IN OUT DOUBLE PRECISION, C IN OUT VARCHAR, D IN OUT DATE,
     E IN OUT TIME, F IN OUT BINARY) AS LANGUAGE CPLUSPLUS
   NAME CSPLib::testInOut");

CallableStatement cstmt =
          conn.prepareCall("{call TESTINOUT1(?, ?, ?, ?, ?, ?)}");

cstmt.registerOutParameter(1, Types.INTEGER);
cstmt.registerOutParameter(2, Types.DOUBLE);
cstmt.registerOutParameter(3, Types.VARCHAR);
cstmt.registerOutParameter(4, Types.DATE);
cstmt.registerOutParameter(5, Types.TIME);
cstmt.registerOutParameter(6, Types.BINARY);

//setting parameters to null values
cstmt.setNull(1, Types.INTEGER);
cstmt.setNull(2, Types.DOUBLE);
cstmt.setNull(3, Types.VARCHAR);
cstmt.setNull(4, Types.DATE);
cstmt.setNull(5, Types.TIME);
cstmt.setNull(6, Types.BINARY);

for(int i = 0; i < 5; i++) {
 //executing the procedure. The parameters will be modified inside the procedure
 cstmt.execute();
 int a = cstmt.getInt(1);
 double b = cstmt.getDouble(2);
 String c = cstmt.getString(3);
 Date d = cstmt.getDate(4);
 Time e = cstmt.getTime(5);
 byte [] f = cstmt.getBytes(6);
}
```

13.9 Using .Net Stored Procedures

The .Net environment enables you to create stored procedures from any .Net language, such as C++, C#, C, and Visual Basic .Net. You create procedures and functions based on methods of a .Net class that is stored in an external DLL. Unlike C++ procedures, you don't need a fixed signature. Instead, the procedure can receive arguments and return values for any supported data type.

Note: Windows CE devices have the following limitations:

- You cannot pass delegates to native code when using the Compact Framework.
- You cannot start the .Net runtime from native code. The only way to use .Net on a Windows CE device is to start it within a C# application before the stored procedures are invoked.

The following sections detail how to build your .Net stored procedures:

- Section 13.9.1, "Creating the .Net Source for Your Stored Procedure"
- Section 13.9.2, "Building Your .Net Stored Procedures"
- Section 13.9.3, "Define Your .Net Stored Procedure"
- Section 13.9.4, "Dropping .Net Stored Procedures"
- Section 13.9.5, ".Net Stored Procedure Example"

13.9.1 Creating the .Net Source for Your Stored Procedure

When you are creating a .Net stored procedure, you can use Oracle-specific .Net extension classes to access the database and transaction objects. The .Net extension classes discussed in the following sections are `OracleData`, `OracleDataRow`, and `OracleSPManager`.

The following sections demonstrate how to develop your .Net stored procedures:

- Section 13.9.1.1, "Defining Methods, Imports and Namespace"
- Section 13.9.1.2, "Access and Modify Database Using .Net Extension Classes In Stored Procedures"
- Section 13.9.1.3, "Access and Modify Database Using OracleSPManager Inside Triggers"

13.9.1.1 Defining Methods, Imports and Namespace

1. When you create your .Net source file, be sure to import the `Oracle.DataAccess.Lite` namespace.
2. All stored procedures are declared as public static methods of the class.

The following example defines public static methods and includes the `Oracle.DataAccess.Lite` namespace:

```
using System;
using Oracle.DataAccess.Lite;

public class SPClass
{
 //function which multiplies two integers
  public static int multiply(int a, int b)
  {
   return a * b;
  }

  //returns string length, as in C++ procedure example
  public static int strlen(string s)
    {
 return s.Length;
    }

//stores sum of first two arguments in the third argument
//as in C++ procedure example
    public static void trigSum(int a, int b, out int c)
    {
            c = a + b;
    }
public static void testInOut1(ref int dInt, ref double dDouble,
    ref string dStr, ref DateTime dDate, ref DateTime dTime,
    ref byte [] dBin)
    {
    dInt += 10;
        dDouble += 12.34;
        dStr += "aaaaa";

        dDate = dDate.AddYears(1);
        dDate = dDate.AddMonths(1);
        dDate = dDate.AddDays(1);

        dTime = dTime.AddMinutes(1);
        dTime = dTime.AddSeconds(1);

        int len = dBin == null ? 0 : dBin.Length;
        byte [] newBin = new byte[len + 5];
        if (dBin != null)
        Array.Copy(dBin, 0, newBin, 0, len);
        for(int i = len; i < newBin.Length; i++)
        newBin[i] = (byte)'x';
            dBin = newBin;
    }
}
}
```

13.9.1.2 Access and Modify Database Using .Net Extension Classes In Stored Procedures

The following are Oracle-specific .Net extension classes:

- OracleData—A .Net version of the SODA `DBData` class.
- OracleDataRow—encapsulates and Oracle Lite database row.

OracleData

The `OracleData` object is a .Net version of the SODA `DBData` class.

Data types that are supported by Oracle Database Lite can be used in the `OracleData` object. These types can be implicitly cast to other compatible types; the cast must still follow normal database SQL casting rules. Casting between scalar types are implicit; casting to array types are explicit. If you try to cast an incompatible type, and `OracleException` is thrown.

This includes the following data types:

Table 13–5 Data Type For OracleData Object

Data Type			
int, int[]	byte, byte[] database type binary	short, short[]	bool, bool []
long, long[]	double, double[]	string, string[]	
DateTime, DateTime[]	OracleDataRow, OracleDataRow[]	OracleBlob, Oracle Blob[] (Oracle Lite Blob object)	

For example, the following code shows how you can cast a `String` to an `Integer` using the `OracleData` object:

```
string s = "10";
OracleData d = new OracleData(s);
int i = d;
```

OracleDataRow

The `OracleDataRow` object encapsulates and Oracle Lite database row. You can query and modify column values in place, instead of using SQL. The `OracleDataRow` implements indexes on the row, which returns an `OracleData` object.

To create an object query on a table, implement `OracleDataReader`, an Oracle extension of the ADO.Net `DataReader` object, to return `OracleDataRow` objects. The following example uses the `GetDataRow` method of the `OracleDataReader` object to retrieve the desired rows. To set up the query more efficiently, set the `RowQuery` attributes of `Table` and `Filter` before executing the query, as follows:

Note: For more information on the ADO.Net classes, see Chapter 11, "Oracle Database Lite ADO.NET Provider".

```
OracleConnection conn =
    new OracleConnection("DSN=POLITE;UID=SYSTEM;PWD=MANAGER");
conn.Open();
OracleCommand cmd = (OracleCommand)conn.CreateCommand();
//set properties for row query of table name and where clause
cmd.RowQuery.Table = "T1"; //table name
cmd.RowQuery.Filter = "C1 < 10"; //where clause

//execute the query and retrieve the desired rows
OracleDataReader rd = (OracleDataReader)cmd.ExecuteQuery();

//While there are rows, process each row
while(rd.Read())
{  //Retrieve each row with the GetDataRow method into the OracleDataRow object
    OracleDataRow row = rd.GetDataRow();
    //query and modify in place columns C1 and C2 of the row.
    //implicit conversion to and from OracleData
    // Retrieve the integer in column C1
    int i = row["C1"];
    // Add 5 to the value in C1 and store it in column C2
    row["C2"] = i + 5;
    //convert the value to a string and write it out
    string s = row["C2"];
    Console.WriteLine(s);
}
rd.Close();
conn.Close()
```

You can retrieve and modify the row by performing the following:

1. Retrieve the row with the `GetDataRow` method of the `OracleDataReader` class.
2. Query and modify the retrieved row within the `OracleDataRow` object.

13.9.1.3 Access and Modify Database Using OracleSPManager Inside Triggers

When you have a stored procedure that is executed by a trigger, the actual row that caused the trigger is accessible through the `OracleSPManager`. Thus, you do not have to create a SQL statement to retrieve the desired row. Instead, use the `GetDataRow` method of the `OracleSPManager` object. Also, you can use the `GetConnection` method of this object to retrieve the current `Connection` object.

The `OracleSPManager` class contains the following static methods, which you can ues to retrieve the connection or the row:

```
public static OracleConnection GetConnection();
public static OracleDataRow GetDataRow();
```

The following example uses the `OracleSPManager` static methods to retrieve the connection and row:

```
public static void log1(int a, int b, int c)
{
    //get current connection from .Net procedure manager
    OracleConnection conn = OracleSPManager.GetConnection();

    //get current row for trigger
    OracleDataRow r = OracleSPManager.GetDataRow();

    if (r[0] != a || r[1] != b || r[2] != c)
        throw new OracleException("Invalid row");

    if (ia == 0 && ib == 0 && ic == 0)
       throw new OracleException("Invalid row");

    OracleCommand cmd = (OracleCommand)conn.CreateCommand();
    cmd.CommandText = "INSERT INTO T1_LOG VALUES(?, ?)";
    cmd.Parameters.Add(new OracleParameter(a));
    cmd.Parameters.Add(new OracleParameter(a + b + c));
    cmd.ExecuteNonQuery();
    //do not close connection here
 }
}
```

13.9.2 Building Your .Net Stored Procedures

You can either build your stored procedure using Visual Studio .Net or by using the `olsp.mak` makefile. See the Visual Studio documentation for how to build using Visual Studio .Net.

If you want to build using the `olsp.mak` file, follow the directions as given for the C++ stored procedures in Section 13.8.2.2, "Automatically Build Your Stored Procedure". There are a few directions in that section that are specific to the .Net environment, as follows:

1. If you are building .Net procedures, then set the `NETFRKDIR` environment variable in the makefile to point to your .Net Framework directory
2. If building for the Compact Framework, then define the `CFK` macro and set the `CFSDKDIR` environment variable in the makefile to point to your Compact Framework SDK directory.
3. When building the `SPClass.dll` C# class, use the following syntax for the make:

   ```
   nmake -f olsp.mak SPClass.dll
   ```

4. Move the resulting DLL into the appropriate place, which is either the application directory or in the global assembly cache. Use the `gacutil.exe` executable if you want to install this DLL in the global assembly cache.

13.9.3 Define Your .Net Stored Procedure

When you want to define your .Net stored procedure, perform the following:

- Section 13.9.3.1, "Create the .Net Class Object in the Oracle Lite Database"
- Section 13.9.3.2, "Define Methods With a Call Specification"

13.9.3.1 Create the .Net Class Object in the Oracle Lite Database

Before you can create the call specification for the .Net stored procedure, you must first create the class within the Oracle Lite database. Use the following syntax:

```
CREATE [OR REPLACE] DOTNET CLASS USING BFILE('AssemblyName', 'ClassName');
```

Where:

- `AssemblyName` is the assembly file name, such as `SPClass.DLL`.
- `ClassName` is the name of the class, such as `SPClass`. However, if the class is defined within a namespace, prefix the namespace name before the classname, as follows: `MyNameSpace.SPClass`.

For example, the following creates the `SPClass` within the Oracle Lite database.

```
create dotnet class using bfile('SPClass.dll', 'SPClass');
```

13.9.3.2 Define Methods With a Call Specification

Define the methods in the class that you want to call from SQL with a call specification, which is created with either the `CREATE FUNCTION` or `CREATE PROCEDURE` commands.

Perform the following to define your .Net stored procedures:

1. Define any methods in the .Net class that you want to make accessible to SQL by creating call specifications for these methods. By defining a method, you associate a SQL name to the method. SQL applications use this name to invoke the method.
2. Invoke the stored procedure through a SQL DML statement.

Define any static method in the class that you want to call from SQL by creating a call specification for it. The call specification maps the method's name, parameter types, and return types to SQL counterparts.

To create a call spec, use the SQL commands `CREATE FUNCTION` for methods that return a value or `CREATE PROCEDURE` for methods that do not return a value. The `CREATE FUNCTION` and `CREATE PROCEDURE` statements have the following syntax:

```
CREATE [OR REPLACE]
   { PROCEDURE <proc_name> [([<sql_parms>])] |
   FUNCTION <func_name> [([<sql_parms>])]
   RETURN <datatype> }
   AS LANGUAGE DOTNET NAME
   '<class_name>.<method_name>)';
   /
```

Where:

- `<proc_name>` is a SQL procedure name; `<func_name>` is the name of the function in the DLL used for this procedure.
- `<sql_parms>` can be a maximum of 32 arguments. All arguments passed to the procedures are given as DBData values to the function, which must cast the arguments to the appropriate data type. The syntax has the following format:

  ```
  <arg_name> [IN | OUT | IN OUT] <datatype>
  ```

- `<datatype>` is the datatype.
- `<class_name>.<method_name>` is the name of the class and method that is used for the procedure or function.

For example:

The following call specification defines the method to SQL:

```
CREATE PROCEDURE bu1 (
    oc1 int,
    nc1 int,
    oc2 int,
    nc2 int)
  AS LANGUAGE DOTNET
  NAME 'SPClass.bu1';
   /
```

13.9.4 Dropping .Net Stored Procedures

To drop a .Net class object from the database, delete it with the following drop statement:

```
drop dotnet class 'ClassName';
```

13.9.5 .Net Stored Procedure Example

The following examples show how to create, build and define the stored procedures.

- Section 13.9.5.1, ".Net Stored Procedure and Trigger Example One"
- Section 13.9.5.2, ".Net Stored Procedure and Trigger Example Two"

13.9.5.1 .Net Stored Procedure and Trigger Example One

The following example does the following:

1. Creates the .Net `SPClass`.
2. Creates the `t1` table.
3. Creates the call specification of `bu1` for the .NET stored procedure `bu1` in the class.method: `SPClass.bu1`.
4. Creates a `BEFORE UPDATE` trigger, `foo`, which calls the `bu1` .Net stored procedure before the values of `c1` and `c2` in the table are updated.

```
create dotnet class using bfile('SPClass.dll', 'SPClass');

create table t1(c1 int, c2 int);

create procedure bu1(oc1 int, nc1 int, oc2 int, nc2 int)
   as language dotnet name 'SPClass.bu1';

create trigger foo before update of c1,c2 on t1
  for each row bu1(old.c1, new.c1, old.c2, new.c2);
```

The following demonstrates how the trigger is executed, which in turn invokes the .Net stored procedure:

```
insert into t1 values(1,2);
insert into t1 values(10,2);

--trigger fired here
update t1 set c1 = 10, c2 = 20 where c1 = 1;
update t1 set c1 = 100 where c1 = 10;
```

13.9.5.2 .Net Stored Procedure and Trigger Example Two

The following example does the same as Example 1, but with a more complicated trigger. The trigger and procedure are dropped at the end of this example.

```
create table t3(c1 int, c2 int);

create procedure bc2(tabref varchar, tranid int, opseq int, c1 int, c2 int) as
language dotnet name 'SPClass.bc2';

--special trigger columns here
create trigger foo2 before insert on t3 for each row bc2(OL__TABLEREF, OL__
TRANSID, OL__OPSEQ, new.c1, new.c2);

--trigger fired here
insert into t3 values(1,2);
insert into t3 values(10,20);
insert into t3(c1) values(100);
insert into t3(c2) values(100);

drop dotnet class 'SPClass';
```

13.10 Loading and Defining C, C++ or C# Stored Procedures

Once you create the C, C++ and C# stored procedures, you still need to load and define them on the client Oracle Lite database. Since C, C++ and C# languages are native code, any stored procedure created from these languages cannot be loaded into the database. However, you still need to perform the following against the Oracle Lite database for these stored procedures to be loaded and defined:

1. Load the stored procedure by copying it directly to the client.
2. For C# .Net stored procedures, create the class within the Oracle Lite database with the `CREATE DOTNET CLASS` SQL command.
3. Define the stored procedures similar to the Java stored procedures by using one of the following methods:
   - Execute the `CREATE FUNCTION` or `CREATE PROCEDURE` SQL command to define a call specification, which declares a method so that it may be called from SQL. The call specification tells Oracle which method to invoke when a call is made.
   - Execute the `CREATE TRIGGER` SQL command to specify that the stored procedure executes when a specific event occurs on the table, such as an insert, update, or delete.

Note: Attach is not supported for C, C++ and C# stored procedures.

14 Configure Security for the Oracle Lite Database

The following sections detail how to encrypt the Oracle Lite database:

14.1 Providing Security for the Mobile Client

The introduction of handheld devices within the corporate environment can pose a security threat to an organization. Devices are now used to store not only company contacts; but, with external cards, may store up to 60 gigabytes of information or more. Devices also provide a mobile point of entry into the organizational network that is located outside the network security perimeter. It is essential to secure this data if a device is lost or compromised.

Securing a device involves a layered approach. You must secure not only access to the device, but data stored on the device and communications across the network. Most aspects of security for a mobile device must be incorporated before Oracle Database Lite is even involved within the security infrastructure.

1. Security needs to start with the device itself. Authentication on the device must be implemented through pin or password authentication, biometric readers, secure digital media for storage, and even how the device is stored, transported, and accounted for.
2. Once access is gained to the device, further security needs to be implemented within the mobile application to prevent the application from being able to retrieve invalid data. Technologies, such as the Microsoft.Net Compact Framework, incorporate API calls that may be used to encrypt and decrypt any data that will be stored or retrieved from the device.

Oracle Database Lite provides several security features that may be utilized to help in securing data. These features aid in protecting information during both synchronization, and once access to a device has been obtained. The two most important aspects of security provided by Oracle Database Lite for the mobile infrastructure are the following:

1. Use Secure Socket Layer (SSL) to protect the transmission of data during the synchronization process. For full details, see Section 11.4, "Configuring for Secure

Socket Layer (SSL) Communication" in the *Oracle Database Lite Administration and Deployment Guide*.

2. Use one of the Oracle Database Lite encryption options to protect the actual database files. See Section 14.2, "Encrypting the Oracle Lite Database" for full details.

14.2 Encrypting the Oracle Lite Database

When you encrypt the Oracle Lite database using any of the encryption techniques in this section, the Oracle Lite database is encrypted using a 128 bit Advanced Encryption Standard (AES) encryption. This does not encrypt the data stored within the Oracle Lite database itself; it only encrypts the database as a whole.

In the default server configuration, Mobile clients do not automatically encrypt the snapshot ODB files. The following sections demonstrate how to encrypt the Oracle Lite database:

- Section 14.2.1, "Configuring for Automatic Encryption of the Oracle Lite Database"
- Section 14.2.2, "Create a Command to Initiate Automatic Encryption of the Oracle Lite Database"
- Section 14.2.3, "Execute EncrypDB Command to Encrypt Database"

14.2.1 Configuring for Automatic Encryption of the Oracle Lite Database

The synchronization engine can automatically encrypt the Oracle Lite database used with the Mobile client. To configure for automatic encryption of the snapshot ODB files after initial synchronization, set the `ENCRYPTDB` parameter in the `SYNC` section in the `POLITE.INI/POLITE.TXT` file.

For details on what value to use for the `ENCRYPTDB` parameter in the `POLITE.INI/POLITE.TXT` file, see Section E.2.12, "ENCRYPTDB" in the *Oracle Database Lite Administration and Deployment Guide*.

14.2.2 Create a Command to Initiate Automatic Encryption of the Oracle Lite Database

On the server, you can configure for automatic encryption of the snapshot ODB files after initial synchronization by performing the following:

1. Logon to the Mobile Server as an Administrator and launch the Mobile Manager tool.
2. Click on Mobile Devices, followed by Administration.
3. Click on Command Management.
4. Click **Create Command**.
5. Create the following new Command:

```
Name: EncryptDB
Command: updt_conf.otl
Description: Encrypt Database
```

6. Edit the newly created command `EncryptDB`, as follows:

```
Command: updt_conf?app=sync&key=ENCRYPTDB&val=1
```

7. Apply the changes.

8. Edit the `DeviceInfo` Command. Insert the new Command `EncryptDB` and click **OK**.

For more information on sending commands to the Mobile device, see Section 7.6, "Sending Commands to Your Mobile Device" in the *Oracle Database Lite Administration and Deployment Guide*.

14.2.3 Execute EncrypDB Command to Encrypt Database

As described in Section C.4, "ENCRYPDB", you can execute the `encrypdb` command on the client to encrypt the Oracle Lite database. If you are using the database as an embedded database and not for synchronization, then you can provide the Mobile user password for the encryption. However, if you are using this database with the Mobile Server for synchronization, do not provide a password, as modifying this password will create an issue for synchronization.

14.3 Providing Your Own Encryption Module for the Client Oracle Lite Database

The database on the client—also known as the Oracle Lite database—uses Advanced Encryption Standard (AES) for encrypting the database. However, you can provide your own encryption module for the client database.

The following sections describe how to implement and plug-in your own encryption module.

- Section 14.3.1, "Encryption Module APIs"
- Section 14.3.2, "Plug-In Custom Encryption Module"

14.3.1 Encryption Module APIs

Oracle Database Lite invokes your encryption APIs when performing encryption duties, instead of the internal AES encryption module. Thus, you must develop and include the following APIs in your customized encryption module:

- Section 14.3.1.1, "Initialize the Encryption Module"
- Section 14.3.1.2, "Delete Encryption Context"
- Section 14.3.1.3, "Create the Encryption Key"
- Section 14.3.1.4, "Encrypt Data"
- Section 14.3.1.5, "Decrypt Data"

Note: All of the functions in this section are in Windows format. Adjust appropriately if developing on a UNIX environment.

14.3.1.1 Initialize the Encryption Module

Implement the `encCreateCtxt` function to initialize the external encryption module. Oracle Database Lite invokes this function when initializing encryption. This function returns an encryption context handle to Oracle Database Lite, which it passes back on all subsequent API calls. The context handle is displayed as a `void*`, so that you can make it any type of structure you desire.

```
extern "C" __declspec(dllexport) void* encCreateCtxt()
```

14.3.1.2 Delete Encryption Context

When Oracle Database Lite is finished with the encryption module, it invokes the `encDeleteCtxt` function to delete the encryption context—which was created with the `encCreateCtxt` function.

```
extern "C" __declspec(dllexport) void encDeleteCtxt(void * ctx)
```

14.3.1.3 Create the Encryption Key

Oracle Database Lite invokes your `encCreateKey` function to create the encryption key within the encryption context, as follows:

```
extern "C" __declspec(dllexport) void encCreateKey (void* ctx,
      const unsigned char* key, int len, int dir)
```

Where the input parameters are as follows:

- `ctx`—The encryption context, which is created in the `encCreateCtxt` function.
- `key`—Pointer to the key to be created.
- `len`—Length of the encryption key.
- `dir`—Encryption direction or type, where 1: encryption, 2: decryption, 3: both encryption and decryption.

14.3.1.4 Encrypt Data

Oracle Database Lite invokes your `encEncryptData` function to encrypt the data that is to be sent, as follows:

```
extern "C" __declspec(dllexport) int encEncryptData (void* ctx,
    const unsigned char* data, int len, unsigned char* out)
```

Where the input parameters are as follows:

- `ctx`—The encryption context, which is created in the `encCreateCtxt` function.
- `data`—Pointer to the data to be encrypted.
- `len`—Length of the data in bytes.
- `out`—Output buffer.

This function returns the number of bytes copied to the output buffer.

14.3.1.5 Decrypt Data

Oracle Database Lite invokes your `encDecryptData` function to decrypt the data that it receives. This function copies the result to the output buffer.

```
extern "C" __declspec(dllexport) int encDecryptData (void* ctx,
      const unsigned char* data, int len, unsigned char* out)
```

Where the input parameters are as follows:

- `ctx`—The encryption context, which is created in the `encCreateCtxt` function.
- `data`—Pointer to the data to be decrypted.
- `len`—Length of the data in bytes.
- `out`—Output buffer.

This function returns the number of bytes copied to the output buffer.

14.3.2 Plug-In Custom Encryption Module

Once implemented, you can plug-in your custom encryption module by adding the `[All Databases]` section to the `POLITE.INI` configuration file. You must either implement your encryption module into a DLL for the Windows environment or into a Shared Object (.SO) for the UNIX environment.

For example, if you created the encryption module as a DLL called `my_enc.dll`, which is located in the `C:\my_dir` directory, then you would add this module as the default encryption module in the `POLITE.INI` configuration file, as follows:

```
[All Databases]
EXTERNAL_ENCRYPTION_DLL=C:\my_dir\my_enc.dll
```

14.4 Pre-Configure Branch Office Passwords

When you install the Branch Office Manager on the Windows machine, it creates the `OracleDatabaseLite` user account with the minimum set of privileges required to execute the Oracle Database Lite software. This prevents Oracle Database Lite Branch Office executing under the `SYSTEM` account, which has broad privileges within the system and can make the system vulnerable.

Both the 'Oracle Lite Multiuser Service' is created as well as the normal Web-to-Go service executes under the privileges of the `OracleDatabaseLite` user. The Oracle Lite Multiuser Server enables remote clients to connect to the Oracle Lite database.

Normally, when installed, the password for the `OracleDatabaseLite` user is randomly generated during the setup. You can either pre-configure this password before the Branch Office installation or modify it after the configuration. See Section 3.5.3, "Defining Password for OracleDatabaseLite User for Branch Office on Windows Machine" in the *Oracle Database Lite Getting Started Guide*.

15

Oracle Database Lite Transaction Support

When an application connects to the local client database—Oracle Database Lite—it begins a transaction with the database. There can be a maximum of 64 connections to Oracle Database Lite. Each connection to Oracle Database Lite maintains a separate transaction, which conform to ACID requirements.

A transaction can include a sequence of database operations, such as `SELECT`, `UPDATE`, `DELETE`, and `INSERT`. All operations either succeed and are committed or are rolled back. Oracle Database Lite only updates the database file when the commit is executed. If an event, such as a power outage, interrupts the commit, then the database is restored during the next connection.

15.1 Locking

Oracle Database Lite supports row-level locking. Whenever a row is read, it is read locked. Whenever a row is modified, it is write locked. If a row is read locked, then different transactions can still read the same row. However, a transaction cannot access a row if it is a write locked row by another transaction.

15.2 What Are the Transaction Isolation Levels?

Each transaction is isolated from another. Even though many transactions run concurrently, transaction updates are concealed from other transactions until the transaction commits. You can specify what level of isolation is used within the transaction, as listed in Table 15–1:

Table 15–1 Isolation Levels

Isolation Level	Description
Read Committed	In Oracle Database Lite, a `READ COMMITTED` transaction first acquires a temporary database level read lock, places the result of the query into a temporary table, and then releases the database lock. During this time, no other transaction can perform a commit operation. No data objects are locked. All other transactions are free to perform any DML operation—except `commit`—during this time. Since a commit operation locks the database in `intent` exclusive mode, a read committed transaction, while gathering the query result, will block another transaction that is trying to commit or vice versa. A `READ COMMITTED` transaction provides the highest level of concurrency, as it does not acquire any data locks and does not block any other transaction from performing any DML operations. In addition, the re-execution of the same query (`SELECT` statement) may return more or less rows based on other transactions made to the data in the result set of the query. **Note**: A `SELECT` statement containing the `FOR UPDATE` clause is always executed as if it is running in a `REPEATABLE READ` isolation level. A `SELECT` statement can execute Java stored procedures. If the transaction executing the Java stored procedure is in the `READ COMMITTED` isolation level and the Java stored procedure updates the database, then the `SELECT` statement that executes the Java stored procedure must have a `FOR UPDATE` clause. Otherwise, Oracle Database Lite issues an error. **Note**: If you are retrieving a large object, such as a BLOB, within a READ COMMITTED transaction, see Section 4.3.46.9, "Select Statement Behavior When Retrieving BLOBs in a READ COMMMITTED transaction" section in the *Oracle Database Lite SQL Reference*.
Repeatable Read	In this isolation level, a query acquires read locks on all of the returned rows. More rows may be read locked because of the complexity of the query itself, the indexes defined on its tables, or the execution plan chosen by the query optimizer. The `REPEATABLE READ` isolation level provides less concurrency than a `READ COMITTED` isolation level, transaction because the locks are held until the end of the transaction. A "phantom" read is possible in this isolation level, which can occur when another transaction inserts rows that meet the search criteria of the current query and the transaction re-executes the query. If a `FOR UPDATE` clause is used in a query, a short-term update lock is acquired on the current row(s) being selected. If a row is updated, the lock is converted into an exclusive lock. An exclusive lock prevents any other transaction running in an isolation level other than `READ COMMITTED` to access this row. If the row is not updated but the next row is fetched, the update lock is downgraded to a read lock, permitting other transactions to read the row.
Serializable	This isolation level acquires shared locks on all tables participating in the query. The same set of rows is returned for the repeated execution of the query in the same transaction. Any other transaction attempting to update any rows in the tables in the query is blocked.
SingleUser	In this isolation level only one connection is permitted to the database. The transaction has no locks and consumes less memory.

Refer to the documentation for ODBC for more information on isolation levels.

15.3 Configuring the Isolation Level

The default isolation level is `READ COMMITTED`. You can modify the isolation level for a data source name (DSN) by using the ODBC Administrator—which you can bring up by executing `odbcad32`—or by manually editing the `ODBC.INI` file. We recommend that you use the `odbcad32` tool, as it will inform you if you have an incorrect combination of isolation level and cursor type. See Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types" for more information.

When you bring up the ODBC Administrator, under the User DSN tab, double-click the Oracle Lite 40 ODBC driver for which you want to modify the isolation level. Select the default cursor type from the pull-down list.

If you decide to edit the `ODBC.INI` file by hand, then set the isolation level as follows:

```
IsolationLevel = XX
```

where the value for `XX` is Read Committed, Repeatable Read, Serializable, or Single User.

Note: The `ODBC.INI` file is available in Windows under `%WINDIR%` and in Linux under `$OLITE_HOME/bin`. For the Linux platform, you must have write permissions on the directory where this is located to be able to modify them.

Alternatively, you can define the isolation level of a transaction by using the following SQL statement:

```
SET TRANSACTION ISOLATION LEVEL <ISOLATION_LEVEL>;
```

where `ISOLATION_LEVEL` is `READ COMMITTED`, `REPEATABLE READ`, `SERIALIZABLE`, or `SINGLE USER`.

See Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types", for information on how certain isolation levels and scrollable cursors sometimes cannot be used in combination.

15.4 Supported Combinations of Isolation Levels and Cursor Types

If you use the ODBC Administrator—which you can bring up by executing `odbcad32`—then this tool informs you if you are using an incorrect combination of isolation level and cursor type.

We support these types of cursors

- Forward only cursors allow you to only move forward through the returned result set. You cannot go backwards, nor can you view any additional modifications. To return to a row, you would have to close the cursor, reopen it and then move to the row you wanted to see. However, it is the fastest cursor for moving through a result set.
- Scrollable cursors are the most flexible as they allow you to go forward and backward through the returned result set, but are also expensive. The other advantage of using a scrollable cursor is you can see modifications directly after they occur.

The three supported types of scrollable cursors are as follows:

- Static—The result set appears to be static; that is, it does not detect modifications made to the membership, order, or values of the result set after the cursor is opened. This cursor can detect its own modifications, just not the modifications of others.
- Dynamic—Any modifications to the result set can be detected and viewed when the row is re-fetched.
- Keyset Driven—The abilities of this cursor is between the static and dynamic. It can detect modifications to the values in the rows of the result set; however, it cannot detect changes to the membership and order of the result set.

Refer to the documentation for ODBC for more information on cursor types.

For some cursors, you cannot combine them with certain isolation levels. Table 15–2 shows the supported combinations of isolation levels and cursor types. Unsupported combinations generate error messages.

Table 15–2 Supported Combinations

	Forward Only Cursor	**Scrollable Static Cursor**	**Scrollable Keyset Driven Cursor**	**Scrollable Dynamic Cursor**
Isolation Level				
Read Committed	Supported	Supported	Unsupported	Unsupported
Repeatable Read	Supported	Unsupported	Supported	Supported
Serializable	Supported	Unsupported	Supported	Supported
Single User	Supported	Supported	Supported	Supported

16 Improving SQL Query Performance for the Oracle Lite Database

The following sections describe the methods you can manage the performance of your SQL Queries against the Oracle Lite database:

16.1 Determining Performance of Client SQL Queries With the EXPLAIN PLAN

To improve performance when accessing data on the local client Oracle Lite database, use the EXPLAIN PLAN. The EXPLAIN PLAN to determine the performance of your SQL query execution on the Oracle Lite database. To execute a SQL statement, Oracle might need to perform several steps. Each of these steps either physically retrieves rows of data from the database or prepares them in some way for the user issuing the statement. The combination of the steps Oracle uses to execute a statement is called an execution plan, which includes an access path for each table that the statement accesses and an ordering of the tables (the join order) with the appropriate join method. The execution plan shows you exactly how Oracle Database Lite executes your SQL statement.

The components of an execution plan include the following:

- An ordering of the tables referenced by the statement.
- An access method for each table mentioned in the statement.
- A join method for tables affected by join operations in the statement.

The EXPLAIN PLAN command stores the execution plan chosen by the Oracle Database Lite optimizer for SELECT, UPDATE, INSERT, and DELETE statement.

For information on how to generate an Explain Plan, see the Section 1.11 "Tuning SQL Statement Execution with the EXPLAIN PLAN" in the *Oracle Database Lite SQL Reference*.

16.2 Determine SQL Query Execution Through Oracle Database Lite Tracing

By setting the parameter `OLITE_SQL_TRACE = YES` in the `polite.ini` or `polite.txt` file on the client device, Oracle Database Lite generates a trace file named `oldb_trc.txt` that shows the following:

- The order tables are accessed by a query.
- The table scan access method used.
- The value of any bind variables utilized by the query.
- The time it takes for the first record to be retrieved.

See Section 7.8, "Enable Tracing for the Oracle Lite Database" for full details.

16.3 Optimizing SQL Queries for the Oracle Lite Database

You can optimize the following SQL queries in your application:

- Single-table queries
- Join queries
- Order By and Group By clauses

The following sections provide tips on improving the performance of the application SQL queries against the Oracle database:

- Section 16.3.1, "Optimizing Single-Table Queries"
- Section 16.3.2, "Optimizing Join Queries"
- Section 16.3.3, "Optimizing with Order By and Group By Clauses"
- Section 16.3.4, "Advanced Optimization Techniques for SQL Queries in Oracle Database Lite"

The tip examples use the database schema listed in Table 16–1:

Table 16–1 Database Schema Examples

Tables	Columns	Primary Keys	Foreign Keys
LOCATION	LOC#	LOC#	
	LOC_NAME		
EMP	SS#	SS#	
	NAME		
	JOB_TITLE		
	WORKS_IN		WORKS_IN references DEPT (DEPT#)
DEPT	DEPT#	DEPT#	
	NAME		
	BUDGET		
	LOC		LOC references LOCATION (LOC#)
	MGR		MGR references EMP (SS#)

16.3.1 Optimizing Single-Table Queries

To improve the performance of a query that selects rows of a table based on a specific column value, create an index on that column. For example, the following query performs better if the `NAME` column of the `EMP` table has an index.

```
SELECT *
FROM EMP
WHERE NAME = 'Smith';
```

An index may ruin performance if selecting more than 10% of the rows of the indexing columns is poor. For example, an index on `JOB_TITLE` may not be a good choice even if the query is as follows.

```
SELECT *
FROM EMP
WHERE JOB_TITLE='CLERK'
```

16.3.2 Optimizing Join Queries

The following can improve the performance of a join query (a query with more than one table reference in the FROM clause).

16.3.2.1 Create an Index on the Join Column(s) of the Inner Table

In the following example, the inner table of the join query is `DEPT` and the join column of DEPT is `DEPT#`. An index on `DEPT.DEPT#` improves the performance of the query. In this example, since `DEPT#` is the primary key of `DEPT`, an index is implicitly created for it. The optimizer will detect the presence of the index and decide to use `DEPT` as the inner table. In case there is also an index on `EMP.WORKS_IN` column the optimizer evaluates the cost of both orders of execution; `DEPT` followed by `EMP` (where `EMP` is the inner table) and `EMP` followed by `DEPT` (where `DEPT` is the inner table) and picks the least expensive execution plan.

```
SELECT e.SS#, e.NAME, d.BUDGET
FROM EMP e, DEPT d
WHERE e.WORKS_IN = DEPT.DEPT#
AND e.JOB_TITLE = 'Manager';
```

16.3.2.2 Bypassing the Query Optimizer

Normally, the optimizer selects the best execution plan, an optimal order of tables to be joined. In case the optimizer is not producing the best execution plan, you can control the order of execution using the `HINTS` feature. For more information, see the *Oracle Database Lite SQL Reference*.

For example, if you want to select the name of each department along with the name of its manager, you can write the query in one of two ways. In the first example which follows, the hint `/*+ordered*/` says to do the join in the order the tables appear in the `FROM` clause.

```
SELECT /*+ordered*/ d.NAME, e.NAME
FROM DEPT d, EMP e
WHERE d.MGR = e.SS#
```

or:

```
SELECT /*+ordered*/ d.NAME, e.NAME
FROM EMP e, DEPT d
WHERE d.MGR = e.SS#
```

Suppose that there are 10 departments and 1000 employees, and that the inner table in each query has an index on the join column. In the first query, the first table produces 10 qualifying rows (in this case, the whole table). In the second query, the first table produces 1000 qualifying rows. The first query will access the EMP table 10 times and scan the DEPT table once. The second query will scan the EMP table once but will access the DEPT table 1000 times. Therefore the first query performs better. As a rule of thumb, tables should be arranged from smallest effective number of rows to largest effective number of rows. The effective row size of a table in a query is obtained by applying the logical conditions that are resolved entirely on that table.

In another example, consider a query to retrieve the social security numbers and names of employees in a given location, such as New York. According to the sample schema, the query would have three table references in the FROM clause. The three tables could be ordered in six different ways. Although the result is the same regardless of which order you choose, the performance could be quite different.

Suppose the effective row size of the LOCATION table is small, for example `select count(*) from LOCATION where LOC_NAME = 'New York'` is a small set. Based on the above rules, the LOCATION table should be the first table in the FROM clause. There should be an index on LOCATION.LOC_NAME. Since LOCATION must be joined with DEPT, DEPT should be the second table and there should be an index on the LOC column of DEPT. Similarly, the third table should be EMP and there should be an index on EMP#. You could write this query as:

```
SELECT /*+ordered*/ e.SS#, e.NAME
FROM LOCATION l, DEPT d, EMP e
WHERE l.LOC_NAME = 'New York' AND
l.LOC# = d.LOC AND
d.DEPT# = e.WORKS_IN;
```

16.3.3 Optimizing with Order By and Group By Clauses

Various performance improvements have been made so that SELECT statements run faster and consume less memory cache. Group by and Order by clauses attempt to avoid sorting if a suitable index is available.

16.3.3.1 IN Subquery Conversion

Converts IN subquery to a join when the select list in the subquery is uniquely indexed.

For example, the following IN subquery statement is converted to its corresponding join statement. This assumes that c1 is the primary key of table t2:

```
SELECT c2 FROM t1 WHERE
c2 IN (SELECT c1 FROM t2);
```

becomes:

```
SELECT c2 FROM t1, t2 WHERE t1.c2 = t2.c1;
```

16.3.3.2 ORDER BY Optimization with No GROUP BY

This eliminates the sorting step for an ORDER BY clause in a select statement if ALL of the following conditions are met:

1. All ORDER BY columns are in ascending order or in descending order.
2. Only columns appear in the ORDER BY clause. That is, no expressions are used in the ORDER BY clause.

3. `ORDER BY` columns are a prefix of some base table index.
4. The estimated cost of accessing by the index is less than the estimated cost of sorting the result set.

16.3.3.3 GROUP BY Optimization with No ORDER BY

This eliminates the sorting step for the grouping operation if GROUP BY columns are the prefix of some base table index.

16.3.3.4 ORDER BY Optimization with GROUP BY

When ORDER BY columns are the prefix of GROUP BY columns, and all columns are sorted in either ascending or in descending order, the sorting step for the query result is eliminated. If GROUP BY columns are the prefix of a base table index, the sorting step in the grouping operation is also eliminated.

16.3.3.5 Cache Subquery Results

If the optimizer determines that the number of rows returned by a subquery is small and the query is non-correlated, then the query result is cached in memory for better performance. For example:

```
select * from t1 where
t1.c1 = (select sum(salary)
from t2 where t2.deptno = 100);
```

16.3.4 Advanced Optimization Techniques for SQL Queries in Oracle Database Lite

Note: This section is provided for those administrators and developers who are already very familiar with optimization techniques for SQL queries. Thus, this material is advanced and not for a beginner.

Unlike procedural languages—such as Java or C—SQL is a declarative language. It states what to do, but does not tell how to do it. This frees developers from writing navigation code to retrieve data. The responsibility of navigation falls on the database management system (DBMS).

The query optimizer—a component of the DBMS—is responsible to come up with an efficient plan to execute the query. Since there are several ways to perform a query, the query optimizer and query execution engine decide how to deliver the result in the quickest time. In a perfect world, the query optimizer will always be right and the database will always be infallible. However, this is not the case. The developer needs to think about the characteristics and peculiarities of the query optimizer. When you do run into performance issues, you can improve the performance as simply as creating some indexes, dropping additional indexes, or re-writing the query. Oracle Database Lite constantly improves the optimizer, so that you do not have to re-write the query.

This section introduces you to the Oracle Database Lite Query Optimizer, briefly covers the architecture of Oracle Lite database, and then provides details of the query compilation and optimization. Lastly, we provide tips on improving query performance. For further information, there are several excellent articles in technical journals that cover SQL query optimization in great technical detail. Some of these journals are listed in the reference section of this document.

16.3.4.1 Oracle Lite Database Application Architecture

The basic database architecture components from the point of view of the application developer are outlined below:

Figure 16–1 Components in applications using ODBC or JDBC

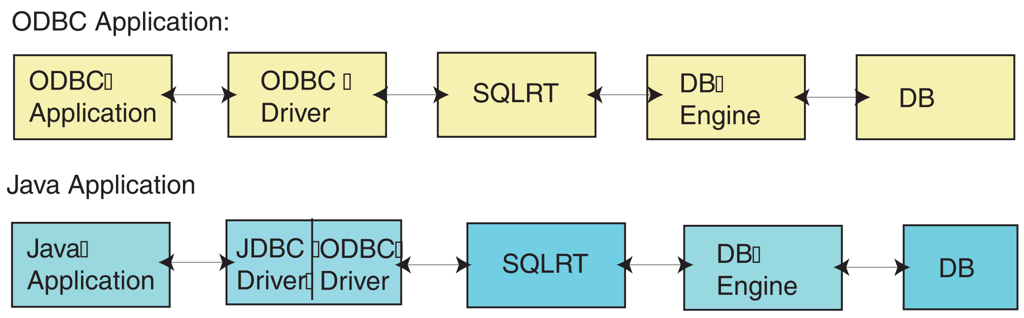


Table 16–2 Architecture Components

Component	Description
<<	Indicates a request and response information flow.
ODBC application	Typically a C or C++ application that issues ODBC API calls.
Java Application	A piece of code written in Java that uses JDBC API to manipulate the database.
ODBC driver	The driver that implements the ODBC API. It calls into the SQL runtime engine (SQLRT).
SQLRT	The SQL Runtime Engine that implements SQL functionality using the capabilities of the underlying database engine.
DB engine and DB	Database engine and Database

16.3.4.1.1 ODBC Application The ODBC application is usually written in C, or C++, or Visual Basic. Third party tools, such as Power Builder, can also generate code that invokes ODBC. The ODBC driver implements ODBC API semantics and uses internal SQLRT APIs to call into SQLRT.

16.3.4.1.2 SQLRT SQLRT, the Oracle Lite SQL engine, is implemented in the `olsql40.dll`. SQLRT implements SQL functionality using the capabilities of underlying database engine. This is covered in some detail in the following sections.

16.3.4.1.3 DB Engine The Oracle Lite database engine implements the object kernel API (also known as OKAPI). The database engine implements an object view of the world; that is, it implements classes, attributes, and iterators.

- Instead of creating a table—which contains a set of columns—you create a class containing a set of attributes.
- Instead of creating a cursor on a table, you create an iterator on a class or a group.

All classes belong to a group, which is a collection of classes.

The DB Engine maintains its own set of Meta catalogs (Meta classes) to store declarative information about classes, attributes, and indexes. For example, see the table below:

Table 16–3 Database Engine Meta Classes

Class	Description
okClass	Information about every class
okAttribute	Information for every attribute in all classes
okIndex	Information for every index created in the database
okGroup	Information about every group in the database

Note: All object kernel Meta classes belong to the MetaGroup group, which is case sensitive. The DB Engine is responsible to implement the ACID properties of a transaction.

16.3.4.2 Overview of SQL Runtime

The SQL Runtime is responsible for providing a SQL interface to the database. It maps SQL entities to the appropriate object kernel entities and translates all SQL operations into a sequence of basic object kernel primitives. For example, a table is mapped to a class; its table columns are mapped to attributes within the class. The mapping between SQL operations and object kernel APIs is not defined here, as this is not the focus of the document.

Execution of a SQL statement involves the following steps:

1. Compile

   You can compile a SQL statement into an internal representation that is easy and efficient to execute. A SQL statement can be one of the following:

   - DDL (data definition language): An example of a DDL statement is "`CREATE INDEX emp_last_name ON employee (last_name, first_name)`".
   - DML (data manipulation) statement: Examples of DML statements are `SELECT`, `INSERT`, `UPDATE`, `DELETE` and `COMMIT` statements.

2. Bind

   A SQL statement may contain markers (such as "`?`"), which are used as placeholders for parameters that can be supplied before execution of the statement. Binding sets the value for each marker in the SQL statement.

3. Execute

   This is when a previously compiled statement is executed. Execution involves interpretation of the internal representation of the SQL statement and making all calls into the database engine to achieve the desired result. The following are examples of what the execution means for particular statements:

   - For an index creation statement, the index is created.

- For an `INSERT` statement, the row is inserted into the table. In the object terminology, a new object is created.
- For a `SELECT` statement, the statement is executed, where a row is available for retrieval. The execution of a `SELECT` statement produces a result set, which is a set of rows. It is not necessary that all rows be materialized. However, for a `READ COMMITED` isolation level transaction, all rows are materialized at this step.

4. Fetch

   This step is required for a `SELECT` statement. Every fetch call returns one row to the caller.

5. Close

   Close the result set created in the execute step. Any remaining rows, if any, are discarded and any resources tied to the processing of the statement are freed up.

16.3.4.2.1 Compilation Compilation is somewhat like translating a JAVA program into byte code. In SQLRT, we translate a SQL statement into an internal data structure called a tree. The following are the steps SQLRT goes through to generate the execution tree, which determines the best method to execute that statement:

1. Parsing: The input statement is scanned and is converted into an abstract syntax tree. Grammatically incorrect statements are flagged and any syntax error is returned.
2. Normalization: The tree is walked, normalized and decorated. Transformation is carried out and semantic information is added to the tree nodes. Any tautologies are evaluated.

   For Example ((1 = 0) AND C1 > 5 OR C2 = 7) is evaluated to (C2 = 7). Any semantic error is caught during the tree traversal, such as Data type mismatches in expressions or SQL operations, references to non existing tables, or columns, unsupported SQL operations, and so on.
3. View expansion: Any references to views are expanded in line and the view tree is walked.
4. View Optimization: If possible, the view expansions are collapsed into the main queries. For example, the statement "SELECT * FROM v1,v2 where v1.c1=v2.c2" is resolved to a query on the base tables in v1 and v2. The transformation takes place on the query tree. This merging may not be possible. For example, if a view selects aggregate functions (COUNT, AVG, and so on.) or contains UNION or MINUS operators, it cannot be collapsed.
5. Subquery optimization: You can re-write the query to eliminate the subquery. This technique is called subquery un-nesting. The tables and filter conditions in the where clause are moved to the parent query block. This is possible only when the subquery does not contains any aggregates, `UNION`, or `MINUS` operations and SQLRT can make sure that the subquery does not return any duplicate rows.
6. Transitive Closure of Predicates: Predicates are analyzed and extra inferences are added to the `WHERE` clause, which helps the optimizer in selecting the best execution plan.
7. Predicate Push: The predicates are pushed down from top to bottom, which helps the queries on top of views. When a view contains any `UNION`, `MINUS` and `GROUP BY` clauses, it helps to push the filtering condition to the source of data or base tables.

8. Execution Plan Generation: The query is now analyzed to generate the best execution plan, which is based on a cost-based approach.
9. Query Execution: The execution plan generated is used to execute the query.

16.3.4.2.2 Query Tree Transformations or Query Re-write Examples These are examples of query tree transformations or query re-writes.

- View Optimization Example for View Replacement or Merging
- View Expansion and Predicate Push
- Subquery Transformation

View Optimization Example for View Replacement or Merging

Consider the following statements:

1. `SQL> CREATE VIEW v_dept_emp AS SELECT emp.*, dept.dname, loc FROM emp, dept WHERE emp.deptno=dept.deptno;`
2. `SELECT  * FROM v_dept_emp WHERE loc = 'DALLAS';`

The query tree transformation process substitutes the definition of view `v_dept_emp` into the select query and collapses the query into single level query. The query then becomes as follows:

```
SELECT  emp.*, dept.dname, dept.loc FROM emp, dept WHERE emp.deptno=dept.deptno
and loc = 'DALLAS'
```

Note: The final query does not refer to the view.

View Expansion and Predicate Push

Consider the following example:

```
SQL> CREATE VIEW v_sal_expense (dno, name, total_sal) AS SELECT dept.deptno,
dept.dname, sum(sal)  FROM emp, dept WHERE emp.deptno=dept.deptno group by
dept.deptno, dname;

SELECT  * FROM v_sal_expense WHERE  total_sal > 10000;
```

Since the query involves aggregation, it cannot be collapsed into the main query and the query after re-write is as follows:

```
SELECT  * FROM (
   SELECT dept.deptno, dept.dname, sum(sal)  total_sal
   FROM emp, dept
   WHERE emp.deptno=dept.deptno
   group by dept.deptno, dname)  temp_view
WHERE temp_view.total_sal > 10000;
```

Note: The predicate `total_sal > 10000` was not pushed into the inner query block as `total_sal` refers to an aggregate `sum(sal)` column in the view definition.

Consider the following query on the same view:

```
SELECT  * FROM v_sal_expense WHERE  dno = 10;
```

The query after the re-write is as follows:

```
SELECT  * FROM (
   SELECT dept.deptno, dept.dname, sum(sal)  total_sal
   FROM emp, dept
   WHERE emp.deptno=dept.deptno and
   dept.deptno = 10
   group by dept.deptno, dname)  temp_view
WHERE dno=10;
```

The predicate `dept.deptno = 10` is pushed down into the nested view expansion, which demonstrates the Predicate Push optimization. The aggregation is performed for the department number 10 only; therefore, this query performs better.

Subquery Transformation

Consider the following query:

```
SELECT * FROM emp WHERE emp.deptno IN (SELECT deptno
          FROM dept WHERE loc = 'DALLAS');
```

Since the subquery selects a unique key column (`deptno`), it can be converted into a join query. In general, a join query provides more flexibility in optimization and performs better. This query could be transformed as follows:

```
SELECT emp.* FROM emp, dept
   WHERE emp.deptno = dept.deptno AND dept.loc = 'DALLAS';
```

Note: The above subquery is a non-correlated subquery; that is, the subquery does not make a reference to columns from the tables in the outer query. For a non-correlated query, Oracle Database Lite does not always transform it to a join query. Instead, sometimes it decides to cache the query result into memory and perform the IN operation at run-time. A correlated subquery, if it meets the correctness requirements, is always transformed into a join query, as follows:

```
SELECT * FROM emp WHERE emp.deptno IN (SELECT deptno FROM dept
WHERE loc = 'DALLAS' AND emp.deptno = dept.depno);
```

Which is transformed into the following:

```
SELECT emp.* FROM emp, dept WHERE emp.deptno = dept.deptno AND
dept.loc = 'DALLAS' AND emp.deptno = dept.depno;
```

16.3.4.3 Execution Plan Generation

Execution plan generation is the last step of query compilation. It is the responsibility of the query optimizer to find the least expensive plan. It generates all plausible execution plans and picks the least expensive plan. As the number of tables in a query increases, the cost of evaluating all possible orders of execution increases exponentially. The optimizer uses its own heuristics to reduce the search space. The query optimizer considers only I/O costs for comparing the different execution plans. It does not consider the CPU time used to perform different operations. The I/O cost is computed based on the statistical information available to it; therefore, the quality of cost estimation depends upon the quality of statistics available.

- Section 16.3.4.3.1, "Statistics"
- Section 16.3.4.3.2, "Access Methods"
- Section 16.3.4.3.3, "Single Table I/O Cost"
- Section 16.3.4.3.4, "Join Query Optimization"

16.3.4.3.1 Statistics The Oracle Database Lite engine maintains the following statistics at all times. You do not have to run a separate command to update the statistics.

Table 16–4 Oracle Database Lite Engine Statistic Parameters

Parameter	Description
npg	Number of data pages allocated to each table.
nrows	Number of rows in the table.
ndk	For each index, number of distinct keys.
nrangeSize	For each index, OKAPI supports an API to estimate the number of rows selected for a given range of key values.
nMaxKey	Maximum value of a key (an OKAPI call is used to estimate it).
nMinKey	Minimum value of a key (an OKAPI call is used to estimate it).

Selectivity Factor

To estimate I/O cost, the optimizer estimates the number of pages that will be read or written to satisfy the query. It evaluates the disk I/O costs for different execution plans before selecting the best one. It assigns a selectivity factor to each predicate (also called a factor in boolean algebra), which is defined as an expected fraction of rows and satisfies the predicate. That is, the selectivity factor is defined as follows:

```
Selectivity factor = (expected-number-rows)/(total-number-of-rows)
```

The current values of the selectivity factor are as follows:

Note: The values are subject to change without any notice.

Table 16–5 Selectivity Factor Values

Condition	Example	Default	With Index
Equality	Name = ‘Smith’	1/5	1/ndk
Range	C1 > 5	1/2	Pretty good estimate
Between	C1 between (4,10) orC1 > 4 and C1 < 10	1/3	Pretty good estimate
Is Null	C1 is NULL	1/10	1/10
Like	Name Like ‘Sm%’	1/3	Estimate*

Like: A like predicate is transformed into a range and like. The range predicate is then appropriately optimized. For example, Name like ‘S%’ is converted into Name like ‘S%’ AND Name >= ‘S’ AND Name < ‘T’. Now the range (‘S’, ‘T’) for Name can be used to calculate the selectivity.

Not Equal: Selectivity factor for not equal is as follows: (1-Selectivity factor for the equal operator).

Caveat With Bind Variables

When bind variables are present, then the selectivity factor for "range", "between" and "like" cannot be correctly estimated and the default selectivity factor is used.

16.3.4.3.2 Access Methods An important component of an execution plan is the "access method" used for each table involved in the execution plan. The Oracle Lite database engine supports the following access methods:

1. A Full table scan: All pages of the table are searched. Therefore, the cost of retrieval is equal to npg (the number pages) in the table.
2. Index access method: A key value or key range—such as, price between (10,15)—is used to retrieve the qualifying rows. The key or key range is used to find the row-ids of the matching rows. Each row-id uniquely identifies the location of the row in the database. The rows are fetched in increasing or decreasing order of the key, which is useful when optimizing queries containing order by or group by clauses.

Cost of Access Methods

The I/O Cost can be computed in terms of the following parameters:

Table 16–6 I/O Access Method Cost

Parameter	Description
npg	Number of data pages.
nrows	Number of rows in the table.
h	Height of the index. It is also called depth of an index tree.
nlf	Number of leaf pages in an index tree.
sf	Expected Fraction of the number of rows selected. It is between 0 and 1.

Since the values for "h" and "nlf" are not available, its values are improvised based on nrows and estimated key size.

Cost of a Full Table Scan

The cost of a full table scan is the number of data pages, as follows: `Cost = npg`.

Cost of an Index Scan

The cost of an index scan is approximated to be as follows:

```
Cost = the number of index pages read + the number of data pages read
```

Where: `number of index pages read = (h-2) + ceil(nlf * sf)`. The value for `h` is calculated based on the estimated key size and number of rows.

It is assumed that the root of index tree is always in memory. Thus, the cost of reading the root page is ignored. Assuming that only a small number of rows are selected by the index access method, we approximate the number of leaf pages read to be one. This is performed sine we do not have information about nlf. Even for a range scan, we approximate it to be one.

For a primary key index or for an index with ndk/nrows close to one, we assume the data to be clustered on the key column values and we estimate the number of data pages read as follows:

```
Number of data pages read = ceil(sf * npg)
```

If the index is not a primary key index, then there is a good chance that the consecutive key values will not be on the same data page. Each new row access can potentially read a new page. The number of data pages read will be in between `sf` `*npg` and `sf * nrows`. We use the following formula as an approximation to actual number of data pages read:

```
Number of data pages read = ceil (sf * sqrt (npg, nrows))
```

Therefore, the cost of index access is as follows:

- For a clustered index, the cost is = `(h-1) + ceil(sf * npg).`
- For a non-clustered index, the cost is =
  `(h-1) + ceil(sf*sqrt(npg,nrows)).`

16.3.4.3.3 Single Table I/O Cost To find the optimal execution plan for a single table query, the costs for each possible access methods are evaluated and the least expensive access method is picked. For example:

```
SELECT * FROM T1 where C1 between 1 and 5 AND C2 > 5 and C2 < 100;
```

Assuming that the indexes exist on C1 and C2, then the optimizer estimates the selectivity for predicates "C1 between 1 and 5" and "C2 > 5 and C2 < 100". It then computes the I/O cost for retrieving the rows using a full table scan, an index scan on C1, and an index scan on C2. The access method that produces the least amount of I/O is chosen.

Interesting Order Optimization

For a single table query that contains "order by" or "group by" clause, the interesting order optimization technique is used to influence which access method is chosen. The result set size and sorting cost are estimated. Sorting can be avoided, if an index is available that can return the rows in the right order. If it is less costly to use an execution plan that does not involve any sort, then it will be chosen. The size of the result set is given by the following:

```
Number of rows in the result set =
     nrows * min(selectivity values for each predicate in the where clause)
```

16.3.4.3.4 Join Query Optimization The join query optimization involves evaluation of a large number of query execution plans. The number of possible plans increases exponentially with the number of tables. The following query illustrates this:

```
SELECT e.empno, e.ENAME, d.dname
FROM EMP e, DEPT d
WHERE e.deptno = DEPT.DEPTNO
AND e.JOB = 'MANAGER'
AND e.sal > 2000;
```

Here both possible orders of (`EMP`, `DEPT`) or (`DEPT`, `EMP`) exist for the execution. If EMP is chosen as the driving table, then the rows qualifying (`EMP.JOB_TITLE = 'Manager' AND EMP.sal > 5000`) are retrieved one by one from the `EMP` table. The optimizer considers the three possible access methods for `EMP` table, as follows:

1. Sequential scan of `EMP` table.
2. Index access using index on `EMP.JOB_TITLE` if one exists.
3. Index access using index on `EMP.SAL` if one exists.

The optimizer picks the method that produces least amount of I/O. Based on the selectivity factor assigned to each predicate, it estimates the number of rows selected for the `EMP` table. Then, it estimates the cost of retrieving a set of matching rows for each outer row in the `EMP` table. The total cost of execution using this order is as follows:

```
Cost = npg_emp + est_row_emp * ( cost_per_row_dept)
```

Where:

Table 16–7

Parameter	Description
est_rowemp	Estimated number of rows fetched from EMP table
cost_per_row_dept	Cost of index access into DEPT to retrieve matching department rows for each row fetched from EMP

The same calculation is repeated for the order `DEPT`, `EMP`. Whichever order produces the lowest cost is chosen. As the number of predicates and tables increase the cost of computing, the different possibilities grow exponentially. To reduce the compilation time, Oracle Lite uses aggressive heuristics to prune the search space, thereby sometimes landing into a sub-optimal execution plan. Also, unreliable statistics values, skewed data, and unavailability of selectivity factors for non-index columns can contribute to sub-optimal execution plan generation.

The following are the main tasks performed during a join query optimization:

1. The optimizer isolates local predicates (the predicates on a single table) from join predicates. In addition, the optimizer estimates the effective table sizes by the applying the selectivity factor of local predicates to the table. Local predicates are predicates that refer to columns from one table only. Whenever an index is available, the calculation of selectivity factor is fairly accurate. Oracle Lite assumes that the data is uniformly distributed; however, when the data is skewed, the estimate can go wrong and the execution plan chosen may not be optimal. When an index is not available, it uses default selectivity for computation.
2. A driving table—the table with the smallest effective cardinality—is picked first. Its optimal access method is picked. The table is put in the set of "outer" tables.
3. The query is examined to discover which possible tables can be joined to the tables in the current outer tables. The cross product is not considered. The I/O cost is estimated for all possible joins. The least costly join is chosen and is added, along with the chosen table, to the outer table set. The step is repeated until it has selected all tables in the query. By the end, it has computed the execution order and access methods for each table in the query.
4. The optimizer saves the current execution plan and picks a new driving table, whose effective cardinality is the second lowest. It repeats step 3 and selects the least expensive execution plan of the two plans. Again, it repeats step 3 with the third, fourth and fifth smallest table—always keeping a copy of the current least expensive execution plan.
5. The optimizer creates hints for when to create an index for intermediate results of a view. This is useful when joining a view that is not collapsed to another table or view.
6. When two tables are outer joined, the master table has to be scanned before the slave table (the table whose column has "+" in the joining column).

Interesting Order Optimization

An interesting order optimization eliminates the final sorting step for queries containing order by or group by clauses. If a suitable index exists that can eliminate the sorting step, then the cost is estimated the following ways:

1. Sorting + the best execution plan.

2. Pick a drive table that has columns from order by or group by clause, such that an index can be used to retrieve the data in the right order. Estimate the execution plan cost.

The least expensive plan is then chosen.

16.3.4.4 Query Execution Engine

The SQL Runtime engine relies on the database engine to implement simple data filtering, access methods, and sorting. A single table query execution of a query involves the following steps:

1. Decide if a temporary table is necessary to store the result. For a `READ COMMITED` isolation level transaction, a temporary table is required to store the result. While the result is being materialized, all other transactions are blocked from committing to preserve the read committed semantics. A temporary table is necessary when sorting is required. The DBE can only sort full tables.
2. Create the iterator on the table. Push the maximum number filter conditions to the database engine. This way, the smaller result set is returned to `SQLRT`.
3. If there are any complex filters that cannot be evaluated by DBE, evaluate them now and reject any rows that do not qualify. Examples of complex filters are SQL functions and subquery operators, such as `UPPER(NAME) = 'SMITH'`, or `zipcode IN (SELECT zipcode from TX where ….)`.
4. If the temporary table is created, then store all qualifying rows into this table. Once all rows are inserted into the temporary table, then the result is returned from this table.

16.3.4.4.1 Join Query Execution The `SQLRT` implements the join operation by executing the query in a nested loop fashion. The optimizer has already picked the optimal order of tables. The execution begins with the first (outer most) table in the list. An iterator is created on this table. A qualifying row is retrieved. The next table is picked from the list and a new iterator is created using the qualifying values from the row already fetched. A new qualifying row is retrieved from the second table. If there are more tables in the list, then the process continues until you reach end of the list. This provides the first matching row for the SELECT statement. Find the next matching row from the last table. If you do not find any qualifying rows, then return to the previous table in the list and repeat the process. Every time you advance to the next table in the list, you create a new iterator. Every time you do not find any more matching rows on a table, then close the iterator and return to the previous table in the list. If you exhaust all rows in the outer most table, then you have found all rows. The execution is analogous to nested loops execution in a programming language, which is why it is called a nested loop join.

16.3.4.4.2 Nested View Execution Oracle Database Lite does not distinguish between dynamic views (the query block in the `FROM` clause) or a view table being used in the `FROM` clause. Both are processed in the same way. If a nested view cannot be merged with the containing query and it is not the first to be picked in the execution order, then `SQLRT` materializes the view into a temporary table and creates a temporary index on the joining column(s). The index is used for joining outer tables with the view. Since the index is created at runtime, the optimizer does not have access to selectivity factors for view columns. The order chosen by the optimizer is based on default selectivity factors and estimated number of rows in the view.

16.3.4.5 Optimization Tips

This section provides guidelines to debug performance problems and helps you design better queries for the Oracle Lite database. Query optimization refers to the techniques used to find best execution plan.

- Section 16.3.4.5.1, "Index Access Method"
- Section 16.3.4.5.2, "Identifying The Bottleneck"
- Section 16.3.4.5.3, "Single Table Query Blocks"
- Section 16.3.4.5.4, "Query Blocks Containing Multiple Tables"
- Section 16.3.4.5.5, "Known Limitations"

16.3.4.5.1 Index Access Method An index access method can be used for equality, as well as range predicates. A predicate has to be one of the following forms in order for it to be considered for index access:

- column_1 = value1
- column_1 rel-op value
- column_1 = value1 AND column_2 = value2 AND …
- column_1 = value1 AND column_2 = value2 AND … column_n rel-op value-n

Where:

- rel-op—One of "=". ">", "<", ">=", <="
- column_n—Prefix columns of an index key. The value is an expression that can be evaluated to a constant value. For example, UPPER(name) = 'TOM' cannot be used with an index access method, UPPER(name) is not a column name, but an expression on the column name. Whereas name = UPPER('TOM') can be used as an index predicate; the right hand side is a constant expression.

> **Note:** You should not create indexes on a column that has multiple duplicate values; that is, the ratio of `nrows/ndk` to `ndk` is large.

16.3.4.5.2 Identifying The Bottleneck The largest problem of solving a query optimization problem is identifying the performance bottlenecks. Where is the CPU spending time? A typical customer query contains multiple tables, predicates, and subqueries. It may also contain outer joins and `UNION` operations. We recommend the following steps to debug the problem:

1. Replace all synonyms with base objects. Expand all views by corresponding view definitions. Imagine how `SQLRT` processes the query and carries out all possible transformations. Identify all query blocks, where each query block contains one `SELECT` statement.
2. Experiment with different query blocks one by one and find the slowest performing query block.
3. Optimize the problematic query block by examining the indexes already existing on columns involved in the query block. Determine if creating new indexes or dropping some indexes improves the performance. Check the order of tables selected by the optimizer (See the "Tools" section). Can it be improved if the query is executed using a different execution order? You can use a HINT to force the execution order of tables.
4. Once the bottleneck is resolved, repeat the process for the next bottleneck.

Tools

In Oracle Database Lite, you can dump query execution plan information by enabling `SQL TRACING`, which is enabled by including the following line in the `polite.ini` configuration file.

```
OLITE_SQL_TRACE= yes
```

This creates an `oldb_trc.txt` file in the current working directory of the process accessing the database. If the file already exists, then it opens the file for appending the dump information. The dump contains the following basic information:

1. Text of the SQL statement and every views in the SQL statement.
2. The time taken to compile the query.
3. The value of each bind variables.
4. Order of joining the tables.
5. Temporary tables created.
6. Access method used for each table. For an index access method, it prints the index name and index number. If the index name is blank, then you can use `idxinfo.exe` to discover the index information.

16.3.4.5.3 Single Table Query Blocks For a single table query, the query optimizer does not select the best available access method. However, it does collect statistics for all available indexes. The job for selecting the best index is left to the DBEngine, which uses a rule-based approach to select the appropriate index. Ensure that the index is available for the highest selective columns, as shown in the following example:

```
SELECT * FROM EMP WHERE NAME = 'Smith' and  EmpNo  between 1 and 1000;
```

Assuming that the total number of employees is a few thousand, then we would expect the predicate `NAME = 'Smith'` to return fewer rows than the predicate `EmpNO` between 1 and 1000. Therefore, we should create an index on the `NAME` column.

Note: Since the DBEngine is following a rule-based approach and EMPNO is a primary key column, it may not select the index on the NAME column.

16.3.4.5.4 Query Blocks Containing Multiple Tables Due to limitations of availability of statistics, and inherent assumptions made about the distribution of data, the execution plan chosen is not always optimal. Also, when suitable indexes are not present, the Oracle Lite Database Engine uses a sequential scan, as opposed to an index access method. To illustrate the importance of the index, see the following query:

```
SELECT e.empno, e.ENAME, d.dname FROM EMP e, DEPT d
WHERE e.deptno = DEPT.DEPTNO AND e.JOB = 'MANAGER' AND e.sal > 2000;
```

cost = npg_{emp} + est_row_{emp} * (cost_per_row_dept)

Let us assume that EMP has 1000 rows with 50 rows per page; that is, the npgemp = 20. Let us assume that the est_row_{emp} is 50, npg_{dept} = 10 and the cost of the index access into the department is 2. The cost calculation is tabulated, as follows:

Table 16–8 Cost Calculation Tabulation

npg_{emp}	est_row_{emp}	npg_{dept}	Access Method dept	cost_per_row_dept	cost
20	50	10	Sequential	10	520 pages
20	50	10	Indexed	2	120 pages

The cost of execution changes dramatically when an index is present. Therefore, the biggest challenge to improve the performance of a query in an Oracle Lite database is as follows:

1. Find the right set of indexes.
2. Optimal order for execution of tables.

There is no easy answer to the above tasks. It requires a deep understanding of the query that you are writing. The first action is to figure out the driving table, which is usually a table with many conditions involving constant values. For example, in the above table, it is most likely the EMP table. On the other hand, if there are only couple of rows in the DEPT table, then the DEPT table is a good candidate for the driving table. Once you select a driving table, the next task is to figure out the possible tables that can be joined to this table. What indexes will help in joining the current result set to the new table? Try joining these two tables and test if the time you receive makes sense. Now, add the third table and repeat the process. To force a specific join order, you can use the HINT clause supported by the Oracle Lite Database. Refer to the *Oracle Database Lite SQL Reference* for more information.

16.3.4.5.5 Known Limitations

1. In the process of finding the maximum and minimum values for an index key, the optimizer can spend too much time if there are large number of duplicates values near maximum and minimum values.
2. Sorting cost calculation is arbitrary.
3. In the presence of host variables, the selectivity for a range or like predicate cannot be correctly estimated.

16.3.4.6 Glossary

- API - Application Programming Interface
- ACID - ACID properties refer to atomicity, consistency, isolation, and durability
- A Correlated Subquery - A subquery that references columns from tables that are not present in its own "from" clause.
- Cross Product - When you join two tables without any joining condition, you produce a cross product. The cross product of a table containing m rows with another table containing n rows produces (m x n) rows.
- OKAPI -- Object Kernel Application Program Interface is implemented by the Oracle Lite Database Engine, which you can use to program your database application.
- Predicate – A boolean condition that evaluates to "true", "false" or unknown. Examples are: (Emp.Salary > 50000), (Dept.DepNo = 10 AND Emp.HireDate > '17-Nov-2001')
- SQLRT – Oracle Lite SQL Runtime Engine that is responsible for implementing SQL functionality on top of Oracle Lite database engine.

16.3.4.7 References

1. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A.,Price T.G. Access Path Selection in a Relational Database System. In Readings in Database Systems. Morgan Kaufman. This is a classical paper and must read for any one who wants to learn about query optimization.
2. Surajit Chaudhuri, An Overview of Query Optimization in Relational Systems, Microsoft Research

A
POLITE.INI Parameters for the Oracle Lite Database

You can customize Oracle Database Lite by modifying the parameter values defined in your POLITE.INI file, which is available in Windows under %WINDIR%\POLITE.INI and in Linux under $ORACLE_HOME/bin. You must have write permissions on the directory where this file is located to be able to modify the POLITE.INI file.

Note: On the WinCE and EPOC platforms, this file is named POLITE.TXT, so that you can double-click on it to open the file.

The following discusses the parameters in the different sections in the POLITE.INI file:

- Section A.1, "POLITE.INI File Overview"
- Section A.2, "All Databases Section"
- Section A.3, "Sample POLITE.INI File"

A.1 POLITE.INI File Overview

The POLITE.INI file centralizes database volume ID assignments and defines parameters for all databases on a system. When you install Oracle Database Lite, the installation creates the POLITE.INI file in your Windows home directory. On Windows CE and EPOC, the file name is POLITE.TXT.

The installation automatically sets the parameters in your POLITE.INI file, but you can modify them to customize the product behavior. To modify the POLITE.INI file, use an ASCII text editor.

A.2 All Databases Section

The All Databases section describes the behavior of your Oracle Lite database. The following describes these parameters:

- Section A.2.1, "CACHE_SIZE"
- Section A.2.2, "DATA_DIRECTORY"
- Section A.2.3, "DATABASE_ID"
- Section A.2.4, "DB_CHAR_ENCODING"

- Section A.2.5, "EXTERNAL_ENCRYPTION_DLL"
- Section A.2.6, "FLUSH_AFTER_WRITE"
- Section A.2.7, "MAX_INDEX_COLUMNS"
- Section A.2.8, "MAX_ROWS"
- Section A.2.9, "MESSAGE_FILE"
- Section A.2.10, "NLS_DATE_FORMAT"
- Section A.2.11, "NLS_LOCALE"
- Section A.2.12, "NLS_SORT"
- Section A.2.13, "OLITE_SERVER_LOG"
- Section A.2.14, "OLITE_SERVER_TRACE"
- Section A.2.15, "OLITE_SQL_TRACE"
- Section A.2.16, "OLITE_WRITE_VERIFY"
- Section A.2.17, "OLITE_READ_VERIFY"
- Section A.2.18, "SQLCOMPATIBILITY"
- Section A.2.19, "TEMP_DB"
- Section A.2.20, "TEMP_DIR"
- Section A.2.21, "SERVICE_PORT"
- Section A.2.22, "SERVICE_WDIR"
- Section A.2.23, "PAGE_FILL_FACTOR"

A.2.1 CACHE_SIZE

Specifies the size of the object cache in kilobytes. The minimum is 128. If not set, the default is 4096 (4 megabytes).

A.2.2 DATA_DIRECTORY

On the WinCE platform, you may wish to define where the Oracle Lite database is installed. By default, the storage card is used—to preserve memory—and the storage card with the maximum free space is used. At least 32 MB of free space must be available. If there is not enough memory on the storage card, then the directory defaults to `\Orace`. If you want to specify the directory where the database is created, specify the directory in the `DATA_DIRECTORY` parameter, as follows:

```
DATA_DIRECTORY=\Orace
```

A.2.3 DATABASE_ID

Defines the next Database Volume ID number to be assigned the `CREATE DATABASE SQL` command. `DATABASE_ID` numbers must be unique for each database file on the system.

A.2.4 DB_CHAR_ENCODING

Specifies the Oracle Database Lite character set. If set to `NATIVE`, the default is the system default character set.

Table A–1 lists the supported code pages and their corresponding values of DB_CHAR_ENCODING for all supported languages.

Table A–1 Supported Code Pages and Values

Code Page	DB_CHAR_ ENCODING	Language
N/A	UTF8	All languages
(1250)	ee8mswin1250	(Croatian, Czech, Hungarian, Polish, Romanian, Slovak, and Slovenian)
(1251)	cl8mswin1251	(Bulgarian, Russian, and Ukranian)
(1252)	we8mswin1252	(English (United States), Catalan, Danish, Dutch (Netherlands), English (United Kingdom), Finish, French (France), German (Germany), Icelandic, Italian (Italy), Malay (Malaysia), Norwegian (Bokmal), Portuguese (Brazil), Portuguese (Portugal), Spanish (Mexico), Spanish (Spain), and Swedish)
(1253)	el8mswin1253	(Greek)
(1254)	tr8mswin1254	(Turkish)
(1255)	iw8mswin1255	(Hebrew)
(1256)	ar8mswin1256	(Arabic (Egypt), and Arabic (UAE))
(1257)	blt8mswin1257	(Estonian and Lithuanian)
(932)	ja16sjis	(Japanese)
(936)	zhs16gbk	(Chinese (PRC) and Chinese (Singapore))
(949)	ko16mswin949	(Korean)
(950)	zht16mswin950	(Chinese (Taiwan) and Chinese (Hong Kong))

A.2.5 EXTERNAL_ENCRYPTION_DLL

You can plug-in a custom encryption module for the Oracle Lite database by adding the EXTERNAL_ENCRYPTION_DLL parameter to the POLITE.INI configuration file. Use this if you do not want to use the default AES encryption provided for the client database.

You must either implement your encryption module into a DLL for the Windows environment or into a Shared Object (.SO) for the UNIX environment.

For example, if you created the encryption module as a DLL called my_enc.dll, which is located in the C:\my_dir directory, then you would add this module as the default encryption module in the POLITE.INI configuration file, as follows:

```
[All Databases]
EXTERNAL_ENCRYPTION_DLL=C:\my_dir\my_enc.dll
```

For more information, see Section 14.3, "Providing Your Own Encryption Module for the Client Oracle Lite Database".

A.2.6 FLUSH_AFTER_WRITE

Syntax

```
FLUSH_AFTER_WRITE=TRUE|FALSE
```

Default Value

`FALSE`

By default, the parameter `FLUSH_AFTER_WRITE` is disabled. Hence, writes to a database are not flushed. The last write operation during a `COMMIT` operation always flushes file buffers, thereby eliminating the danger of losing data. For devices that are unreliable, users can enable this flag and set the parameter to `TRUE`. When enabled, every write action flushes file buffers. However, this setting degrades the database `COMMIT` performance.

Note: This parameter applies to the WinCE platform only.

A.2.7 MAX_INDEX_COLUMNS

Defines the number of columns used in the index creation statement. For more information, see "Index Creation Options" in the *Oracle Database Lite SQL Reference*.

A.2.8 MAX_ROWS

This parameter only applies for WinCE only.

The number of rows displayed in the `msql` GUI tool in the tables tab. By default, this value is 20. If you want more than 20 rows displayed at a time, modify this value.

A.2.9 MESSAGE_FILE

Use the `MESSAGE_FILE` parameter to specify the location of the message file used for the client Oracle Lite database. The default is where the binaries are installed. You may want to modify where the message file is located if you want to test another language. Modifying the `MESSAGE_FILE` parameter means that you do not have to move files around to test other languages.

Configure the path and the name of the message file, as follows:

```
MESSAGE_FILE=C:\Olite\Mobile\Sdk\BIN\OLITE40.MSB
```

A.2.10 NLS_DATE_FORMAT

Allows you to use a date format other than the Oracle Database Lite default. When a literal character string appears where a date value is expected, the Oracle Database Lite tests the string to see if it matches the formats of Oracle, SQL-92, or the value specified for this parameter in the `POLITE.INI` file. Setting this parameter also defines the default format used in the `TO_CHAR` or `TO_DATE` functions when no other format string is supplied.

For Oracle, the default is `dd-mon-yy` or `dd-mon-yyyy`. For SQL-92, the default is `yy-mm-dd` or `yyyy-mm-dd`.

Using `RR` in the format forces two digit years less than or equal to 49 to be interpreted as years in the 21st century (2000–2049), and years 50 and over, as years in the 20th century (1950–1999). Setting the `RR` format as the default for all two digit year entries allows you to become year-2000 compliant. For example,

```
NLS_DATE_FORMAT='RR-MM-DD'
```

You can also modify the date format using the `ALTER SESSION` command. For more information, see the *Oracle Database Lite SQL Reference*.

A.2.10.1 Date Format

A date format includes one or more of the elements listed in the following table. Elements that represent similar information cannot be combined, for example, you cannot use `SYYYY` and `BC` in the same format string. Table A–2 lists date formats and their corresponding description.

Table A–2 Date Formats

Format	Description
`AM` or `P.M.`	Meridian indicator, periods are optional.
`PM` or `P.M.`	Meridian indicator, periods are optional.
`CC` or `SCC`	Century, "S" prefixes BC dates with "-".
`D`	Day of week.
`DAY`	Name of day, padded with blanks to length of 9 characters.
`DD`	Day of month (1-31).
`DDD`	Day of year (1-366).
`DY`	Abbreviate name of day.
`IW`	Week of year (1-52 or 1-53) based on the ISO standard.
`IYY`, `IY`, or `I`	Last 3, 2, or 1 digit(s) of the ISO year, respectively.
`IYYY`	4-digit year, based on the ISO standard.
`HH` or `HH12`	Hour of the day (1-12).
`HH24`	Hour of the day (0-23).
`MI`	Minute (0-59).
`MM`	Month (01-12, for example, JAN=01).
`MONTH`	Name of the month, padded with blanks to length of 9 characters.
`MON`	Abbreviated name of the month.
`Q`	Quarter of the year, (1,2,3,4, for example, JAN-MAR=1).
`RR`	Last 2 digits of the year, for years in other countries. This forces two-digit years less than or equal to 49 to be interpreted as years in the 21st century (2000-2049), and years 50 and over, as years in the 20th century (1950-1959).
`WW`	Week of the year (1-53), where 1 starts on the first day of the year and continues to the seventh day of the year.
`SS`	Second (0-59).
`SSSSS`	Seconds past midnight (0-86399).
`Y` or `YYY`	Year with comma in this position.
`YEAR` or `SYEAR`	Year, spelled out. "S" prefixes BC dates with "-".
`YYYY` or `SYYYY`	4-digit year. "S" prefixes BC dates with "-".
`YYY`, `YY`, or `Y`	Last 3, 2, or 1 digit(s) of the year.

A.2.10.2 Date Format Examples

Listed below are sample variations of the `NLS_DATE_FORMAT` parameter.

1. `YYYY-MONTH-DAY:HH24:MI:P.M.`

2. `YYYY/MONTH/DD, HH24:MI A.M.`
3. `YYYY-MONTH-DAY:HH24:MI:PM`
4. `MM D, YYY, HH:MI A.M.`
5. `MM, WW, RR, HH:MI A.M.`
6. `MM, IW, RR, HH:M1 A.M.`
7. `MM, DY, RR, HH:MI A.M.`
8. `MM; DY; IYY, HH:MI A.M.`
9. `MON WW, RR, HH:MI A.M.`
10. `MONTH.DD, SYYYY, HH:MI A.M.`
11. `MONTH/DD, YYYY, HH:MI A.M.`
12. `MONTH|DD, YYYY, HH:MI A.M.`
13. `MONTH DD, YYYY, HH:SSSSS:MI A.M.`
14. `MONTH DD, HH:SS::MI CC`
15. `MONTH DD, HH:SS:MI SCC`
16. `MONTH W, YYYY, HH:MI A.M.`
17. `MONTH WW, YYYY, HH:MI A.M.`
18. `MONTH WW, RR, HH:MI A.M.`
19. `MONTH WW, Q, HH:MI A.M.`
20. `MONTH WW, RR, HH:MI A.M.`

A.2.11 NLS_LOCALE

Defines the `NLS_LOCALE` parameter in the `POLITE.INI` file to specify the locale data of Oracle Database Lite. Oracle Database Lite locale data includes the following items:

- Decimal character and group separator
- Locale currency symbol and ISO currency symbol
- Day, week, month names, and their abbreviations

For example, `NLS_LOCALE=FRENCH_FRANCE` specifies the locale data of `FRENCH_FRANCE` in Oracle Database Lite. Table A–3 describes the supported locale and corresponding values of the `NLS_LOCALE` setting.

Table A–3 Supported Locales and Values

Locale	NLS_LOCALE
English (United States)	`AMERICAN_AMERICA`
Arabic (Egypt)	`ARABIC_EGYPT`
Arabic (UAE)	`ARABIC_UNITED ARAB EMIRATES`
Bulgarian	`BULGARIAN_BULGARIA`
Catalan	`CATALAN_CATALONIA`
Chinese (PRC)	`SIMPLIFIED CHINESE_CHINA`
Chinese (Singapore)	`SIMPLIFIED CHINESE_SINGAPORE`

Table A–3 (Cont.) Supported Locales and Values

Locale	NLS_LOCALE
Chinese (Taiwan)	TRADITIONAL CHINESE_TAIWAN
Chinese (Hong Kong)	TRADITIONAL CHINESE_HONG KONG
Croatian	CROATIAN_CROATIA
Czech	CZECH_CZECH REPUBLIC
Danish	DANISH_DENMARK
Dutch (Netherlands)	DUTCH_THE NETHERLANDS
English (United Kingdom)	ENGLISH_UNITED KINGDOM
Estonian	ESTONIAN_ESTONIA
Finnish	FINNISH_FINLAND
French (France)	FRENCH_FRANCE
German (Germany)	GERMAN_GERMANY
Greek	GREEK_GREECE
Hebrew	HEBREW_ISRAEL
Hungarian	HUNGARIAN_HUNGARY
Icelandic	ICELANDIC_ICELAND
Italian (Italy)	ITALIAN_ITALY
Japanese	JAPANESE_JAPAN
Korean	KOREAN_KOREA
Lithuanian	LITHUANIAN_LITHUANIA
Malay (Malaysia)	MALAY_MALAYSIA
Norwegian (Bokmal)	NORWEGIAN_NORWAY
Polish	POLISH_POLAND
Portuguese (Brazil)	BRAZILIAN PORTUGUESE_BRAZIL
Portuguese (Portugal)	PORTUGUESE_PORTUGAL
Romanian	ROMANIAN_ROMANIA
Russian	RUSSIAN_CIS
Slovak	SLOVAK_SLOVAKIA
Slovenian	SLOVENIAN_SLOVENIA
Spanish (Mexico)	MEXICAN SPANISH_MEXICO
Spanish (Spain)	SPANISH_SPAIN
Swedish	SWEDISH_SWEDEN
Turkish	TURKISH_TURKEY
Ukrainian	UKRANIAN_UKRAINE

A.2.12 NLS_SORT

This parameter can be used to define the collation sequence for databases created on the Oracle Database Lite instance. Collation is referred as ordering strings into a culturally acceptable sequence. A collation sequence is a sequence of all collation elements from an alphabet from the smallest collation order to the largest.

```
NLS_SORT=[collation sequence]
```

When this parameter is used, all databases created with the `CREATEDB` command line utility or those that are replicated from the Mobile Server are enabled for the collation sequence unless a different collation sequence is specified when using the utility. Collation sequences currently supported are `BINARY` (default), `FRENCH`, `GERMAN`, `CZECH`, and `XCZECH`. You can only perform a linguistic sort on Oracle Lite databases that have the collation sequence of FRENCH, GERMAN, CZECH, OR XCZECH. You cannot do a linguistic sort on a BINARY collation sequence, which is used with all languages, except the three previously listed.

Note: Unless you require your databases to have linguistic sort enabled for a supported collation sequence, it is recommended that you use the `CREATEDB` utility with the `NLS_SORT <collation sequence>` parameter, which overrides this `POLITE.INI` parameter. Setting the `NLS_SORT` using the `POLITE.INI` file means that your databases have the specified collation sequence enabled. There is currently no way to convert a database from one collation sequence to another.

For a complete description of this feature, see Section C.2, "CREATEDB" and Section 7.4, "Support for Linguistic Sort".

A.2.13 OLITE_SERVER_LOG

The server log file contains the status of `oldaemon` processes including start, launch time, abort time, and executed processes. If any errors occurred, then the exception information is included. To forward all log information for a Multi-User Service on a LINUX machine, designate the filename of the logfile, as follows:

```
OLITE_SERVER_LOG = <path_and_filename>
```

A.2.14 OLITE_SERVER_TRACE

To debug the multi-user service, set this parameter to true, as follows:

```
OLITE_SERVER_TRACE = TRUE
```

A.2.15 OLITE_SQL_TRACE

Generates the SQL statement text, compilation time, execution plan, and the bind value.

For example:

```
OLITE_SQL_TRACE = TRUE
```

SQL trace output is dumped to a trace file named `oldb_trc.txt` in the current working directory of the database process. For a database service on Windows, Windows NT or the Oracle Database Lite daemon for a Linux platform, the current working directory is specified by the `wdir` parameter during the database startup service or daemon. Applications that use an embedded connection to connect to the database contain a working directory. This working directory is the application working directory. To implement the tracing feature, the database process must contain permissions to create the trace file in the current working directory. The trace output is always included in the trace file. If the trace file does not exist, it is created automatically.

To modify the working directory, see Section A.2.22, "SERVICE_WDIR".

A.2.16 OLITE_WRITE_VERIFY

You can perform diagnostics if you experience database corruption due to file system write errors, I/O errors, or a media device problem. Setting `OLITE_WRITE_VERIFY` to `TRUE` generates error reporting if a checksum error occurs on the device for the client.

If you receive a `POL-3207` error, then you may wish to execute the `validatedb` tool to see if the error message came about because of a checksum error. The `validatedb` tool deciphers if a checksum error has occurred. To further diagnose the checksum error, you can set `OLITE_WRITE_VERIFY` to perform further diagnostics to see if it is a filesystem, I/O, or media problem. After you set this to true on the client, then all write operations are verified that the checksum is valid. If not, then an error is written to a log file named `<odb_file>.odb_fserr.log` in the same directory as the Oracle Lite database (ODB). At this point, only metadata is written to this log file. However, if the file has a size greater than zero, then you know that a checksum error has occurred and there is a problem on your client device.

Note: Be careful in setting this parameter to `TRUE`, that you only use it while performing your diagnostic tests and that you change it back to `FALSE` when the problem is found. The error checking performed for this diagnostic effects your performance.

For example:

```
OLITE_WRITE_VERIFY = TRUE
```

A.2.17 OLITE_READ_VERIFY

This parameter specifies whether the checksum calculation is enabled or disabled during reading the database into the internal memory. This can be set to either TRUE or FALSE. The performance of the database improves notably if the checksum calculation is disabled.

Syntax:

```
OLITE_READ_VERIFY=TRUE | FALSE
```

Default Value

`FALSE`

A.2.18 SQLCOMPATIBILITY

Oracle Database Lite supports both Oracle SQL and SQL-92 features. For more information on Oracle SQL and SQL-92, see the *Oracle Database Lite SQL Reference*.

If there is a conflict between Oracle SQL and SQL-92, the `SQLCOMPATIBILITY` flag is referenced. If you specify `ORACLE` for the parameter, Oracle SQL is favored, and if you specify `SQL92`, SQL-92 is favored. If you do not include this parameter in the `POLITE.INI`, Oracle SQL is favored, by default.

A.2.19 TEMP_DB

The temporary database is created **by default** in virtual memory. This improves the performance of some queries that require the use of temporary tables. Unless you explicitly choose to create the temporary database in the file system with the `TEMP_DB` parameter, the `poltempx.odb` files are not created. The `*.slx` files that are sometimes used to store savepoint information are also not created. If you plan to create a large result set, you must either have enough swap space to hold the result, or choose the file option for the temporary database.

You can specify that the temporary database files are written to the file system either with the `TEMP_DB` or `TEMP_DIR` parameters. The `TEMP_DB` parameter enables you to define the name of the database files; the `TEMP_DIR` parameter allows you only to specify the directory to which the temporary database files are written.

To include this option, use the following syntax in the `POLITE.INI` file.

```
TEMP_DB=<path_and_temporary_database_name>
```

For example,

```
TEMP_DB=c:\temp\olite_
```

As a result of the example setting, Oracle Database Lite creates temporary databases as given below.

```
c:\temp\olite_0.odb, c:\temp\olite_1.odb, ...
```

A.2.20 TEMP_DIR

Specifies the directory where the temporary database `poltemp.odb` is created. If not set, the default is any `TEMP`, `TMP` or `WINDIR` setting defined in your environment. See Section A.2.19, "TEMP_DB" for more information.

A.2.21 SERVICE_PORT

Syntax

```
SERVICE_PORT=<port_number>
```

Default Value

The default port number is 1160.

Modify the default port of the multi-user service with this parameter.

A.2.22 SERVICE_WDIR

Syntax

```
SERVICE_WDIR=C:\WINDOWS\SYSTEM32
```

Modify the default working directory of the multi-user service with this parameter.

A.2.23 PAGE_FILL_FACTOR

The page fill factor for the database can be configured with this parameter. The higher the page fill factor, the smaller is the size of the database. The optimum value for this parameter is 80. The page fill factor determines the size of the database file, the performance of the database for a full table scan and how quickly an existing table can be updated.

Syntax:

```
PAGE_FILL_FACTOR=<page_fill_factor>
```

Default value: 80

A.3 Sample POLITE.INI File

The following content is displayed from a sample `POLITE.INI` file.

```
[All Databases]
DATABASE_ID=128
DB_CHAR_ENCODING=NATIVE
CACHE_SIZE=4096
MAX_INDEX_COLUMNS=5
SQLCOMPATIBILITY=SQL92
NLS_DATE_FORMAT=RR/MM/DD H24,MI,SS
NLS_LOCALE=ENGLISH
TEMP_DB=c:\temp\olite_
TEMP_DIR=D:\TMP
```

B

Catalog Views for the Oracle Lite Client

The following are the SQLRT and other system catalogs that are included in the client Oracle Lite database:

- Section B.1, "ALL_COL_COMMENTS"
- Section B.2, "ALL_CONSTRAINTS"
- Section B.3, "ALL_CONS_COLUMNS"
- Section B.4, "ALL_DEPENDENCIES"
- Section B.5, "ALL_INDEXES"
- Section B.6, "ALL_IND_COLUMNS"
- Section B.7, "ALL_OBJECTS"
- Section B.8, "ALL_PRIVILEGES"
- Section B.9, "ALL_SEQUENCES"
- Section B.10, "ALL_SYNONYMS"
- Section B.11, "ALL_TABLES"
- Section B.12, "ALL_TAB_COLUMNS"
- Section B.13, "ALL_TAB_COMMENTS"
- Section B.14, "ALL_USERS"
- Section B.15, "ALL_VIEWS"
- Section B.16, "POL__ALLOBJ"
- Section B.17, "POL__COLUSAGE"
- Section B.18, "POL__COMMENT"
- Section B.19, "POL__CONS"
- Section B.20, "POL__DATABASE_PARAMETERS"
- Section B.21, "POL__INDICES"
- Section B.22, "POL__INDICESDT"
- Section B.23, "POL__PROCEDURES"
- Section B.24, "POL__PROCEDURE_COLUMNS"
- Section B.25, "POL__SCHEMATA"
- Section B.26, "POL__SEQ"

B.1 ALL_COL_COMMENTS

Table B–1 ALL_COL_COMMENTS Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(128)	NOT NULL	Owner
TABLE_NAME	VARCHAR(128)	NOT NULL	Name of the table
COLUMN_NAME	VARCHAR(128)	NOT NULL	Name of the column
COMMENTS	VARCHAR(4096)	NULL	Description

B.2 ALL_CONSTRAINTS

Table B–2 ALL_CONSTRAINTS Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(128)	NOT NULL	Owner
CONSTRAINT_NAME	VARCHAR(128)	NOT NULL	Constraint name
CONSTRAINT_TYPE	VARCHAR(128)	NOT NULL	Constraint type
TABLE_NAME	VARCHAR(128)	NOT NULL	Name of the table
SEARCH_CONDITION	VARCHAR(1000)	NULL	Search condition
R_OWNER	VARCHAR(128)	NULL	Owner of Referenced Primary Key Constraint
R_CONSTRAINT_NAME	VARCHAR(128)	NULL	Reference Name of Primary Constraint
DELETE_RULE	VARCHAR(128)	NULL	Delete rule
STATUS	VARCHAR(20)	NOT NULL	Status
VALIDATED	VARCHAR(13)	NOT NULL	Validated

B.3 ALL_CONS_COLUMNS

Table B–3 ALL_CONS_COLUMNS Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(128)	NULL	Owner
CONSTRAINT_NAME	VARCHAR(128)	NULL	Constraint name
TABLE_NAME	VARCHAR(128)	NULL	Table name
COLUMN_NAME	VARCHAR(128)	NULL	Column name
POSITION	VARCHAR(10)	NULL	Position

B.4 ALL_DEPENDENCIES

Table B–4 ALL_DEPENDENCIES Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(30)	NOT NULL	Owner
NAME	VARCHAR(30)	NOT NULL	Name
TYPE	VARCHAR(16)	NULL	Type
REFERENCED_OWNER	VARCHAR(30)	NULL	Referenced owner
REFERENCED_NAME	VARCHAR(30)	NOT NULL	Referenced name
REFERENCED_TYPE	VARCHAR(16)	NULL	Referenced type

B.5 ALL_INDEXES

Table B–5 ALL_INDEXES Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(128)	NOT NULL	Owner
INDEX_NAME	VARCHAR(128)	NOT NULL	Index name
TABLE_OWNER	VARCHAR(128)	NOT NULL	Table owner
TABLE_NAME	VARCHAR(128)	NOT NULL	Table name
TABLE_TYPE	VARCHAR(10)	NULL	Table type
UNIQUENESS	VARCHAR(128)	NOT NULL	Index Uniqueness

B.6 ALL_IND_COLUMNS

Table B–6 ALL_IND_COLUMNS Parameters

Column	Datatype	Null	Description
INDEX_OWNER	VARCHAR(128)	NOT NULL	Index owner
INDEX_NAME	VARCHAR(128)	NOT NULL	Index name
TABLE_OWNER	VARCHAR(128)	NOT NULL	Table owner
TABLE_NAME	VARCHAR(128)	NOT NULL	Table name
COLUMN_NAME	VARCHAR(128)	NOT NULL	Column name
COLUMN_POSITION	VARCHAR(10)	NOT NULL	Column position

B.7 ALL_OBJECTS

Table B–7 ALL_OBJECTS Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(128)	NOT NULL	Owner
OBJECT_NAME	VARCHAR(128)	NOT NULL	Object name

Table B–7 (Cont.) ALL_OBJECTS Parameters

Column	Datatype	Null	Description
OBJECT_TYPE	VARCHAR(128)	NULL	Object type
CREATED	DATE	NULL	When object was created
STATUS	VARCHAR(128)	NULL	Status

B.8 ALL_PRIVILEGES

Table B–8 ALL_PRIVILEGES Parameters

Column	Datatype	Null	Description
SCHEMA	VARCHAR(128)	NOT NULL	Schema
TABLE	VARCHAR(128)	NOT NULL	Table
COLUMN	NUMBER(11)	NOT NULL	Column
GRANTOR1	VARCHAR(128)	NOT NULL	Grantor
GRANTEE1	UNKNOWN(8)	NOT NULL	Grantee
OBJTYPE	TINYINT(3)	NOT NULL	Object type
GTYPE	TINYINT(3)	NOT NULL	Grant type
GTABLE	TINYINT(3)	NOT NULL	Grant table
TO	UNKNOWN(8)	NOT NULL	Not used

B.9 ALL_SEQUENCES

Table B–9 ALL_SEQUENCES Parameters

Column	Datatype	Null	Description
SEQUENCE_OWNER	VARCHAR(128)	NOT NULL	Sequence owner
SEQUENCE_NAME	VARCHAR(128)	NOT NULL	Sequence name
MIN_VALUE	VARCHAR(10)	NOT NULL	Minimum value
MAX_VALUE	VARCHAR(10)	NOT NULL	Maximum value
INCREMENT_BY	VARCHAR(10)	NOT NULL	Increment value for sequence

B.10 ALL_SYNONYMS

Table B–10 ALL_SYNONYMS Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(128)	NULL	Owner
SYNONYM_NAME	VARCHAR(128)	NULL	Synonym name
TABLE_OWNER	VARCHAR(128)	NULL	Table owner
TABLE_NAME	VARCHAR(128)	NULL	Table name
DB_LINK	VARCHAR(128)	NULL	Database link

B.11 ALL_TABLES

Table B–11 ALL_TABLES Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(128)	NOT NULL	Owner
TABLE_NAME	VARCHAR(128)	NOT NULL	Table name
TABLESPACE_NAME	VARCHAR(128)	NULL	Tablespace name
CLUSTER_NAME	VARCHAR(128)	NULL	Cluster name
PCT_FREE	NUMBER(10)	NULL	Minimum percentage of free space in a block
PCT_USED	NUMBER(10)	NULL	Minimum percentage of used space in a block. Note: If the space in data block is less than PCT_FREE, no new rows will be added in that block until amount of space in table is less than PCT_USED
INI_TRANS	NUMBER(10)	NULL	Initial number of transactions
MAX_TRANS	NUMBER(10)	NULL	Maximum number of transactions
INITIAL_EXTENT	NUMBER(10)	NULL	Set of Contiguous blocks in a database segment that is automatically allotted when a segment is created
NEXT_EXTENT	NUMBER(10)	NULL	Block allocated after initial extent
MIN_EXTENTS	NUMBER(10)	NULL	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER(10)	NULL	Maximum extent that can be allocated
PCT_INCREASE	NUMBER(10)	NULL	Default percentage increase in extent size
BACKED_UP	VARCHAR(1)	NULL	Has table being backed up since last modification
NUM_ROWS	NUMBER(10)	NULL	Number of rows
BLOCKS	NUMBER(10)	NULL	Number of used blocks in the table
EMPTY_BLOCKS	NUMBER(10)	NULL	Number of empty blocks in the table
AVG_SPACE	NUMBER(10)	NULL	Average available free space in the table
CHAIN_CNT	NUMBER(10)	NULL	Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER(10)	NULL	Average row length, including row overhead

B.12 ALL_TAB_COLUMNS

Table B–12 ALL_TAB_COLUMNS Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(128)	NOT NULL	Owner
TABLE_NAME	VARCHAR(128)	NOT NULL	Table name
COLUMN_NAME	VARCHAR(128)	NOT NULL	Column name
DATA_TYPE	VARCHAR(30)	NULL	Data type
DATA_LENGTH	NUMBER(10)	NULL	Data length
DATA_PRECISION	NUMBER(10)	NULL	Data precision
DATA_SCALE	NUMBER(10)	NULL	Data scale
NULLABLE	VARCHAR(1)	NULL	Nullable
COLUMN_ID	NUMBER(10)	NOT NULL	Column ID
DEFAULT_LENGTH	NUMBER(10)	NULL	Default length
DATA_DEFAULT	VARCHAR(4096)	NULL	Default value for column
NUM_DISTINCT	NUMBER(10)	NULL	Number of distinct values this column holds
LOW_VALUE	NUMBER(10)	NULL	Low value
HIGH_VALUE	NUMBER(10)	NULL	High value
IS_HIDDEN	VARCHAR(1)	NULL	If item is hidden

B.13 ALL_TAB_COMMENTS

Table B–13 ALL_TAB_COMMENTS Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(128)	NOT NULL	Owner
TABLE_NAME	VARCHAR(128)	NOT NULL	Table name
TABLE_TYPE	VARCHAR(128)	NOT NULL	Table type
COMMENTS	VARCHAR(4096)	NOT NULL	Description

B.14 ALL_USERS

Table B–14 ALL_USERS Parameters

Column	Datatype	Null	Description
USERNAME	VARCHAR(30)	NOT NULL	Username
USER_ID	NUMBER(11)	NOT NULL	User ID
CREATED	DATE	NOT NULL	When user was created

B.15 ALL_VIEWS

Table B–15 ALL_VIEWS Parameters

Column	Datatype	Null	Description
OWNER	VARCHAR(128)	NOT NULL	Owner
VIEW_NAME	VARCHAR(128)	NOT NULL	View name
TEXT_LENGTH	NUMBER(10)	NOT NULL	Text length
TEXT	VARCHAR(1000)	NOT NULL	Text

B.16 POL__ALLOBJ

Table B–16 POL__ALLOBJ Parameters

Column	Datatype	Null	Description
CATALOG_NAME	VARCHAR(128)	NOT NULL	Catalog name
SCHEMA_NAME	VARCHAR(128)	NOT NULL	Schema name
OBJECT_NAME	VARCHAR(128)	NOT NULL	Object name
OBJECT_ID	NUMBER(10)	NOT NULL	Object ID
OBJECT_TYPE	SMALLINT(5)	NOT NULL	Object type
CREATED	DATE	NOT NULL	Date when created
LAST_DDL_TIME	DATE	NOT NULL	Timestamp for the last modification of the object resulting from a DDL command

B.17 POL__COLUSAGE

Table B–17 POL__COLUSAGE Parameters

Column	Datatype	Null	Description
COLUMN	VARCHAR(128)	NOT NULL	Column
TBLUSAGEREF	UNKNOWN(8)	NOT NULL	Reference to its POL__TBLUSAGE object
POSITION	NUMBER(11)	NOT NULL	Column position in Constraint definition. Used to identify order of column in the key

B.18 POL__COMMENT

Table B–18 POL__COMMENT Parameters

Column	Datatype	Null	Description
COLREF	UNKNOWN(8)	NOT NULL	Column Reference
COMMENTS	VARCHAR(4096)	NOT NULL	Comment

B.19 POL__CONS

Table B–19 POL_CONS Parameters

Column	Datatype	Null	Description
CATALOG	VARCHAR(128)	NOT NULL	Catalog
SCHEMA	VARCHAR(128)	NOT NULL	Schema
CONSTRAINT	VARCHAR(128)	NOT NULL	Constraint
TYPE	NUMBER(11)	NOT NULL	Type
TBLCONS	UNKNOWN(8)	NOT NULL	Reference to POL__TBLCONS that points to constrained table
ASSERTCONS	UNKNOWN(8)	NOT NULL	Reserved
DOMAINCONS	UNKNOWN(8)	NOT NULL	Reserved
FLAGS	NUMBER(11)	NOT NULL	For update rules
TO	UNKNOWN(8)	NOT NULL	Used by foreign key constraint to point to parent table primary key constraint object
EXTENSIONS	TINYINT(3)	NOT NULL	Not used

B.20 POL__DATABASE_PARAMETERS

Table B–20 POL__DATABASE_PARAMETERS Parameters

Column	Datatype	Null	Description
PARAMETER	VARCHAR(30)	NOT NULL	Parameter
TYPE	TINYINT(3)	NOT NULL	Type
VALUE	VARCHAR(128)	NULL	Value

B.21 POL__INDICES

Table B–21 POL__INDICES Parameters

Column	Datatype	Null	Description
CATALOG_NAME	VARCHAR(128)	NOT NULL	Catalog name
SCHEMA_NAME	VARCHAR(128)	NOT NULL	Schema name
INDEX_NAME	VARCHAR(128)	NOT NULL	Index name
TABLE_SCHEMA	VARCHAR(128)	NOT NULL	Table schema
TABLE_NAME	VARCHAR(128)	NOT NULL	Table name
FLAGS	NUMBER(11)	NOT NULL	Options for Index
INDEXOBJ	UNKNOWN(8)	NOT NULL	Index object
EXTENSIONS	TINYINT(3)	NOT NULL	Not used

B.22 POL__INDICESDT

Table B–22 POL__INDICESDT Parameters

Column	Datatype	Null	Description
COLUMN_NAME	VARCHAR(128)	NOT NULL	Column name
DIRECTION	TINYINT(3)	NOT NULL	Ascending or descending direction
INDEX_POSITION	NUMBER(10)	NOT NULL	Relative position in index
MASTER	UNKNOWN(8)	NOT NULL	Master index table
EXTENSIONS	TINYINT(3)	NOT NULL	Not used

B.23 POL__PROCEDURES

Table B–23 POL__PROCEDURES Parameters

Column	Datatype	Null	Description
CATALOG_NAME	VARCHAR(128)	NOT NULL	Catalog name
SCHEMA_NAME	VARCHAR(128)	NOT NULL	Schema name
PROCEDURE_NAME	VARCHAR(128)	NOT NULL	Procedure name
TABLE_CLASS	UNKONWN(8)	NOT NULL	Table class
PROCEDURE_TYPE	SMALLINT(5)	NOT NULL	Procedure type
PROCEDURE_REF	UNKNOWN(8)	NOT NULL	Procedure reference

B.24 POL__PROCEDURE_COLUMNS

Table B–24 POL__PROCEDURE_COLUMNS Parameters

Column	Datatype	Null	Description
MASTER	UNKNOWN(8)	NOT NULL	Master Index table
COLUMN_TYPE	SMALLINT(5)	NOT NULL	Column type
DATA_TYPE	SMALLINT(5)	NOT NULL	Data type
TYPE_NAME	VARCHAR(128)	NOT NULL	Type name
PRECISION	NUMBER(10)	NOT NULL	Significant digits for Column
LENGTH	NUMBER(10)	NOT NULL	Length
SCALE	SMALLINT(5)	NOT NULL	Significant digits right of decimal
RADIX	SMALLINT(5)	NOT NULL	Base of the system
POSITION	SMALLINT(5)	NOT NULL	Position

B.25 POL__SCHEMATA

Table B–25 POL__SCHEMATA Parameters

Column	Datatype	Null	Description
NAME	VARCHAR(128)	NOT NULL	Name
OWNER	VARCHAR(128)	NOT NULL	Owner
SCHEMA_ID	NUMBER(11)	NOT NULL	Schema ID
CREATED	DATE	NOT NULL	Date when created

B.26 POL__SEQ

Table B–26 POL__SEQ Parameters

Column	Datatype	Null	Description
SCHEMA_NAME	VARCHAR(128)	NOT NULL	Schema name
SEQ_NAME	VARCHAR(128)	NOT NULL	Sequence name
SEQ_REF	UNKNOWN(8)	NOT NULL	Sequence reference
INC_BY	NUMBER(10)	NOT NULL	Increment by value
START_WITH	NUMBER(10)	NOT NULL	Number to start sequence with
MIN_VALUE	NUMBER(10)	NOT NULL	Minimum value for sequence
MAX_VALUE	NUMBER(10)	NOT NULL	Maximum value for sequence
IS_ACCESSED	TINYINT(3)	NOT NULL	if accessed

B.27 POL__SYNONYM

Table B–27 POL__SYNONYM Parameters

Column	Datatype	Null	Description
SYN	VARCHAR(128)	NOT NULL	Synonym
OBJECT	VARCHAR(128)	NOT NULL	Object

B.28 POL__TBLCONS

Table B–28 POL__TBLCONS Parameters

Column	Datatype	Null	Description
CATALOG	VARCHAR(128)	NOT NULL	Fully qualified database file name
SCHEMA	VARCHAR(128)	NOT NULL	Schema
TABLE	VARCHAR(128)	NOT NULL	Table
CONSREF	UNKNOWN(8)	NOT NULL	Reference to POL__CONS object
FLAGS	NUMBER(11)	NOT NULL	Reserved
TYPE	NUMBER(11)	NOT NULL	Type
SEARCH	VARCHAR(128)	NOT NULL	Used by Check constraint to store user constraint string

Table B–28 (Cont.) POL__TBLCONS Parameters

Column	Datatype	Null	Description
SEARCHTREE	TINYINT(3)	NOT NULL	Internal representation of constraint clause
EXTENSIONS	TINYINT(3)	NOT NULL	Not used
CLASSREF	UNKNOWN(8)	NOT NULL	Reference to Class Object for the table
GROUPREF	UNKNOWN(8)	NOT NULL	Reference to Group Object for the table

B.29 POL__TBLUSAGE

Table B–29 POL_TBLUSAGE Parameters

Column	Datatype	Null	Description
CATALOG	VARCHAR(128)	NOT NULL	Catalog
SCHEMA	VARCHAR(128)	NOT NULL	Schema
TABLE	VARCHAR(128)	NOT NULL	Table
TBLCONSREF	UNKNOWN(8)	NOT NULL	Reference to POL__TBLCONS Object
DOMCONSREF	UNKNOWN(8)	NOT NULL	Not used
ASSERTCONSREF	UNKNOWN(8)	NOT NULL	Not used
EXTENSIONS	TINYINT(3)	NOT NULL	Not used

B.30 POL__TRIGGERS

Table B–30 POL__TRIGGERS Parameters

Column	Datatype	Null	Description
TRIGGER_NAME	VARCHAR(128)	NOT NULL	Trigger name
TABLE_CLASS	UNKNOWN(8)	NOT NULL	Table class
TRIGGER_KIND	SMALLINT(5)	NOT NULL	Trigger kind
METHOD	UNKNOWN(8)	NOT NULL	Method

B.31 POL__VIEWS

Table B–31 POL__VIEWS Parameters

Column	Datatype	Null	Description
NAME	VARCHAR(30)	NOT NULL	View name
VIEWDEF	VARCHAR(128)	NOT NULL	View definition
POSMAPS	NUMBER(11)	NOT NULL	Position Map

B.32 POL__USERS

Table B–32 POL__USERS Parameters

Column	Datatype	Null	Description
`USERNAME`	`VARCHAR(30)`	`NOT NULL`	User name
`PASSWORD`	`VARCHAR(30)`	`NOT NULL`	Password
`USER_ID`	`NUMBER(11)`	`NOT NULL`	User ID
`CREATED`	`DATE`	`NOT NULL`	Date when created

C Oracle Lite Database Utilities

This appendix describes how to use the following Oracle Lite database utilities for the Windows 32 and Windows CE platforms. Table C–1 lists all of the utility names:.

Table C–1 Database Tools and Utilities

Utility	Description
Section C.1, "The mSQL Tool"	Allows users to execute SQL statements against the Oracle Lite database.
Section C.2, "CREATEDB"	Use this to create your Oracle Lite database.
Section C.3, "REMOVEDB"	Use this to remove your Oracle Lite database.
Section C.4, "ENCRYPDB"	Use this to encrypt your Oracle Lite database.
Section C.5, "DECRYPDB"	Use this to decrypt your Oracle Lite database.
Section C.6, "BACKUPDB"	Use this to backup your Oracle Lite database.
Section C.7, "DefragDB to Defragment and Reduce Size of the Oracle Lite Database"	Defragmenting to reduce the size of the OracleLite database.
Section C.8, "ODBC Administrator and the Oracle Database Lite ODBC Driver"	Use this to manage ODBC connections by creating data source names (DSNs) that associate the Oracle Database Lite ODBC Driver with the Oracle Database Lite that you want to access through the driver.
Section C.9, "ODBINFO"	Use this utility to find out the version number and volume ID of an Oracle Database Lite database.
Section C.10, "VALIDATEDB"	Use this to validate the structure of an Oracle Lite database and find any corruption of the database.
Section C.11, "Transferring Data Between a Database and an External File"	Use either the command-line tool or programmatic APIs to load data from an external file into a table in Oracle Database Lite, or to unload (dump) data from a table in Oracle Database Lite to an external file.
Support for Linguistic Sort	Allows databases to be created with linguistic sort capability enabled. See Section 7.4, "Support for Linguistic Sort" for full details.
Section 13.3.1.4.1, "Using dropjava"	The `dropjava` command-line utility removes Java classes from Oracle Database Lite.
Section 13.3.1.1.1, "loadjava"	The `loadjava` command-line utility loads a Java class into Oracle Database Lite.

C.1 The mSQL Tool

Mobile SQL (mSQL) is a GUI-based application that runs on the client device (laptop and Windows CE). It allows the user to execute SQL statements against the local database. It is a development tool that enables users to execute SQL statements to the Oracle Lite database.

Using mSQL you can accomplish the following:

- Create databases
- View tables
- Execute SQL statements

Note: UTF8 SQL Scripts are not supported in mSQL.

The following sections describe how to use the mSQL tool on two platforms:

- Section C.1.1, "The mSQL Tool for Windows 32"
- Section C.1.2, "The mSQL Tool for Windows CE"

C.1.1 The mSQL Tool for Windows 32

On Windows 32 platform, the mSQL tool accesses the database through JDBC. The following sections describe how to use the mSQL command-line to access the database for the Windows 32 platform:

- Section C.1.1.1, "Starting mSQL"
- Section C.1.1.2, "Populating your Database Using mSQL"
- Section C.1.1.3, "SET TERM {ON|OFF}"
- Section C.1.1.4, "SET TIMING {ON|OFF}"
- Section C.1.1.5, "SET VERIFY {ON|OFF}"

C.1.1.1 Starting mSQL

Start mSQL by opening the *`ORACLE_HOME`*`\Mobile\SDK\Bin` directory and double-click the `msql.exe` file. This starts the command-line interface that accepts standard SQL commands.

You can also start mSQL from the command-line, as follows:

```
msql <username>/<password>[@<JDBC_URL>]
```

Where:

- `<username>/<password>`: The client username and password for the Oracle Lite database.
- `<jdbc_url>`: The JDBC URL is optional. If not specified, then the JDBC URL defaults to the URL defined in the `webtogo.ora` file. You can specify the JDBC URL of a single Oracle database or an Oracle RAC database, as follows:
  - The URL for a single Oracle database has the following structure: `jdbc:oracle:thin:@<host>:<port>:<SID>`
  - The JDBC_URL for an Oracle Lite database using embedded JDBC connection is `jdbc:polite:<localDSN>`

- The JDBC_URL for an Oracle Lite database using Type 2 JDBC driver and the Multi-User (MU) service is `jdbc:polite@<host>:<PortNo>:<serverDSN>`, where the Multi-User Oracle Lite database is on the `<host>`, the MU is listening on `<PortNo>` and the DSN is `<ServerDSN>`.

  The JDBC_URL for an Oracle Lite database using Type 4 JDBC driver and the Multi-User service is `jdbc:polite4@<host>:<PortNo>:<serverDSN>`
- The JDBC URL for an Oracle RAC database can have more than one address in it for multiple Oracle databases in the cluster and follows this URL structure:

```
jdbc:oracle:thin:@(DESCRIPTION=
 (ADDRESS_LIST=
   (ADDRESS=(PROTOCOL=TCP)(HOST=PRIMARY_NODE_HOSTNAME)(PORT=1521))
   (ADDRESS=(PROTOCOL=TCP)(HOST=SECONDARY_NODE_HOSTNAME)(PORT=1521))
 )
 (CONNECT_DATA=(SERVICE_NAME=DATABASE_SERVICENAME)))
```

For more information on commands you can execute within the mSQL command, see the *Oracle Database Lite SQL Reference*.

C.1.1.2 Populating your Database Using mSQL

You can use SQL scripts to create tables and schema, and to insert data into tables. A SQL script is a text file, generally with a `.sql` extension, that contains SQL commands. You can run a SQL script from the mSQL prompt, as follows:

```
msql> @<ORACLE_HOME>\DBS\Poldemo.sql
```

You can also type:

```
msql> START <filename>
```

Note: You do not need to include the `.sql` file extension when running the script.

C.1.1.3 SET TERM {ON|OFF}

Controls the display of output generated by commands executed from a script. OFF suppresses the display so that you can spool output from a script without seeing the output on the screen. ON displays the output. TERM OFF does not affect output from commands you enter interactively.

C.1.1.4 SET TIMING {ON|OFF}

Controls the display of timing statistics. ON displays timing statistics on each SQL command. OFF suppresses timing of each command.

C.1.1.5 SET VERIFY {ON|OFF}

Controls whether to list the text of a SQL statement or PL/SQL command before and after replacing substitution variables with values. ON lists the text; OFF suppresses the listing.

C.1.1.6 SET AUTO {ON|OFF}

Controls whether auto-commit is enabled or not.

C.1.1.7 DESC <table_name>

Describes the table columns displaying the column name, type and whether it can be null.

C.1.1.8 DIR

Displays the list of tables and the owners.

C.1.2 The mSQL Tool for Windows CE

The mSQL tool allows the user to execute SQL statements against the local database. You can use either the mSQL tool as a command-line or GUI tool.

The following sections describe the GUI or the command-line tool:

- Section C.1.2.1, "The mSQL GUI Tool"
- Section C.1.2.2, "Manage Snapshots Using mSQL"

C.1.2.1 The mSQL GUI Tool

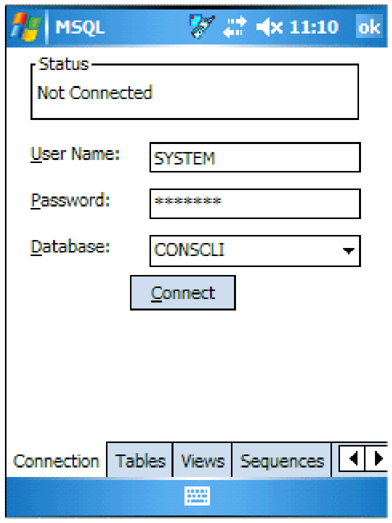
Start the mSQL GUI tool by double-clicking on `msql.exe`. The mSQL GUI tool provides you the ability to perform the following tasks:

- Section C.1.2.1.1, "Connect to the Oracle Lite Database"
- Section C.1.2.1.2, "Execute SQL Statement Against Oracle Lite Database"
- Section C.1.2.1.3, "Create or Encrypt the Oracle Lite Database"
- Section C.1.2.1.4, "Table Contents of the Oracle Lite Database"
- Section C.1.2.1.5, "Views of the Oracle Lite Database"
- Section C.1.2.1.6, "Sequences of the Oracle Lite Database"

C.1.2.1.1 Connect to the Oracle Lite Database Select the Connect tab to connect to the Oracle Lite database on the device.

1. If you have more than one Oracle Lite database on the device, select the appropriate database from the pull-down.
2. Provide the username and password for this database.
3. Click **Connect**.

Figure C–1 Connect to the Oracle Lite Database



Status
Not Connected

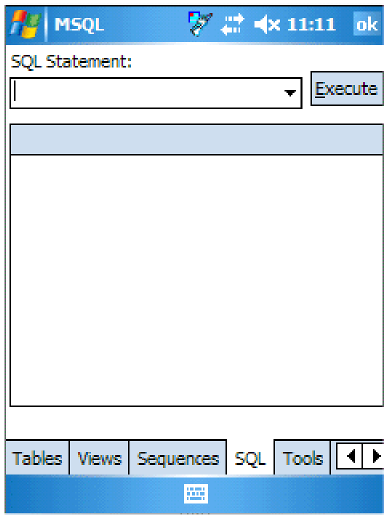
User Name: SYSTEM

Password: ********

Database: CONSCLI

C.1.2.1.2 Execute SQL Statement Against Oracle Lite Database Select the SQL tab to execute a SQL statement against the Oracle Lite database. After connecting, enter your SQL statement and click **Execute**. The statement and any results are displayed in the bottom window. For more information, see the *Oracle Database Lite SQL Reference*.

Figure C–2 Execute a SQL Statement



C.1.2.1.3 Create or Encrypt the Oracle Lite Database The Tools tab enables you to create an Oracle Lite database or to encrypt or validate and existing database.

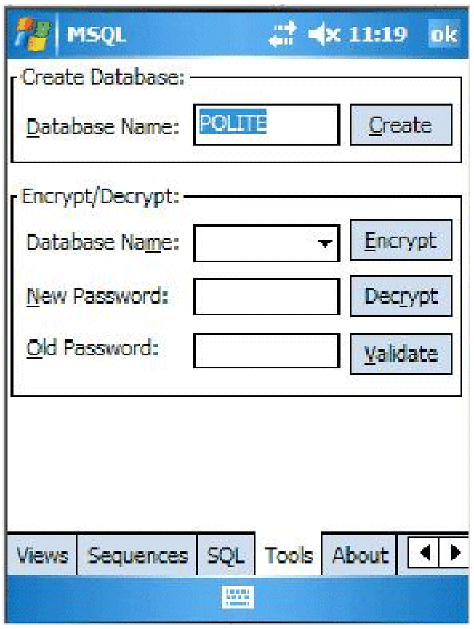
- Create Database: You can create an Oracle Lite database to use embedded within a standalone application.
- Encrypt/Decrypt/Validate: Select the Oracle Lite database that you want to execute the EncrypDB, DecrypDB or ValidateDB commands against. The password that you provide is the user password. Click on the appropriate button for each of these functions.

  See the following sections for details on EncryptDB, DecryptDB or ValidateDB:

  – Section C.5, "DECRYPDB"

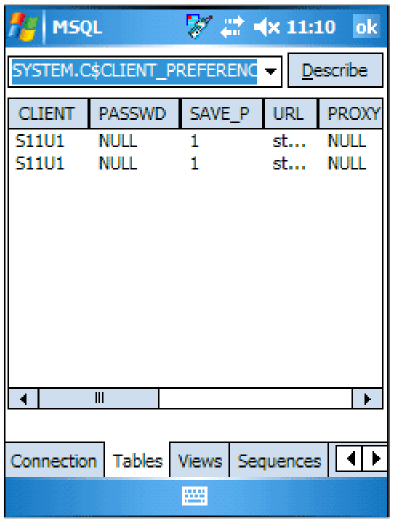
– Section C.6, "BACKUPDB"
– Section C.10, "VALIDATEDB"

Figure C–3 Create, Encrypt, Decrypt, or Validate the Oracle Lite Database



C.1.2.1.4 Table Contents of the Oracle Lite Database When you select the Tables tab, you can select any of the tables in the Oracle Lite database and click **Describe**. The structure and contents of this table is displayed.

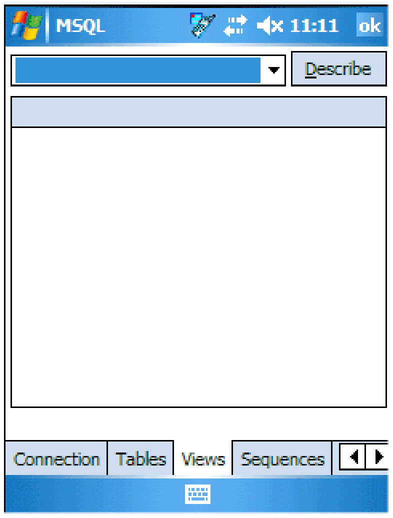
Figure C–4 Table Description for Oracle Lite Database



CLIENT	PASSWD	SAVE_P	URL	PROXY
S11U1	NULL	1	st...	NULL
S11U1	NULL	1	st...	NULL

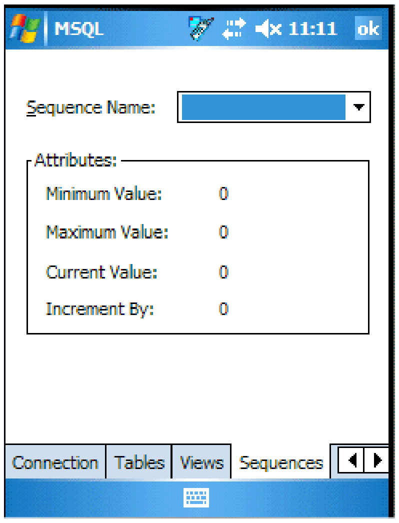
C.1.2.1.5 Views of the Oracle Lite Database When you select the Views tab, you can select any of the views in the Oracle Lite database and click **Describe**. The view definition is displayed.

Figure C–5 Show Views of Oracle Lite Database



C.1.2.1.6 Sequences of the Oracle Lite Database When you select the Sequences tab, you can select any of the sequences in the Oracle Lite database and click **Describe**. The sequence definition is displayed.

Figure C–6 Display Sequence Definition of Oracle Database Lite



Sequence Name:

Attributes:

Minimum Value:	0
Maximum Value:	0
Current Value:	0
Increment By:	0

C.1.2.2 Manage Snapshots Using mSQL

The Oracle Lite database format is the same on Windows 32 and Windows CE platforms. Manage your snapshots, as follows:

1. Create and test your snapshots on Windows 32 using the Windows 32 mSQL command-line utility.
2. Copy the database to the Windows CE platform.
3. Use the Windows CE mSQL tool to manipulate the database that is on your device.

The mSQL tool enables the user to execute SQL statements against the local database and access functionality provided by the interfaces of the underlying Oracle Lite database engine.

C.2 CREATEDB

Description

Utility for creating a database.

Syntax

```
CREATEDB DataSourceName DatabaseName Database_SysUser_Password [[[VolID] DATABASE_
SIZE] EXTENT_SIZE] [collation sequence]
```

Keywords and Parameters

`DataSourceName`

Data source name, used to look up the `ODBC.INI` file for the default database directory.

> **Note:** If you specify an invalid DSN, Oracle Database Lite ignores the DSN and creates the database in the current directory. To access this database through ODBC, you must create a DSN for the database that points to the directory in which the database resides. For instructions on adding a DSN, see Section C.8.1, "Adding a DSN Using the ODBC Administrator".

`DatabaseName`

Name of the database to be created. It can be a full path name or just the database name. If only the database name is given, the database is created under the Data Directory for the data source name specified in the `ODBC.INI` file. The extension for the database name must always be `.ODB`. If a name without the `.ODB` is given, the `.ODB` is appended.

`DATABASE_SysUser_Password`

The database system user password.

`VolID`

When specified, the `VolID` is used as the database ID, instead of the database ID from the `POLITE.INI` file. The ID must be unique for each database. If you specify a volumn id, then you also specify the database and extent sizes. Thus, the createdb executable knows that the volume id, database size and extent size are being specified when three numbers are provided in a row.

> **Note:** For the volume id, database size, and extent size, specify only the number; do not specify name=value. See the examples for more information.

`DATABASE_SIZE`

The database size in bytes. If you want to specify the database size, then you also must specify the volume id and extent size.

`EXTENT_SIZE`

An incremental amount of pages in a database file. When a database runs out of pages in the current file, it extends the file by this number of pages. If you want to specify the extent size, then you also must specify the volume id and database size.

`COLLATION_SEQUENCE`

This parameter is a string constant which creates the database as enabled for linguistic sorting when a value other than the default is used. A collation sequence specified here overrides a collation sequence set using the `NLS_SORT [collation_sequence]` parameter in the `POLITE.INI` file. The string can also be one of the options listed in Table C–2:

Table C–2 Collation Sequence Values

Collation Sequence	Description
`BINARY`	Default. Two strings are compared character by character and the characters are compared using their binary code value. You cannot perform a linguistic sort with an Oracle Lite database that has a binary collation sequence.
`FRENCH`	Two strings are compared according to the collation sequence of French. Supported by ISO 8859-1 or IBM-1252.
`GERMAN`	Two strings are compared according to the collation sequence of German. Supported by ISO 8859-1 or IBM-1252.
`CZECH`	Two strings are compared according to the collation sequence of Czech. Supported by ISO 8859-2 or IBM-1250.
`XCZECH`	Two strings are compared according to the collation sequence of Xczech. Supported by ISO 8859-2 or IBM-1250.

Note: There is no way to alter a collation sequence after the database is created.

Examples

Create the `db1` database with DSN of `polite` and password `manager`: `createdb polite db1 manager`

Create the `db2.odb` database with DSN `polite` and password `manager300`: `createdb polite c:\testdir\db2.odb manager300`

Create `polite` database with DSN polite, password of `manager`, and a collation sequence of `french`: `createdb polite polite manager french`

Create polite database with DSN `polite`, password `manager`, volume id of 199, database size of 1000, and extent size of 1:
`createdb polite polite manager 199 1000 1`

C.3 REMOVEDB

Description

Utility for deleting a database.

Syntax

REMOVEDB DataSourceName Database Name

Keywords and Parameters

DataSourceName

Data source name of the database you want to remove. The DSN can be a dummy argument such as none, in which case the database name must be a fully qualified filename.

DatabaseName

The name of the database to delete. It can be a full path name or just the database name. If only the database name is given, the database is deleted from the Data Directory for the data source name specified in the `ODBC.INI` file.

Examples

```
removedb polite db1
```

```
removedb none c:\testdir\db2.odb
```

C.4 ENCRYPDB

Description

Enables you to encrypt Oracle Database Lite with a password, which prevents unauthorized access to the database and encrypts the database, so that the data stored in the database files cannot be interpreted. To decrypt the database, see Section C.5, "DECRYPDB".

This tool is used by embedded applications to encrypt the database used by the application. You provide the user password for the encryption.

This is more difficult on a handheld as it is sometimes difficult for users to find the RUN option in order to execute the command with arguments.

`ENCRYPDB` uses AES-128 encryption.

Syntax

`ENCRYPDB` *DSN* | `NONE` *DBName* [*New_Password* [*Old_Password*]]

Keywords and Parameters

- `DSN`—Data Source Name of Oracle Database Lite that you want to encrypt. If you specify `NONE`, `DBName` must be a fully qualified database name with the full path name (without the `.ODB` extension). If the `DSN` is a value other than `NONE`, then the name must appear as a data source name in the `ODBC.INI` file.
- `DBName`—Name of the database to be encrypted. If DSN was specified as `NONE`, `DBName` must be entered with the full path name.
- `New_Password and Old_Password`—Optional, the password (or previously used password) for encrypting the database. This password can be 128 characters in length. If you do not enter a password, `ENCRYPDB` prompts you to enter one. Since both passwords are optional in the command line to invoke the utility, the command line could have three different forms:
  - No password given: If the database is already encrypted, then `ENCRYPDB` assumes that the user is trying to change the password of the database. It prompts the user for the old password once and new password twice, and encrypts the database using the new password. If the database is not already

encrypted, ENCRYPDB prompts for the new password twice and encrypts the database using this new password.

- One password given: This password is assumed to be the new password. If the database is already encrypted, ENCRYPDB prompts for the old password and encrypts the database using the new password.
- Both passwords given: ENCRYPDB assumes that the first password is the new password and the second is the old password.

To run from the command line, you must pass in the DSN name and the database name. The following encrypts the `employee` database with DSN of `Employee` with the `test` password:

```
Encrypdb Employee employee test test
```

Comments

If you call this utility from another program, the possible values returned are listed in Table C–3:

Table C–3 ENCRYPDB Return Codes

Return Code	Description
EXIT_SUCCESS	Success
EXIT_USAGE	Command line arguments are not properly used or are in error
EXIT_PATH_TOO_LONG	Path is too long
EXIT_SYSCALL	I/O error while making new encrypted copy on disk
EXIT_BAD_PASSWD	Incorrect password supplied

The default Oracle Database Lite (POLITE.ODB) is not encrypted. After encrypting an Oracle Database Lite, every user that attempts to establish a connection to the encrypted Oracle Database Lite must provide the valid password. If the password is not provided, Oracle Database Lite returns an error. An Oracle Database Lite database cannot be encrypted if there are any open connections to the database.

You should consider the following when encrypting and decrypting Oracle Database Lite:

- You cannot decrypt an encrypted database without the password. Make sure you back up your database in a secure place before you encrypt it. Another user of the same database can create a copy with a new user name for a user who loses their password, otherwise, there is no method to recover a database where the passwords are lost.
- A password encrypts the entire database. It is not a user-specific password.
- Database encryption does not prevent a third party from removing an Oracle Lite Database. That is, `removedb` and `rmdb` remove a database without checking the password. Use tools that protect unauthorized users from manipulating your file system.
- ODBC applications that connect to an encrypted Oracle Database Lite database need to specify a valid password. It is customary to prompt for the password at runtime rather than to code it in the application. Most ODBC applications can use the `SQLDriverConnect` function with the `DRIVER=` option, rather than the `SQLConnect` function, if the applications require the Oracle Database Lite ODBC driver to prompt for the password at runtime.

- All sample applications provided with this release of Oracle Database Lite are designed to run against a database that is not encrypted.
- You can use `DECRYPDB` and `ENCRYPDB` (in this order) to change the password of a database. However, `DECRYPDB` creates an Oracle Database Lite database in plain text before `ENCRYPDB` encrypts it. This results in a database in plain text form, for a short period of time, and is not recommended.
- For encrypted databases, all user names and passwords are written to a file named `DSN.OPW`. Each user can then use the password as a "key" to unlock the `.OPW` file before the `.ODB` file is accessed. When you copy or back up the database, you should include the `.OPW` file.

C.5 DECRYPDB

Description

This tool allows you to decrypt an encrypted Oracle Lite database used with an embedded application. For more information, see Section C.6, "BACKUPDB".

This tool is used by embedded applications to decrypt the database used by the application. To encrypt an Oracle Lite database used by a client, see the `ENCRYPDB` executable in the Section C.4, "ENCRYPDB".

SYNTAX

DECRYPDB *DSN* | NONE *DBName* [*Password*]

Keywords and Parameters

`DSN`

Data Source Name of Oracle Database Lite that you want to decrypt. If you specify `NONE`, you must the enter the `DBName` with the full path name (without the `.ODB` extension).

`DBName`

Name of the database to be decrypted. If DSN was specified as `NONE`, the `DBName` must be entered with the full path name.

`Password`

Optional. The password used previously to encrypt Oracle Database Lite. If you do not enter the password, `DECRYPDB` prompts you to enter it.

Comments

An Oracle Database Lite database cannot be decrypted if there is any open connection to the database.

If you call this utility from another program, the possible values returned are listed in Table C–4:

Table C–4 DECRYPDB Return Codes

Return Code	Description
`EXIT_SUCCESS`	Success
`EXIT_USAGE`	Command line arguments are not properly used or are in error
`EXIT_PATH_TOO_LONG`	Path is too long

Table C–4 (Cont.) DECRYPTDB Return Codes

Return Code	Description
EXIT_SYSCALL	I/O error while making new decrypted copy on disk
EXIT_BAD_PASSWD	Incorrect password supplied

For more information, see the comments in Section C.6, "BACKUPDB".

C.6 BACKUPDB

Description

You can back up the Oracle Lite database either by using the `backupdb` utility or by copying the files to another location.

Oracle Database Lite uses the ODB and OBS files with dependent log files that can be backed up by copying to another location. Before any files can be copied, disconnect all applications that access the database and shut down the multi-user service, if running. Once that has been accomplished, execute the `backupdb` utility, which copies the `*.odb`, `*.obs`, and `*.opw` files to the filename of your choice to make a backup of the database.

```
BACKUPDB DSN|NONE DBName backup_filename [DB_password]
```

If you want to restore the backup, then execute the `backupdb` executable, with `NONE` and reversing the `filename` and the `dbname`, as follows:

```
BACKUPDB NONE backup_filename DBName
```

This is more difficult on a handheld as it is sometimes difficult for users to find the RUN option in order to execute the command with arguments.

Syntax

```
BACKUPDB DSN | NONE DBName <backup_filename> [<DB_password>]]
```

Keywords and Parameters

- `DSN`—Data Source Name of Oracle Database Lite that you want to backup. If you specify `NONE`, `DBName` must be a fully qualified database name with the full path name (without the `.ODB` extension).
- `DBName`—Name of the database to be encrypted. If DSN was specified as `NONE`, `DBName` must be entered with the full path name.
- `backup_filename`—File where you want the backup to be stored. This can include an absolute or relative path. If no path is included, then the file is stored in the directory where the command is executed.
- `DB_password`—The password you used for encrypting the database. This is only required if the database you are backing up is encrypted. You can either supply this password on the command line or when prompted during execution.

To run from the command line, you must pass in the DSN name and the database name. The following backs up the `employee` database with DSN of `Employee` into the `backupemployee` file:

```
Backupdb Employee employee backupemployee
```

C.7 DefragDB to Defragment and Reduce Size of the Oracle Lite Database

On each client device, an Oracle Lite database stores the application data. You can optimize the Oracle Lite database with the DefragDB utility, as follows:

- Reduce size of Oracle Lite databases by defragmenting the Oracle Lite database.
- Remove any BLOB data from the Oracle Lite database. All BLOB data—both binary and character— and indexes are stored in separate files with the extension of `.obs` for Oracle Blob Store. This changes the size limit on your device to either the operating system file size limitations or 16 terabytes.

Note: This tool removes any Blob data currently in your Oracle Lite database and stores it in its own `.obs` file. However, if you do not run this tool, you can continue to work seamlessly. Any new Blob data is stored in an `.obs` file; any pre-existing Blob data can continue to reside in the `.odb` file.

Use the DefragDB tool to defragment Oracle Lite databases, which reduces their size by compacting them and removing any Blob data from within the database into its own `.obs` file. The DefragDB tool is a UI dialog which allows you to choose which databases to defragment. This tool defragments databases by dumping each database into a file and then reloading it from this file. Alternatively, you can use the command-line interface: `olmig.exe`. Both tools exist in the `<ORACLE_HOME>/Mobile/Sdk/bin` on your desktop or in `\OraCE` on a WinCE device.

Note: Currently the tool runs on Win32 desktop and Windows CE devices.

The following sections describe this tool:

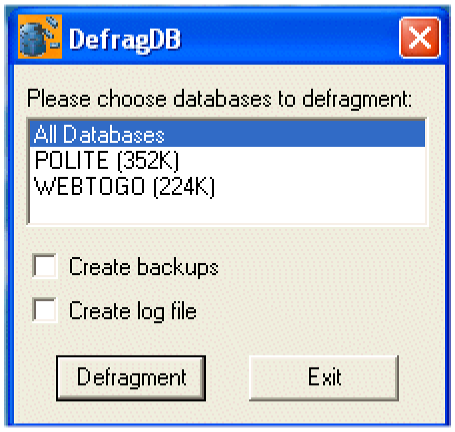
- Section C.7.1, "Execute DefragDB"
- Section C.7.2, "Pause or Cancel Defragmentation"
- Section C.7.3, "Execute DefragDB With Command-Line"

C.7.1 Execute DefragDB

To start the tool, either double-click on the Oracle Database Lite Degramentation icon or execute `DefragDB.exe`, which brings up the following screen:

You can execute the Oracle Database Lite Defragmentation tool on the client for all applications. Before executing this tool, you must stop ALL applications, as the database is erased during this process.

Figure C–7 DefragDB GUI



All application databases are listed on this screen. Select the existing databases on your PC (or WinCE device) on which you want to perfrom the deframentation.

- Click **Defragment** to defragment all databases. This tool performs a defragmentation on one database at a time.

> **Note:** In the worse case scenario, the defragmentation process requires three times the space of the database to complete the process. Thus, if you do not have enough space to defragment your larger databases, you will receive a warning notice about the database that is too large to complete the process. In order to continue, either free up enough space to enable the process to complete or return to the main screen and select all databases except for the offending one.

- To defragment specific databases, select the databases desired from the list and click **Defragment**.

> **Note:** To cancel out without performing any defragmentation, click **Cancel**. See Section C.7.2, "Pause or Cancel Defragmentation" for more information.

In addition, select the following checkboxes, as appropriate:

- Create backup: provides a backup of the original copy of the database before defragmentation. The backup copy has the same name with a `.bak` extension. For example, `C:\orant\oldb40\polite.odb` becomes `C:\orant\oldb40\polite.odb.bak`. Thus, you can restore the database if an error occurs during defragmentation. In addition, the blob storage file (`.obs`) is backed up. To restore to the original version, rename these files back to the original names without the `.bak` extensions.
- Create log: provides a log, `defragdb.log`, of the defragmentation process, which is useful for the developer in diagnosing any problems that may occur during the defragmentation.

When you click **Defragment**, the process initiates, which brings up a window that displays messages about the progress of the defragmentation. This same log can be saved in the `defradb.log` file. See the status bar at the bottom for the final status: defragmenting, success, or fail.

If your database is encrypted, another dialog prompts for the user name and password for the encrypted database. Select the user name from the list of users and enter the password for that user. You only need to enter the password once if all databases are encrypted with the same password.

There are 2 files created during the defrag which should be deleted automatically by DefragDB when it finishes:

- The dump file, which has the same name as the database, but with a `.dmp` extension.
- The newly loaded database, which has the same name as the database, but witha `_defrag` suffix. This file is renamed to the database name once the load completes.

C.7.2 Pause or Cancel Defragmentation

If you are defragmenting large databases, this may take some time. For example, it takes about 5 minutes on a desktop to defragment a 200 MB database; however, on WinCE devices, the defragmentation performs slower than the desktop machine. The amount of time this process takes is proportional to the size of the database. Thus, if you need to pause or cancel this process by clicking the **Cancel** button. This pauses the process. To continue the defragmentation, select **No** on the Cancel/Continue prompt. Select **Yes** on this prompt to stop the process entirely. If you cancel, the database remains in the original state, so your applications still perform normally.

C.7.3 Execute DefragDB With Command-Line

You can use automatic defragmentation, execute defragmentation within your application, or use additional options only available with the command-line tool. The following shows two examples: the first deframents all databases, the second defragments a specific database identified by name.

```
olmig -defrag all
or
olmig -defrag dbname
```

The usage for `olmig.exe` is as follows:

```
olmig -dump|-load|-defrag|-restore <dbName>|all [options]
```

Where:

- `-dump` : dump the database to a dump file (default `dbName.odb.dmp`). You can separate the functionality of the defragmentation into the dumping and loading. You can perform these functions at separate times, if desired.
- `-load` : load database from the dump file. This completes the defragmentation process and should only be executed after a dump is performed.
- `-defrag` : defragment database, which performs both the dump and load for the database.
- `-restore` : restore a database from a backup. If an error occurred, restore the saved original database.
- `<dbName>` or `all` : perform the actions on a specfic database or on all Oracle Lite databases on the machine or device.

Options include:

- `-auto` : exit the dialog when upgrade is done. When you invoke the command-line, a GUI is initiated. If you want this screen to exit when finished, provide the `-auto` option.
- `-backup` : backup the database to `dbName.odb.bak`
- `-readonly` : connect to a database that is read-only. During a dump only, you can dump a read-only database, such as one that may exist on a CD-ROM. Since the dump normally is written to the same location as the database, you must also provide the directory location and filename for the output. Thus, this option is only valid if used in combination with the `-dump` and `-file` options.
- `-nosingle` : do not enter single-user mode. Normally, only a single user can connect to the database while performing a dump, load, or defragmentation. That way, other users are not allowed to update an Oracle Lite database that is currently in the middle of a defragmentation, dump, or load activity. However, if you are performing a dump, you can use this option to enable other users to continue to execute their applications against the database. This cannot be allowed during any load activity.
- `-log logfile` : append messages to the specified file
- `-file dumpfile` : use the specified dump file, instead of the default file
- `-dot interval` : print a dot after processing the number of objects designated by `interval`; set the `interval` to 0 to disable this option.
- `-commit interval` : during the load process, this designates the number of rows after which to perform a commit; set the `interval` to 0 for no commit.
- `-passwd passwd` : specify a connect password for encrypted databases

C.8 ODBC Administrator and the Oracle Database Lite ODBC Driver

A Data Source Name (DSN) associates the Oracle Database Lite ODBC Driver with the Oracle Database Lite database that you want to access through the driver. The Oracle Database Lite installation process creates a default `DSN`, `POLITE`, for the Oracle Database Lite database. You can also create additional DSNs for the additional Oracle Database Lite databases that you create.

Microsoft provides the ODBC Administrator, a tool for managing the `ODBC.INI` file and associated registry entries in `Windows 2003/XP`. The `ODBC.INI` file and the Windows registry store the DSN entries captured through the ODBC Administrator. Using the ODBC Administrator, you can relate a DSN to the Oracle Database Lite ODBC Driver.

Note: This document does not provide instructions on using the ODBC Administrator. See the ODBC Administrator tool online help for this information.

In the ODBC Administrator, in addition to the DSN, you must specify the parameters listed in Table C–5:

Table C–5 ODBC Administrator DSN Parameters

DSN Parameter	Description
`Data Description`	An optional description for the data source.

Table C–5 (Cont.) ODBC Administrator DSN Parameters

DSN Parameter	Description
`Database Directory`	The path to the data directory where the database resides. This is an existing path.
`Database`	Oracle Database Lite database name to be created. Do not include the `.ODB` extension.
`Default Isolation Level`	Determines the degree to which operations in different transactions are visible to each other. For more information on the supported isolation levels, refer the *Oracle Database Lite Developer's Guide*. The default level is "`Read Committed`".
`Autocommit`	Commits every database update operation in a transaction when that operation is performed. Autocommit values are Off and On. The default value is Off. **Note**: In the Microsoft ODBC SDK, the ODBC driver defaults to auto-commit mode. However, the default for Oracle Database Lite is manual-commit mode. In this environment, if you execute SQLEndTrans / SQLTransact call with SQL_COMMIT option using the ODBC driver, you receive a SQL_SUCCESS, because ODBC believes that auto-commit is on. However, no commit actually occurs, because ODBC transfers the transaction to Oracle Database Lite, whose default is manual-commit. You must configure the Microsoft ODBC Driver Manager to transfer control of the SQLEndTrans / SQLTransact API call to Oracle Database Lite by explicitly setting autocommit to OFF in ODBC. When you do this, ODBC does not try to autocommit, but gives control of the transaction to Oracle Database Lite. To set auto-commit to off, execute either the SQLSetConnectAtrr or SQLSetConnectOption method with SQL_AUTOCOMMIT_OFF as the value of the SQL_AUTOCOMMIT option. Then, the SQLEndTrans / SQLTransact calls will commit as defaulted within Oracle Database Lite. Thus, if you want auto-commit on, turn it on only within Oracle Database Lite.
`Default Cursor Type`	■ *Forward Only*: Default. A non-scrollable cursor which only moves forward but not backward through the result set. As a result, the cursor cannot go back to previously fetched rows. ■ *Dynamic*: Capable of detecting changes to the membership, order, or values of a result set after the cursor is opened. If a dynamic cursor fetches rows that are subsequently deleted or updated by another application, it detects those changes when it fetches those rows again. ■ *Keyset Driven*: Does not detect change to the membership or order of a result set, but detects changes to the values of rows in the result set. ■ *Static*: Does not detect changes to the membership, order or values of a result set after the cursor is opened. If a static cursor fetches a row that is subsequently updated by another application, it does not detect the changes even if it fetches the row again.

For example, the DSN entry for `POLITE` in the `ODBC.INI` file may contain:

```
[POLITE]
Description=Oracle Lite Data Source
Data_Directory=C:\ORANT\OLDB40
Database=POLITE
IsolationLevel=Repeatable Read
CursorType=Dynamic
```

Note: The `ODBC.INI` file is available in Windows under `%WINDIR%` and in Linux under `$OLITE_HOME/bin`. For the Linux platform, you must have write permissions on the directory where this is located to be able to modify them.

C.8.1 Adding a DSN Using the ODBC Administrator

To add a DSN using the ODBC Administrator:

1. Start the ODBC Administrator, either by selecting its icon in the Oracle Database Lite program group, or by typing the following at a DOS prompt:

   ```
   C:\>ODBCAD32
   ```

2. Click **Add**.
3. Double-click the Oracle Database Lite ***nn*** **ODBC Driver**, where *nn* is the release number, from the list of Installed ODBC Drivers.
4. Next, add the DSN name and define the parameters in the ODBC driver setup dialog. Refer the preceding table for help in defining the parameters.

C.8.2 Adding a DSN which points to Read-Only Media (CD-ROM)

1. Create the DSN as explained in Section C.8.1, "Adding a DSN Using the ODBC Administrator".
2. Add the following line to the new DSN in the `ODBC.INI` file:

   ```
   ReadOnly = True
   ```

Note: You can define a DSN which points to a file on a CD-ROM. Simply point the DSN to the CD-ROM drive and directory and provide the file name of the database file. Then modify the `ODBC.INI` file to add the line `ReadOnly=True` to the data source definition. ODBC programmers can call the following before opening the database to enable this feature (instead of adding the line to the `ODBC.INI` file):

```
SQLSetConnectOption( hdbc, SQL_ACCESS_MODE, SQL_
MODE_READ_ONLY )
```

Setting a database file to read-only suppresses the creation of log files. Updates, insertions, deletions, or commits appear to work on the in-memory image of tables. However, when you commit, these changes are not written to the database file. If you exit your application, reconnect, and issue your query, you see your original data.

C.9 ODBINFO

Description

You can use `ODBINFO` to find out the version number and volume ID of an Oracle Database Lite database. `ODBINFO` can also display and set several parameters.

Syntax

To display current information without making any changes use the syntax:

```
odbinfo [-p passwd]DSN DBName
```

You can also use:

```
odbinfo [-p passwd] NONE dbpath\dbanme.odb
```

For example:

```
odbinfo -p tiger polite polite
```

```
odbinfo NONE c:\orant\oldb40\polite.odb
```

If your database is encrypted you need to include the password.

Parameters

To set or clear parameters, use one or more "+" or "-" parameter arguments before the DSN or NONE. For example:

```
odbinfo +reuseoid -pagelog -fsync polite polite
```

You can use the parameters listed in Table C–6 with the `ODBINFO` utility:

Table C–6 ODBINFO Parameters

Parameter	Description
`pagelog`	By default, a commit backs up modified database pages to `filename.plg` before actually writing the changes to `filename.odb`. If an application or the operating system experiences a failure during a commit, the transaction is cleanly rolled back during the next connect. If `-pagelog` is specified, no backup is created and the database can become corrupted if a failure occurs.
`fsync`	Oracle Database Lite generally forces the operating system to write all the modified buffers associated with the database back to disk during a commit. If this option is disabled (`-fsync`), the operating system can keep the changes in memory until a later time. If the system (but not the application) crashes before the buffers are flushed, the database can become corrupted. Using `odbinfo -fsync -pagelog` improves the performance of applications that use many small transactions (with autocommit on) or ones with massive updates. However, if the database is corrupted, there is no straightforward way to repair it or recover the data. Therefore these two options should only be cleared during initial loading of the database, if (1) the `.ODB` file is backed up on regular basis, or (2) the data in the database can be recovered from some other source. Using this option has no effect on applications that seldom update the database. Setting the transaction isolation level to `SINGLE USER` has more impact in this case.

Table C–6 (Cont.) ODBINFO Parameters

Parameter	Description
`reuseoid`	By default, Oracle Database Lite does not reuse the `ROWID` of any row that exists in a table until the table is dropped. The "Slot Deleted" error is returned when accessing a deleted object. This uses two bytes of storage for each deleted object, causing performance and disk space usage to degrade over time if rows are constantly inserted and deleted. If you use `odbinfo +reuseoid`, new rows can reuse `ROWIDs` of previously deleted rows. However, this may not free all the space in a table that already has many deleted objects. For best results, you should set this option immediately after you create your database. This option is safe for pure relational applications. However, SQL applications that use `ROWID` and `OKAPI` applications that use direct pointers between objects need to verify that all references to an object are set to `NULL` before the object is deleted. Otherwise, dangling references may eventually point to some other, unrelated object.
`compress`	This option (which is "on" by default) enables run-length compression of objects. Run-length compression takes very little CPU time, so you should only deselect (`-compress`) this option if: ■ Operating system-level file compression is used, such as `DriveSpace` or a `NTFS` compressed attribute. In this case not compressing the same data twice provides a better compression ratio. ■ Most objects in the database are frequently updated to a highly compressible state (for example, all columns set to `NULL`), and the data cannot be compressed well (such as binary columns with random data). In these cases, using this option (`+compress`) can result in highly fragmented tables. Changing this option does not compress or decompress any existing objects in the database.

C.10 VALIDATEDB

This command-line tool validates the structures within the database file and if the database structure is found to be corrupted, lists the errors found in a file designated by the user. The tool checks the following:

- Objects - Header information for database objects. Flags are checked for consistency in case the object was moved or compressed. Object length is checked against a valid range. If the object is a BLOB, the object's frames are checked against the volume page bitmap.
- Index page entries - Checks that the creation of an index page entry results in the correct number of nodes or list of object identifiers.
- Index pages - Checks that all key values on the page are sorted. All objects contained on the page are validated. Page descriptor information such as the number of objects, the number of free bytes, and the number of entries are checked against the actual objects on the page.
- Groups - As each page is validated, the group descriptor information is checked against the actual number of pages and objects.

- Indexes - All the pages are validated against the btree. The tool also validates all page pointers. All levels of the btree are checked to validate that key values are in the sorted order as a whole. For leaf elements of the btree, all OIDs from the leaf page entries are checked for consistency with the actual group objects.

Syntax

validatedb *DSName DBName* [-p *password*] [-l username:password] [-t *schemaname.tablename] -file outputfilename*

Keywords and Parameters

DSName

The data source name. This can also be `NONE` if no DSN is present.

DBName

If there is a DSN present, this is the database file name (without the .odb extension) if it is different from the default filename for the DSN. If there is no DSN, then `VALIDATEDB` uses the current directory unless the full path is specified. If there is a log file in the same directory as the database file, it is also validated.

-p password

Password for an encrypted database.

-l username:password

Optional. Provide the username/password to log into the Oracle Lite database that you are validating with the `-l username:password` option.

Note: This is not available on Windows CE.

The following details the behavior of this option:

1. If the database is encrypted, and the encryption password (the `-p` option) is not supplied, then the password included in the login (`-l`) option is used as the encryption password.
2. If you do supply an encryption password in the `-p` option as well as a login password with the `-l` option, then the login password is used to verify that the encryption password is correct.

-t schemaname:tablename

Optional.

- schema name. The default schema name is used unless this is specified.
- table name. The specified table is validated along with all of its indexes. If no table name is specified, the entire database is validated.

-file outputfilename

Optional filename for the text file where all errors and other related information revealed by `VALIDATEDB` are saved. The default is stdout.

Examples

```
validatedb polite polite -t emp -file out.txt
```

C.11 Transferring Data Between a Database and an External File

You can transfer data between an external file and the Oracle Lite database through either a command-line tool or programmatic APIs, as described in the following sections:

- Section C.11.1, "OLLOAD"
- Section C.11.2, "Oracle Database Lite Load Application Programming Interfaces (APIs)"

C.11.1 OLLOAD

The Oracle Database Lite Load Utility (`OLLOAD`) is a command-line tool, which enables you to load data from an external file into a table in Oracle Database Lite or to unload (dump) data from a table in Oracle Database Lite to an external file. Unlike SQL*Loader, `OLLOAD` does not use a control file in which you supply all data parameters and format information on the command-line.

When loading data, `OLLOAD` takes an input file that contains one record per line with a separator character between fields. The default field separator is a comma (,). These records can also include fields with values that are quoted strings. The default value is single quote (′). For more information on data parsing, see "Data Parsing".

Before executing this tool, you must stop all applications. This includes the Oracle Database Lite applications, such as the Sync Agent, DM agent, and so on. To stop the Sync Agent, see Section 5.4.2, "Start, Stop, or Get Status for Automatic Synchronization" in the *Oracle Database Lite Administration and Deployment Guide*.

C.11.1.1 Syntax

Loading a Datafile

To load a datafile, use the following syntax.

olload [options] -load *dbpath tbl* [*col1 col2 ...*] [<*datafile*]

Unloading (dump) to an Outfile

olload [options] -dump *dbpath tbl* [col1 col2 ...] [>*outfile*]

C.11.1.2 Keywords and Parameters

This section describes keywords and parameters that are available for the `OLLOAD` utility.

[options]

For a list of options, see Section C.11.1.2.1, "Options".

-load

To use the load utility.

-dump

To use the unload (dump) utility.

dbpath

The path to the Oracle Database Lite (`.odb`) file.

tbl

The table name. `OLLOAD` first attempts to find a table name in the user-specified case. If this fails, it searches for the upper-case of the user-specified name.

Note: The default user is `SYSTEM`. To specify an `OLLOAD` operation for another user name's tables, prefix the `tbl` parameter with the user name and a dot (.).

---

col1 col2

The column names. `OLLOAD` first attempts to find a column name in the user-specified case. If this fails, it searches for the upper-case of the user-specified name.

[datafile] [outfile]

The source or destination file for the load or unload operations. If you do not specify a datafile or outfile, `OLLOAD` displays the output on the screen.

C.11.1.2.1 Options This section describes keyword and parameter options that are available for the `OLLOAD` utility.

-sep *character*

The field separator. If you do not specify this option, `OLLOAD` assumes that the separator character is a comma (,).

-quote *character*

The quote character. If you do not specify this option, `OLLOAD` assumes that the quote character is a single quote (’).

-file *filename*

Use this option when loading and unloading data to specify the source or destination file name. When loading data, filename specifies the source file to load into Oracle Database Lite. When unloading (dumping) data, it is the destination file for the unloaded data.

Note: To unload data from Oracle Database Lite and load (or pipe) it to another Oracle Database Lite, do not specify a file name for this option. For a description of sample syntax, see "Examples".

-log *logfile*

Specify this option if you want to produce a log file listing rows that OLLOAD could not insert during load. If you do not specify a log file, loading stops at the first error.

-passwd *passwd*

The connection password for an encrypted database. You need to supply this password so that loading and unloading can occur.

-nosingle

Specify this option when you do not want to use single user mode. This degrades performance but allows other connections to the database.

-readonly
Specify this option when unloading data from a read-only Oracle Database Lite, for example, one located on a CD-ROM.

-commit *count*
Use this option if you want OLLOAD to commit after processing a specified number of rows. The default is 10000. OLLOAD prints an asterisk (*) to the screen each time it commits the specified number of rows. To disable the commit operation specify 0.

-mark *count*
Use this option if you want OLLOAD to print a dot on the screen after processing the specified number of records. The default is 1000. To disable this feature specify 0.

Data Parsing
Table C–7 lists examples for `OLLOAD` data parsing.

Table C–7 Data Parsing Examples

Input	Data	Explanation
'Redwood Shores, CA'	Redwood Shores CA	Enclosing the input string in quotes preserves spaces and punctuations within a string.
'O''Brien'	O'Brien	Represent a single quote with its escape sequence, two single quotes.
fire fly	firefly	Spaces in data that is not quoted is ignored.
,	NULL,NULL	Empty fields are NULL.
1,,3	1,NULL,3,NULL	Empty fields are NULL.
	[no row inserted]	Completely empty lines are ignored.

If there are more values than database columns, extra values are ignored. Any missing values at the end of the line are set to NULL.

OLLOAD Utility Restrictions
`OLLOAD` does not support tab-delimited input files and LONG datatypes.

Examples
```
olload -quote \" -file p_kakaku.csv -load c:\orant\oldb40\polite.odb skkm01
olload -dump c:\orant\oldb40\polite.odb emp empno ename | olload -load myfile.odb
myemp
```

C.11.2 Oracle Database Lite Load Application Programming Interfaces (APIs)
This document describes the Oracle Database Lite Load APIs. Each section of this document presents a different topic. These topics include:

- Section C.11.2.1, "Overview"
- Section C.11.2.2, "Oracle Database Lite Load APIs"
- Section C.11.2.3, "File Format"
- Section C.11.2.4, "Limitations"

C.11.2.1 Overview

The Oracle Database Lite Load APIs allow you to load data from an external file into a table in Oracle Database Lite, or to unload (dump) data from a table in Oracle Database Lite to an external file. For information on using the command line tool `OLLOAD`, see Section C.11.1, "OLLOAD". You can use the API calls presented in this document to make your own customizations.

C.11.2.2 Oracle Database Lite Load APIs

The Oracle Database Lite Load APIs include:

- Section C.11.2.2.1, "Connecting to the Database: olConnect"
- Section C.11.2.2.2, "Disconnecting from the Database: olDisconnect"
- Section C.11.2.2.3, "Deleting All Rows from a Table: olTruncate"
- Section C.11.2.2.4, "Setting Parameters for Load and Dump Operations: olSet"
- Section C.11.2.2.5, "Loading Data: olLoad"
- Section C.11.2.2.6, "Dumping Data: olDump"
- Section C.11.2.2.7, "Compiling"
- Section C.11.2.2.8, "Linking"

The normal mechanism for unloading and loading a table is as follows:

1. Declare local variable, `DBHandle`.
2. Connect to the database using `olConnect`.
3. Optionally, set parameters for load or unload.
4. Dump or load the data using `olDump` or `olLoad`. You may optionally delete all rows from a table by calling `olTruncate`.
5. Disconnect from the database using `olDisconnect`.

C.11.2.2.1 Connecting to the Database: olConnect Use this API to connect to the database. This is the first API that you have to call. It creates a load and unload context that is used in subsequent APIs to influence the load and unload behavior. This returns an initialized database handle DBHandle.

Syntax

```
olError olConnect (char *database_path, char *password, DBHandle &dbh);
```

The arguments for `olConnect` are listed in Table C–8:

Table C–8 olConnect Arguments

Argument	Description
`database_path`	The full path to the database file (directory path and filename).
`password`	The password used for the encrypted database, for any other database the password = NULL.
`dbh`	The application handle for the current database connection. This allows multiple database connections for one application thread (each connection has a different handle).

Return Values

(short) integer error code

Values from -1 to -8999 are used for the error codes returned by the database, values from -9000 and below are used for olLoad-specific error codes.

C.11.2.2.2 Disconnecting from the Database: olDisconnect Disconnects from the database.

Syntax

```
olError olDisconnect (DBHandle dbh);
```

The arguments for `olDisconnect` are listed in Table C–9:

Table C–9 olDisconnect Arguments

Argument	Description
`dbh`	The current application handle.

Return Value

(short) integer error code

C.11.2.2.3 Deleting All Rows from a Table: olTruncate This API can be used to delete all rows from an existing table.

> **Note:** Records removed from the server through a truncate command will not be removed from the client unless a complete refresh is triggered. The truncate command is considered a DDL operation. Consequently, the necessary DML triggers do not fire and therefore the operations are not logged for fast refresh.

Syntax

```
olError olTruncate (DBHandle  dbh, char* table );
```

The arguments for `olTruncate` are listed in Table C–10:

Table C–10 olTruncate Arguments

Argument	Description
`dbh`	The current application handle.
`tablename`	The name of the table in the form: owner_name.table_name. where owner_name is the name of the owner of the table.

Return Value

(short) integer error code

C.11.2.2.4 Setting Parameters for Load and Dump Operations: olSet This is an optional API. This sets optional parameters for load and unload.

Syntax

```
olError olSet (DBHandle  dbh, char * parameter_name, char *parameter_value);
```

The arguments for `olSet` are listed in Table C–11:

Table C–11 olSet Arguments

Argument	Description
`dbh`	The current application handle.
`parameter_name`	The name of the given parameter. This is not case sensitive. See Section C.11.2.3.2, "Parameters" for a list of parameter names and their default values.
`parameter_value`	The value to be set. This is not case sensitive for most parameters.

Return Value

(short) integer error code

C.11.2.2.5 Loading Data: olLoad `OlLoad` loads data from a file into a table using current parameter settings.

Syntax

```
olError olLoad (DBHandle dbh, char *table, char *file);
```

The arguments for `olLoad` are listed in Table C–12:

Table C–12 olLoad Arguments

Argument	Description
dbh	The current application handle.
table	The table information in the form: `owner_name.table_name(col1,col2,...)` where `col1,col2,...` is the list of column names to load. This allows you to load and dump certain columns instead of the entire table. If the entire table is to be dumped, the column list need not be specified.
file	The path to the file from which loading takes place.

Note: If table = NULL, `olLoad` tries to find the table description in the file header.

Return Value

(short) integer error code

C.11.2.2.6 Dumping Data: olDump `OlDump` dumps data from a table into a file using current parameter settings.

Syntax

```
olError olDump (DBHandle dbh, char *table, char *file);
```

The arguments for `olDump` are listed in Table C–13:

Table C–13 olDump Arguments

Argument	Description
`dbh`	The current application handle.

Table C–13 (Cont.) olDump Arguments

Argument	Description
`table`	The table information in the same form as `olLoad`.
`file`	The file to which dump data is written.

Return Value

(short) integer error code

C.11.2.2.7 Compiling The declarations for the DBHandle, parameter constants and flags, and error message codes are given in the file **olloader.h** in the *ORACLE_HOME*\Mobile\SDK\include directory. For compilation of your product include olloader.h in your main source file.

C.11.2.2.8 Linking Linking use the file **olloader40.dll** and the library file **olloader40.lib**. Include these files in your project settings.

C.11.2.3 File Format

The Oracle Database Lite Load APIs support three file formats FIXEDASCII, BINARY and CSV. Each file contains an optional header followed by zero or more rows of data.

C.11.2.3.1 Header Format The header has the following format (comments are in bold):

```
$$OL_BH$$ [begins header]
VERSION=xx.xx.xx.xx   [version number]
TABLE=T1(C1, C2, ...)... [table name with list of column names dumped]
FILEFORMAT=FIXEDASCII
SEPARATOR=,
[any other parameters in the parameter list can be listed here]
$$OL_EH$$ [ends header]
```

The following is a header example:

```
$$OL_BH$$
VERSION=01.01.01.01
TABLE=T1(EMPNO,SALARY)
FILEFORMAT=BINARY
BITARRAY=TRUE
HEADER=TRUE
RDONLY=FALSE
LOGFILE=
COMMITCOUNT=-1
NOSINGLE=TRUE
$$OL_EH$$
```

The header lines can be in any order and all lines except $$OL_BH$$ and $$OL_EH$$ can be considered optional. Although, during the dump, if the header flag is on, table information and all parameter settings are dumped into the header.

When executing load, parameter information in the header overwrites current parameter settings. If the table argument in `olLoad` is NULL, the table name and list of columns in the header prevails, otherwise the table argument of `olLoad` prevails over the header.

C.11.2.3.2 Parameters Header file parameters listed in Table C–14 are not case sensitive.

Table C–14 Parameters

Parameter	Description
`FILEFORMAT`	Input and output file format. The following formats are supported: ▪ FixedASCII - text file with fixed field width for each datatype. ▪ CSV – comma separated values format. ▪ Binary - binary file format. These key word values are not case sensitive.
`SEPARATOR`	The separator between the values (one character), comma by default.
`QUOTECHAR`	The quote character for the string datatype values in the file, single quote (') by default.
`LOGFILE`	The log file name. NULL by default (no log file produced and loading stops at the first error).
`NOSINGLE`	FALSE for single user mode (the default), or TRUE for no single user mode.
`READONLY`	FALSE (the default). TRUE to dump the data from read-only database (such as CD-ROM).
`COMMITCOUNT`	The number of rows processed after which `olLoad`, `olDump`, and `olTruncate` commit. The default value is -1, not to commit at all. Value 0 commits at the end of the operation, and values above 0 commit after the specified number of rows.
`HEADER`	FALSE (the default). TRUE to create a header in the beginning of the file during `olDump`.
`BITARRAY`	TRUE (the default) to support writing and reading nulls in binary format. During the dump, a bit array with the null information is dumped before each row. For FALSE `olDump` provides an error trying to write nulls in binary.
`NONULL`	TRUE (the default) when trying to read or write nulls `olLoad` and `olDump` return an error. When the flag is set to FALSE nulls are supported, including binary format since the default BITARRAY value is TRUE.
`DATEFORMAT`	The string for which date and timestamp columns should be written into the file and read from the file in FIXED ASCII and CSV formats. Such formats as "YYYYMMDD", "YYYY-MM-DD", and "YYYY/MM/DD" are supported. The default value is empty string (which can also be set using NULL), and the default date format is "YYYY-MM-DD". (In Oracle mode, date is treated the same as timestamp so that the date format is the default timestamp format which is "YYYY-MM-DD HH:MM:SS.SSSSSS".)

C.11.2.3.3 Data Format The data format can be comma separated value (CSV), fixed ASCII, or binary. The following cases apply:

- CSV Format: Each row of the table is represented as a separate line in the file. Each line is separated by a carriage return and a line feed character on the Windows platform. Each value in the row is separated by a separator character which by default is a comma.

  Each value is also quoted by a quote character. Nulls are represented by an empty quoted string " ". The number of quoted strings in the file should be the same as the number of columns in the table, `olLoad` gives an error otherwise.

- FixedAscii Format: Each row of the table is represented as a separate line in the file. Each line is separated by a carriage return and a line feed character on the Windows platform. Each line is of the same size. The datatype of a column governs its format or representation in the file. Nulls are represented by a string of *n* '\0' (null) characters, where *n* is the fixed size of the field. Table C–15 describes data representation for each data type. The total record length for each line in the file should be the same as the sum of field lengths (precision) of each column, otherwise `olLoad` returns an error.

Table C–15 Datatypes

Datatype	Description
`CHAR(n)`	Length of the field in *n* characters. Data is left aligned and padded with blanks on the right.
`VARCHAR(n)`	Length of the field in n characters. Data is left aligned. It is padded with a null byte ('\0').
`NUMERIC(p,s)`	The default mode: length of the field is p+1 characters if scale s is zero or is not present. Otherwise, the length of the field is (p+2) characters. The value is right aligned in the output field. Format is optional negative sign, followed by zeros if required, followed by significant digits. If there is no negative sign, then '0' instead, for example, Number(5,2) 12.3 -> ' 012.30' -12.3 -> '-012.30' 1.23 -> ' 001.23' -1.23 -> '-001.23' The custom mode: the field length is one less: p if scale is not present, or zero and p+1 otherwise. The actual number stored in the file is of type NUMERIC(p-1, s). Correspondingly, `olDump` gives an error trying to insert a number within the range of NUMERIC(p, s), but out of the range of NUMERIC(p-1, s). Therefore, the first character in the NUMERIC field must be '0' or '-'; `olLoad` gives an error otherwise.
`DECIMAL(p,s)`	The same as NUMERIC(p,s).
`INTEGER`	Length of the field is 11 characters. A negative sign or space followed by 10 digits. Leading digits are filled with zeros.
`SMALLINT`	Field length is 6 characters. Minus sign or space followed by 5 digits.
`FLOAT`	Field length is 23 characters. In Oracle mode, it is minus sign or space, followed by leading zeroes, followed by some number of digits, followed by dot, followed by some number of digits. For example: 0 -> ' 0000000000000000000000' -12.34 -> '-0000000000000000012.34' In SQL92 mode the E (exponent) is always present and there is only 1 digit before the decimal point. For example: 0 -> ' 00000000000000000000E0' -12.34 -> '-000000000000001.234E10'
`REAL`	The same format as for double precision except that the total field length is only 16 characters instead of 23.

Table C–15 (Cont.) Datatypes

Datatype	Description
`DOUBLE PRECISION`	Field length is 23 characters. Minus sign or space followed by 22 characters which are digits, dot, or E, floating point number followed by E, followed by the exponent digits. In Oracle mode, if the number is small enough to fit in the field without using the exponent, E is not used. In SQL92 mode, E is always used. There is always one meaningful digit before the floating point, except 0. For example, in SQL92 mode: 0 -> ' 00000000000000000000E0' -1.79E10 -> '-0000000000000001.79E10' 12 -> ' 00000000000000001.2E10' For example, in Oracle mode: 1.2E75 -> ' 00000000000000001.2E75' -1.33333 -> '-0000000000000001.33333' -1.79E10 -> '-00000000000017900000000'
`DATE`	In SQL92 mode: YYYY-MM-DD, 10 characters long, for example: October 1, 1999 -> 1999-10-01 In Oracle mode the date is dumped as timestamp. If it is not the default date format parameter, the date format corresponds to the specified date format string, for example: DATEFORMAT = "YYYYMMDD" October 1, 1999 -> 19991001
`TIME`	HH:MM:SS, 8 characters long, for example: 5:01:58 p.m. is 17:01:58
`TIMESTAMP`	Date format, space, time format, dot, 6 digits after dot (precision of microseconds), total length of 26 characters: YYYY-MM-DD HH:MM:SS.SSSSSS If it is not the default date format parameter, the timestamp format corresponds to the specified date format string. If no time is specified in the date format string, the time information in the timestamp is omitted when dumping into a file. Note: `TIMESTAMP WITH TIME ZONE` is not supported.

C.11.2.4 Limitations

Currently `olLoad` does not support the following features:

- Columns of the datatype Interval, Time with time zone, Timestamp with time zone, BLOB, and CLOB.
- Binary data is not supported.
- The only "var" type supported is varchar.

Index