Using Oracle® Forms Developer, Oracle® Reports Developer and the Oracle Open Client Adapter to access ODBC Data sources

Overview

The Oracle Open Client Adapter for ODBC (OCA) allows Forms Developer and Reports Developer on Microsoft Windows 95 and Windows NT to access ODBC-compliant data sources through ODBC drivers.

Using the Oracle Open Client Adapter, an application can access different data sources in a consistent manner. This allows an application developer to build an application that can run unmodified against one of several databases. Alternatively, the application can be targeted at a specific database, and take advantage of features particular to that system.

This document explains how to build Forms Developer and Reports Developer applications that access data stored in ODBC data sources. The first chapter, Getting Started, gives general guidelines that apply to any ODBC data source. The second chapter, "Building Oracle Forms Developer and Oracle Reports Developer Applications for ODBC Data Sources", gives instructions for building applications that can access any ODBC data source. For *additional* information about specific ODBC data sources please refer to the database-specific information chapters.

The absence of a specific chapter for a given database does not mean we do not support ODBC access of that database using the OCA. There are a limited number of databases we can test and document, but the OCA should work with *any ODBC data source*, provided the ODBC driver is compliant with ODBC version 2.0 or above.

How To Use This Document

This document is meant to assist you in using the Oracle Open Client Adapter. The document is divided into chapters on certain areas, but you should be able to start using the OCA just by reading the first two chapters: "Getting Started" and "Building Oracle Forms Developer and Oracle Reports Developer Applications for ODBC Data Sources".

Several issues concerning usage of the OCA are discussed in the appendices of this document because they deal with situations many users will not meet.

Table of Contents

1. Getting Started

1.1 Supported Data Sources and ODBC Compliance

The Oracle Open Client Adapter (OCA) is an ODBC 2.0 application. It should therefore be used with drivers which are ODBC version 2.0 or above. The OCA requires the ODBC driver to be ODBC API Level 1-compliant. The OCA also uses certain ODBC API level 2 functionality to achieve greater performance if that capability is provided by the ODBC driver. Level 2 functions are required for calling database stored procedures.

In addition to these requirements, we also strongly recommend using drivers which support an unlimited number of active statements per database connection (that is, drivers which report a value of 0 when SQLGetInfo is called for information type SQL_ACTIVE_STATEMENTS). The OCA does support drivers which only support one active statement per connection, but there are certain issues and limitations. Please refer to "Appendix F: Notes on drivers supporting only one active statement per connection" if you intend to use such a driver.

The Open Client Adapter allows Oracle Developer to access *any ODBC data source*, of which there are many, provided the driver meets the requirements stated above. Oracle has tested the OCA with the following data sources, and provides some additional information for them:

- Oracle Lite
- Oracle Rdb
- IBM DB2
- Informix
- Ingres
- Microsoft SQL Server
- Microsoft Access
- Sybase System 10

1.2 Installation, Setup and System Requirements

Using Forms Developer and Reports Developer to access an ODBC data source typically requires installation of components from Oracle, from the data source you wish to access, and from your ODBC driver vendor.

1.2.1 OCA Installation Directories

The location for the OCA is $(ORACLE_HOME)\OCA60 in Forms Developer and Reports Developer release 6.0, unless you specified a different location when you installed it.

It contains the following directories:

sql		consists of several subdirectories that contain SQL scripts to create the data dictionary views used by the Forms Developer and Reports Developer Builders.
plsqllib		contains the following files:
	opendb.pll	PL/SQL library of functions for making applications datasource independent
drivers		contains programs and subdirectories pertinent to ODBC drivers (if installed).
	odbcad32.exe	Microsoft ODBC Administrator (redistributed with any ODBC driver)
	odbctst.exe	Oracle ODBC test program distributed with the Oracle Rdb driver.
	rdbcnv32.exe	Oracle Rdb conversion utility (from prior driver versions).
	Rdb60	subdirectory containing documentation, license and sample files distributed with the Oracle Rdb driver.
demo		The OCA demos are included in the Forms Developer and Reports Developer standard demonstration (located under $(ORACLE_HOME)\tools). The OCA demo directory only exists for compatibility with previous releases. It contains just one file:
	open_db.fmb	Form containing property classes for different data sources. This is now OBSOLETE and is only supplied to allow existing applications which reference the classes to continue to work.

1.2.2 Setup for Forms Developer, Reports Developer and Oracle Net8 Client

Oracle Forms Developer, Oracle Reports Developer, Oracle Net8 Client[1], and the Open Client Adapter must be installed on the machine where Forms Developer

[1] Oracle Net8 Client is required for Oracle Developer release 6.0, even if only OCA (ODBC) connections are to be used. This is a change from previous releases where SQLNet was not required. It is not necessary to install any Protocol Adapters however.

and Reports Developer are to run. In addition, the Oracle software distribution includes ODBC drivers from INTERSOLV that allow access to many different ODBC data sources. Use the Oracle Installer to install them.

1.2.2.1 Net8 Client Configuration to minimize connect time

For Forms Developer and Reports Developer release 6*i* it is preferable to make a change to the sqlnet.ora configuration file once Oracle Net8 has been installed (the need for this will be eliminated in a future release). The change will minimize the time taken to connect to ODBC data sources by instructing Net8 not to attempt to contact an Oracle Names server when resolving database connect strings. If this change is not made, the connection time may be as long as 20 or 30 seconds instead of just a few seconds. The sqlnet.ora file is usually found in %ORACLE_HOME%\net80\admin. Please modify the file as follows:

Remove "HOSTNAME" from the names.directory_path entry. For example, if the file has the entry:

names.directory_path = (HOSTNAME, TNSNAMES)

change it to:

names.directory_path = (TNSNAMES)

1.2.3 Setup for ODBC Data Sources

1.2.3.1 Client Setup

Many data sources require special setup for their clients. This setup will often include the installation of networking libraries, the installation of client side tools (like an interactive SQL tool[1]), and configuring the client software to point at the correct server. This procedure differs substantially from database to database, and you should refer to the documentation provided with your database for more information.

1.2.3.2 ODBC Setup

You need to set up an ODBC data source before you can access it using Forms Developer and Reports Developer . To configure ODBC data sources, run the ODBC Administrator (installed with your ODBC driver), choose the driver you wish to use, and create a new data source.

When you set up an ODBC data source, you must give it a name. This name will be used in the connect string you specify to establish a connection to the data source (see information on connect strings later in this chapter in the section "How to specify an OCA Data Source").

The other information you usually specify includes the server name, default login ID, and whatever other driver-specific information is required. See your driver documentation for more information.

---

[1] Oracle supplies SQL*Plus for this purpose. Microsoft SQL Server and Sybase SQL Server supply ISQL.

You can create several ODBC data sources that access the same DBMS (perhaps in order to access different databases and/or different database servers). Each data source will have a unique name.

1.2.3.3 Server Setup

When you install a database server, you can set a number of options, such as sort order and language to be used, that help customize the database.

There are several things you can do at server setup time to better prepare your database for use with Oracle Forms Developer and Oracle Reports Developer .

If you data source supports case-sensitive objects, you should configure your data source in case insensitive mode (if it is available and feasible). If your data source cannot be configured to be case insensitive, create all database objects (tables, columns, views, and stored procedures) in upper case. User names should be created in upper case if possible, to remove the need to quote them if used in client-side PL/SQL. If it is not possible or feasible to configure the data source to be case insensitive, or to create objects with upper case names, you can still use Forms Developer and Reports Developer , but you should refer to the ppendix A: Case Sensitivity Issues appendix for more information.

1.2.3.4 Data Dictionary Setup

The Forms Developer and Reports Developer *builder* tools will benefit if certain views are present in the data source in order to properly access data dictionary information. They use this information to display lists of tables, columns, views, etc.. If these views are not present, you will be unable to see lists of tables, columns, etc., though you may still develop applications and reference any object in the database in your applications.

The Oracle Open Client Adapter provides scripts for generating these data dictionary tables on some popular data sources. If there is no script provided for your data source, you can build your own, basing it on the supplied scripts for other data sources[1].

The following scripts are provided:

Database	**View Creation Script**
Sybase SQL Server 4.x	`$ORACLE_HOME\oca60\sql\sqlsrv\sqsvubld.sql`
Microsoft SQL Server 4.x	`$ORACLE_HOME\oca60\sql\sqlsrv\sqsvubld.sql`
Microsoft SQL Server 6.x	`$ORACLE_HOME\oca60\sql\sqlsrv\sqs60bld.sql`
Sybase System 10 or 11	`$ORACLE_HOME\oca60\sql\sybase\sybvubld.sql`
DB2 for AS 400	`$ORACLE_HOME\oca60\sql\db2_400\db4vubld.sql`

[1] Some data sources, like Microsoft Access, do not expose data dictionary information in accessible SQL tables and views. For these data sources, it is impossible to build data dictionary views.

DB2 for AIX[1]	`$ORACLE_HOME\oca60\sql\db2_aix\db2vubld.sql`
DB2 for MVS	`$ORACLE_HOME\oca60\sql\db2_mvs\dbmvubld.sql`
Informix	`$ORACLE_HOME\oca60\sql\informix\infvubld.sql`
Ingres	`$ORACLE_HOME\oca60\sql\ingres\ingvubld.sql`
Non-Stop SQL (Tandem)	`$ORACLE_HOME\oca60\sql\nonstop\nssvubld.sql`
Oracle Rdb (6.1 and above)	`$ORACLE_HOME\oca60\sql\rdb\rdbvubld.sql`
Oracle Lite (3.0 and above)	`$ORACLE_HOME\oca60\sql\polite\polvubld.sql`

Run these scripts using the SQL*Plus utility. The example below is for Microsoft SQL Server 6.5, creating the tables in the pubs database:

```
plus80.exe scott/tiger1@odbc:sql65:pubs
      @c:\orant\oca60\sql\sqlsrv\sqs60bld.sql
```

Create these views in all databases that will be accessed by Forms Developer and Reports Developer during application *development.*

Runtime components do not use the data dictionary views, so they are not required on production systems. However, they may access a table called "DUAL" if the PL/SQL built-in functions SYSDATE or USER are used, or if any application code references the DUAL table (something which is frequent in applications initially developed for use against Oracle) . The "DUAL" table is a table with only one row in it. We have provided a script valid for all data sources to create this table. You can find it at:

```
$ORACLE_HOME\oca60\sql\dual.sql
```

and you should run it using the SQL*Plus utility. The example below is for Microsoft SQL Server 6.5, creating the table in the pubs database, owned by the user scott.

```
plus80.exe scott/tiger1@odbc:sql65:pubs
      @c:\orant\oca60\sql\dual.sql
```

1.3 Connecting to an ODBC Data Source

You should verify successful connectivity to your data source in two stages. First, verify that you have native (non-ODBC) connectivity using native data source tools (For example, for Microsoft SQL Server, use the ISQL utility). Then verify Forms Developer and Reports Developer connectivity by using the SQL*Plus utility.

1.3.1 How to specify an OCA Data Source

When connecting to an Oracle database from Forms Developer and Reports Developer tools (using Oracle Net), the connection information is provided in the form:

```
<username>/<pwd>@<connect_string>
```

[1] You should be able to use this script against DB2 for WindowsNT and DB2 for OS2.

where <connect_string> is an Oracle Net TNS alias. When connecting through the Open Client Adapter, the connect string must be of the form (items in square brackets[] are optional):

```
ODBC:<DataSourceName>[:<DataBaseName>]
```

Where <DataSourceName> is the name of the ODBC data source you defined using the ODBC administrator. See "Appendix B: Advanced OCA Connect Strings" for information about advanced usage of connect string formats.

1.3.2 Verifying Oracle Forms Developer and Oracle Reports Developer OCA Connectivity with SQL*Plus

You can test connectivity using any component, or with SQL*Plus. A simple first test is to execute the following command from a command shell:

```
plus80 <username>/<password>@odbc:<datasource>
```

If SQL*Plus generates an error message and asks for a user-name, then you have failed to connect. Otherwise, SQL*Plus will generate some status messages, and should connect you to the data source.

2. Building Oracle Forms Developer and Oracle Reports Developer Applications for ODBC Data Sources

2.1 Overview

This chapter describes the differences in building Forms Developer and Reports Developer applications against Oracle7 or Oracle8 (connected via SQL*Net), and any ODBC data source accessed via the Open Client Adapter. The following general restrictions and differences in behavior apply to all the Forms Developer and Reports Developer builder components:

- A non-Oracle database cannot be used as a repository for storing Forms, Reports or Graphics modules. You should store your modules in the file system.
- Trigger information cannot be accessed from the Database Objects Node of the Object Navigator.
- You can view stored procedure text only for data sources that can emulate the Oracle ALL_SOURCE table (Examples are Microsoft SQL Server and Sybase). You can never edit database stored procedure text.
- You cannot drag and drop PL/SQL program units from the client to a non-Oracle data source.
- Forms Developer and Reports Developer cannot use primary and foreign key constraint information from OCA data sources for default creation of Master-Detail relationships or for marking Primary Key items. These must be identified manually where required.
- Optimizer hints (/*hint*/ style comments) will not cause errors but are ignored by any data source connected to via the OCA (because the OCA SQL parser removes comments from SQL statements).
- Oracle8-specific features (such as user-defined data types and object extensions to SQL) will not work against OCA connections

2.2 Using SQL Plus with OCA Connections

2.2.1 Avoiding truncation of DATE, TIME and TIMESTAMP values

SQL Plus 8.0 fetches all column values as character strings, relying on the data source to perform the conversion from native to character format. The ODBC standard is for ODBC drivers to return DATE values in the format 'YYYY-MM-DD', TIME values as 'HH24:MI:SS' and DATETIME (or TIMESTAMP) values as 'YYYY-MM-DD HH24:MI:SS'. Fractional seconds may also be present in some cases.

All native columns which map to the Oracle DATE type are displayed in SQL Plus 8.0 only up to a certain maximum length. This maximum length is determined from the client-side Oracle NLS environment parameters, notably NLS_LANG and NLS_DATE_FORMAT (if present). If a date, time or datetime value exceeds this maximum length, it will be truncated when displayed by SQL Plus.

The length to which values are truncated is equal to the length of the format mask given in NLS_DATE_FORMAT, or, if that is absent, by the length of the default date format mask for the territory specified in NLS_LANG. Note that the actual *contents* of the mask are ignored by SQL Plus. For example, if NLS_DATE_FORMAT is not set and the territory part of NLS_LANG is America, then the default date mask is 'DD-MON-YY' which is of length 9, so all date, time or datetime values will be truncated to 9 characters. This is insufficient to display date and time values as they are typically returned by the ODBC datasources, for example, January 31st 1997 would appear as "1997-01-3".

2.2.1.1 Recommended action

We recommend explicitly setting the Oracle environment variable NLS_DATE_FORMAT to a mask which is long enough for all required date, time or datetime values. For example, "YYYY-MM-DD" (length 11) will allow date and time values such as "1997-01-31" or "10:15:05" to be displayed fully. A longer value (such as "YYYY-MM-DD HH24:MI:SS") would be needed for displaying datetime data.

On Windows 32-bit platforms (Windows 95, 98 and NT) the NLS_DATE_FORMAT value should be created (or updated) in the Windows Registry under the "ORACLE" key, using a registry editor such as regedit or regedt32.

2.2.2 Supported SQL statements

SQL Plus passes most SQL statements straight through to the data source, so in general it is possible to use any SQL statements, whether or not they conform to Oracle SQL syntax. However, there are some special cases which it is important to note. These are detailed in the following sections.

2.2.2.1 How to terminate commands which look like PL/SQL

By default, SQL Plus assumes that input SQL statements are terminated by the character ";" (semi-colon), *unless* they have the appearance either of an Oracle SQL command containing PL/SQL (Create Procedure, Create Function or Create Package) or of an anonymous PL/SQL block (starting with "BEGIN" or "DECLARE") . If the command has this appearance, then embedded semi-colons will be accepted as part of the command and will not be interpreted as indicating the end of the command. So such commands should instead be terminated with a single line containing just the character "/" (to execute the command) or "." (to

terminate command input without immediately executing it). This is in fact good practice for all commands anyway.

As an example, a SQL*Plus script to create a Sybase stored procedure might be written as follows:

```
CREATE PROCEDURE sptest_emp_2parms @empno_in
smallint, @empno_out smallint output
AS
SELECT @empno_out = @empno_in
return 33
/
```

Note the use of "/" to terminate (and execute) the command instead of ";".

2.2.2.2 Issuing SQL commands with embedded semi-colons (";")

By default, SQL Plus assumes that input SQL statements are terminated by (i.e. end before) the character ";" (semi-colon), *unless* they have the appearance of an Oracle SQL command containing PL/SQL (see previous section).

So in order to execute statements containing semi-colons and which do not resemble PL/SQL, such as RDB "Create Module" statements, it is necessary to change the SQL command terminator to something else. This can be done, for example, as follows (sets the terminator to "\"):

```
set sqlterminator "\"
```

2.2.2.3 Issuing SQL commands with embedded "@" characters

Input lines starting with the character "@" (including lines which are part of a SQL statement) will be interpreted as an instruction to execute a file of SQL*Plus commands. To avoid this problem, arrange the SQL statement so that no lines start with "@". For example, when creating a stored procedure in Sybase, the following will fail because SQL*Plus will attempt to interpret the files empno_in.sql, etc:

```
CREATE PROCEDURE sptest_emp_2parms
   @empno_in smallint,
   @empno_out smallint output
AS
SELECT @empno_out = @empno_in
return 33
/
```

However, it *will* work if input as:

```
CREATE PROCEDURE sptest_emp_2parms @empno_in
smallint, @empno_out smallint output
AS
SELECT @empno_out = @empno_in
return 33
/
```

2.2.2.4 SQL commands starting with non-standard keywords

SQL*Plus recognizes SQL statements by their leading keyword or keywords. Certain data source-specific (non-standard) SQL commands starting with non-standard keywords are not recognized, and therefore *cannot be executed* under SQL*Plus. For example, CALL statements against RDB are rejected as

"unknown commands". These commands should be executed using the native SQL interpreter supplied with the DBMS.

2.3 Writing PL/SQL for use with OCA (ODBC) connections

2.3.1 SQL within PL/SQL

SQL statements embedded in PL/SQL program units must conform to both Oracle SQL and the SQL dialect of the data source (or data sources) against which the application is to run. Any statements which would fail against Oracle will cause the PL/SQL to fail to compile, and any statements using syntax not supported by the data source will fail at execution time. Exceptions to this are the SYSDATE and USER functions which are specific to Oracle SQL. These will be translated by the OCA to the corresponding ODBC functions and will work against all data sources[1].

If your data source has case sensitive names for items in the database, and some are not upper case, you should refer to "Appendix A: Case Sensitivity Issues" for details on how to handle these in PL/SQL.

2.3.2 "Where current of <cursor>" is not supported

The syntax "Where current of <cursor_name>" is sometimes used in "UPDATE" and "DELETE" SQL statements within PL/SQL against Oracle. However, it will *not* work against ODBC connections and should be avoided in all applications which are required to work against ODBC data sources. The reason is that it is implemented using Oracle rowid's which are not portable to other data sources. The solution is to use primary keys instead. For example, in the code:

```
Declare
  Cursor C is select empno, ename from emp for update;
  v_empno emp.empno%type;
  v_ename emp.ename%type;
Begin
  Open C;
  Fetch C into v_empno, v_ename;
  Update emp set ename = 'Updated'
  where current of C; --This is forbidden against ODBC
connections
  Close C;
End;
```

the Update statement should be changed to read:

```
  Update emp set ename = 'Updated'
  where empno = v_empno; --Use primary key to identify row
```

[1] except when the ODBC driver does not support the now() function, but almost all drivers do support it.

2.3.3 Native SQL and multiple database connections

If you wish to issue SQL statements using syntax specific to the data source (and conflicting with Oracle syntax), you may do so using one of the pass-through SQL techniques described in section 2.4 below. Note that the EXEC_SQL package, which is a built-in package in the Forms Developer and Reports Developer tools, not only allows native SQL, but may also be used to handle multiple simultaneous database connections and to fetch result sets returned from stored procedures.

2.3.4 Writing code branches specific to a data source

When using native SQL you can benefit from functionality specific to a particular data source. However, to allow the application to continue to work against other data sources, you may wish to ensure that such statements are issued only against the data source which supports them. The OPENDB PL/SQL library (also supplied with the Open Client Adapter) provides functions which allow you to find out which data source you are connected to at a given time. You may use these functions to write data source specific branches in your code. Please refer to the chapter "How to use the OPENDB PL/SQL library for data source independence" for more details.

2.3.5 Referencing tables from more than one database

Many databases (e.g. Microsoft SQL Server) allow you to access table in a database other than the one you are connected to, by using a three-part tablename syntax like:

```
database.owner.tablename
```

PL/SQL does not recognize the three-part table name syntax, and your client-side program unit or trigger will not compile. To work-around this problem, enclose the three-part name in double quotes, after calling the appropriate function from the OPENDB PL/SQL library to instruct the OCA to strip double quotes. Refer to the chapter "How to use the OPENDB PL/SQL library for data source independence" for details about the OPENDB library.

For example:

```
dbinfo.set_quote_char(NULL);
select ename, sal into v_ename, v_sal from “hrdb.scott.emp”;
```

2.3.6 Using PL/SQL predefined exceptions

Most predefined (i.e. built-in) PL/SQL exceptions work the same way against ODBC connections as against native Oracle connections, even exceptions relating to database access. Examples are: `NO_DATA_FOUND`, `CURSOR_ALREADY_OPEN`, `INVALID_CURSOR`, `TOO_MANY_ROWS` and `ROWTYPE_MISMATCH`.

However, some predefined exceptions are never raised against ODBC connections. These are:

`DUP_VAL_ON_INDEX`, `INVALID_NUMBER` and `TIMEOUT_ON_RESOURCE`.

Instead, an exception is raised with an error code and error message specific to the Open Client Adapter (and not corresponding to a predefined PL/SQL exception). The native error message from the data source or ODBC driver will be included as part of the error message. Most often the OCA error code reported is -30021 ("error preparing/executing SQL statement"). This will cause an unhandled exception unless there is a "WHEN OTHERS" exception handler or a specific handler for the -30021 error code.

2.3.6.1 Example exception handler for unique key violation

One exception which will not fire against ODBC data sources is `DUP_VAL_ON_INDEX` (corresponding to Oracle error ORA-00001). So for example, if an `INSERT` statement is attempted against Oracle Lite which would result in a duplicate value for a primary key, the statement will fail with error code -30021, but `DUP_VAL_ON_INDEX` will not be raised. The error message is:

```
OCA-30021: error preparing/executing SQL statement
[POL-3220] duplicate keys in unique/primary index
```

Ways of catching this error are: (a) use the "WHEN OTHERS" exception clause, or (b) define and catch a named exception corresponding to error code -30021 (using the syntax "pragma exception_init"). The error message may then either be displayed to the user or analyzed further to determine the cause of the error. Here is an example:

```
EXCEPTION
  When dup_val_on_index
  --Duplicate key value against native Oracle connection
  then
    message('This department number already exists !');
  When Others then
      If sqlcode = -30021 /* OCA statement execution error */
      and (upper(sqlerrm)) like '%DUPLICATE%'
      --Duplicate key value against ODBC connection
      then
        message('This department number already exists !');
      else
        message('Unexpected error: ' || sqlerrm);
      end if;
END;
```

2.3.7 PL/SQL REF CURSOR variables

REF CURSOR variables may be used in client-side PL/SQL against ODBC data sources. However, they may not be passed into or out of the data source (as stored procedure parameters) because they are Oracle-specific; they are not recognized by the ODBC standard, nor by any non-Oracle database management systems. As already mentioned, the `ROWTYPE_MISMATCH` exception works as expected.

2.4 Pass-through SQL: techniques for executing native SQL

We have already seen in the previous sections that whilst SQL*Plus allows the execution of almost any native SQL command, PL/SQL does not. There are

however ways of executing native SQL statements from within PL/SQL with no limitations, by using special built-in routines. These are illustrated in the following table, which shows ways of executing a statement specific to Microsoft SQL Server (sets current database to "pubs"):

Forms:	`FORMS_DDL  ('use pubs');`
Reports:	`SRW.DO_SQL ('use pubs');`
Graphics:	`DO_SQL     ('use pubs');`

Any PL/SQL:
 	
(Forms, Graphics, Reports or Procedure Builder)
	`DECLARE`
	`  v_curs EXEC_SQL.CursType;`
	`  v_numrows INTEGER;`
	`BEGIN`
	`  v_curs := EXEC_SQL.OPEN_CURSOR;`
	`  EXEC_SQL.PARSE(v_curs, 'use pubs');`
	`  v_numrows := EXEC_SQL.EXECUTE(v_curs);`
	`  EXEC_SQL.CLOSE_CURSOR(v_curs);`
	`END;`

Note that bind variables are not allowed when using the FORMS_DDL built-in.

The most general technique is to use the EXEC_SQL package, which offers full support for bind variables and for re-executing statements efficiently without having to re-parse. The EXEC_SQL package is documented in the Procedure Builder on-line help and documentation. It may also be used to open and handle multiple simultaneous database connections, and to fetch result sets returned by stored procedures.

2.5 Using Oracle Forms with OCA Connections

2.5.1 Form, data block and item properties

2.5.1.1 Automatic runtime adjustment of Form and Data Block properties

The Oracle RDBMS offers locking and concurrency features that some other databases do not. As a result, the way Forms interacts with the database when running against Oracle may not be appropriate for other data sources.

In order to deal with this, Oracle Forms automatically detects certain characteristics of the data source at connect time and changes it's behavior accordingly. Certain Form and Data Block properties are automatically adjusted. The changes are documented in in "Appendix E: Automatic Adjustments in Forms Behavior for OCA Data Sources" for completeness. It is not normally necessary for users to make any other changes[1]. However, the Form-level "Isolation Mode" property may optionally be used to change the transaction

[1] The only known case where it is necessary to make other changes (by calling OPENDB.INIT_FORM) is for certain forms running against Rdb. Please refer to the chapter on RDB for further details.

isolation level from read committed (the default) to serializable. If set to serializable, the OCA will set the transaction isolation level to TXN_SERIALIZABLE (if supported) or, failing that, leave it unchanged.

Please note that for most data sources it is still necessary to set the item-level Primary Key property to TRUE for items based on primary key columns (see Item Properties below).

2.5.1.2 Item Properties

Primary Key (Item) Identifies which items are primary key items. For OCA data sources other than RDB, it is a *requirement* to set this property to *TRUE* for all items which are based on primary key columns in the database. Otherwise runtime errors will occur when attempting to update or delete rows, as Forms will be unable to uniquely identify a row in the database.

2.5.1.3 Data Block properties

In certain (exceptional) circumstances it may be necessary to modify the following properties for base-table data blocks :

Update Changed Columns When an item in a row is changed, Forms will build an update statement that includes all columns. This improves performance by allowing the same cursor to be used for all updated rows, even though different columns may have been updated in each. However, if there are primary key items, and the data source does not allow updating of primary keys, the update may fail with the error 'Primary key column *colname* cannot be updated'. If you encounter this error in your OCA application, set the *Update Changed Columns* property to *True* so that Forms will include only columns that changed in the update statement.

Records Fetched (Block) Some ODBC drivers have limits on the number of records fetched at a time. For this reason, the *Records Fetched* property for base table blocks should be set no higher than the maximum number of records your ODBC documentation says the driver can fetch. In general you will obtain best apparent performance by leaving this property at its default of zero, which fetches enough records to fill one screen at a time.

2.5.2 Data Types

The OCA automatically maps Oracle data types to the corresponding data type for the data source. Please see Appendix C: Oracle Data Type Matching for details.

Some ODBC drivers can only support one LONG column per SELECT statement, and some additionally require that that LONG column be last in the SELECT list. If your ODBC driver has this limitation (see your driver's documentation), then you must only have one LONG column per base table data block.

2.5.3 Multiple sessions

Multiple *sessions* against a *single database connection* are NOT supported for ODBC connections (because this is very Oracle-specific). If a new form is opened using "OPEN_FORM(…,SESSION)", then any database interaction in the opened form will be done in the *same* database session (and transaction) as the calling form. In addition, errors may occur as "OCA-30002: ubofscr function not supported". It is therefore strongly recommended to use OPEN_FORM(…,NO_SESSION)" when running against an ODBC database connection.

2.6 Using Oracle Reports with OCA Connections

2.6.1 Referencing tables from more than one database

Some ODBC data sources allow you to access tables in a database other than the one you are connected to. For example, in SQL Server, you can access tables in other databases using the syntax *database.owner.tablename*. Reports supports this syntax (which is very data source dependent) in queries, but does not usually generate correct SQL if you build master-detail relationships between them. To work around this, build the master-detail relationship using a group link instead of column links (click on the link tool and drag the mouse from the master to the detail groups), and explicitly specify the master-detail relationship. For more on group links, consult the Reports documentation.

2.6.2 Executing DML from within Reports

In order to avoid acquiring table-level locks during the execution of a report, Reports runs in Auto-commit mode. If you use DML inside Reports, for example to log progress or populate temporary tables, be aware that each statement will be committed immediately after execution

Other than the above notes, and the general differences described earlier, Reports can be used against OCA data sources with no other changes to its default processing.

2.7 Using Oracle Graphics with OCA Connections

Other than the general differences between Oracle and non-Oracle databases described earlier, Oracle Graphics may be used against OCA data sources with no changes to its default processing.

2.8 Using Oracle Procedure Builder with OCA Connections

Other than the general differences between Oracle and non-Oracle databases described earlier, Procedure Builder may be used against any OCA data source with no changes to its default processing. Although no editing of database stored procedures is possible, Procedure Builder is nonetheless a valuable tool for managing client-side PL/SQL libraries, and testing stand-alone client-side PL/SQL modules and libraries.

2.9 Connecting to different databases

Some data sources, such as Microsoft SQL Server, allow one database server to manage several databases. You can usually specify which database to connect to when you set up your ODBC driver, but sometimes you may want to switch databases when running an application.

You can specify which database to connect to by using the following syntax for the connect string[1]:

```
ODBC:<DataSourceName>:<DataBasename>
```

Alternatively, to change the current database when the connection has already been established, you may execute a native (database-specific) SQL command for switching databases in a client-side trigger or program unit. Use one of the pass-through SQL techniques described in section 2.4.

2.10 Transaction isolation levels

The ODBC specification defines various transaction isolation levels which describe different types of transactional support. Examples of transaction isolation levels are SQL_TXN_READ_COMMITTED and SQL_TXN_SERIALIZABLE. Often a given data source can support several isolation levels depending on the needs of the application. Which levels which are supported depends on the ODBC driver and on the characteristics of the underlying DBMS. Higher isolation levels (like Serializable) give better transaction isolation but often at the expense of throughput (because they cause more locking of data to occur).

2.10.1 Default transaction isolation level

On connecting to an ODBC data source, the OCA makes no attempt to set or change the transaction isolation level so it will remain set to whatever is the default for the ODBC data source. Some drivers (such as the Oracle Lite driver) allow

[1] See " Appendix B: Advanced OCA Connect Strings" for more information on this format of connect strings.

the default transaction isolation level to be configured for each data source using the ODBC Administrator utility.

2.10.2 Setting the transaction isolation level

It is possible for applications to change the transaction isolation level dynamically. This can be done in one of several ways when using the Open Client Adapter (listed in order of preference):

1. Use the Isolation Mode property in Forms (see next section)
2. Issue one of the following Oracle-style ALTER SESSION commands using one of the pass-through SQL techniques described in section 2.4:
   ```
   ALTER SESSION SET ISOLATION LEVEL=SERIALIZABLE
   ALTER SESSION SET ISOLATION LEVEL=READ COMMITTED
   ```
   The OCA will intercept these commands and issue the appropriate ODBC command to set the isolation level to SQL_TXN_SERIALIZABLE or SQL_TXN_READ_COMMITTED respectively. If the requested level is not supported by the ODBC driver, then the OCA leaves the level unchanged and returns Oracle error 2248 (invalid option for ALTER SESSION) with a corresponding OCA error message.
3. Issue a native SQL command (if the DBMS has one) to change the isolation level, using one of the pass-through SQL techniques described in section 2.4. This method has the disadvantage of being specific to a particular DBMS.

2.10.3 SERIALIZABLE Isolation Mode in Forms

In Forms it is possible to set the Form-level "Isolation Mode" property to either "Read Committed" (the default) or "Serializable".

The default ("Read Committed") value causes the data source's *default* isolation level to be used (as described in section 2.10.1 above). *This will not necessary actually be Read Committed.*

If the "Serializable" value is selected, Forms runtime will attempt to set the transaction isolation level to SERIALIZABLE by issuing the command "`ALTER SESSION SET ISOLATION LEVEL=SERIALIZABLE`". If this succeeds, no more lock-record processing will be done in base table data blocks (because it is essentially redundant when transactions are SERIALIZABLE because lost updates cannot then occur). The Forms Developer and Reports Developer Forms documentation gives further details on this.

2.11 OCA Error Handling

Open Client Adapter error messages have the same general form as Oracle error messages. Their format is as follows:

```
OCA-XXXXX: Error Message Text
```

Where XXXXX is a 5 digit number for the OCA error code (in the range 30000 to 31000).

If the error being reported was an error reported by the ODBC driver, the OCA appends the ODBC error message to the end of the error message. For example, the following error message can be returned by the OCA:

```
OCA-30037: datasource not available
[Microsoft][ODBC Driver Manager] Data source name not found
and no default driver specified
```

This information greatly assists in identifying the source of the error.

Note that the OCA attempts to map its errors to corresponding Oracle error codes where possible. This means the reported error code (SQLCODE in PL/SQL or embedded SQL) after a failed database operation will be set to an Oracle error code if there is a suitable corresponding one. Otherwise it will be set to the OCA code, in the range 30000 to 31000 (which is also the number which appears in the error message text).

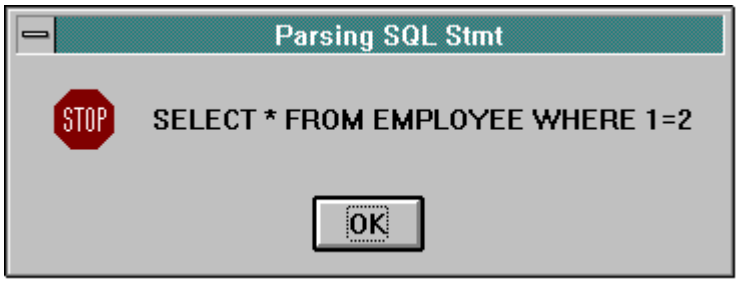
For a list of possible OCA error messages, and for the mapping with Oracle error codes, see "Appendix G: Error Messages".

2.12 Debugging an Application using the OCA

2.12.1 OCA debug options

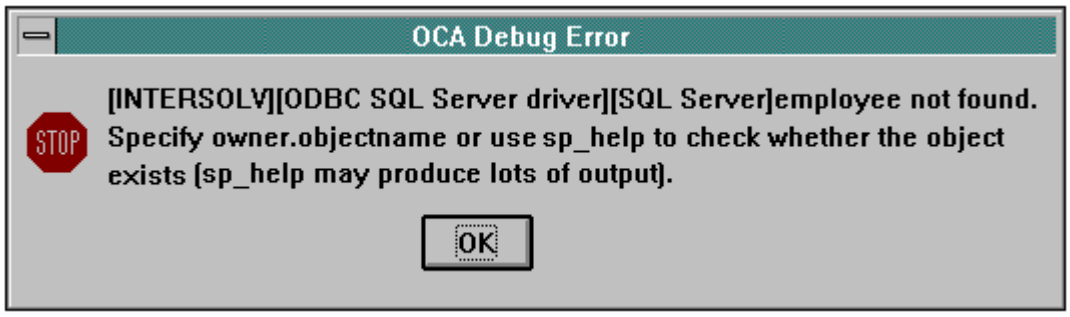
The Open Client Adapter supports two environment entries which cause it to display useful debugging information. Set these entries in ORACLE.INI on Windows 3.1, or in the registry under Windows 95 or NT (under the sub-key \\HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE[1]).

Setting OCA_DEBUG_SQL=TRUE will result in each SQL statement that is sent to the ODBC driver being displayed in a dialog before it is transmitted. For example:



Setting OCA_DEBUG_ERROR=TRUE will result in any errors returned by the ODBC driver being displayed in a dialog before it is passed back to the Forms Developer and Reports Developer tools. This allows you to inspect the error before it is processed in any way by the tool. For example:

[1] All of these variables are String Values



Please note when using OCA_DEBUG_ERROR that some errors are to be expected (i.e. do not indicate that there is a problem), for example, when a SQL statement is being described the columns are sometimes described one by one until an error indicates there are no more columns.

Using OCA_DEBUG_SQL and OCA_DEBUG_ERROR in combination will help you pinpoint the cause of SQL failures when using Forms Developer and Reports Developer against OCA data sources.

2.12.2 Using ODBC Driver Manager tracing

Under certain circumstances, it can be useful to generate a trace of all ODBC commands which are being executed (by the OCA). Oracle Technical Support may ask you to do this if you encounter problems specific to a particular data source or ODBC driver.

Tracing may be requested by running the Microsoft ODBC 3.0 administrator (ODBCAD32.EXE). This is included and installed with your ODBC driver, for example, the Intersolv ODBC drivers and the Oracle RDB ODBC driver which are included in the Forms Developer and Reports Developer distributions. It is also usually available in the Windows Control Panel (marked for example "32-bit ODBC"). To turn on ODBC tracing, quit all ODBC applications, run the ODBC Administrator, select the tab marked "Tracing", type a suitable trace file name into the field marked "Log file path", and click on the button marked "Start tracing now". Leave the ODBC administrator running and run the application you need to trace. Then return to the ODBC administrator and click on the button which is now marked "Stop tracing now" to turn off tracing. The log file will contain a trace of all ODBC calls which were made during tracing (this includes ALL running ODBC applications, so it is best to run only one). Copy it to a new location to avoid it being overwritten or expanded by any future tracing which may be done.

3. How to use the OPENDB PL/SQL library for data source independence

3.1 Overview

This chapter explains how to use the OPENDB library (opendb.pll) from within Forms Developer and Reports Developer applications to write data source independent applications. This library allows you to :

1. Obtain the DBMS and ODBC Driver names and versions for the current connection.
2. Override the default behavior of the Open Client Adapter.

In addition, a routine called "opendb.init_form" is provided which automatically adjusts Data Block properties at runtime to suit the data source. This routine is only needed when running against RDB (although it does no harm against other data sources). Please refer to the chapter on RDB for more details.

The OPENDB library is primarily intended for use against non-Oracle datasources (via ODBC), but it may also be used when directly connected to Oracle (i.e. not through ODBC). This allows applications to be developed and deployed against a mixture of Oracle and other datasources.

3.2 Obtaining details about the Datasource

The following routines (which are in the DBINFO package) are available for specific programming needs. They allow you to obtain information about the datasource, as reported by the ODBC driver. They all return NULL if the current connection is a native Oracle connection (i.e. not ODBC).

3.2.1 DBINFO.GetDBMSName

```
FUNCTION GETDBMSNAME
RETURN VARCHAR2; /* DBMS name */
```

3.2.2 DBINFO.GetDBMSVer

```
FUNCTION GETDBMSVER
RETURN VARCHAR2; /* DBMS version */
```

3.2.3 DBINFO.GetDriverName

```
FUNCTION GETDRIVERNAME
RETURN VARCHAR2; /* Driver name */
```

3.2.4 DBINFO.GetDriverVer

```
FUNCTION GETDRIVERVER
RETURN VARCHAR2; /* Driver version */
```

3.2.5 DBINFO.GetCursorMode

```
FUNCTION GETCURSORMODE
RETURN PLS_INTEGER; /* Cursor mode, either DBINFO.CURSOR_CLOSE
or
                    ** DBINFO.CURSOR_OPEN */
```

3.2.6 DBINFO.GetSavepointMode

```
FUNCTION GETSAVEPOINTMODE(ConnID in pls_integer)
RETURN PLS_INTEGER; /* Savepoint mode, either
DBINFO.SP_SUPPORTED
                    ** or DBINFO. SP_NOT_SUPPORTED */
```

Of these, only dbinfo.GetCursorMode requires any further explanation. If this function returns dbinfo.CURSOR_CLOSE, this means that the data source behavior is such that whenever a database transaction is COMMITTED (save) or ROLLED BACK (clear all), the positions of all open cursors are lost. In practice, this means that if a user executes a query in a data block, then makes changes and saves (commit), they will have to RE-EXECUTE the query before they can scroll down any further. An attempt to scroll down without re-querying will result in an error message. The dbinfo.GetCursorMode function allows applications to trap this condition and handle it in a more user-friendly manner (for example, by automatically clearing the block or automatically re-executing the query).

3.3 Overriding the default behavior of the OCA

The following routines (which are in the DBINFO package) allow you to change the behavior of the Open Client Adapter.

3.3.1 DBINFO.SetQuoteChar

DBINFO.SetQuoteChar allows you to set the character (or characters) used to delimit quoted identifiers. The OCA transforms identifier names within double quotes within SQL statements by changing the double quotes into the appropriate delimiter for the data source (as reported by the ODBC driver). Occasionally it is necessary to override the character used, for example you may wish to strip the double quotes altogether to allow the use of three-part table names in PL/SQL in the form “database.user.table”. This may be done by supplying a value of NULL for the quotechar parameter :

```
DBINFO.SetQuoteChar(NULL); /* OCA will now strip double quotes
*/
```

Syntax:

```
PROCEDURE SetQuoteChar  (quotechar IN VARCHAR2);
```

3.4 Using the OPENDB library in Oracle Forms Developer and Oracle Reports Developer applications

When the Open Client Adapter for ODBC is installed, the installer copies the OPENDB PL/SQL library to the plsqllib directory under the OCA home directory. It also adds this directory to the path used by Forms to search for PL/SQL libraries (FORMS60_PATH). This ensures that the library will always be found by Oracle Forms when attaching it to new forms and when opening or running existing forms which use it.

4. Calling Stored Procedures through the Oracle Open Client Adapter

4.1 Overview

Oracle Forms Developer and Oracle Reports Developer allow users to call stored procedures in non-Oracle databases from PL/SQL in much the same way as they can call stored procedures in Oracle databases. However, due to differing kinds of stored procedures depending on data sources, there are some limitations of this support.

For example, some databases (like Rdb) do not support stored procedures that return a value (a FUNCTION in PL/SQL). Others, like Microsoft SQL Server, can return multiple result sets from a stored procedure.

This chapter discusses some of the aspects of stored procedure support in Oracle Forms Developer and Oracle Reports Developer , and shows how to deal with any limitations in calling non-Oracle stored procedures.

4.2 Calling Stored Procedures from PL/SQL

Oracle Forms Developer and Oracle Reports Developer client-side PL/SQL allows you to call database stored procedures as if they were just regular PL/SQL procedures. For example, a Transact SQL (Microsoft SQL Server or Sybase) procedure like this:

```
create proc demoproc @var1 varchar(20), @var2 integer output
as
  select @var2 = convert(integer, @var1)
```

can be called from PL/SQL as[1]:

```
DECLARE
  v_ret INTEGER;
  v_var1 VARCHAR2(20);
  v_var2 NUMBER;
BEGIN
  v_var1 := '134.35';
  v_ret := demoproc(v_var1, v_var2);
END;
```

The PL/SQL compiler will figure out what kind of stored procedure you are trying to invoke, and will make sure that all arguments match. For example, if you try to execute:

```
v_ret := demoproc(v_var2, v_var1);
```

the compiler will return an error because of the mismatched parameters.

[1] All SQL Server and Sybase stored procedures are treated as FUNCTIONS by PL/SQL, as they always return an INTEGER value.

4.2.1 Handling Stored Procedures That Return Result Sets

Some non-Oracle databases allow stored procedures to return result sets directly, instead of via REF CURSOR variables as used by Oracle stored procedures. PL/SQL does not implicitly allow you to access these non-Oracle result sets.

Therefore, if you call a stored procedure like (using Transact SQL again):

```
create proc demoproc2 @var1 varchar(20), @var2 integeroutput
as
  select empno from emp
  select @var2 = convert(integer, @var1)
  select deptno from dept
  return 99
```

You will be unable to retrieve the results sets generated. However, all variables will be correctly set.

If you need to retrieve a result set returned by a stored procedure, you must use the EXEC_SQL package (see the section "Calling Stored Procedures Using The EXEC_SQL Package" later in this chapter).

4.2.2 Compiling Non-Oracle Stored Procedure calls in PL/SQL

When PL/SQL attempts to compile a call to a non-Oracle stored procedure, it asks the OCA to provide a PL/SQL wrapper function of the same name that calls the database stored procedure using dynamic SQL methods. The OCA constructs a PL/SQL wrapper with the same arguments that the ODBC driver reports for the stored procedure. PL/SQL then takes this wrapper, and makes sure that all arguments match the invocation given by the user.

For example, for the Transact SQL procedure demoproc2 above, the OCA would generate a wrapper PL/SQL function that has the signature:

```
FUNCTION demoproc2 ( p1 IN VARCHAR2, p2 IN OUT NUMBER)
  RETURN NUMBER IS
...
```

Sometimes the arguments in the wrapper function do not exactly match what might be expected. To help the user compile calls to Non-Oracle stored procedures, he/she can turn on OCA SQL debugging (see the section "Debugging an Application using the OCA" in chapter 2). The code for all wrapper functions will then be displayed when PL/SQL attempts to compile. The user can then compare the order and types of arguments in the wrapper function to make sure they agree.

4.3 Calling Stored Procedures Using The EXEC_SQL Package

If you want to retrieve a result set from a stored procedure in an ODBC data source, you must use the EXEC_SQL built-in package. Please refer to the Procedure Builder documentation or on-line help for complete information about the EXEC_SQL package.

To call a stored procedure through the OCA using EXEC_SQL, you must use the following syntax[1]. Specifically, your call should be of the form (where items in square brackets [] are optional):

{ [:*parameter*] = call *procedure-name* [([:*parameter*][,*parameter*...])] }

For example, a call to the stored procedure demoproc2 above, would be:

```
{ :ret_val = call demoproc2(:v_val1, :v_val2) }
```

The following code is an example of executing a stored procedure against SQL Server, fetching all result sets, and getting all output variables. It assumes you are familiar with the basics of EXEC_SQL:

```
DECLARE
  v_cursor EXEC_SQL.CursType;
  v_empno VARCHAR2(20);
  v_deptno VARCHAR2(20);
  v_retval PLS_INTEGER;
  v_val2   NUMBER;
  v_numrows PLS_INTEGER;
BEGIN
  -- Opens a new cursor to do all work on.
  v_cursor := EXEC_SQL.OPEN_CURSOR;

  -- Parses the call to the stored procedure.
  EXEC_SQL.PARSE(v_cursor,
    '{ :ret_val = call demoproc2 ( :v_val1, :v_val2 ) }');

  -- Binds a PL/SQL variable to each parameter in the stored
  -- procedure call.

  -- Bind the return value
  EXEC_SQL.BIND_VARIABLE(v_cursor, ':ret_val', v_retVal);

  -- Bind the first parameter.  Since it is an input only
  -- parameter, we don't bother specifing a variable, and
  -- instead just give the value we care about.
  EXEC_SQL.BIND_VARIABLE(v_cursor, ':v_val1', '1344.5');

  -- Bind the second parameter.  The second parameter is only
  -- used for OUTPUT, but we need to BIND it, to inform
EXEC_SQL
  -- it will need to retrieve the value later.
  EXEC_SQL.BIND_VARIABLE(v_cursor, ':v_val2', v_val2);

  -- Now, execute the stored procedure
  v_numrows := EXEC_SQL.EXECUTE(v_cursor);

  -- For the first result set, we define one column.
  -- For VARCHAR columns, we need to specify the maximum
  -- length we will retrieve into v_ename.
  EXEC_SQL.DEFINE_COLUMN(v_cursor, 1, v_ename, 20);

  -- And we keep fetching the rows until we're done.
```

[1] The syntax is a mixture of ODBC stored procedure SQL syntax and Oracle SQL syntax. Notably, all parameter markers must be in Oracle Form (:parameter) instead of ODBC form (?).

```
  WHILE EXEC_SQL.FETCH_ROWS(v_cursor) != 0 LOOP
    -- Retrieve each column value into our PL/SQL variable.
    EXEC_SQL.COLUMN_VALUE(v_cursor, 1, v_ename);
    TEXT_IO.PUT_LINE('Ename: ' || v_ename);
  END LOOP;

  -- We are done fetching, but need to prepare the next
  -- result set.
  IF NOT EXEC_SQL.MORE_RESULT_SETS(v_cursor) THEN
    -- Error.  We expected another result set.
    TEXT_IO.PUT_LINE('Error.  No second result set');
    RAISE EXEC_SQL.PACKAGE_ERROR;
  END IF;

  -- Define the new column
  EXEC_SQL.DEFINE_COLUMN(v_cursor, 1, v_dname, 20);

  -- And fetch the rows again.
  WHILE EXEC_SQL.FETCH_ROWS(v_cursor) != 0 LOOP
    -- Retrieve each column value into our PL/SQL variable.
    EXEC_SQL.COLUMN_VALUE(v_cursor, 1, v_dname);
    TEXT_IO.PUT_LINE('Dname: ' || v_dname);
  END LOOP;

  -- Now, retrieve our output variables.
  EXEC_SQL.VARIABLE_VALUE(v_cursor, ':ret_val', v_retVal);
  EXEC_SQL.VARIABLE_VALUE(v_cursor, ':v_val2', v_val2);
  TEXT_IO.PUT_LINE('Stored Procedure returned '
    || TO_CHAR(v_retVal));
  TEXT_IO.PUT_LINE('Val2 is ' || TO_CHAR(v_val2));

  -- And close the cursor
  EXEC_SQL.CLOSE_CURSOR(v_cursor);
EXCEPTION
  WHEN EXEC_SQL.PACKAGE_ERROR THEN
    TEXT_IO.PUTLINE('Error (' ||
      TO_CHAR(EXEC_SQL.LAST_ERROR_CODE) ||
      '): ' ||
      EXEC_SQL.LAST_ERROR_MESG);
    EXEC_SQL.CLOSE_CURSOR(v_cursor);
  WHEN EXEC_SQL.INVALID_CONNECTION THEN
    TEXT_IO.PUT_LINE('Invalid connection');
END;
```

Note: Some data sources do not set output variables until all result sets have been fetched. Be sure not to call VARIABLE_VALUE before all result sets have been fetched, or the value you receive may not be valid.

5. Using Oracle Forms Developer and Oracle Reports Developer with Oracle Lite

5.1 Installation, Setup and System Requirements

Using Oracle Forms Developer and Oracle Reports Developer to access an ODBC data source requires the installation of components from the Forms Developer and Reports Developer CD-ROM. In addition, Oracle Lite must be properly installed and configured.

5.1.1 Installation of Forms Developer, Reports Developer and Oracle Lite

Forms Developer, Reports Developer , the Open Client Adapter, Oracle Lite and the ODBC driver for Oracle Lite must be installed on the same machine, since Oracle Lite is a local database which must run on the same machine as the client applications.

First use the Oracle Installer to install Oracle Lite runtime (which includes the ODBC driver) and any other Oracle Lite components you wish to use. *Then* use the Oracle Installer to install Forms Developer, Reports Developer and the Open Client Adapter from the Forms Developer and Reports Developer CD-ROM[1].

5.1.2 Setup for Oracle Lite

When you install Oracle Lite, a default database and an ODBC Data Source (called “POLITE”) are automatically created along with a schema (database user) called “system” and a demo schema called "scott". No additional system setup is required to use Oracle Lite, although it is possible to create and use other databases and/or users (please refer to the Oracle Lite documentation). Additional ODBC data sources may be created to access new databases (this is required if they are in different directories).

5.2 Using Forms Developer and Reports Developer with Oracle Lite

5.2.1 Data Types

The Oracle Lite date types (DATE, TIMESTAMP and TIME) are initially mapped to Forms’ DATE item data type. You should be aware of the following points :

- The DATE type contains “YEAR", "MONTH" and "DAY” fields and is correctly mapped. The is ***no*** time component (unlike Oracle dates).

[1] If Oracle Developer is already installed, there is no need to reinstall it: just install the Open Client Adapter from the Oracle Developer distribution *after* installing Oracle Lite.

- The TIMESTAMP type contains fields "YEAR" to "SECOND" and fractional seconds up to 6 decimal places. When creating a table, the fraction is specified as TIMESTAMP(*frac*), where *frac* is a number between 0 and 6. TIMESTAMP without a *frac* qualifier defaults to TIMESTAMP(6). To handle "YEAR to SECOND" fields, allowing updating, map TIMESTAMP to Forms' DATETIME with a format mask. To handle fractional seconds, map TIMESTAMP to Forms' CHAR type and set the item's *Max Length* and *Query Length* properties to 26. Updating is correctly handled in this case although the user must respect the Oracle Lite input format for timestamps (YYYY-MM-DD HH24:MI:SS.FFFFFF where FFFFFF is the fraction).

- The TIME type contains "HOUR to SECOND" fields. There is no fractional component. Map TIME to Forms' item data type DATETIME, and set the corresponding item's *Format Mask* property to a suitable format (e.g. "HH24:MI:SS"), and set the *Max. Length* property to the corresponding length (e.g. 8), and the *Query Length* property to the same value or greater.

Text and image (LONG and LONG RAW) columns of length greater than 64k cannot be updated from within client-side PL/SQL (this is a PL/SQL limitation). They can be updated via base table image or text items however.

To change the data type of an item (or any other property), go to its property sheet, and choose the desired value for its *Data Type* property (or other property).

5.2.2 Implications of Oracle Lite's Locking Model

Oracle Lite versions 3.0 and above has full transaction support, as described in the Oracle Lite documentation. The standard SQL-92 transaction isolation levels READ COMMITTED, REPEATEABLE READ and SERIALIZABLE are supported. By default, READ COMMITTED is used. However, it is possible to configure it in the ODBC Administrator. That way, it can be set to Read Committed (the default), Repeatable Read or Serializable. To change it, run the ODBC administrator, select the relevant data source (for example, "POLITE") and click the <Configure> button. In the dialogue which appears, use the drop list marked "Default Isolation level:"

Oracle Forms Developer automatically uses the ***Immediate*** locking mode for base table data blocks when running against Oracle Lite. This means a shared lock is taken out on a row (by issuing "SELECT…FOR UPDATE…") as soon as the user starts to modify it on the screen, to ensure no other process can modify it. At the same time, a check is done to see if the row has been changed since it was last read. If it has changed, an error is raised and the row must be re-queried before it can be modified. If lost updates are not a concern, or if the datasource has been configured to use isolation levels REPEATABLE READ or SERIALIZABLE (which garantee against lost updates anyway), then a block-level or form-level ON-LOCK trigger may be created with the PL/SQL code:

```
null;
```

to avoid the “lock record” processing altogether and improve performance. Alternatively, SERIALIZABLE transaction isolation can be requested on a Form by Form basis by setting the Form-level property "Isolation Mode" to Serializable. This will also eliminate "lock record" processing. Please refer to section 2.10.3 for more details.

Note that if one process has already started to modify a row (i.e has locked it) then an attempt by a second process to post an update to the same row will *block* until the first process has ended it's transaction.

5.3 SQL Differences between Oracle and Oracle Lite

5.3.1 Connecting to Different Databases

An Oracle Lite ODBC data source may specify a database name. In this case, the user is always connected to that database when connecting to that data source. However, if the data source does NOT specify a database name, then it is possible to choose the database at connect time using the following syntax for the connect string:

```
ODBC:<DataSourceName>:<Dbname>
```

Alternatively, to change the current database when a connection to a given data source has already been established, execute a 'SET CATALOG <*dbname*>' SQL statement in a client-side trigger or program unit. See the section "Connecting to different databases" in chapter 2.

5.3.2 SQL Constructs

Use of ANSI standard SQL constructs is recommended from applications if they are to run against multiple database systems. However, most Oracle7 SQL dialect is supported by Oracle Lite (e.g. decode, to_char, to_date, to_number, user, sysdate, ||, nvl, etc.), so there are very few restrictions if the application is only to run against Oracle Lite and/or Oracle. Oracle Lite-specific SQL within client-side PL/SQL will fail with syntax errors, unless executed via pass-through functions. Some other differences which you may encounter are:

- User-supplied object names can be longer than 30 characters in Oracle Lite. In Oracle the maximum length of user-supplied names is 30. **However, use of table or column names longer than 30 characters *will not work* with Forms Developer and Reports Developer or SQL*Plus.**
- The following Oracle-specific features are not supported by Oracle Lite: database roles, pseudo column ROWNUM.

You should refer to the Oracle Lite SQL documentation (reference manual or on-line help) for a complete description of the differences between Oracle Lite and Oracle SQL.

5.4 Client-side PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. Forms Developer and Reports Developer use PL/SQL as its programming language. PL/SQL supports Oracle8 SQL, which is a superset of Oracle7 SQL syntax. Since Oracle Lite supports almost all Oracle7 constructs, there are very few restrictions when using PL/SQL with Oracle Lite, provided Oracle8-specific syntax is avoided.

If you need to use Oracle Lite-specific SQL constructs not conforming to Oracle SQL, you need to use a pass-through interface, such as the interface the EXEC_SQL package provides. The EXEC_SQL package is documented in the Procedure Builder on-line help and documentation.

5.4.1 PL/SQL and Oracle Lite Data Types

The following PL/SQL datatypes may be used freely in client-side PL/SQL program units:

- NUMBER and its subtypes DECIMAL (DEC), DOUBLE PRECISION, FLOAT, INTEGER
- (INT), NUMERIC, REAL, and SMALLINT.
- CHAR and its subtypes CHARACTER and STRING. Note that a CHAR value in client-side PL/SQL can have a maximum length of 32767, but a CHAR column in the database has a maximum length of 2048.
- VARCHAR and VARCHAR2. Note that a VARCHAR value in client-side PL/SQL can have a maximum length of 32767, but a VARCHAR column in the database has a maximum length of 2048.
- LONG. This is similar to VARCHAR, except that while it is limited to a maximum size of 32767 on the client, a database value can be up to 2 Gigabytes in size.
- RAW. A RAW value in client-side PL/SQL can have a maximum length of 32767, but a RAW column in the database has a maximum length of 2048.
- LONG RAW. Analogous to LONG, but for RAW data.
- BOOLEAN
- DATE

Oracle Lite data types can be mapped directly to their equivalent PL/SQL data types, with the following exceptions and caveats:

Oracle Lite Datatype	**PL/SQL Datatype**
TINYINT	NUMBER(3)
SMALLINT	NUMBER(5)
INT	NUMBER(10)

BIGINT	FLOAT or NUMBER(19)
LONG VARCHAR	LONG
DATE TIMESTAMP TIME	DATE (Mapping to CHAR is required to handle fractional seconds)

5.4.2 Supported PL/SQL Statements and Functions

The following PL/SQL statements and attributes are supported with Oracle Lite:

COMMIT	LOOP
CLOSE	OPEN
CURSOR	ROLLBACK
DECLARE	ROLLBACK TO
DELETE	SAVEPOINT
FETCH	SELECT INTO
GOTO	UPDATE
INSERT	

%FOUND	%NOTFOUND
%ROWCOUNT	%ISOPEN
%TYPE	%ROWTYPE

as well as most exceptions and all looping and conditional constructs.

Oracle Lite (versions 2.3 and above) supports almost all Oracle7 SQL functions (the Oracle Lite documentation gives an exact comparison of Oracle Lite and Oracle7 SQL). SQL functions common to both Oracle Lite SQL and Oracle8 SQL (by name, number and type of arguments) can be used in SQL statements within PL/SQL. Oracle8 SQL is a superset of Oracle7 SQL.

5.4.3 SYSDATE and USER on the client, DUAL table

The client-side constructs SYSDATE and USER reference the dual table in the database. As in Oracle, the dual table always exists in Oracle Lite, so these constructs, and any SELECT statements referencing the dual table, can be used freely in PL/SQL without worrying about manually creating the dual table.

6. Using Oracle Forms Developer and Oracle Reports Developer with Microsoft SQL Server

6.1 Installation, Setup and System Requirements

Using Forms Developer and Reports Developer to access an ODBC data source requires the installation of components from Oracle, Microsoft SQL Server and the ODBC driver vendor on the client machine, and proper configuration of Microsoft SQL Server.

6.1.1 Setup for Forms Developer and Reports Developer

Forms Developer and Reports Developer and the Open Client Adapter must be installed on the client. In addition, Forms Developer and Reports Developer include ODBC drivers from INTERSOLV that allow access to Microsoft SQL Server databases. If you choose to use one of these drivers, you should run the Oracle Installer to install it from the Forms Developer and Reports Developer CD-ROM. For more information on this process, please refer to the *Oracle Forms Developer and Oracle Reports Developer Installation Guide*.

6.1.2 Setup for Microsoft SQL Server

6.1.2.1 Client Requirements and Setup

The INTERSOLV DataDirect driver for Microsoft SQL Server uses the Microsoft SQL Server DB-Library interface to access SQL Server databases. You must install the Microsoft SQL Server client-side net libraries before configuring the ODBC driver and accessing the data source.

If you will be using an ODBC driver other than that provided with Forms Developer and Reports Developer , install it according to the instructions from the supplier, and install any necessary database access software.

6.1.2.2 Server Requirements and Setup

If possible, we recommend configuring your Microsoft SQL Server database to use case-insensitive sort order. This enables the use of both upper, lower and mixed case names for database objects. If your Microsoft SQL Server database cannot conveniently be configured to use case-insensitive sort order, try to use upper case names for database objects (table, columns, views, and stored procedures). User names should be also created in upper case if possible, to remove the need to quote them if used in client-side PL/SQL. If it is necessary to use lower or mixed case object names, you will need to take certain precautions during application development. These are detailed in the appendix ppendix A: Case Sensitivity Issues.

If you will be calling Transact-SQL stored procedures from Forms Developer and Reports Developer , you should raise the procedure cache server configuration

parameter from its default of 20 to 40. This parameter controls how much of the cache is used for stored procedures, and how much is used for data.

6.2 Microsoft ODBC Driver Versus INTERSOLV ODBC Driver

Oracle provides the INTERSOLV ODBC Driver for Microsoft SQL Server on the Forms Developer and Reports Developer CD-ROM. However, Microsoft also provides its own ODBC driver with its SQL Server product. There are some differences between the two drivers that are documented here to help you choose which driver to use.

Characteristic	Microsoft Driver	INTERSOLV Driver
Open Cursors	The Microsoft driver, when used with the OCA, uses Server Cursors for all select statements. This allows the driver to have multiple active statements per connection when running against SQL Server versions 6.0 and above. This means all COMMITs and ROLLBACKs are safe, and the chance of locking yourself out in an application is vasty reduced.	The INTERSOLV driver uses Client Cursors, which means it can only have one active statement per connection. When an application needs to open another cursor, another connection to the database is created by the OCA. (see the Appendix F: Notes on drivers supporting only one active statement per connection Appendix). COMMITs and ROLLBACKs may not be transactionally pure in case of failure, and there is a chance of deadlock if two different SQL statements modify rows on the same data page without committing in between .
Handling of Long Data	The Microsoft driver can insert LONG or LONG RAW values greater than 64K.	The INTERSOLV driver truncates all LONG and LONG RAW data to 64K

6.3 Using Oracle Forms Developer and Oracle Reports Developer - Forms with Microsoft SQL Server

6.3.1 Data Types

The NCHAR, NVARCHAR, SENSITIVITY, and SENSITIVITY_BOUNDARY data types are not supported.

When creating a data block based on a table or view, all other Microsoft SQL Server data types are automatically mapped to Forms items of the equivalent Oracle data type. This mapping is usual satisfactory but you may wish to override it for DATETIME and SMALLDATIME columns which are mapped to Forms' DATE items. The data displayed in this case is limited to days, months and years. You may want to change the item's data type depending on what portion of the time information you wish to display and/or edit.

- To display and update only days, months and years, the mapping to DATE items is sufficient. The default format mask in this case depends on the setting of NLS_LANG. It is *dd-mon-yy* for territory=America.
- To display and update hours, minutes and seconds (instead of or in addition to days, months and years) use Forms' DATETIME data type. Specify a format mask to display the fields of interest.
- If you need to display and/or update fractional seconds, use CHAR items with length 26. This will display the data in a canonical date format, which should be respected if the user wishes to edit the data.

An item based on a Microsoft SQL Server TIMESTAMP column must be query only and of data type CHAR. The value will be fetched as a hexadecimal string, and attempting to update such a column will fail.

To change the data type of an item, go to its property sheet, and choose the desired value for its ***Data Type*** property.

6.3.2 Implications of Microsoft SQL Server's Locking Model

Microsoft SQL Server SQL does not support a lock time-out feature to return an error if a connection attempting to get a write lock on a page is blocked by another connection. Instead, a connection attempting to get a lock on a resource waits until the locks already held on that resource are released. If two or more Forms applications are trying to update the same page in the database, they will hang until the locks are released. If they are on the same workstation (two instances of Forms runtime, for example) a deadlock condition might completely lock up the machine. For this reason, we recommend using Forms' ***Delayed*** locking mode, and will default to it unless you specify otherwise.

6.4 Using Oracle Forms Developer and Oracle Reports Developer - Reports with Microsoft SQL Server

SQL Server does not support duplicate column references in the ORDER BY clause (see the "Additional Restrictions" section in this chapter). This can sometimes cause problems for Reports. To avoid this problem, there should be no ORDER BY on a column which is also a break column or a target of a link in a child query.

SQL Server does not support implicit type conversion as Oracle does, so you cannot compare a character value to a number value without first converting one of the two. When creating a link, make sure the proper conversion is specified in the child column in the query.

For example, if you have a SQL statement like:

```
SELECT ename FROM emp WHERE empno >= :ID_EMPNO
```

Where empno is a number field, make sure that :ID_EMPNO is also a number field.

6.5 SQL Differences between Oracle and Microsoft SQL Server

6.5.1 Connecting to Different Databases

A Microsoft SQL Server server can control multiple databases. Each user has a default database, to which they are connected at logon. A user can switch to other databases controlled by the same server by executing a 'USE <*dbname*>' SQL statement.

In Forms Developer and Reports Developer , there are two ways to connect to a database other than the user's default. The database name can be specified at connect time, as in:

```
ODBC:<DataSourceName>:<Dbname>
```

Alternatively, execute a 'USE <*dbname*>' SQL statement in a client-side trigger or program unit. . See the section "Connecting to different databases" in chapter 2.

6.5.2 SQL Constructs

Use of ANSI standard SQL constructs is recommended from Forms Developer and Reports Developer applications when using Microsoft SQL Server. Microsoft SQL Server-specific SQL will fail with syntax errors, unless used via pass-through functions. In particular, the following Microsoft SQL Server-specific SQL constructs are not supported:

SELECT… HOLDLOCK

SELECT…INTO(although Oracle-style SELECT…INTO statements **are** supported within PL/SQL)

SELECT… FOR BROWSE

COMPUTE

When issuing an INSERT statement from within Forms Developer and Reports Developer , always include the INTO clause, as in INSERT INTO <*table*> VALUES…

6.5.3 SQL Functions and Operators

Only those SQL functions and operators common to Oracle SQL and Microsoft SQL Server Transact-SQL can be used in SQL statements, except via pass-through functions. These common functions include:

ABS	RTRIM
ASCII	SIGN
AVG	SIN
COS	SOUNDEX
COUNT	SQRT
LOWER	SUM
LTRIM	SYSDATE
MAX	TAN
MIN	UPPER

ROUND USER

6.5.4 Comparison Operators

The following comparison operators are either not supported, or have an Oracle or ANSI equivalent which should be used from within an application:

Operator	**Equivalent**
!>	<=
!<	>=
LIKE ‘a[x-z]’	Not Supported
LIKE ‘a[^x-z]’	Not Supported
= NULL	IS NULL
!= NULL	IS NOT NULL
!> ANY	<= ANY
!< ANY	>= ANY
!> ALL	<= ALL
!< ALL	>= ALL

6.5.5 Arithmetic Operators

The modulo operator (%) can not be used in SQL statements within Forms Developer and Reports Developer .

6.5.6 String Operators

Microsoft SQL Server uses “+” for string concatenation, whereas Oracle uses “||”. Reports and Graphics allow the use of “+” for string concatenation in queries, but PL/SQL does not permit its use. As a workaround, you can select the columns individually, and concatenate them on the client side.

6.5.7 Bit Operators

The &, |, ^, and ~ operators cannot be used in SQL statements within Forms Developer and Reports Developer .

6.6 Microsoft SQL Server Transactions and Locking

Forms applications run with AUTO-COMMIT mode OFF. All SQL statements are executed in the context of transactions, which run in unchained mode. A SQL statement issued immediately after connecting to the database or issued after a commit or rollback implicitly starts a transaction.

Microsoft SQL Server uses exclusive locks for data modifications and shared locks for non-update or read operations. The granularity of locking is page-level. Page-level locks are escalated to table level locks if more than a specified percentage of pages are locked by a transaction.

SELECT statements obtain shared locks on pages. This prevents other statements from obtaining an exclusive lock on those pages. Hence a SELECT statement blocks other data modification operations as long as the transaction that includes

the SELECT statement does not commit or roll back. There is no lock time-out feature in Microsoft SQL Server. This means that applications waiting for a lock on a resource will block until the locks held on that resource are released. This might cause a Windows 3.1 client system to appear to hang until the lock is attained.

6.7 Handling of DDL statements

SQL Server only allows the execution of DDL statements (such as CREATE TABLE…) when running with AUTO-COMMIT mode ON. If auto-commit mode is OFF and the Open Client Adapter is requested to execute a DDL command, it will temporarily set the mode to ON in order to process the command without provoking an error. This is done to allow the execution of DDL commands from tools, such as SQL Plus and Forms applications, where AUTO-COMMIT mode is usually OFF. An important consequence is that DDL statements cause an implicit COMMIT (as is the case against Oracle). The exact processing of the execution of DDL statements (when AUTO-COMMIT mode is OFF) is as follows:

1. If a transaction is pending, COMMIT
2. Set auto-commit mode ON
3. Execute the DDL
4. Set auto-commit mode back to OFF

6.8 Client-side PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. Forms Developer and Reports Developer use PL/SQL as its programming language. Certain restrictions and limitations will apply when using PL/SQL with Microsoft SQL Server via ODBC.

PL/SQL is case insensitive. To use PL/SQL effectively, either the database has to be configured to use case-insensitive sort order, or object names have to be created in upper case, or object (table and column) names used in PL/SQL must be surrounded with double quotes (" "). The PL/SQL compiler will then preserve the case. For example, if you have a column Ename in a table Emp in a case-sensitive database, the following statement will work within PL/SQL :

```
Cursor C IS SELECT "Ename" FROM "Emp"
```

Use of standard SQL constructs (as opposed to Oracle or Microsoft SQL Server-specific) is recommended when using PL/SQL with Microsoft SQL Server databases.

If you need to use SQL constructs not conforming to Oracle SQL, you need to use a pass-through interface, such as the interface the EXEC_SQL package provides. The EXEC_SQL package is documented in the Procedure Builder on-line help and documentation.

6.8.1 PL/SQL and Microsoft SQL Server Data Types

The following PL/SQL data types can be used freely in client-side PL/SQL program units:

- NUMBER and its subtypes DECIMAL (DEC), DOUBLE PRECISION, FLOAT, INTEGER (INT), NUMERIC, REAL, and SMALLINT.
- CHAR and its subtypes CHARACTER and STRING. Note that a CHAR value in client-side PL/SQL can have a maximum length of 32767, but a CHAR column in the database has a maximum length of 255.
- VARCHAR and VARCHAR2. Note that a VARCHAR value in client-side PL/SQL can have a maximum length of 32767, but a VARCHAR column in the database has a maximum length of 255.
- LONG. This is similar to VARCHAR, except that while it is limited to a maximum size of 32767 on the client, the equivalent SQL Server data type, TEXT, can be up to 2 Gigabytes in size.
- RAW. A RAW value in client-side PL/SQL can have a maximum length of 32767, but a RAW column in the database has a maximum length of 255.
- LONG RAW. Analogous to LONG, but for RAW data.
- BOOLEAN
- DATE

Microsoft SQL Server data types can be mapped to their equivalent PL/SQL data type, with the following exceptions and caveats:

Microsoft SQL Server Data Type	**PL/SQL Data Type**
TINYINT	SMALLINT
BIT	RAW(1)
TEXT	LONG (Max length < 32768)
IMAGE	RAW (Max length < 32768)
BINARY	RAW
VARBINARY	RAW
DATETIME	DATE
SMALLDATETIME	DATE
MONEY	FLOAT
SMALLMONEY	FLOAT

NCHAR, NVARCHAR, SENSITIVITY and SENSITIVITY_BOUNDARY data types are not supported.

6.8.2 Supported PL/SQL Statements and Functions

The following PL/SQL statements and attributes are supported with Microsoft SQL Server:

COMMIT	LOOP
CLOSE	OPEN
CURSOR	ROLLBACK
DECLARE	ROLLBACK TO[1]
DELETE	SAVEPOINT[1]
FETCH	SELECT INTO
GOTO	UPDATE
INSERT	

%FOUND	%NOTFOUND
%ROWCOUNT	%ISOPEN

as well as all exception, looping and conditional constructs

Only SQL functions common to both Microsoft SQL Server Transact-SQL and Oracle SQL (by name, number and type of arguments) can be used in SQL statements within procedures. All other PL/SQL functions can be used in the body of procedures only.

6.8.3 Calling Stored Procedures from Client-side PL/SQL

Stored procedures can be called from client-side PL/SQL program units and triggers. Stored Procedures with IN, OUT, IN/OUT parameters or return values are fully supported. Result sets cannot be retrieved when calling a stored procedure directly from PL/SQL, but you can use the EXEC_SQL package to execuate a stored procedure and get result sets back. See the chapter "Calling Stored Procedures through the Oracle Open Client Adapter" for an example.

For example, this Microsoft SQL Server stored procedure:

```
create proc testproc @arg1 int, @outarg2 float out
as
begin
  select @outarg2=sum(sal) from emp where deptno = @arg1
end
```

could be called from a client-side program unit as follows:

```
FUNCTION compute_deptsal (Param1 IN NUMBER) RETURN FLOAT IS
  Param2 FLOAT;
  retVal INTEGER;
BEGIN
  retVal := testproc (Param1, Param2);
  RETURN (Param2);
END;
```

[1] But only if the ODBC driver supports multiple active statements per connection.

Note: All Microsoft SQL Server functions must be called from PL/SQL as functions, because they all return an integer value (even without an implicit return statement in the procedure body).

Note: Microsoft SQL Server stored procedure arguments are always of type IN or INOUT. There are no OUTPUT-only arguments.

Note: Due to what appears to be a bug in the INTERSOLV driver for Microsoft SQL Server, the return value returned by a Microsoft SQL Server 4.2 stored procedure is always NULL. However, you still must specify it when you call the stored procedure from PL/SQL. Return values are returned correctly for SQL Server Version 6.0 and above.

6.9 Additional Restrictions

ORDER BY Within ORDER BY clauses, Microsoft SQL Server permits positional references to columns only if the columns are explicitly listed.

For example,

```
SELECT * FROM TABLE_NAME ORDER BY <column_number>
```

will raise an error.

Suggestion: Replace the "*" with explicit column names, or substitute the <column_number> with a <column_identifier>.

In addition, Microsoft SQL Server does not permit duplicate column references in the order by clause, such as:

```
SELECT C1, C2, C3 from TABLE_NAME ORDER BY C2, 2
```

Global Variables Microsoft SQL Server global variables cannot be used with Forms Developer and Reports Developer .

Transact SQL Statements Use of the following Transact SQL statements is restricted:

- The PRINT statement cannot be used with PL/SQL, although I/O routines can be called from within user exits.
- The WAIT FOR {DELAY 'time' | TIME 'time' | ERROREXIT | PROCESSEXIT | MIRROR EXIT} statement cannot be used within PL/SQL.
- The WHERE CURRENT OF <cursor> clause is not supported with Microsoft SQL Server (or indeed with any ODBC data source).
- The FOR UPDATE… clause in SELECT statements may be used, but is automatically removed by the OCA before the statement is passed to the database, because it is not supported by MS SQL Server.

Using IMAGE and TEXT data with the Microsoft ODBC Driver Some versions of the Microsoft ODBC driver for SQL Server would incorrectly return data for IMAGE or TEXT columns which actually contained NULL data when using server cursors. This problem has been fixed as of Version 3.00.0075 of the Microsoft ODBC Driver. You should acquire this version or above (you can find it at http://www.microsoft.com).

7. Using Oracle Forms Developer and Oracle Reports Developer with Sybase

7.1 Installation, Setup and System Requirements

Using Forms Developer and Reports Developer to access an ODBC data source requires installation of components from Oracle, Sybase and the ODBC driver vendor on the client machine, and proper configuration of the Sybase server.

7.1.1 Setup for Oracle Forms Developer and Oracle Reports Developer

Forms Developer, Reports Developer and the Open Client Adapter must be installed on the client. In addition, Forms Developer and Reports Developer include ODBC drivers from INTERSOLV that allow access to Sybase databases. Use the Oracle Installer to install them from the Forms Developer and Reports Developer CD-ROM to the client. For more information on this process, refer to the *Oracle Forms Developer and Oracle Reports Developer Installation Guide.*

7.1.2 Setup for Sybase

7.1.2.1 Client Requirements and Setup

The INTERSOLV DataDirect driver for Sybase System 10 uses the Sybase CT-Library interface to access System 10 or System 11. You must install the Sybase Open Client Library/C version 10.0.2 or later, and the corresponding Net-Library, before configuring the ODBC driver and accessing the data source.

For System 10, you will need to set the client environment variable SYBASE to the directory where the client libraries are installed. Add the server name entries to the SQL.INI file in the $*SYBASE*\INI directory. Also make sure that the client library DLL's (contained in $*SYBASE*\DLL or $*SYBASE*\BIN) are on the path, either by editing the PATH environment variable, or copying them to a directory already on the path.

If you will be using an ODBC driver other than that provided with Forms Developer and Reports Developer , install it according to the instructions from the supplier, and install any necessary database access software.

7.1.2.2 Server Requirements and Setup

If possible, we recommend configuring your Sybase database to use case-insensitive sort order. This enables the use of both upper, lower and mixed case names for database objects. If your Microsoft SQL Server database cannot conveniently be configured to use case-insensitive sort order, try to use upper case names for database objects (table, columns, views, and stored procedures). User names should also be created in upper case if possible, to remove the need to quote them if used in client-side PL/SQL. If it **is** necessary to use lower or mixed case object names, you will need to take certain precautions during application

development. These are detailed in the appendix ppendix A: Case Sensitivity Issues.

If you will be calling Transact-SQL stored procedures from Forms Developer and Reports Developer , you should raise the procedure cache server configuration parameter from its default of 20 to 40. This parameter controls how much of the cache is used for stored procedures, and how much is used for data.

7.2 Using Oracle Forms Developer and Oracle Reports Developer - Forms with Sybase

7.2.1 Data Types

The NCHAR, NVARCHAR, SENSITIVITY, and SENSITIVITY_BOUNDARY data types are not supported.

All other Sybase data types are automatically mapped to their equivalent Oracle data types with the exception of DATETIME and SMALLDATIME which are mapped to Forms' DATE type. The data displayed in this case is limited to days, months and years. You may want to change the mapping depending on what portion of the time types you want to edit.

- To display and update only days, months and years, the mapping to DATE is sufficient. The default format mask in this case is *dd-mon-yy*.
- To display and update the hours, minutes and seconds, use Forms' DATETIME data type. Specify a format mask to display the fields of interest.
- If you need to display and update fractional seconds, map these data types to CHAR(26). This will display the data in a canonical date format, which should be respected if the user wishes to edit the data.

An item based on a Sybase TIMESTAMP column must be query only and of data type CHAR. The value will be fetched as a hexadecimal string, and attempting to update such a column will fail.

To change the data type of an item, go to its property sheet, and choose the desired value for its ***Data Type*** property.

Text and image type columns of length greater than 64k will be truncated to 64k if using certain combinations of databases and drivers. The following table describes which driver/database combinations support full access to long data from Forms Developer and Reports Developer :

Database	**Driver**	**Longs > 64K**
Sybase SQL Server 4.x	INTERSOLV	No
Sybase System 10, 11	INTERSOLV	Yes

7.2.2 Implications of Sybase's Locking Model

Sybase SQL does not support a lock time-out feature to return an error if a connection attempting to get a write lock on a page is blocked by another connection. Instead, a connection attempting get a lock on a resource waits until the locks already held on that resource are released. If two or more Forms applications are trying to update the same page in the database, they will hang until the locks are released. If they are on the same workstation (two instances of Forms runtime, for example) a deadlock condition might completely lock up the machine. For this reason, we recommend using Forms' ***Delayed*** locking mode, and will default to it unless you specify otherwise.

7.3 Using Oracle Forms Developer and Oracle Reports Developer - Reports with Sybase

Sybase does not support duplicate column references in the ORDER BY clause (see the "Additional Restrictions" section in this chapter). This can sometimes cause problems for Reports. To avoid this problem, there should be no ORDER BY on a column which is also a break column or a target of a link in a child query.

Sybase does not support implicit type conversion like Oracle does, so you cannot compare a character value to a number value without first converting one of the two. When creating a link, make sure the proper conversion is specified in the child column in the query.

For example, if you have a SQL statement like:

```
SELECT ename FROM emp WHERE empno >= :ID_EMPNO
```

Where empno is a number field, make sure that :ID_EMPNO is also a number field.

7.4 SQL Differences between Oracle and Sybase

7.4.1 Connecting to Different Databases

A Sybase server can control multiple databases. Each user has a default database, to which they are connected at logon. A user can switch to other databases controlled by the same server by executing a 'USE <*dbname*>' SQL statement.

In Forms Developer and Reports Developer , there are two ways to connect to a database other than the user's default. The database name can be specified at connect time, as in:

```
ODBC:<DataSourceName>:<Dbname>
```

Alternatively, execute a 'USE <*dbname*>' SQL in a client-side trigger or program unit. See the section "Connecting to different databases" in chapter 2.

7.4.2 SQL Constructs

Use of ANSI standard SQL constructs is recommended from applications when using Sybase. Sybase-specific SQL will fail with syntax errors, unless used via

pass-through functions. In particular, the following Sybase-specific SQL constructs are not supported:

SELECT… HOLDLOCK
SELECT… INTO
SELECT… FOR BROWSE
COMPUTE

When issuing an INSERT statement from within Forms Developer and Reports Developer , always include the INTO clause, as in INSERT INTO <*table*> VALUES…

7.4.3 SQL Functions and Operators

Only those SQL functions and operators common to Oracle SQL and Sybase Transact-SQL can be used in SQL statements, except via pass-through functions. These common functions include:

ABS	RTRIM
ASCII	SIGN
AVG	SIN
COS	SOUNDEX
COUNT	SQRT
LOWER	SUM
LTRIM	SYSDATE
MAX	TAN
MIN	UPPER
ROUND	USER

7.4.4 Comparison Operators

The following comparison operators are either not supported, or have an Oracle or ANSI equivalent which should be used from within an application:

Operator	**Equivalent**
!>	<=
!<	>=
LIKE ‘a[x-z]’	Not Supported
LIKE ‘a[^x-z]’	Not Supported
= NULL	IS NULL
!= NULL	IS NOT NULL
!> ANY	<= ANY
!< ANY	>= ANY
!> ALL	<= ALL
!< ALL	>= ALL

7.4.5 Arithmetic Operators

The modulo operator (%) can not be used in SQL statements within Forms Developer and Reports Developer .

7.4.6 String Operators

Sybase uses "+" for string concatenation, whereas Oracle uses "||". Reports and Graphics allow the use of "+" for string concatenation in queries, but PL/SQL does not permit its use. As a workaround, you can select the columns individually, and concatenate them on the client side.

7.4.7 Bit Operators

The &, |, ^, and ~ operators can not be used in SQL statements within Forms Developer and Reports Developer .

7.5 Sybase Transactions and Locking

Forms applications run with AUTO-COMMIT mode OFF. All SQL statements are executed in the context of transactions, which run in unchained mode. A SQL statement issued immediately after connecting to the database or issued after a commit or rollback implicitly starts a transaction.

Sybase uses exclusive locks for data modifications and shared locks for non-update or read operations. The granularity of locking is page-level. Page-level locks are escalated to table level locks if more than a specified percentage of pages are locked by a transaction.

SELECT statements obtain shared locks on pages. This prevents other statements from obtaining an exclusive lock on those pages. Hence a SELECT statement blocks other data modification operations as long as the transaction that includes the SELECT statement does not commit or roll back. There is no lock time-out feature in Sybase. This means that applications waiting for a lock on a resource will block until the locks held on that resource are released. This might cause a Windows 3.1 client system to appear to hang until the lock is attained.

7.6 Handling of DDL statements

Sybase only allows the execution of DDL statements (such as CREATE TABLE…) when running with AUTO-COMMIT mode ON. If auto-commit mode is OFF and the Open Client Adapter is requested to execute a DDL command, it will temporarily set the mode to ON in order to process the command without provoking an error. This is done to allow the execution of DDL commands from tools, such as SQL Plus and Forms applications, where AUTO-COMMIT mode is usually OFF. An important consequence is that DDL statements cause an implicit COMMIT (as is the case against Oracle). The exact processing of the execution of DDL statements (when AUTO-COMMIT mode is OFF) is as follows:

1. If a transaction is pending, COMMIT
2. Set auto-commit mode ON
3. Execute the DDL
4. Set auto-commit mode back to OFF

7.7 Client-side PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. Forms Developer and Reports Developer use PL/SQL as its programming language. Certain restrictions and limitations will apply when using PL/SQL with Sybase via ODBC.

PL/SQL is case insensitive. To use PL/SQL effectively, either the database has to be configured to use case-insensitive sort order, or object names have to be created in upper case, or object (table and column) names used in PL/SQL must be surrounded with double quotes (" "). The PL/SQL compiler will then preserve the case. For example, if you have a column Ename in a table Emp in a case-sensitive database, the following statement will work within PL/SQL :

```
Cursor C is SELECT "Ename" from "Emp";
```

Use of standard SQL constructs (as opposed to Oracle or Sybase-specific) is recommended when using PL/SQL with Sybase databases.

If you need to pass non-Oracle specific SQL constructs, you need to use a pass-through interface, such as the interface the EXEC_SQL package provides. The EXEC_SQL package is documented in the Procedure Builder on-line help and documentation.

7.7.1 PL/SQL and Sybase Data Types

The following PL/SQL data types can be used freely in client-side PL/SQL program units:

- NUMBER and its subtypes DECIMAL (DEC), DOUBLE PRECISION, FLOAT, INTEGER (INT), NUMERIC, REAL, and SMALLINT.
- CHAR and its subtypes CHARACTER and STRING. Note that a CHAR value in client-side PL/SQL can have a maximum length of 32767, but a CHAR column in the database has a maximum length of 255.
- VARCHAR and VARCHAR2. Note that a VARCHAR value in client-side PL/SQL can have a maximum length of 32767, but a VARCHAR column in the database has a maximum length of 2000.
- LONG. This is similar to VARCHAR, except that while it is limited to a maximum size of 32767 on the client, a database value can be up to 2 Gigabytes in size.
- RAW. A RAW value in client-side PL/SQL can have a maximum length of 32767, but a RAW column in the database has a maximum length of 255.
- LONG RAW. Analogous to LONG, but for RAW data.
- BOOLEAN
- DATE

Sybase data types can be mapped to their equivalent PL/SQL data type, with the following exceptions and caveats:

Sybase Data Type	**PL/SQL Data Type**
TINYINT	SMALLINT
BIT	RAW(1)
TEXT	LONG (Max length < 32768)
IMAGE	RAW (Max length < 32768)
BINARY	RAW
VARBINARY	RAW
DATETIME	DATE
SMALLDATETIME	DATE
MONEY	FLOAT
SMALLMONEY	FLOAT

NCHAR, NVARCHAR, SENSITIVITY and SENSITIVITY_BOUNDARY data types are not supported.

7.7.2 Supported PL/SQL Statements and Functions

The following PL/SQL statements and attributes are supported with Sybase:

COMMIT	LOOP
CLOSE	OPEN
CURSOR	ROLLBACK
DECLARE	ROLLBACK TO[1]
DELETE	SAVEPOINT[1]
FETCH	SELECT INTO
GOTO	UPDATE
INSERT	

%FOUND	%NOTFOUND
%ROWCOUNT	%ISOPEN

as well as all exception, looping and conditional constructs

Only SQL functions common to both Sybase Transact-SQL and Oracle SQL (by name, number and type of arguments) can be used in SQL statements within procedures. All other PL/SQL functions can be used in the body of procedures only.

7.7.3 Calling Stored Procedures from Client-side PL/SQL

Stored procedures can be called from client-side PL/SQL program units and triggers. Stored Procedures with IN, OUT, IN/OUT parameters or return values

[1] But only if the ODBC driver supports multiple active statements per connection.

are fully supported. . Result sets cannot be retrieved when calling a stored procedure directly from PL/SQL, but you can use the EXEC_SQL package to execute a stored procedure and get result sets back. See the chapter "Calling Stored Procedures Using The EXEC_SQL Package" for an example.

For example, this Sybase stored procedure:

```
create proc testproc @arg1 int, @outarg2 float out
as
begin
  select @outarg2=sum(sal) from emp where deptno = @arg1
end
```

could be called from a client-side program unit as follows:

```
FUNCTION compute_deptsal (Param1 IN NUMBER) RETURN FLOAT IS
  Param2 FLOAT;
  retval INTEGER;
BEGIN
  retval := testproc (Param1, Param2);
  RETURN (Param2);
END;
```

Note: All Sybase stored procedures must be called as functions from PL/SQL, as they can all return an integer value.

Note: Sybase stored procedure arguments are always of type IN or INOUT. There are no OUTPUT-only arguments.

Note: Due to what appears to be a bug in the INTERSOLV driver for Sybase, the return value returned by a Sybase stored procedure is always NULL. However, you still must specify it when you call the stored procedure from PL/SQL.

7.8 Additional Restrictions

ORDER BY Within ORDER BY clauses, Sybase permits positional references to columns only if the columns are explicitly listed.

For example,

```
SELECT * FROM TABLE_NAME ORDER BY <column_number>
```

will raise an error.

Suggestion: Replace the "*" with explicit column names, or substitute the <column_number> with a <column_identifier>.

In addition, Sybase does not permit duplicate column references in the order by clause, such as:

```
SELECT C1, C2, C3 from TABLE_NAME ORDER BY C2, 2
```

Global Variables Sybase global variables cannot be used with Forms Developer and Reports Developer .

Transact SQL Statements Use of the following Transact SQL statements is restricted:

- The PRINT statement cannot be used with PL/SQL, although I/O routines can be called from within user exits.

- The WAIT FOR {DELAY 'time' | TIME 'time' | ERROREXIT | PROCESSEXIT | MIRROR EXIT} statement cannot be used with PL/SQL.
- The FOR UPDATE ... WHERE CURRENT OF statement does not work with Sybase.

8. Using Oracle Forms Developer and Oracle Reports Developer with Oracle Rdb

8.1 Installation, Setup and System Requirements

Using Forms Developer and Reports Developer to access an ODBC data source requires the installation of components from Forms Developer and Reports Developer , Oracle Rdb and the ODBC driver vendor on the client machine, and proper configuration of the Oracle Rdb server.

8.1.1 Setup for Oracle Forms Developer and Oracle Reports Developer

Forms Developer, Reports Developer and the Open Client Adapter must be installed on the client. In addition, Forms Developer and Reports Developer include the Oracle ODBC driver for Rdb that allows access to Oracle Rdb databases. If you choose to use this driver, you should run the Oracle Installer to install it from the Forms Developer and Reports Developer CD-ROM. For more information on this process, please refer to the *Oracle Forms Developer and Oracle Reports Developer Installation Guide.*

8.1.2 Setup for Oracle Rdb

8.1.2.1 Client Requirements and Setup

When you install the Oracle ODBC driver for Rdb from the Forms Developer and Reports Developer CD-ROM, all necessary client software is installed. No additional system setup is required to use Rdb.

The Rdb server may be accessed via TCP/IP, SPX/IPX or DecNET networking. Consult your Rdb documentation for supported networking software vendors.

If you will be using an ODBC driver other than that provided with Forms Developer and Reports Developer , install it according to the instructions from the supplier, and install any necessary database access software.

8.1.2.2 ODBC Driver Configuration

The generic steps required to configure an ODBC data source are detailed in the ODBC Setup section of the Getting Started chapter. In the case of the Oracle ODBC driver for Rdb you need to additionally edit the driver's configuration information in the RDBODBC.INI file that is found in the Windows directory (usually C:\WINDOWS or C:\Win95.). Set or modify the following parameters:

NetworkBufferSize=5000	Use larger network packets for improved performance.
RemoveControlChars=YES	Permits blank spaces in SQL statements. Rdb does not ordinarily accept control characters such as additional spaces in SQL statement.

FetchAhead=NO	Multiple SQL statements cannot be active when fetch ahead is set to YES, and applications will usually have multiple open cursors.
LockTimeOut=10	This will allow applications to time out instead of hanging indefinitely in the case of a deadlock. If your applications time out even in the absence of a potential deadlock, increase this value.

An example RDBODBC.INI might look like this:

```
[Oracle ODBC Driver for Rdb]
Transport=2=tcp/ip,1=decnet
Client Logging=0
Driver Logging=0
LockTimeOut=10
FetchAhead=NO
NetworkBufferSize=5000
RemoveControlChars=YES
Proxy Access=0
```

Note: Depending on the version of the Oracle ODBC driver for Rdb, this information may also be recorded in the file WIN.INI, also found in the Windows directory. You should check this file for an "Oracle ODBC driver for Rdb" section, and if it is present, edit the parameters to match those of RDBODBC.INI.

8.1.2.3 Server Requirements and Setup

Install Rdb on the server, or identify the Rdb server that you will be accessing. The Rdb ODBC driver accesses Rdb via the SQL/Services API. Rdb and SQL/Services must be installed and running on the server to access Rdb from Forms Developer and Reports Developer .

8.2 Connecting to Different Databases

An Rdb server can control multiple data files. When users configure the Oracle ODBC driver for Rdb, they indicate which data file is attached to at logon. You can connect to a database other than the default. The data file name can be specified at connect time, as in:

```
ODBC:<DataSourceName>:<FileSpec>
```

For example, if your ODBC data source name is `Rdb61`, and the default data file is `personnel.rdb`, your normal connect string is:

```
ODBC:Rdb61
```

However, to attach data file `qa.rdb` at logon, you specify the following connect string:

```
ODBC:Rdb61:qa.rdb
```

Alternatively, you can switch to other databases controlled by the same server by executing the attach SQL statement in a client-side trigger or program unit. See the section "Connecting to different databases" in chapter 2.

8.3 Using Oracle Forms Developer with Oracle Rdb

8.3.1 Form, Data Block and Item properties

Oracle Forms runtime automatically adjusts Form and Data block level properties according to the data source. Against RDB, the block-level Key Mode is set to Unique and rows are identified using the DBKEY values, so it is *not* usually necessary to mark items based on primary key columns as Primary Key.

However, please note that due to a limitation in the current ODBC driver for RDB, the use of DBKEY is not possible for data blocks containing large binary items based on LIST OF BYTE VARYING columns. Typically these will be image, sound or OLE Container items. Forms containing such items should attach the *OPENDB* library and call the `opendb.init_form` procedure, which will automatically set the Key Mode to Updateable Primary Key on those data blocks which contain the large binary items. In addition, these data blocks *must* have their primary key items marked.

8.3.1.1 How to use the OPENDB.init_form procedure

The INIT_FORM procedure is provided in the OPENDB PL/SQL library (which is documented in chapter 3). This library should be attached to the form and the INIT_FORM procedure should be called at the start of processing for the form (either from the PRE-FORM or the WHEN-NEW-FORM-INSTANCE triggers).

Example PRE-FORM trigger (at Form level) :

```
opendb.init_form;
```

In addition to this call, INIT_FORM should also be called if the database connection is changed at runtime (this is unusual, but can be done by calling the Forms built-ins logout and logon).

8.3.1.2 Advanced usage of OPENDB.init_form

The full specification of the init_form procedure is as follows :

```
Procedure Init_Form(
  Strip_Double_Quotes IN BOOLEAN := FALSE,
  RDB7_Set_Oracle_Dialect IN BOOLEAN := TRUE);
```

The Strip_Double_Quotes parameter defaults to FALSE but if set to TRUE will cause the Open Client Adapter to strip double quotes from SQL statements before passing them to the database.

The RDB7_Set_Oracle_Dialect parameter defaults to TRUE meaning that, if running against Rdb version 7.0 or higher, init_form will issue an Rdb-specific

SQL command to turn on Oracle dialect (by issuing the SQL command `SET DIALECT 'ORACLE LEVEL1'`). Some Rdb applications may use SQL statements incompatible with Oracle dialect, in which case init_form may be called as follows to avoid the Oracle dialect being set :

```
opendb.init_form(FALSE, FALSE);
```

8.3.2 Data Types

When creating a data block based on a table or view, all other Oracle Rdb data types are automatically mapped to Forms items of the equivalent Oracle data type. This mapping is usual satisfactory but you may wish to override it for DATE VMS, DATE ANSI, TIMESTAMP and TIME columns which are mapped to Forms' DATE items. The data displayed in this case is limited to days, months and years. You may want to change the item's data type depending on what portion of the time information you wish to display and/or edit.

The DATE ANSI type contains YEAR TO DAY fields and is correctly mapped.

- The DATE VMS type is a time stamp containing YEAR TO SECOND and fractional seconds up to hundredths of a second. Map this data type to Forms' DATETIME type.

- The TIMESTAMP type contains fields YEAR TO SECOND and fractional seconds up to hundredths of a second. The fraction is specified as TIMESTAMP(*frac*), where *frac* is a number between 0 and 2. TIMESTAMP without an interval qualifier defaults to TIMESTAMP(2). To handle YEAR TO SECOND fields, map TIMESTAMP to Forms' DATETIME. To handle fractional seconds map TIMESTAMP to Forms' CHAR type and set the corresponding item's ***Max Length*** and ***Query Length*** properties to 25.

- The TIME type contains HOUR TO SECOND fields and fractional seconds up to hundredths of a second. The fraction is specified as TIME(*frac*), where *frac* is between 0 and 2. TIME without an interval qualifier defaults to TIME(0). Map TIME(0) to Forms' CHAR type, and set the corresponding item's ***Max Length*** and ***Query Length*** properties to 9. Map TIME(2) to Forms' CHAR type and set the corresponding item's ***Max Length*** and ***Query Length*** properties to 12. Although this mapping for TIME does not display the fractional seconds, inserting and updating fractional seconds is possible.

Text and image type columns of length greater than 64k cannot be updated from within client-side PL/SQL in Forms applications.

To change the data type of an item, go to its property sheet, and choose the desired value for its ***Data Type*** property.

8.4 SQL Differences between Oracle and Oracle Rdb

8.4.1 SQL Constructs

- Use of ANSI standard SQL constructs is recommended from applications when using Rdb 6.1 and earlier. Most of the Oracle SQL dialect can be used, however, with Rdb version 7.0 and above (if you are using Rdb Version 7, you can set the database dialect to "ORACLE LEVEL 1", which will allow you to use Oracle7 SQL syntax). Rdb-specific SQL will fail with syntax errors, unless used via pass-through functions. In particular, the following Rdb-specific SQL constructs are not supported:

  SELECT… LIMIT TO
  SELECT… OPTIMIZE FOR
  CASE, IF, LOOP
  INNER JOIN
  OUTER JOIN
  FULL JOIN
  LEFT JOIN
  RIGHT JOIN
  CALL <*procedure*>

- In Rdb 6.1 and earlier, use COUNT(DISTINCT <colname>), not COUNT(<colname>). In Rdb 7.0 and later, both COUNT(<colname>) and COUNT(DISTINCT <colname>) are supported. This Rdb problem was also fixed in patches that are available for Rdb 6.0 and Rdb 6.1. You can obtain Rdb patches by contacting Oracle Customer Support.

- User-supplied object names can be 31 characters long in Rdb. In Oracle the maximum length of user-supplied names is 30.

- If you need to support delimited identifiers (in double quotes) set the SQL dialect to SQL92 or Oracle using one of the following SQL statements via a pass-through function:

  SET DIALECT 'SQL92'
  SET DIALECT 'ORACLE LEVEL1'

- Be aware that in Rdb, trailing spaces are ignored in quoted identifiers, whereas in Oracle they are significant.

- Rdb versions 5.1 and below do not support column name aliases.

- The following Oracle-specific features cannot be used with Rdb: sequences, synonyms, database links, security roles, set difference operators MINUS and INTERSECT, and recursive queries.

- In versions of Rdb prior to 7.0, the ROWNUM pseudo-column could not be used. In Rdb 7.0 and above, using the "ORACLE LEVEL 1" dialect,

rownum may be used in the where clause of a query (it may not be selected).

8.4.2 SQL Functions and Operators

In Rdb 6.1 and earlier, only those SQL functions and operators common to Oracle SQL and Rdb SQL can be used in SQL statements, except via pass-through functions. These common functions include:

AVG
COUNT
LOWER
MAX
MIN
SUM
UPPER
USER

Rdb 6.0 and above supports user-defined functions, so you could implement a user-defined function with the same name and semantics as an Oracle function, and use it in SQL statements.

In Rdb 7.0, the following Oracle functions have been added:

ABS	INSTR	RAWTOHEX
ADD_MONTHS	INSTRB	REPLACE
ASCII	LAST_DAY	ROUND
CEIL	LEAST	RPAD
CHR	LENGTH	RTRIM
CONCAT	LENGTHB	SIGN
CONVERT	LN	SIN
COS	LOG	SINH
COSH	LPAD	SQRT
DECODE	LTRIM	SUBSTR
EXP	MOD	SUBSTRB
FLOOR	MONTHS_BETWEEN	SYSDATE
GREATEST	NEW_TIME	TAN
HEXTORAW	NEXT_DAY	TANH
INITCAP	POWER	TRUNC

8.5 Oracle Rdb Transactions and Locking

By default, the Open Client Adapter uses the READ COMMITTED isolation mode with Rdb. If you need to use a different isolation level (REPEATABLE READ or SERIALIZABLE, for example), you can override the default mode issuing a DECLARE TRANSACTION or SET TRANSACTION statement, or in Forms you can use the Isolation Mode property. Section 2.10 and its sub-sections give more details about using different transaction isolation levels.

In Rdb 7.0, you can set the SQL dialect to Oracle in order to acquire row level locks and to use Forms' immediate locking mode. In Forms this is done automatically by the "OPENDB.init_form" routine (as mentioned earlier in this chapter). It can also be done more generally by using the "SET DIALECT" SQL statement.

To execute SQL statements that are native to Rdb use one of the techniques described in section 2.4. For example, your POST_LOGON trigger might include this code:

```
begin
  forms_ddl ('COMMIT');  -- Make sure no transaction is in
progress
  forms_ddl ('SET DIALECT ''ORACLE LEVEL1''');
  forms_ddl ('DECLARE TRANSACTION READ WRITE WAIT 5
              ISOLATION LEVEL READ COMMITTED');
end;
```

Note: A DECLARE TRANSACTION statement does not start a transaction. It specifies the default characteristics of a transaction. These apply to all transactions (except those started by SET TRANSACTION) until another DECLARE TRANSACTION is issued. SET TRANSACTION starts a transaction and specifies its characteristics. A transaction should not be in progress when issuing a DECLARE TRANSACTION statement, thus the COMMIT statement in the example above. For more information on transaction control in Rdb, refer to the Rdb SQL Reference Manual.

8.6 Client-side PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. Forms Developer and Reports Developer use PL/SQL as its programming language. Certain restrictions and limitations will apply when using PL/SQL with Oracle Rdb via ODBC.

Use of standard SQL constructs (as opposed to Oracle or Rdb-specific) is recommended when using PL/SQL with Oracle Rdb databases.

If you need to use SQL constructs not conforming to Oracle SQL, you need to use a one of the SQL pass-through techniques described in section 2.4

8.6.1 PL/SQL and Oracle Rdb Data Types

The following PL/SQL data types can be used freely in client-side PL/SQL program units:

- NUMBER and its subtypes DECIMAL (DEC), DOUBLE PRECISION, FLOAT, INTEGER (INT), NUMERIC, REAL, and SMALLINT.
- CHAR and its subtypes CHARACTER and STRING. Note that a CHAR value in client-side PL/SQL can have a maximum length of 32767, but a CHAR column in the database has a maximum length of 255.

- VARCHAR and VARCHAR2. Note that a VARCHAR value in client-side PL/SQL can have a maximum length of 32767, but a VARCHAR column in the database has a maximum length of 2000.
- LONG. This is similar to VARCHAR, except that while it is limited to a maximum size of 32767 on the client, the equivalent Oracle Rdb data type, LIST OF BYTE VARYING, can be up to 2 Gigabytes in size.
- RAW. A RAW value in client-side PL/SQL can have a maximum length of 32767, but a RAW column in the database has a maximum length of 255.
- LONG RAW. Analogous to LONG, but for RAW data.
- BOOLEAN
- DATE

Oracle Rdb data types can be mapped to their equivalent PL/SQL data type, with the following exceptions and caveats:

Oracle Rdb Data Type	**PL/SQL Data Type**
TINYINT	NUMBER(3)
SMALLINT	NUMBER(5)
INT	NUMBER(10)
BIGINT	FLOAT or NUMBER(19)
LONG VARCHAR	LONG
LIST OF BYTE VARYING	RAW (Max length < 32768) LONG LONG RAW (PL/SQL V2) depending on the data
DATE VMS DATE ANSI TIMESTAMP TIME	DATE (Mapping to CHAR with a format mask is required to handle fractional seconds)

NCHAR, NVARCHAR, SENSITIVITY and SENSITIVITY_BOUNDARY data types are not supported.

8.6.2 Supported PL/SQL Statements and Functions

The following PL/SQL statements and attributes are supported with Oracle Rdb:

COMMIT	INSERT
CLOSE	LOOP
CURSOR	OPEN
DECLARE	ROLLBACK
DELETE	SELECT INTO
FETCH	UPDATE

GOTO

%FOUND %NOTFOUND
%ROWCOUNT %ISOPEN

as well as all exception, looping and conditional constructs

Only SQL functions common to both Rdb SQL and Oracle SQL (by name, number and type of arguments) can be used in SQL statements within procedures. Note that this set of functions will depend on the Rdb dialect in use, as well as Rdb version - Rdb 7.0 supports most Oracle SQL functions. All other PL/SQL functions can be used in the body of procedures only.

8.6.3 Calling Stored Procedures from Client-side PL/SQL

Stored procedures can be called from client-side PL/SQL program units and triggers. Stored Procedures with IN, OUT, IN/OUT parameters or return values are fully supported. Result sets can be retrieved from stored procedures using the EXEC_SQL package. See the chapter "Calling Stored Procedures through the Oracle Open Client Adapter" for an example.

For example, this Oracle Rdb stored procedure:

```
CREATE MODULE TESTMOD
  LANGUAGE SQL
  PROCEDURE TESTPROC
    (IN  :DNUM    INTEGER,
     OUT :TOT_SAL FLOAT);
  BEGIN
    SELECT SUM(SAL) INTO :TOT_SAL FROM EMP WHERE DEPTNO =
:DUM;
  END;
END MODULE;
```

could be called from a client-side program unit as follows:

```
FUNCTION compute_deptsal (Param1 IN NUMBER) RETURN FLOAT IS
  Param2 FLOAT;
BEGIN
  testproc (Param1, Param2);
  RETURN (Param2);
END;
```

9. Using Oracle Forms Developer and Oracle Reports Developer with Informix

9.1 Installation, Setup and System Requirements

Using Forms Developer and Reports Developer to access an ODBC data source requires the installation of components from Oracle, Informix and the ODBC driver vendor on the client machine, and proper configuration of the Informix server.

9.1.1 Setup for Oracle Forms Developer and Oracle Reports Developer

Forms Developer, Reports Developer and the Open Client Adapter must be installed on the client. In addition, Forms Developer and Reports Developer include ODBC drivers from INTERSOLV that allow access to Informix databases. If you choose to use one of these drivers, you should run the Oracle Installer to install it from the Forms Developer and Reports Developer CD-ROM. For more information on this process, please refer to the *Oracle Forms Developer and Oracle Reports Developer Installation Guide.*

9.1.2 Setup for Informix

9.1.2.1 Client Requirements and Setup

The INTERSOLV DataDirect drivers for Informix 5 and Informix 7 included with Forms Developer and Reports Developer use version 5.x of the Informix-Net (or Informix-Star) product to access Informix databases. You must install Informix-Net before configuring a data source.

- The environment variable INFORMIXDIR must be set to the directory where Informix-Net is installed. The message files RDS.IEM and SQL.IEM must be located under *$INFORMIXDIR*\MSG.
- Add $INFORMIXDIR\BIN to your path, either in AUTOEXEC.BAT or via the control panel.
- Use the SETNET.EXE program to set INFORMIX.INI entries - *hostname, username, servicename, protocolname*, password options and any environment variables required for your system. You may also need to add a *servicename* entry to the DRIVERS\ETC\SERVICES file in the Windows system directory. Consult your Informix documentation for more information on configuring Informix-Net.

If you will be using an ODBC driver other than that provided with Forms Developer and Reports Developer , install it according to the instructions from the supplier, and install any necessary database access software.

9.1.2.2 ODBC Driver Configuration

The generic steps required to configure an ODBC data source are detailed in the ODBC Setup section of the Getting Started chapter. In the case of the INTERSOLV DataDirect drivers for Informix, if the required client libraries are not installed or the *$INFORMIXDIR*\BIN is not in the path, the ODBC Administrator program will display an error such as "*The setup routines for the INTERSOLV INFORMIX5 ODBC driver could not be loaded. You may be low on memory and need to quit a few applications*'. To correct this error, review the regarding client-side setup. If the problem persists you may need to reinstall either the database client and net libraries or the ODBC driver.

9.1.2.3 Server Requirements and Setup

If using Informix SE, the database must be started in transaction mode (use the `WITH LOG IN` option). If using Informix Online, the database must be started in `Quiescent` mode.

9.2 Using Oracle Forms Developer with Informix

9.2.1 Data Types

The Informix `INTERVAL` data type is not supported. All other Informix datatypes are automatically mapped to their equivalent Oracle data types with the exception of DATETIME which is mapped to Forms' DATE type. The data displayed in this case is limited to days, months and years (unless you explicitly specify a format mask on the item including time information, such as 'DD-MON-YY HH24:MI:SS').

If you need to display or update hours, minutes and seconds, use Forms' DATETIME. To display and update fractional seconds, map these data types to CHAR(25), which will use a standard format mask showing all components. To change the data type of an item, go to its property sheet, and choose the desired value for its ***Data Type*** property.

If your table includes an image column, or a long text column, you need to take a special steps to ensure that it will be fetched and updated reliably by making sure that it is the last column referenced in the SQL statement. To do this:

- Make the item the last item in the data block, by dragging and dropping it after all other items.

9.2.2 Implications of Informix's Locking Model

By default, Informix locks are at the page level. Row-level locking can be enabled (albeit at a performance premium) on a table-by-table basis with the "LOCK MODE [PAGE | ROW]" clause to the CREATE TABLE and ALTER TABLE statements.

In addition, Informix SQL does not support a lock time-out to escape a potential deadlock -- a connection attempting to get a lock on a resource hangs until the locks already held on that resource are released.

In light of this, you should structure your application to hold locks for as short a period as possible, by using Forms' ***Delayed*** locking mode, and bundling changes made from client-side PL/SQL to occur just before a commit. Note that ***Delayed*** locking mode is automatically used by Forms runtime against Informix if you leave the data blocks' locking mode property set at the default value of "automatic".

9.3 SQL Differences between Oracle and Informix

9.3.1 Connecting to Different Databases

An Informix Online server can control multiple databases. Each user has a default database, to which they are connected at logon. A user can switch to other databases controlled by the same server by executing a 'database <*dbname*>' SQL statement.

In Forms Developer and Reports Developer , there are two ways to connect to a database other than the user's default. The database name can be specified at connect time, as in:

```
ODBC:<DataSourceName>:<Dbname>
```

Alternatively, execute a 'database <*dbname*>' SQL statement in a client-side trigger or program unit. See the section "Connecting to different databases" in chapter 2.

9.3.2 SQL Constructs

Use of ANSI standard SQL constructs is recommended from applications when using Informix. Informix-specific SQL will fail with syntax errors, unless used via pass-through functions.

9.3.3 Naming Rules

Informix limits names of database objects to 18 characters, as opposed to Oracle's 30. If trying to create a portable application, restrict your Oracle object names to 18 characters.

Informix is case insensitive, but the data dictionary is maintained in lower case. If using double quotes around table names, make sure that the name within is in lower case.

You can only use double quotes around table names -- column names cannot be quoted.

9.3.4 SQL Syntax Differences

- NULL cannot be used in the select list of a SELECT statement. For example, "select null from emp" will fail to execute.

- UNIQUE cannot be used in a SELECT statement. Use DISTINCT instead.
- Informix's outer join syntax is different from Oracle's and cannot be used in SQL statements.
- Informix does not support the INTERSECT and MINUS set operators.
- You cannot use the "EXECUTE PROCEDURE" construct in INSERT statements.
- Informix doesn't allow expressions in the INSERT values. Because of that you cannot concatenate strings to insert long values.

9.3.5 SQL Functions and Operators

Only those SQL functions and operators common to Oracle and Informix SQL can be used in SQL statements, except via pass-through functions. These common functions include:

AVG	SUM
COUNT	TRUNC
MAX	USER
MIN	LENGTH
ROUND	

9.3.6 Comparison Operators

The following comparison operators are either not supported, or have an Oracle or ANSI equivalent which should be used from within a application:

Operator	**Equivalent**
MATCHES "a%"	LIKE 'a%'
MATCHES "a?"	LIKE 'a_'
MATCHES "a[x-z]"	Not Supported
LIKE "a[^x-z]"	Not Supported
NOT MATCHES ...	`NOT LIKE ...`
= NULL	IS NULL
!= NULL	IS NOT NULL

9.3.7 Arithmetic Operators

The modulo (%) and exponent (**) operators can not be used in SQL statements within Forms Developer and Reports Developer .

9.4 Client-side PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. Forms Developer and Reports Developer use PL/SQL as its programming language. Certain restrictions and limitations will apply when using PL/SQL with Informix via ODBC.

Use of standard SQL constructs (as opposed to Oracle or Informix-specific) is recommended when using PL/SQL with Informix databases.

If you need to use SQL constructs not conforming to Oracle SQL, you need to use a pass-through interface, such as the interface the EXEC_SQL package provides. The EXEC_SQL package is documented in the Procedure Builder on-line help and documentation.

9.4.1 PL/SQL and Informix Data Types

The following PL/SQL data types can be used freely in client-side PL/SQL program units:

- NUMBER and its subtypes DECIMAL (DEC), DOUBLE PRECISION, FLOAT, INTEGER (INT), NUMERIC, REAL, and SMALLINT.
- CHAR and its subtypes CHARACTER and STRING. Note that a CHAR value in both client-side PL/SQL and the database can have a maximum length of 32767.
- VARCHAR and VARCHAR2. Note that a VARCHAR value in client-side PL/SQL can have a maximum length of 32767, but a VARCHAR column in the database has a maximum length of 255.
- LONG. This is similar to VARCHAR, except that while it is limited to a maximum size of 32767 on the client, the equivalent Informix data type, TEXT, can be up to 2 Gigabytes in size.
- LONG RAW. Analogous to LONG, but for RAW data.. The equivalent Informix data type, BYTE, can be up to 2 Gigabytes in size.
- BOOLEAN
- DATE

Informix data types can be mapped to their equivalent PL/SQL data type, with the following exceptions and caveats:

Informix Data Type	**PL/SQL Data Type**
BYTE	RAW
INTERVAL	Not Supported
MONEY	FLOAT
TEXT	LONG

Columns of length greater than 64K cannot be updated from client-side PL/SQL.

9.4.2 Supported PL/SQL Statements and Functions

The following PL/SQL statements and attributes are supported with Informix:

COMMIT	INSERT
CLOSE	LOOP
CURSOR	OPEN

DECLARE	ROLLBACK
DELETE	SELECT INTO
FETCH	UPDATE
GOTO	

%FOUND	%NOTFOUND
%ROWCOUNT	%ISOPEN

as well as all exception, looping and conditional constructs

Only SQL functions common to both Informix and Oracle SQL (by name, number and type of arguments) can be used in SQL statements within procedures. All other PL/SQL functions can be used in the body of procedures only.

9.4.3 Calling Stored Procedures from Client-side PL/SQL

Informix stored procedures can be called from client-side PL/SQL program units and triggers. These procedures must either be parameterless or take only input parameters (which is consistent with Informix stored procedures handling). Output values can be retrieved from stored procedures in the same way as result sets using the EXEC_SQL package. See the chapter "Calling Stored Procedures through the Oracle Open Client Adapter" for an example.

10. Using Oracle Forms Developer and Oracle Reports Developer with Microsoft Access

10.1 Installation, Setup and System Requirements

Using Forms Developer and Reports Developer to access an ODBC data source requires installation of components from one or more software providers on the client and in most cases, the server. In the case of Microsoft Access, the client and server are on the same machine, although the database files may reside on a networked or shared volume or drive.

10.1.1 Setup for Oracle Forms Developer and Oracle Reports Developer

Forms Developer, Reports Developer and the Open Client Adapter must be installed on the client.

10.1.2 Setup for Microsoft Access

10.1.2.1 Client Requirements and Setup

The Microsoft Access ODBC driver is part of the ODBC Desktop Database Driver pack, available from Microsoft. It is included as part of Microsoft Office and several other application programs. Alternatively, you can install the Access ODBC driver that is bundled with the Microsoft Access database product.

10.1.2.2 Server Requirements and Setup

Microsoft Access is a single-user, single-machine system. As such, there is no notion of a database server and database client, and no special setup is required. You do, however, need to use Microsoft Access to create the database file, which defines the schema - tables and columns - of the database. Once the database file has been created, the MS Access ODBC driver manipulates the database file directly.

10.2 Data dictionary views

SQL support in Access is limited compared to that of Oracle. Forms Developer's and Reports Developer's design-time components use certain views in the database to access user and schema information. These views cannot be created in Access due to inadequate exposure of Access' catalog information via SQL. Therefore it is not possible to browse available schema and tables in the Forms Developer and Reports Developer Builders. For example, when using the Data Block wizard in Forms to create a new data block, it is necessary to type in the name of the table manually because the list of tables will not work.

10.3 Using Oracle Forms Developer with Microsoft Access

10.3.1 Data Types

Text and image type columns of length greater than 64k will be truncated to 64k if using the 16-bit MS Access driver. There is no truncation when using the 32-bit MS Access driver, however.

10.4 SQL Differences between Oracle and Microsoft Access

10.4.1 SQL Constructs

SQL support in Access is limited, and there are also syntactical differences between Access SQL and Oracle SQL. Use of ANSI standard SQL constructs is recommended from Forms Developer and Reports Developer applications when using Microsoft Access . Microsoft Access-specific SQL will fail with syntax errors, unless used via pass-through functions. In particular, the following Access-specific SQL statements are not supported:

DISTINCTROW	IN (specifying table location)
INNER JOIN	LEFT JOIN
RIGHT JOIN	PARAMETERS
TRANSFORM	WITH OWNER ACCESS OPTION
PROCEDURE	

Only those SQL functions and operators common to Oracle SQL and Access SQL can be used in SQL statements, except via pass-through functions. These common functions include:

ABS	MIN
ASCII	ROUND
AVG	RTRIM
COUNT	SIGN
FLOOR	SOUNDEX
LOWER	SQRT
LTRIM	SUM
MAX	UPPER

10.5 Microsoft Access Transactions and Locking

Access has limited locking control, and it does not support control of locking mechanisms through SQL. Access' two locking modes, `Read Only` and `Exclusive`, are specified when the database is opened in the ***Open Database*** dialog box, or in the ***ODBC Access Setup*** dialog box. Multi-user locking modes such as `No Locks`, `All Records`, or `Edited Record` are specified on a per-table basis, and there is no mechanism to change these attributes through SQL at runtime.

As a result, your application cannot depend upon runtime control of locking behavior. If you are porting an application from a data source where this control

is possible, you will need to disable this functionality, or implement a runtime check for the data source.

10.6 Client-side PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. Forms Developer and Reports Developer use PL/SQL as its programming language. Certain restrictions and limitations will apply when using PL/SQL with Microsoft Access via ODBC.

Use of standard SQL constructs (as opposed to Oracle or Access-specific) is recommended when using PL/SQL with Microsoft Access databases.

If you need to use SQL constructs not conforming to Oracle SQL, you need to use a pass-through interface, such as the interface the EXEC_SQL package provides. The EXEC_SQL package is documented in the Procedure Builder on-line help and documentation.

10.6.1 PL/SQL and Microsoft Access Data Types

The following PL/SQL data types can be used freely in client-side PL/SQL program units:

- NUMBER and its subtypes DECIMAL (DEC), DOUBLE PRECISION, FLOAT, INTEGER (INT), NUMERIC, REAL, and SMALLINT.
- CHAR and its subtypes CHARACTER and STRING. Note that a CHAR value in both client-side PL/SQL and the database can have a maximum length of 255.
- VARCHAR and VARCHAR2. Note that a VARCHAR value in client-side PL/SQL can have a maximum length of 32767, but a VARCHAR column in the database has a maximum length of 2000.
- LONG. This is similar to VARCHAR, except that while it is limited to a maximum size of 32767 on the client, a database value can be up to 2 Gigabytes in size.
- RAW. A RAW value in client-side PL/SQL can have a maximum length of 32767, but a RAW column in the database has a maximum length of 255.
- LONG RAW. Analogous to LONG, but for RAW data..
- BOOLEAN
- DATE

Microsoft Access data types can be mapped to their equivalent PL/SQL data types, with the following exceptions and caveats:

Microsoft Access Data Type	**PL/SQL Data Type**
Memo	CHAR
OLE	RAW

10.6.2 Supported PL/SQL Statements and Functions

The following PL/SQL statements and attributes are supported with Microsoft Access:

COMMIT	INSERT
CLOSE	LOOP
CURSOR	OPEN
DECLARE	ROLLBACK
DELETE	SELECT INTO
FETCH	UPDATE
GOTO	

%FOUND	%NOTFOUND
%ROWCOUNT	%ISOPEN

as well as all exception, looping and conditional constructs

Only SQL functions common to both Access and Oracle SQL (by name, number and type of arguments) can be used in SQL statements within procedures. All other PL/SQL functions can be used in the body of procedures only.

11. Using Oracle Forms Developer and Oracle Reports Developer with Ingres

11.1 Installation, Setup and System Requirements

Using Forms Developer and Reports Developer to access an Ingres data source requires installation of components from Oracle, Ingres and the ODBC driver vendor on the client machine, and proper configuration of the Ingres server.

11.1.1 Setup for Oracle Forms Developer and Oracle Reports Developer

Forms Developer and Reports Developer and the Open Client Adapter must be installed on the client. In addition, Forms Developer and Reports Developer include ODBC drivers from INTERSOLV that allow access to Ingres databases. Use the Oracle Installer to install them from the Forms Developer and Reports Developer CD-ROM to the client. For more information on this process, refer to the *Oracle Forms Developer and Oracle Reports Developer Installation Guide.*

11.1.2 Setup for Ingres

11.1.2.1 Client Requirements and Setup

The INTERSOLV DataDirect driver for Ingres requires you to setup Ingres's client side networking. See the driver documentation for details.

If you will be using an ODBC driver other than that provided with Forms Developer and Reports Developer , install it according to the instructions from the supplier, and install any necessary database access software.

11.2 Using Oracle Forms Developer with Ingres

11.2.1 Data Types

The object_key and table_key data types are not supported.

All other Ingres data types are automatically mapped to their equivalent Oracle data types with the exception of DATE which is mapped to Forms' DATE type. The data displayed in this case is limited to days, months and years. You may want to change the mapping depending on what portion of the time types you want to edit.

- To display and update only days, months and years, the mapping to DATE is sufficient. The default format mask in this case is *dd-mon-yy*.
- To display and update the hours, minutes and seconds, use Forms' DATETIME data type. Specify a format mask to display the fields of interest.

- If you need to display and update fractional seconds, map these data types to CHAR(26). This will display the data in a canonical date format, which should be respected if the user wishes to edit the data.

To change the data type of an item, go to its property sheet, and choose the desired value for its ***Data Type*** property.

11.3 Using Oracle Reports Developer with Ingres

Ingres does not support duplicate column references in the ORDER BY clause (see the "Additional Restrictions" section in this chapter). This can sometimes cause problems for Reports. To avoid this problem, there should be no ORDER BY on a column which is also a break column or a target of a link in a child query.

11.4 SQL Differences between Oracle and Ingres

11.4.1 SQL Constructs

Use of ANSI standard SQL constructs is recommended from Forms Developer and Reports Developer applications when using Ingres. Ingres-specific SQL will fail with syntax errors, unless used via pass-through functions.

When issuing an INSERT statement from within Forms Developer and Reports Developer , always include the INTO clause, as in INSERT INTO *<table>* VALUES…

11.4.2 SQL Functions and Operators

Only those SQL functions and operators common to Oracle SQL and Ingres SQL can be used in SQL statements, except via pass-through functions. These common functions include:

ABS	SIN
AVG	SQRT
COS	SUM
COUNT	SYSDATE
MAX	USER
MIN	

11.4.3 Comparison Operators

The following comparison operators are either not supported, or have an Oracle or ANSI equivalent which should be used from within an application:

Operator	**Equivalent**
!>	<=
!<	>=
LIKE ‘a[x-z]’	Not Supported
LIKE ‘a[^x-z]’	Not Supported
= NULL	IS NULL
!= NULL	IS NOT NULL

!> ANY	<= ANY
!< ANY	>= ANY
!> ALL	<= ALL
!< ALL	>= ALL

11.4.4 Arithmetic Operators

The modulo operator (%) can not be used in SQL statements within Forms Developer and Reports Developer .

11.4.5 String Operators

Ingres uses "+" for string concatenation, whereas Oracle uses "||". Reports and Graphics allow the use of "+" for string concatenation in queries, but PL/SQL does not permit its use. As a workaround, you can select the columns individually, and concatenate them on the client side.

11.4.6 Bit Operators

The &, |, ^, and ~ operators can not be used in SQL statements within Forms Developer and Reports Developer .

11.5 Ingres Transactions and Locking

Forms applications run with AUTO-COMMIT mode OFF. All SQL statements are executed in the context of transactions, which run in unchained mode. A SQL statement issued immediately after connecting to the database or issued after a commit or rollback implicitly starts a transaction.

Ingres uses exclusive locks for data modifications and shared locks for non-update or read operations. The granularity of locking is page-level. Page-level locks are escalated to table level locks if more than a specified percentage of pages are locked by a transaction.

SELECT statements obtain shared locks on pages. This prevents other statements from obtaining an exclusive lock on those pages. Hence a SELECT statement blocks other data modification operations as long as the transaction that includes the SELECT statement does not commit or roll back. By default Ingres does not time-out waiting for locks. This can be changed by executing:

```
SET LOCKMODE session N
```

where N is the number of seconds you want the application to wait before timing out. Use one of the pass-through SQL techniques described in section 2.4 since this is not standard Oracle syntax.

11.6 Client-side PL/SQL

Forms Developer and Reports Developer use PL/SQL as its programming language. Certain restrictions and limitations will apply when using PL/SQL with Ingres via ODBC.

Use of standard SQL constructs (as opposed to Oracle or Ingres-specific) is recommended when using PL/SQL with Ingres databases.

If you need to pass Ingres specific SQL constructs, you need to use a pass-through interface, such as the interface the EXEC_SQL package provides. The EXEC_SQL package is documented in the Procedure Builder on-line help and documentation.

11.6.1 PL/SQL and Ingres Data Types

The following PL/SQL data types can be used freely in client-side PL/SQL program units:

- NUMBER and its subtypes DECIMAL (DEC), DOUBLE PRECISION, FLOAT, INTEGER (INT), NUMERIC, REAL, and SMALLINT.
- CHAR and its subtypes CHARACTER and STRING. Note that a CHAR value in client-side PL/SQL can have a maximum length of 32767, but a CHAR column in the database has a maximum length of 2000.
- VARCHAR and VARCHAR2. Note that a VARCHAR value in client-side PL/SQL can have a maximum length of 32767, but a VARCHAR column in the database has a maximum length of 2000.
- LONG. This is similar to VARCHAR, except that while it is limited to a maximum size of 32767 on the client, and cannot be inserted into an Ingres database.
- RAW. A RAW value in client-side PL/SQL can have a maximum length of 32767, but cannot be inserted into an Ingres database.
- LONG RAW. Analogous to LONG, but for RAW data.
- BOOLEAN
- DATE

Ingres data types can be mapped to their equivalent PL/SQL data type, with the following exceptions and caveats:

Ingres Data Type	**PL/SQL Data Type**
integer1	SMALLINT
TEXT	VARCHAR2
DATE	DATE
MONEY	FLOAT

object_key and table_key data types are not supported.

11.6.2 Supported PL/SQL Statements and Functions

The following PL/SQL statements and attributes are supported with Ingres:

COMMIT INSERT

CLOSE	LOOP
CURSOR	OPEN
DECLARE	ROLLBACK
DELETE	SELECT INTO
FETCH	UPDATE
GOTO	

%FOUND	%NOTFOUND
%ROWCOUNT	%ISOPEN

as well as all exception, looping and conditional constructs

Only SQL functions common to both Ingres SQL and Oracle SQL (by name, number and type of arguments) can be used in SQL statements within procedures. All other PL/SQL functions can be used in the body of procedures only.

11.6.3 Calling Stored Procedures from Client-side PL/SQL

Stored procedures can be called from client-side PL/SQL program units and triggers. Stored Procedures with IN, OUT, IN/OUT parameters or return values are fully supported.

For example, this Ingres stored procedure:

```
create procedure move_emp (id integer not null) as
begin
  insert into emptrans
    select *
       from employee
       where id = :id;
  delete from employee
       where id = :id;
end;
```

could be called from a client-side program unit as follows:

```
PROCEDURE example_prog (Param1 in number) IS
BEGIN
  MOVE_EMP(Param1);
END;
```

11.7 Additional Restrictions

11.7.1 ORDER BY

Within ORDER BY clauses, Ingres permits positional references to columns only if the columns are explicitly listed.

For example,

```
SELECT * FROM TABLE_NAME ORDER BY <column_number>
```

will raise an error.

Suggestion: Replace the "*" with explicit column names, or substitute the <column_number> with a <column_identifier>.

In addition, Ingres does not permit duplicate column references in the order by clause, such as:

```
SELECT C1, C2, C3 from TABLE_NAME ORDER BY C2, 2
```

11.7.2 DATATYPE OVERFLOW

Ingres does not detect overflow of integers or char fields by default, but you can create rules that try to detect when this will occur.

11.7.3 SAVEPOINT/ROLLBACK TO SUPPORT

Although Ingres supports SAVEPOINTs and ROLLBACKs, it can only do so if there are no currently open cursors. Forms Developer and Reports Developer re-use cursors frequently to help improve performance, and therefore keeps many cursors open at any given time. Therefore, we cannot support SAVEPOINTs and ROLLBACK TO statements for Ingres.

12. Using Oracle Forms Developer and Oracle Reports Developer with DB2/400

12.1 Installation, Setup and System Requirements

Using Forms Developer and Reports Developer to access an ODBC data source requires the installation of components from Forms Developer and Reports Developer , DB2/400 and the ODBC driver vendor on the client machine, and proper configuration of the DB2/400 server.

12.1.1 Setup for Oracle Forms Developer and Oracle Reports Developer

Forms Developer, Reports Developer and the Open Client Adapter must be installed on the client. In addition, Forms Developer and Reports Developer include the Oracle ODBC driver for DB2/400 that allows access to DB2/400 databases. If you choose to use this driver, you should run the Oracle Installer to install it from the Forms Developer and Reports Developer CD-ROM. For more information on this process, please refer to the *Oracle Forms Developer and Oracle Reports Developer Installation Guide.*

12.1.2 Setup for DB2/400

12.1.2.1 Client Requirements and Setup

Forms Developer and Reports Developer have been certified against the IBM Client Access/400 ODBC driver. This is a component of Client Access/400 for Windows 3.1. When you run Client Access/400 Setup, the required driver DLL's are installed into the Client Access/400 product directory.

You must establish a connection between the PC client and the AS/400 server before starting the ODBC driver. To do this, start the QCMN and QSERVER subsystems on the OS/400 server, and then start the Client Access/400 for Windows 3.1 router on the PC.

Client side software requirements:

- Microsoft Windows version 3.1, Microsoft Windows for Workgroups 3.11, Microsoft Windows95, Microsoft WindowsNT 3.5 or Microsoft WindowsNT 4.0
- Client Access/400 for Microsoft Windows
- IBM's LAN Support Program (for Token-Ring and Ethernet connections only)
- Microsoft ODBC Driver Manager version 2.0 or above

Communications (one of the following):

- Token-ring (direct and 5494 Remote Controller)
- Twinaxial (local, 5394 and 5494 Remote Controller)
- SDLC
- Ethernet

If you will be using any other ODBC driver, install it according to the instructions from the supplier, and install any necessary database access software.

12.1.2.2 Server Requirements and Setup

Install DB2/400 on the server, or identify the DB2/400 server that you will be accessing. The DB2/400 ODBC driver accesses DB2/400 via the Client Access/400. DB2/400 and Client Access/400 must be installed and running on the server to access DB2/400 from Forms Developer and Reports Developer .

Server side software requirements:

- OS/400 Version 3 Release 1
- DB2/400 Version 3.1 or higher
- Client Access/400 Family
- Client Access/400 for Windows 3.1

12.2 Using Oracle Forms Developer with DB2/400

12.2.1 Data Types

When creating a data block based on a table or view, DB2/400 data types are automatically mapped to Forms items of the equivalent Oracle data type. This mapping is usually satisfactory but you may wish to override it for DATE, TIMESTAMP and TIME columns which are mapped to Forms' DATE items. The data displayed in this case is limited to days, months and years. You may want to change the item's data type depending on what portion of the time information you wish to display and/or edit.

- To display and update only days, months and years, the mapping to DATE is sufficient. The default format mask in this case is *dd-mon-yy*.
- To display and update the hours, minutes and seconds of DB2/400's TIME and TIMESTAMP, use Forms' DATETIME data type. Specify a format mask to display the fields of interest.
- If you need to display and update microseconds of DB2/400's TIMESTAMP type, map these data types to CHAR(26). The DB2/400 character-string representation of the TIMESTAMP data type has a length of 26, and a display format of *yyyy-mm-dd-hh.mm.ss.zzzzzz*, where *yyyy,*

mm, *dd*, *hh*, *mm*, *ss* and *zzzzzz* represent, respectively, the years, months, days, hours, minutes, seconds, and microseconds.

To change the data type of an item, go to its property sheet, and choose the desired value for its ***Data Type*** property.

DB2/400's types CHAR() FOR BIT DATA and VARCHAR() FOR BIT DATA are mapped to Forms' CHAR type. However, unless the columns actually contain character data, you will not be able to edit them in a meaningful way from within Forms.

12.3 SQL Differences between Oracle and DB2/400

12.3.1 SQL Constructs

- Use of ANSI standard SQL constructs is recommended from Forms Developer and Reports Developer applications when using DB2/400. DB2/400-specific SQL will fail with syntax errors, unless used via pass-through functions.
- NULL cannot be used in the select list of a SELECT statement.
- Object names can be up to 255 characters long in DB2/400 (10 characters for collection names). In Oracle and in PL/SQL the maximum length of object names is 30. Therefore, when designing the DB2/400 data dictionary, shorter names are preferred.
- The ORDER BY clause can reference only a column which is specified in the SELECT list.
- DB2/400 supports neither synonyms nor aliases.
- The following Oracle-specific features cannot be used with DB2/400: sequences, synonyms, database links, security roles, set difference operators MINUS and INTERSECT, and recursive queries.

12.3.2 SQL Functions and Operators

Only those SQL functions and operators common to Oracle SQL and DB2/400 SQL can be used in SQL statements, except via pass-through functions. These common functions include:

AVG	MOD
COS	SIN
COSH	SINH
COUNT	SQRT
EXP	STDDEV
LENGTH	SUBSTR
LN	SUM
LOG	TAN
MAX	TANH

MIN TRANSLATE

12.3.3 The Comparison Operators

The following comparison operators are either not supported, or have an Oracle or ANSI equivalent which should be used from within an application:

Operator	**Equivalent**
CONCAT()	||
!=	<>

12.4 Client-side PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. Forms Developer and Reports Developer use PL/SQL as its programming language. Certain restrictions and limitations will apply when using PL/SQL with DB2/400 via ODBC.

Use of standard SQL constructs (as opposed to Oracle or DB2/400-specific) is recommended when using PL/SQL with DB2/400 databases.

If you need to use SQL constructs not conforming to Oracle SQL, you need to use a pass-through interface, such as the interface the EXEC_SQL package provides. The EXEC_SQL package is documented in the Procedure Builder on-line help and documentation.

12.4.1 PL/SQL and DB2/400 Data Types

The following PL/SQL data types can be used freely in client-side PL/SQL program units:

- NUMBER and its subtypes DECIMAL (DEC), DOUBLE PRECISION, FLOAT, INTEGER (INT), NUMERIC, REAL, and SMALLINT.
- CHAR and its subtypes CHARACTER and STRING. Note that a CHAR value in client-side PL/SQL can have a maximum length of 32767, but a CHAR column in the database has a maximum length of 255.
- VARCHAR and VARCHAR2. Note that a VARCHAR value in client-side PL/SQL can have a maximum length of 32767, but a VARCHAR column in the database has a maximum length of 2000.
- LONG. This is similar to VARCHAR, except that while it is limited to a maximum size of 32767 on the client, the equivalent DB2/400 data type, LIST OF BYTE VARYING, can be up to 2 Gigabytes in size.
- RAW. A RAW value in client-side PL/SQL can have a maximum length of 32767, but a RAW column in the database has a maximum length of 255.
- LONG RAW. Analogous to LONG, but for RAW data.
- BOOLEAN
- DATE

DB2/400 data types can be mapped to their equivalent PL/SQL data type, with the following exceptions and caveats:

DB2/400 Data Type	PL/SQL Data Type
TIME, TIMESTAMP	DATE
CHAR() FOR BIT DATA	CHAR
VARCHAR() FOR BIT DATA	VARCHAR

12.4.2 Supported PL/SQL Statements and Functions

The following PL/SQL statements and attributes are supported with DB2/400:

COMMIT	INSERT
CLOSE	LOOP
CURSOR	OPEN
DECLARE	ROLLBACK
DELETE	SELECT INTO
FETCH	UPDATE
GOTO	

%FOUND	%NOTFOUND
%ROWCOUNT	%ISOPEN

as well as all exception, looping and conditional constructs

Only SQL functions common to both DB2/400 SQL and Oracle SQL (by name, number and type of arguments) can be used in SQL statements within procedures. All other PL/SQL functions can be used in the body of procedures only.

13. Universal Back-end Tester (UBT)

13.1 Overview

UBT is an interactive command driver for testing and exercising the Oracle Open Client Adapter for ODBC. UBT enables you to manipulate UBT-specific and regular SQL commands, within the interpreter, or through command files. Data migration can be performed through a database table-to-table COPY command.

UBT is provided for backwards compatibity. Almost all of the functionality of UBT can be achieved with SQL*Plus, which should be used in preference to UBT [1].

UBT is a line-mode command interpreter similar to SQL*Plus. Commands may be SQL statements or native UBT commands (such as COPY).

13.2 SQL Command Syntax

You may divide your SQL commands into separate lines at any points, as long as individual words are not split between lines. You may terminate commands in the following ways:

- with a semicolon (;) : This indicates you wish to run the command. Type the semicolon at the end of the last line of the command.
- with a blank line: The command is ignored, and the prompt appears.

13.3 UBT Command Syntax

You may continue a long UBT command by typing a hyphen at the end of the line and pressing [Return]. If you wish, you may type a space before typing the hyphen. The line number is displayed for the next line.

You should not end a UBT command with a semicolon. When you finish entering the command, just press [Return].

Note: Any command not mentioned in the following ***UBT Commands*** section is interpreted as a regular SQL command and will be passed to the underlying database. For example:

```
DECLARE
@MYVAR int
BEGIN
      SELECT @MYVAR=8000
END;
```

will be passed to the underlying database with no change, except for the semicolon (a SQL command terminator) which will be stripped off.

[1] The one exception is the COPY command, which only works through UBT for OCA connections.

13.4 UBT Commands

13.4.1 CONNECT

Connects to a given username and database.

13.4.1.1 Syntax

```
CONNECT username[/password]@db_specification
```

username[/password]: User name and password you use to log in to the database.

db_specification: Consists of a SQL*Net or ODBC connection string. The exact syntax for SQL*Net depends on the communications protocol your Oracle installation uses. The syntax for an ODBC connection is "ODBC:*datasource*[:*dbname*]". Note that both *datasource* and *dbname* are preceded by a colon (:).

datasource: Specifies the name of the ODBC datasource you defined using the Microsoft ODBC Administrator.

dbname: Specifies the name of the initial database or data file to connect to. If not supplied the default database defined by DBMS for the user is used. Note that for some databases (MS Access, for example) it may not be possible.

13.4.1.2 Examples

Example 1

The following command connects user SCOTT (password TIGER) to the SYBSYS10 ODBC datasource. Note that SCOTT is automatically connected to the default database defined for the SYBSYS10 datasource during configuration (with the use of the Microsoft ODBC Administrator).

```
CONNECT SCOTT/TIGER@ODBC:SYBSYS10
```

Example 2

The following command connects user JACK (password SMITH) to the SQL60 ODBC datasource. Note that JACK is connected to database SQWQA that may differ from that defined for SQL60 during configuration.

```
CONNECT JACK/SMITH@ODBC:SQL42:SRWQA
```

13.4.2 COPY

Copies the data from a query to a table in a local or remote database.

13.4.2.1 Syntax

```
COPY {FROM username[/password]@db_specification TO
username[/password]@db_specification/
             FROM username[/password]@db_specification/
             TO       username[/password]@db_specification}
            {APPEND/CREATE/INSERT/REPLACE} destination_table
```

```
 [(column, column, column ...)]
USING query
```

username[/password]: User name and password you use to log in to the database. In the FROM clause, it identifies the user name and password to log in to the source of the data; in the TO clause it identifies the user name and password to log in to the destination.

db_specification: Consists of a SQL*Net or ODBC connection string. In the FROM clause *db_specification* represents the database at the source; in the TO clause it represents the database at the destination. The exact syntax for SQL*Net depends on the communications protocol your Oracle installation uses. The syntax for an ODBC connection is "ODBC:*datasource*[:*dbname*]". Note that both *datasource* and *dbname* are preceded by a colon (:).

datasource: Specifies the name of the ODBC datasource you defined using the Microsoft ODBC Administrator.

dbname: Specifies the name of the initial database or data file to connect to. If not supplied the default database defined by DBMS for the user is used. Note that for some databases (MS Access, for example) it may not be possible.

destination_table: Represents the table you wish to create or to which you wish to add data.

(column, column, …): Specifies the names of the columns in *destination_table*. If you specify columns, the number of columns must equal the number of columns selected by *query*. If you do not specify any columns, the copied columns will have the same names in the *destination_table* as they had in the source (if COPY creates *destination_table*).

query: Specifies a SQL query (SELECT command) determining which rows and columns COPY copies.

FROM: If you omit the FROM clause, the source defaults to the database UBT is connected to. You must include the FROM clause to specify a source database other than the default.

TO: If you omit the TO clause, the destination defaults to the database UBT is connected to. You must include the TO clause to specify a destination database other than the default.

APPEND: Inserts the rows from *query* into *destination_table* if the table exists. If *destination_table* does not exist, COPY creates it.

CREATE: Inserts the rows from *query* into *destination_table* after creating the table first. If *destination_table* already exists, COPY returns an error.

INSERT: Inserts the rows from *query* into *destination_table* (if the table exists).

REPLACE: Replaces *destination_table* and its contents with the rows from *query*. If *destination_table* does not exist, COPY creates it. If *destination_table* does already exist, COPY drops the existing table and replaces it with a table containing the copied data.

13.4.2.2 Examples

Example 1

The following command copies the entire EMP table to a table named WESTEMP. Note that the tables are located in two different databases. If WESTEMP already exists, both the table and its contents are replaced. The columns in WESTEMP have the same names as the columns in the source table, EMP.

```
COPY FROM SCOTT/TIGER@ODBC:INFO TO JACK/SMITH@ODBC:HQ -
REPLACE WESTEMP -
USING SELECT * FROM EMP
```

Example 2

The following command copies selected records from EMP to the database to which UBT is connected. Table SALESMEN is created during the copy. UBT copes only the columns EMPNO and ENAME and at the destination names them EMPNO and SALESMAN.

```
COPY FROM SCOTT/TIGER@ODBC:INFO -
CREATE SALESMEN (EMPNO,SALESMAN) -
USING SELECT EMPNO, ENAME FROM EMP -
WHERE JOB='SALESMAN'
```

13.4.3 SPOOL

Stores query results in an operating system file.

13.4.3.1 Syntax

```
SPOOL [filename | OFF]
```

filename: Represents the name for the file to which you wish to spool.

OFF: Stops spooling.

13.4.3.2 Examples

Example 1

To record your displayed output in a file named DIARY.OUT, enter:

```
SPOOL DIARY.OUT
```

Example 2

To stop spooling, type:

```
SPOOL OFF
```

13.4.4 @ ("at" sign)

Runs the specified command file.

13.4.4.1 Syntax

```
@filename
```

filename: Represents the name for the file to which you wish to spool.

13.4.5 SET AUTOCOMMIT

Toggles autocommit on and off.

13.4.5.1 Syntax

```
SET AUTOCOMMIT {ON | OFF}
```

ON: Autocommit mode is ON.

OFF: Autocommit mode is OFF.

13.4.6 DISCONNECT

Commits pending changes to the database and logs the user off the database, but does not exit UBT.

13.4.6.1 Syntax

```
DISCONNECT
```

13.4.7 QUIT / EXIT

Commits all pending database changes, terminates UBT, and returns control to the operating system.

13.4.7.1 Syntax

```
QUIT | EXIT
```

Appendices

Appendix A: Case Sensitivity Issues

Table and column names used in PL/SQL (either in SQL statements or in variable declarations such as v_ename emp.ename%type) are treated as upper case by the PL/SQL compiler. That is, they will only be found to exist in the database if either (a) the database is case insensitive (most are) or (b) the database is case sensitive but the table and column names are defined in upper case. If it is required to use lower or mixed case names in a case sensitive database, you must enclose the names in double quotes. The PL/SQL compiler will then preserve the case.

For example, if you have a column Ename in a table Emp in a case-sensitive database, the following statement will *not* work within PL/SQL :

```
CURSOR C IS SELECT Ename FROM Emp;
```

PL/SQL will fail to find the table Emp in the database, because it converts all column and table names to uppercase. On the other hand, the following statement will work:

```
Cursor C IS SELECT "Ename" FROM "Emp"
```

PL/SQL will compile this correctly, and will run the select statement correctly.

The OCA automatically replaces the double quotes, the standard delimiter in Oracle, with the delimiter used by the data source for quoted identifiers. For example, against Microsoft Access the select statement above would be converted to:

```
SELECT `Ename` from `Emp`;
```

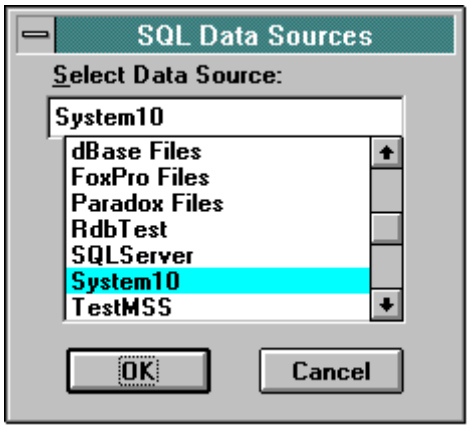
Therefore, you should always use double quotes when you want to quote identifiers (such as table and column names).

Appendix B: Advanced OCA Connect Strings

The normal format for an OCA connect string is:
`ODBC:<data source name>`

However, there are several modifications to this syntax that you can use.

If you specify "*" (asterisk) for the `<data source name>` (example: 'scott/tiger@odbc:*'), you will be presented with a dialog box listing your defined ODBC data sources from which to choose, similar to this one:



You can also specify an optional database name using the following syntax:
`ODBC:<data source name>:<database name>`

The optional <DataBaseName> allows you to indicate which database or data file to connect to. ODBC does not have a standard way of connecting to different databases, but the OCA knows how to connect to different databases for certain data source types (For example, if connected to Microsoft SQL Server, OCA will generate a "USE" statement).

If you specify "*" for the database name (Example: /@odbc:sql65:*) you will be presented with the previous dialog box (with the data source you specified highlighted, or if you specified "*" for the data source name, with no data source highlighted). Once you choose your data source, you will be prompted with the driver-specific connection dialog for that data source.

In this second dialog, you can choose your username, password, database (if the driver allows you to), and configure whatever else the Driver lets you, on a per connection basis.

Example: Connection to any ODBC data source as any user. Specify the following full connection string:
`/@odbc:*:*`

This will allow the user to choose a data source, and then to choose a username and password in the data source log-on box. OCA will automatically return the correct username back to Forms Developer and Reports Developer when requested.

Appendix C: Oracle Data Type Matching

The OCA automatically maps Oracle data types to the corresponding native data types and vice versa. This affects in particular how columns are described (for example, how column types appear when listed by the SQL*Plus "desc" command), because the OCA always returns the corresponding Oracle data types. Please refer to the chapter about your specific data source for more information.

If we have not provided a chapter for your data source, you may still determine how data types are mapped by using the following table. This shows how the OCA maps between Oracle data types and ODBC types. Your ODBC driver's documentation should tell you how the relevant ODBC data types are mapped to the data source's native types.

Oracle Data Type	**ODBC Data Type**
VARCHAR2	SQL_VARCHAR
CHAR	SQL_CHAR
NUMBER	SQL_NUMERIC
	SQL_DECIMAL
	SQL_INTEGER
	SQL_TINYINT
	SQL_SMALLINT
	SQL_BIGINT
	SQL_FLOAT
	SQL_REAL
	SQL_DOUBLE
DATE	SQL_TIMESTAMP
	SQL_DATE
	SQL_TIME
RAW	SQL_BINARY
	SQL_VARBINARY
LONG	SQL_LONGVARCHAR
LONG RAW	SQL_LONGVARBINARY
ROWID	SQL_VARCHAR[1]

[1] Most ODBC data sources have no concept of ROWIDs, but those that do will convert CHAR columns into ROWID automatically.

Appendix D: Notes on DATETIME types with fractional seconds.

Some ODBC data sources support DATETIME (sometimes known as TIMESTAMP) values that allow more granularity than Oracle's DATE type. For example, SQL Server's DATETIME type stores dates up to millisecond granularity, whereas Oracle's DATE type only goes down to seconds.

Should you wish to display or update a datetime column including fractional seconds, you should display it as a VARCHAR or CHAR field in your form, of sufficient length to show all figures.

Please note that when you save the data in this field to the database, you may need to change the format into the format your data source expects. This is because the ODBC specification requires dates *retrieved* as character strings to be in the format 'yyyy-mm-dd hh:mm:ss[.f...], but drivers are not required to accept this as valid character *input*, and often they only accept the format which is native to the data source. Hence, you sometimes need to change the format of datetime values by hand (or by program code) before updating or inserting the value into the database.

Appendix E: Automatic Adjustments in Forms Behavior for OCA Data Sources

Oracle Forms runtime automatically adjusts its behavior, dependent upon what type of data source it is connected to. The adjustments are documented here for your reference.

Changes to Form (i.e. Module) Properties

Savepoint Mode

Oracle uses savepoints to identify a point in a transaction to which the application can later roll back. Savepoints are used in conjunction with the ROLLBACK TO command to rollback portions of the current transactions.

ODBC has no concept of named transactions, or partial rollbacks, but the OCA has added savepoint support for many popular databases . If the OCA knows the syntax for using named savepoints against a certain data source, then it will translate all ROLLBACK TO and SAVEPOINT Oracle-style statements to the appropriate native SQL statements.

The OCA translates for the following data sources, according to the table below:

Data Source	Syntax Used	Comment
Oracle	ROLLBACK TO <name> SAVEPOINT <name>	
SQL Server 6.X Sybase 10 Sybase 11	ROLL TRAN <name> SAVE TRAN <name>	Savepoints are only supported for ODBC drivers which support multiple active statements per connection. See the SQL Server and Sybase chapters for more information
Tandem NonStop SQL	ROLLBACK TRAN <name> SAVE TRAN <name>	

For data sources for which the OCA does not support savepoints, Forms will default to not using savepoints at all.

Cursor Mode

This property is now obsolete, and is left unchanged (i.e. at the default setting of Open), since the OCA now

handles cursor behavior automatically[1]. It should never be set to Close. Please note however that application behavior can still be affected by the cursor characteristics of the data source because, whilst certain databases such as Oracle can keep a cursor open, even after a user commits a transaction, a lot of databases cannot do this. If the data source does not maintain cursors across commits, fetches from cursors after a commit will fail if the cursor is not re-executed beforehand. In practice this mainly affects cursors associated with queries in data blocks based on tables: the user (or application code) must re-query after a commit (i.e. after saving changes) before scrolling down to the next set of records.

Changes to base table Data Block Properties

Certain data blocks, called base table data blocks, are associated with a database table or view, and most items in a base table data block correspond directly to database columns. Certain properties of base table data blocks that affect database interaction are changed when building an application against OCA connections. For more on data blocks, please refer to your Forms documentation.

Key Mode

The *Automatic* value tells Forms to choose whichever Key Mode bests suits the data source it is connected to. This is the default for new forms. For a native (non-ODBC) connection to Oracle, this will be interpreted as *Unique* mode, which will make the application use Oracle's ROWID feature to lock, update and delete rows. For ODBC (OCA) connections, the key mode will be interpreted as whatever works best for the data source, according to the table below.

Data source	Key Mode
Oracle (not via ODBC)	Unique
Rdb	Unique (uses DBKEY)
All others	Updateable Primary Key

You can find out what Key Mode has been chosen by using the GET_BLOCK_PROPERTY function at runtime, but only after making the following call :

```
Set_application_property(
     db_design_properties, PROPERTY_FALSE);
```

[1] Some ODBC data sources (for example, Ingres) actually delete cursors after a commit, meaning that they close the cursor, and the user must re-open and re-parse that cursor to use it again. The OCA handles this transparently by detecting when the cursor is deleted, and automatically re-opening and re-parsing it if an attempt is made to re-execute that cursor.

Otherwise, a call to get_block_property(theBlock, KEY_MODE) will return the value 'AUTOMATIC'.

Updateable primary key means that rows will be identified by those items marked as PRIMARY KEY. *Unique* means that a system-specific internal key is used to identify rows (such as Oracle's *rowid* or Rdb's *dbkey)*. *Unique* key mode is only supported against Rdb and native (i.e. non-ODBC) connections to Oracle.

If the Key Mode property is explicitly set to *Updateable* or *Non-Updateable primary key*, or if it is set to *automatic* but will be interpreted as *updateable primary key* **against at least one of the data sources to be used,** then one or more of the items in the data block must be identified as the primary key, by setting the Item's *Primary Key* property to *True.*

Locking Mode

Defines when rows are locked in the database as they are updated by the user.

The *Automatic* value specifies that Forms should use the locking mode that is optimal for the data source it is connected to. When connected to Oracle, it will use Immediate mode. When connected to an OCA data source, it will use the mode as defined by the following table:

Data source	**Locking Mode**
Oracle	Immediate
All Others	Delayed

You can find out what Locking Mode has been chosen by using the GET_BLOCK_PROPERTY function at runtime, but only after making the following call :

```
Set_application_property(
        db_design_properties,
      PROPERTY_FALSE);
```

Otherwise, a call to get_block_property(theBlock, KEY_MODE) will return the value 'AUTOMATIC'.

The *Immediate* value specifies that Forms should attempt to lock a row as soon as the user modifies an item in the row, or attempts to delete the row.

The *Delayed* value specifies that Forms should lock rows immediately prior to issuing the updates and committing the transaction. With this setting, the record is locked only when the transaction is posted to the database, and not

while the operator is editing the record. As a result, deadlocks or lengthy waits for locks are much less likely.

In order to lock a row, Forms issues a "SELECT…FOR UPDATE" statement selecting the row. Against data sources not supporting the "FOR UPDATE" clause, the OCA will remove this clause. This means the row will not actually be locked by the database, but still allows Forms to ensure that no other user has updated the row since it was queried. This reduces the risk of lost updates.

Appendix F: Notes on drivers supporting only one active statement per connection

While most ODBC data sources allow any number of active statements (or cursors) per connection, some, like certain configurations of Microsoft SQL Server[1], do not. If this is the case for your datasource you should read this appendix.

Lock Outs

If only one active statement is allowed on a connection, you can only ever have one SELECT statement with results waiting on your connection at a time.

All Forms Developer and Reports Developer tools assume they can have multiple active statements on a connection, just by opening new cursors, so the OCA simulates multiple active statements by silently opening a new connection for each opened cursor.

However, a separate connection for each cursor increases an application's chance of locking itself out. For example, if two or more different SQL statements in a PL/SQL block update, insert or delete rows in a table in the same database page (when running against a page-locking database[1]), the application will deadlock as each connection waits for the other one to finish with the page lock.

As a workaround for this problem, you can commit between statements, or execute the same statement several times in a PL/SQL loop rather than using separate statements, or put the statements in a stored procedure and pass the values as arguments to the procedure.

Commit/Rollback Issues

Some databases allow only one cursor per connection. As a result, as mentioned above, the OCA opens additional cursors through separate connections.

Oracle Forms controls master-detail relations through multiple cursors. If a database only allows one cursor per connection, another connection must be established for each detail table. Because each cursor has an individual connection, committing a master-detail application requires coordinating the various cursors. Consequently, a problem could arise if only a subset of the multiple required connections succede in committing their work - a form could be inconsistently updated if a remaining subset of the connections fail to commit or rollback.

If your driver can coordinate two-phase commits between connections, you will not experience this problem.

---

[1] See the chapter on Microsoft SQL Server for more information.

Appendix G: Error Messages

The following table contains OCA error message numbers, the corresponding error message text, and (if there is one) the corresponding Oracle error code which is returned. As described in section "2.11 OCA Error Handling", in the absence of an Oracle error code, the OCA error code is returned..

OCA Error Code	OCA Error Message Text	Oracle Error Code
30000	**general OCA error**	
30001	**invalid session context**	
30002	**function not supported**	**1010**
30003	**NULL username in connect string**	**1017**
30004	**NULL hostname in connect string**	
30005	**all cursors are in use**	
30006	**error creating parse tree**	
30007	**NULL SQL statement**	
30008	**memory allocation failed**	**1019**
30009	**error processing bind variables**	
30010	**Oracle to ODBC datatype conversion error**	
30011	**ODBC to Oracle datatype conversion error**	
30012	**error binding reference value**	
30013	**no parameters to bind**	
30014	**hostname too long**	
30015	**stored procedures not supported for this datasource**	
30016	**sequences not supported by this database**	
30017	**error logging on to non-Oracle database**	**1017**
30018	**username or password too long**	**1017**
30019	**error handling non-Oracle stored procedures**	
30020	**stored procedure name is missing**	
30021	**error preparing/executing SQL statement**	
30022	**error retrieving column**	
30023	**error fetching result set**	
30024	**error retrieving column descriptors**	**1007**
30025	**error getting number of columns in result set**	
30026	**error retrieving procedure descriptors**	
30027	**error setting ODBC connection/statement option**	
30028	**error at commit/rollback**	
30029	**error binding column**	
30030	**error binding parameter**	
30031	**error retrieving information about driver/database**	

30032	**error freeing memory**	
30033	**row status error**	**1444**
30034	**table not found**	**942**
30035	**column not found**	**904**
30036	**invalid authorization specification**	
30037	**datasource not available**	**1034**
30038	**invalid column number**	**1007**
30039	**illegal bundled operation combination**	**2035**
30040	**invalid cursor context**	**1023**
30041	**bind variable does not exist**	**1006**
30042	**no data found**	**1403**
30043	**fetched column value is NULL**	**1405**
30044	**fetched column value was truncated**	**1406**
30045	**error returning result data**	
30046	**no datasource name specified**	**12153**
30047	**unsupported transaction isolation level**	**2248**
30048	**binding arrays not supported**	**1484**
30049	**not logged on**	**1012**
30050	**error freeing SQL statement**	
30051	**error fetching data. You must re-execute after a commit for this data source**	
30052	**parameter value was truncated at execution**	**6502**
30053	**unsupported network call**	**3115**

If an error occurs and the application is unable to read the message from the message file, verify that the following entry is set in the registry on Windows NT and Windows 95 or in the ORACLE.INI file on Windows 3.1:

```
UB=$(ORACLE_HOME)\OCA60
```

Where $(ORACLE_HOME) is replaced with your ORACLE_HOME.