Meridian IVR

SQL Server User's Guide

Publication number: 555-9001-314
Product release: Meridian IVR 2.0/I
Document release: Standard 1.0
Date: February 1996

Printed in the United States of America

Publication history

February 1996

This document is the first standard issue for Meridian IVR release 2.0/I.

Contents

List of figures

List of tables

List of procedures

About this guide

Who should use this manual

This Meridian IVR 2.0/I SQL Server User's Guide has been prepared for application developers who want to interface Meridian IVR 2.0/I applications to an SQL-based ANSI-compliant database management system.

Note: Although use of the SQL server does not require an in-depth knowledge of your database management system, you should be familiar with Standard Query Language (SQL) command syntax and the database that will be accessed (for example, the name of the tables and columns). If you are unfamiliar with your database environment, see your database administrator for assistance.

How to use this manual

Before you begin using the SQL Server cells in your applications, read Chapter 1 for an overview of the SQL Server and a brief discussion of the basic requirements. When you are ready to create an application that will access your database, refer to Chapter 2 to use the graphical application editor or Chapter 3, for a short tutorial. Chapter 4 provides information on accessing remote databases.

Additional Nortel manuals

Related manuals available from Northern Telecom (Nortel) include the following:

- *Meridian IVR Application Development Guide* (555-9001-310)
- *Meridian IVR System Administration Guide* (555-9001-300)

Conventions used in this manual

Throughout this manual, several typographic conventions have been used to highlight certain types of information.

- Items that are parts of the Meridian IVR 2.0/I screens appear in quotes (for example, the "# of Prompts to Play" parameter).
- Items that are file names, messages, or commands you type are shown in bold (for example, the **Enter the name of the database** message).
- Variables shown in command lines appear in italics (for example, *filename* indicates that the word *filename* should be replaced with a specific file name).

Additional sources of information

Bowman, Judith S., Sandra L. Emerson and Marcy Darnovsky. *The Practical SQL Handbook.* 1993.

Date, C. J. *An Introduction to Database Systems*. 1990.

Melton, Jim and Alan R. Simon. *Understanding the New SQL: A Complete Guide*. 1993.

Chapter 1: Introduction

The IVR 2.0/I SQL Server option provides you with additional tools to develop effective voice messaging and interactive voice response (IVR) application solutions. The SQL Server interfaces with several SQL-based ANSI-compliant relational database management systems (DBMS) that allow your applications to retrieve, insert, update, and delete information stored in the supported databases.

This chapter contains the following:

- An overview of the SQL server
- Getting started
- Planning your application

An overview of the SQL server

IVR 2.0/I already supports application access to an information database and provides a method for building a simple database with the System Database Editor (SDE). You may, however, want to develop applications that can access SQL-based relational databases. The SQL Server allows IVR 2.0/I applications to access and update SQL databases. The SQL Server provides this functionality by adding the following ready-to-use cells that you can include in your application:

- QCNT — SQL Select Count Cell
- QDEL — SQL Delete Cell
- QINS — SQL Insert Cell
- QSEL — SQL Select Cell
- QUPD — SQL Update Cell

These cells execute standard SQL COUNT, SELECT, INSERT, UPDATE, and DELETE statements. Each cell corresponds to one SQL transaction. You can define access to database tables or views from within your applications. To make more complex queries, you can use traditional embedded SQL for C (ESQL/C) with IVR 2.0/I User cells.

The SQL Server is designed for an open environment and can interface with Ingres, Oracle, Sybase, and Informix. The following are currently supported:

- Ingres 6.4 (or newer) on SCO 3.0
- Oracle 7.0 (or newer) or 7.1 (or newer) on SCO 3.0
- Informix 5.0 (or newer) on SCO 3.0
- Sybase 10.0 (or newer) on SCO 3.0

Differences between SQL and the SQL server cells

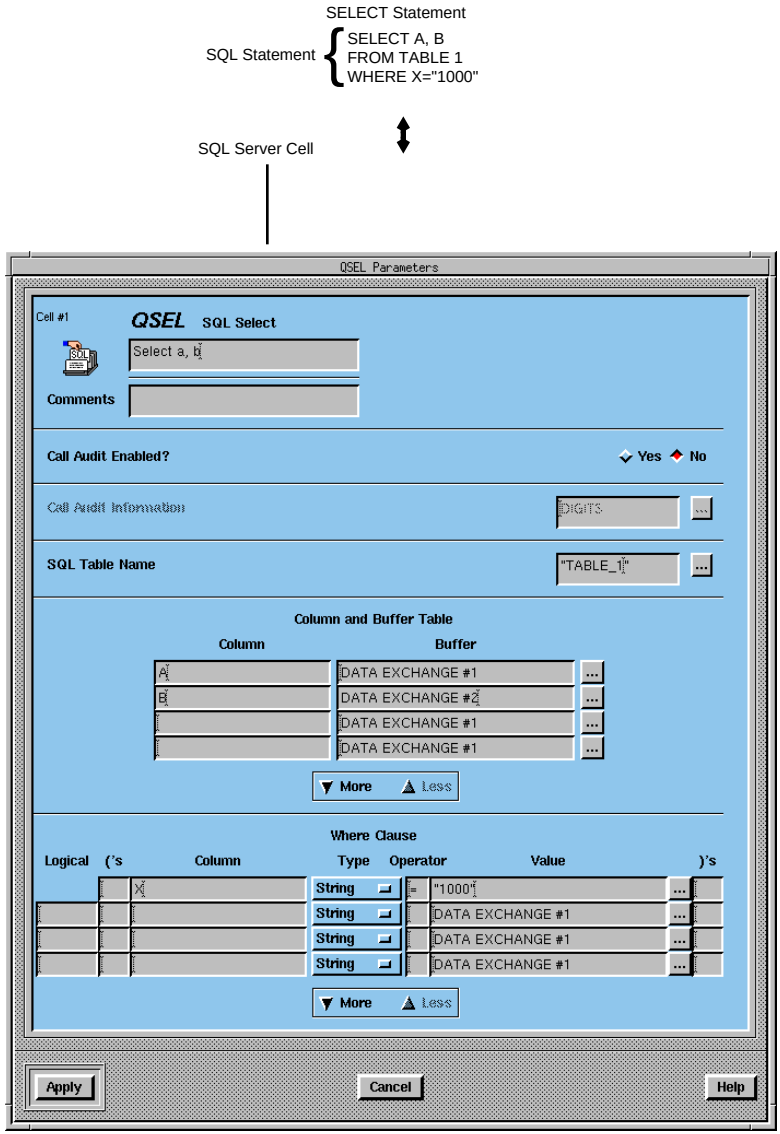
To manipulate information in a database, you normally type one or more SQL statements at your keyboard, or include embedded SQL commands in a C program. The IVR 2.0/I SQL server cells are designed to execute SQL statements as well, but from within a IVR 2.0/I application.

When you create an SQL cell, you are specifying the same information as contained in a normal SQL statement, as shown in Figure 1-1. The cells are simpler to use since there is no need to worry about grammatically correct SQL syntax. All you need to reference is the correct table and column names. The SQL server executes the correct ANSI SQL commands based on the cell parameters that you specify.

The differences between SQL statements and the SQL Server cells follow:

- Only one table per cell is allowed. Consequently, to perform a join, you should build a view, and then reference the name of the view in the appropriate cell's table name parameter. Finally, build a view using your RDBMs.
- Complex queries, such as subqueries, are not supported (except as noted in the previous bullet item, when building a view).
- SQL expression syntax is not completely supported.
- Each cell represents one SQL transaction; you cannot roll back the cell transaction once the cell has completed processing. Each SQL statement is committed if it is successful, and rolled back if it fails.
- The QSEL (SQL select) cell currently returns only the first row of a matched data set.

Figure 1-1
Coding an SQL Statement versus an SQL Server Cell



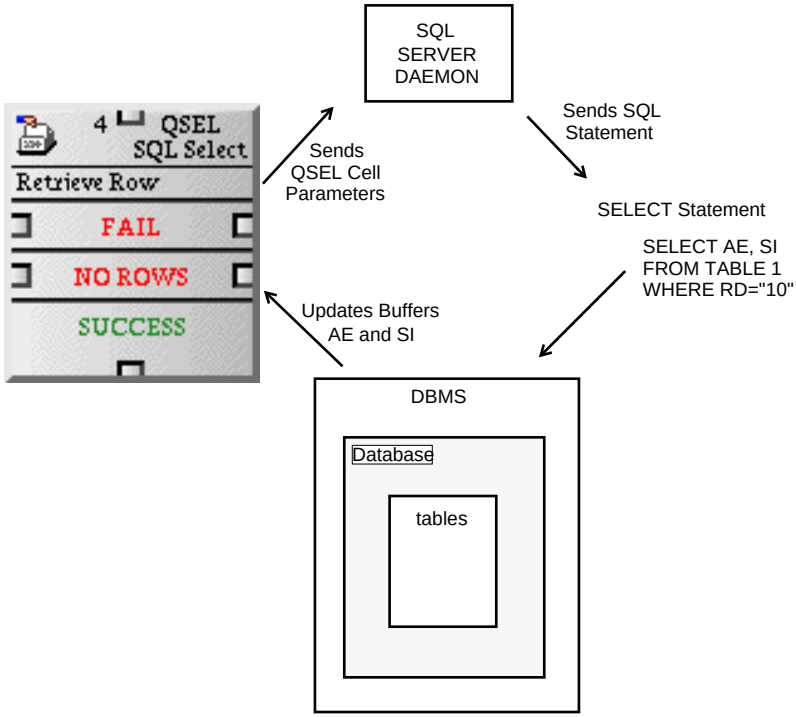
Running in the background

When you boot your system, IVR 2.0/I automatically loads the SQL server as a daemon process (or as middleware if you are using a remote SQL server). The server manages all requests by callers for information stored in the database and executes the corresponding SQL statements in the database management system (DBMS) as illustrated in Figure 1-2. The data that the caller can manipulate is determined by how you define the SQL cells in your application and the privileges you grant to the database user.

For Informix and Sybase, the applications run in the default tablespace defined for the user “vad” with a password of “vad1”. For Ingres, the applications run as the operating system user specified in the default cell. For Oracle, the applications run in the default tablespace defined for the user with a password specified in the default cell.

For both Oracle and Ingres, the password specified in the default cell must be correct. Consult the IVR 2.0/I transaction log verify the accuracy of your password.

Figure 1-2
Managing database queries from callers



In this illustration, when the QSEL (SQL SELECT) cell is processed during execution of the application, the cell parameters are passed to the SQL server daemon. The daemon process submits the corresponding SQL SELECT statement to the DBMS and retrieves the first row that matches the statement's selection criteria. The daemon process updates the output buffers identified by the QSEL cell, and the application continues processing with the next cell.

Getting started

Before you run an application that accesses an SQL-based database, you must meet the following prerequisites:

- Install the Database Management System (DBMS) on the application processor.
- Load (or create) the database that your application will access (or create).
- Create the necessary objects (for example, tables, views).
- Define or create the authorized user account(s), if necessary.
- Grant user access privileges to db objects.

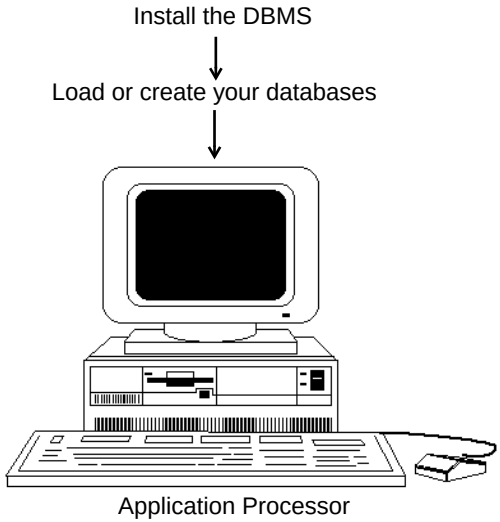
Installing the database management system

You can develop an application that manipulates data in a database, but the application will be useless without the database. IVR 2.0/I SQL Server is compatible with the following database management systems, or the specified platforms, that support ANSI SQL:

- Informix
- Ingres
- Oracle
- Sybase

You must purchase your DBMS separately from IVR 2.0/I and install it on the application processor.

Figure 1-3
Install your DBMS and load the data



Creating or updating the database

Once the DBMS has been installed on the application processor, use the tools associated with your DBMS to create or update the appropriate tables and views. (Refer to the Database Administrator guide accompanying your DBMS, or consult your company's Database Administrator.)

Setting up the user account

The only users who can access your database are those with authorized user accounts. For Informix and Sybase, the applications run in the default tablespace defined for the user "vad" with a password of "vad1". For Ingres, the applications run as the operating system user specified in the default cell. For Oracle, the applications run in the default tablespace defined for the user with a password specified in the default cell.

For both Oracle and Ingres, the password specified in the default cell must be correct. Consult the IVR 2.0/I transaction log verify the accuracy of your password.

Granting access privileges

To ensure that callers can access specific data while preserving the security of the database, the Database Administrator should grant the appropriate access privileges to those tables and views in the database that may be accessed by the user and the user account running IVR 2.0/I.

Planning your application

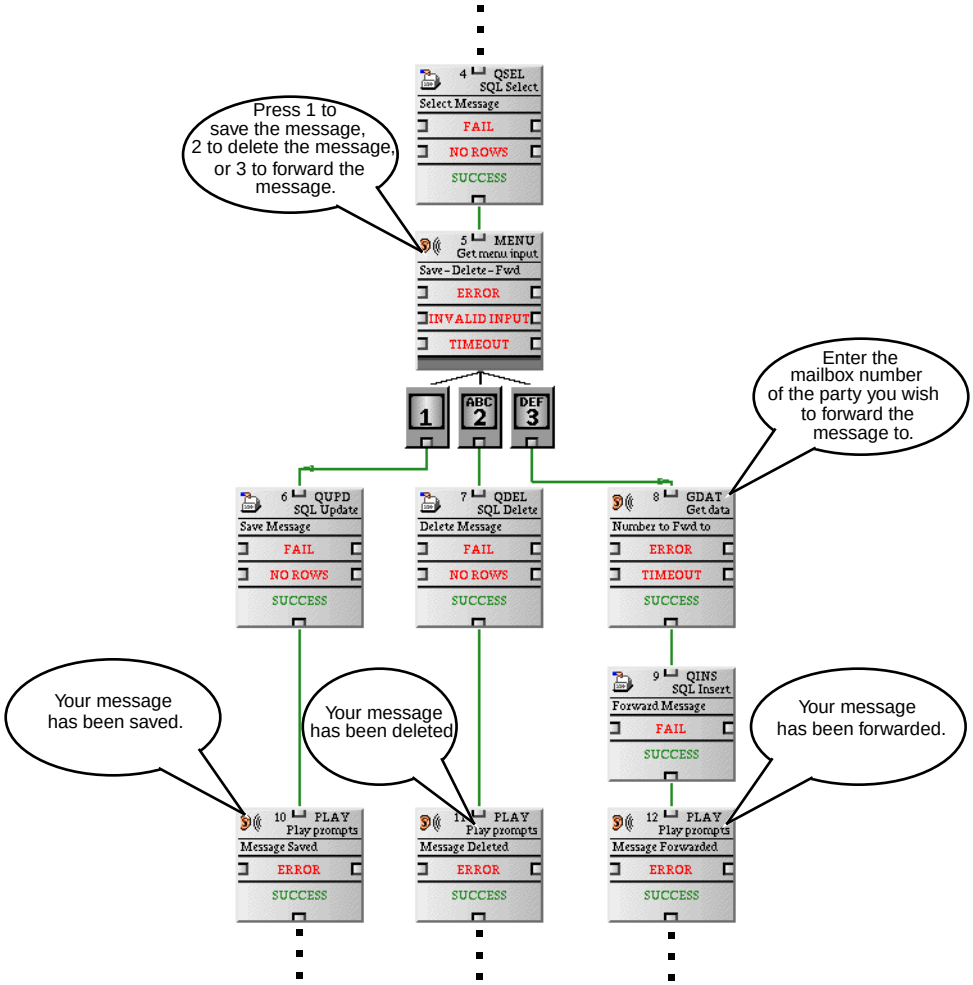
Before you start building your application, you should consider the following:

- What information do you want callers to be able to manipulate?
- Where is this information stored (that is, which database and tables)?
- Should callers be able to retrieve, insert, update, and/or delete data?

Once you have the names of the databases, tables, and columns where the data resides and have granted the appropriate privileges to the IVR 2.0/I accounts, you are ready to build your application.

Figure 1-4 shows an application that inserts, deletes, or updates information into a database based on a comparison made of the caller's input against the contents of several buffers.

Figure 1-4
Sample application showing Select, Update, Delete, and Insert

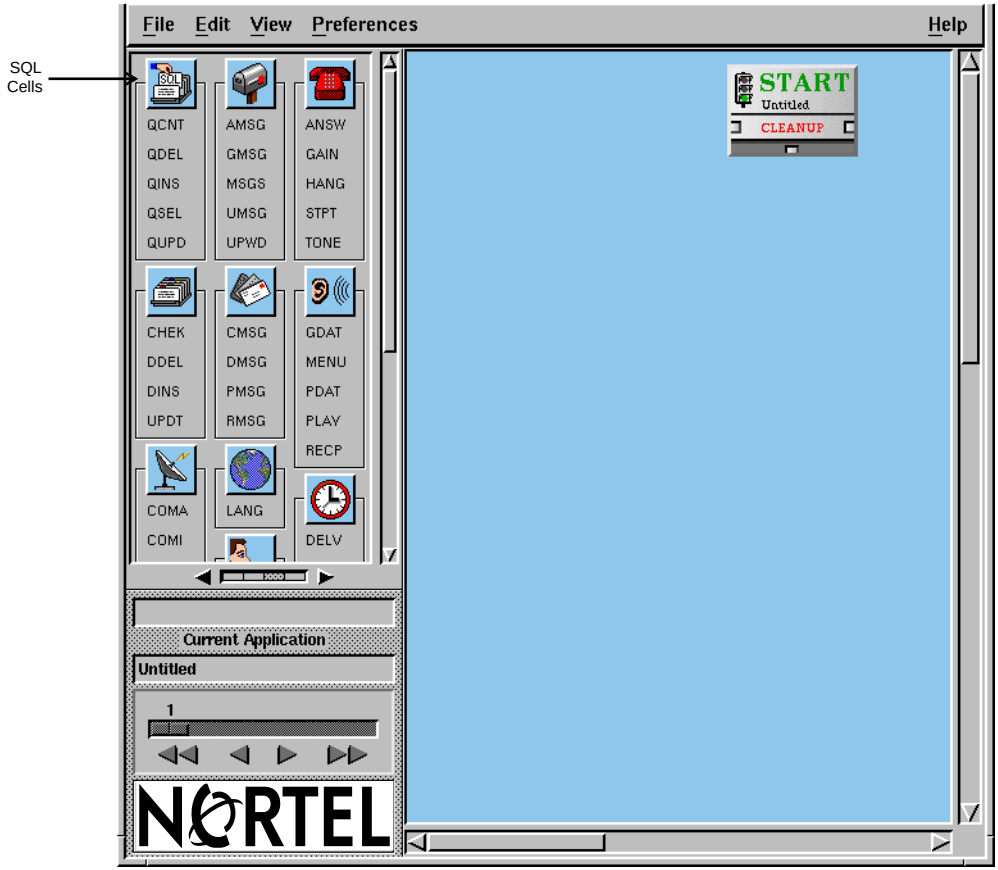


Chapter 2: Building an SQL application

The IVR 2.0/I SQL Server has five cells that you can include in your application to allow callers to manipulate data in your database. Figure 2-1 illustrates the different SQL server cell types as they appear on the Application Editor palette. This chapter describes each of the SQL cells that you can use in addition to the following information:

- Counting selected rows with QCNT (SQL SELECT COUNT)
- Deleting a row with QDEL (SQL DELETE)
- Inserting a new entry with QINS (SQL INSERT)
- Retrieving information with QSEL (SQL SELECT)
- Updating a row with QUPD (SQL Update)

Figure 2-1
The Application Editor palette with SQL cells



The SQL Server cells are designed for your convenience. If you have experience using standard and embedded SQL statements, you can use the SQL Server cells to quickly design IVR 2.0/I applications for accessing your database.

Note: SQL cells work within the limitations specified in the next section.

Common characteristics of SQL server cells

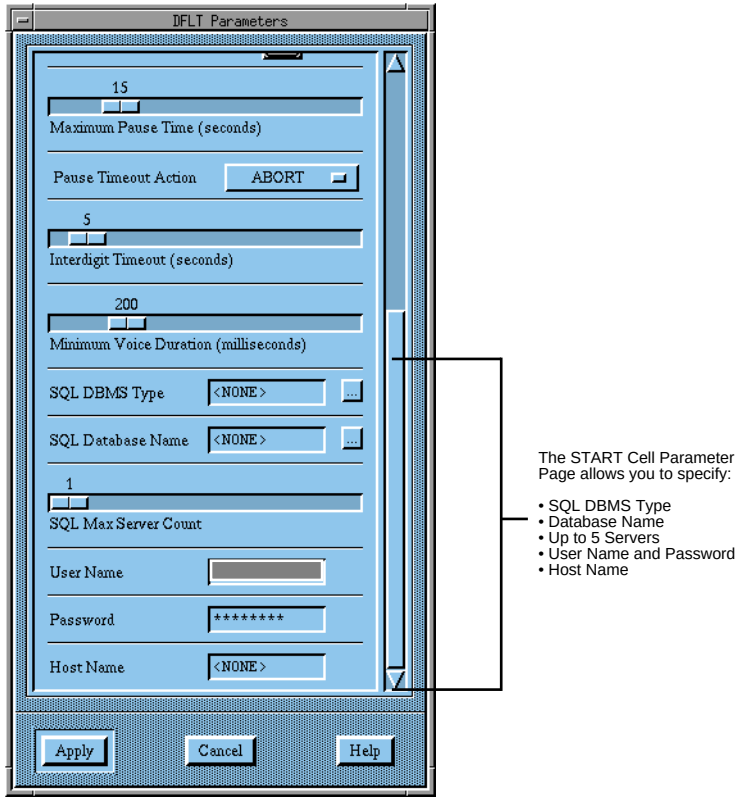
As shown in Figure 2-1, each of the SQL Server cells is represented in the Application Editor as a cell icon. You can place a cell in your application by selecting the appropriate cell type from the palette.

If an application uses SQL cells, specify the DBMS type, the database name, the server count, the user name and the password in the default cell as shown in Figure 2-2.

Each SQL cell includes:

- The table with the data which will be manipulated, as shown in Figure 2-3.
- The columns in those cells that manipulate column data and the associated value or buffer containing the value (that is, inserting, updating, or deleting values), as shown in Figure 2-3.
- The where clause, in those cells requiring selection criteria, as shown in Figure 2-3.

Figure 2-2
Default cell parameter



SQL default cell parameters

You must specify the following SQL parameters in the Default cell:

SQL DBMS Type Specify the DBMS type that you are using for your application. Current DBMS types supported by IVR 2.0/I are: Informix, Ingres, Sybase, and Oracle.

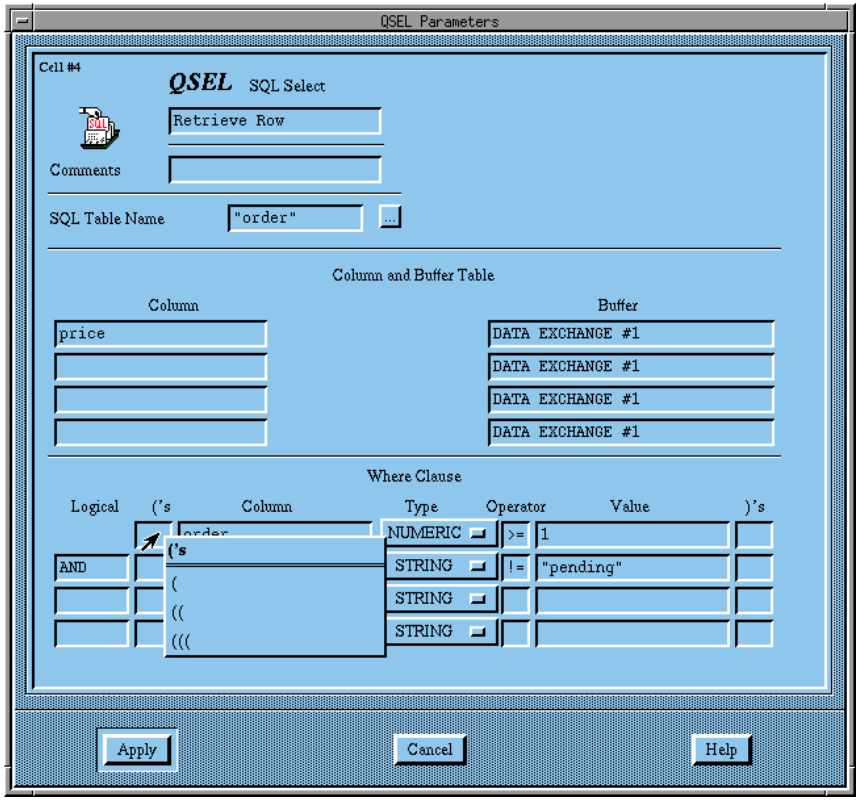
SQL Database Name Specify the database name. The database must be previously set up. Although the buffer allows this name to be up to 31 characters long Informix only allows 8 characters, Sybase allows 12.

SQL Max Server Count Specify the number of servers to use (up to five). The number of servers to be specified depends on the number of queued requests. The performance of your SQL queries will suffer if you do not have an adequate number of SQL servers selected. The recommended number at this time is one.

Host Name Specify the remote host name. Use the host name only if the database is remote.

Figure 2-3
QSEL parameter

Specify the table, the column, and the WHERE clause in the appropriate fields. You can get a popup containing valid values for the where clause fields by clicking the right mouse button on the field. See parenthesis popup below.



Specifying clauses

You can enter one or more clauses. Move the cursor to the field where you wish to enter a value and type a value. You can also select a value by selecting the field and pressing the right mouse button. A pop-up appears containing the valid values for that field. Move the cursor to the value you want and release the right mouse button. The value appears in the field. The possible fields where you can enter a value, as shown in Figure 2-3, are:

LOGICAL Enter a logical (Boolean) operator (**AND**, **NOT**, **ANDNOT**, **ORNOT**, or **OR**) when specifying a compound expression.

(
Enter left parenthesis as single, double, triple [**(**, **((**, or **(((**].

COLUMN
Enter the name of a column, up to 31 characters.

Type
Press the button and choose the data type. Valid choices are:

- string
- numeric
- money
- date
- float

OPERATOR
Enter the relational operator, such as =, **!**=, >, <, >=, <= (equal, not equal, greater than, less than, greater than or equal to, less than or equal to).

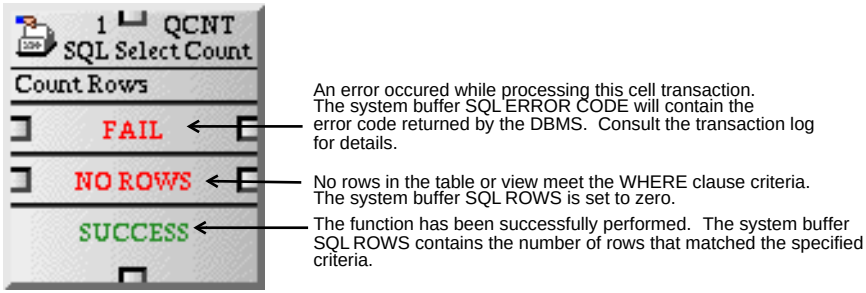
VALUE
Enter a value against which the query is matched. The value can be a system buffer, an application buffer, a number, or a hard coded value surrounded by double quotes.

)
Enter closing parenthesis as single, double, or triple [**)**, **))**, or **)))**].

Counting selected rows with QCNT (SQL select count)

Use the QCNT cell to count the number of rows in a table that match specific criteria. The QCNT cell parameters identify the table to be accessed and the WHERE clause selection criteria. Figure 2-4 shows the QCNT cell and the possible next cells. Figure 2-5 shows the QCNT parameters.

Figure 2-4
QCNT cell



Entering an SQL statement in the QCNT parameter

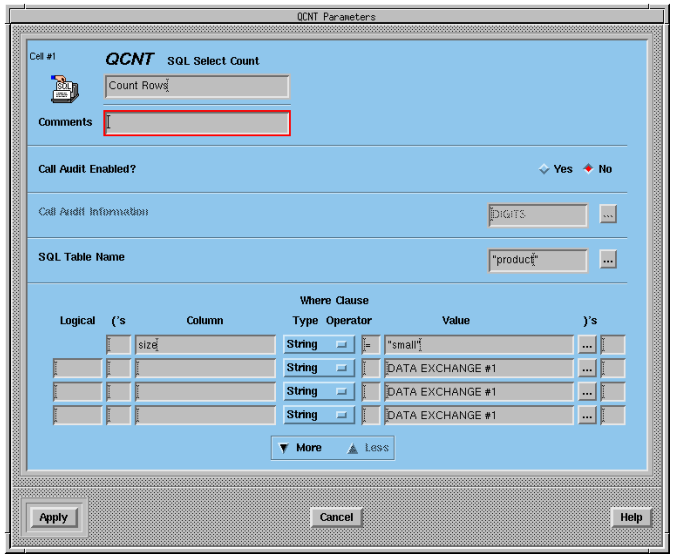
Suppose that you want to count all the rows in table “product” that have the value in the “size” column equal to “small.” Normally you would code the query to the database table by entering the following SQL statement:

```
SELECT COUNT (*) FROM product

WHERE  size = 'small'
```

When you create the QCNT cell, you specify the following values in the cell parameters window as shown in Figure 2-5:

Screen location	Field	Specify
	SQL Table Name	product
Under the Where Clause	Column	size
	Operator	=
	Value (	“small”

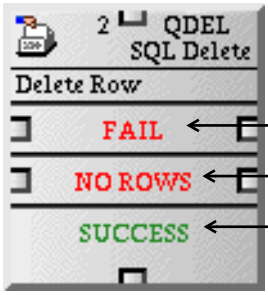
Figure 2-5
QCNT parameters



Deleting a row with QDEL (SQL delete)

Use the QDEL cell to delete a row in a database table. The QDEL parameter identifies the table to be accessed. The rows to be deleted are specified by a WHERE clause. Figure 2-6 shows the QDEL cell and the possible next cells.

Figure 2-6
QDEL cell



An error occurred while processing this cell transaction. The system buffer SQL ERROR CODE will contain the error code returned by the DBMS.

No rows in the table or view meet the WHERE clause criteria. The system buffer SQL ROWS is set to zero.

The delete rows transaction was successful. The system buffer SQL ROWS will contain the number of rows that have been deleted.

Entering an SQL statement in a QDEL parameter

Suppose that you want to enable a caller to delete an order that is currently pending in the database. You would code the following SQL statement to delete the order:

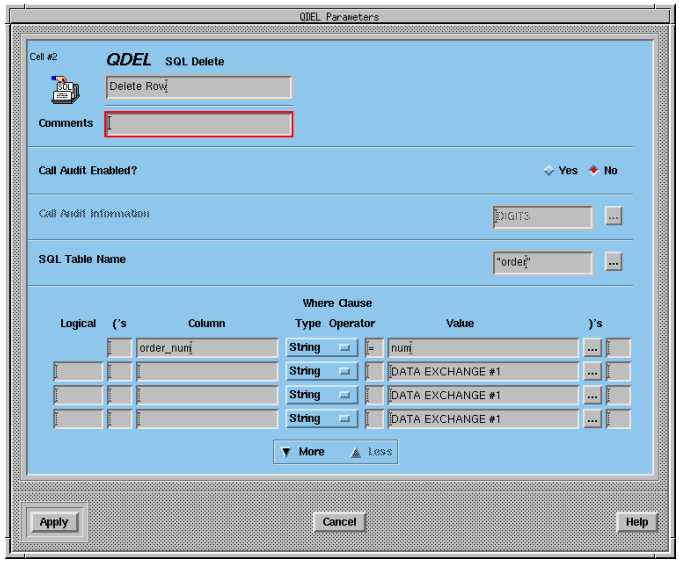
```
DELETE FROM order WHERE order_num = :num;
```

Note: Due to the syntax convention for the SQL DELETE statement, it is possible to delete all rows from a table using the statement: **DELETE from ORDER**. The Meridian IVR SQL cell performs this same statement if the WHERE clause is omitted.

When you create the QDEL cell, you specify the following values in the cell parameters window as shown in Figure 2-7

Screen Location	Field	Specify
	SQL Table Name	order
Under the Where Clause	Column	order_num
	Operator	=
	Value (	num

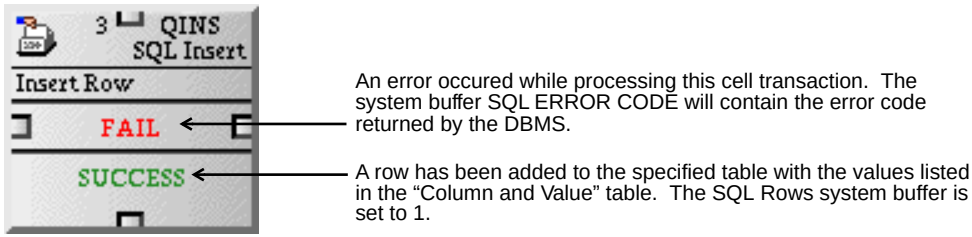
Figure 2-7
QDEL parameter



Inserting a row with QINS (SQL insert)

Use the QINS cell to insert a row of data into a database table. The QINS parameter identifies the table to be accessed. The Column and Value table specifies each column in a new row, the column type, and the corresponding value to be inserted. The column "type" can be numeric, string, money, date, or float. The "value" you specify can be a specific value or a buffer containing the value. Figure 2-8 shows the QINS cell and the possible next cells.

Figure 2-8
QINS cell



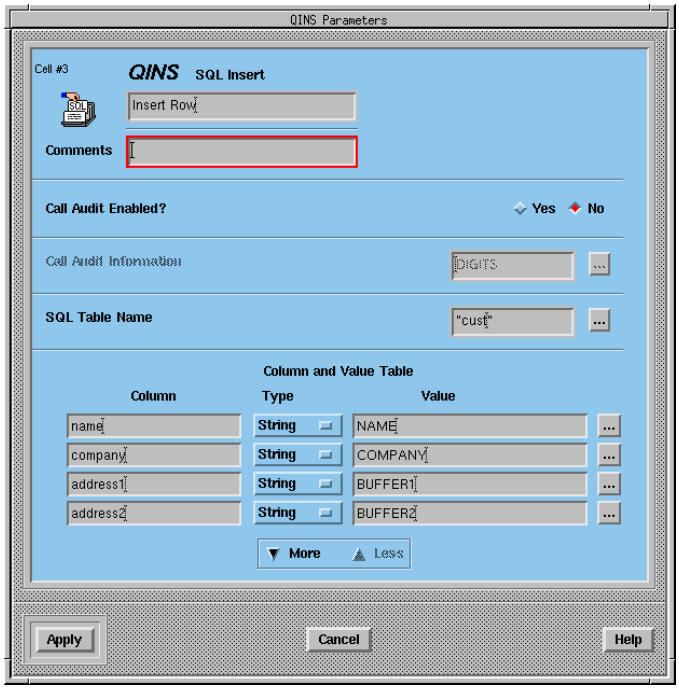
Entering an SQL statement into a QINS parameter

Suppose that you want to enter the name, company, two separate addresses, city, state, zip code, and phone number of a customer into a specific database. You would identify the table and columns that would be affected and the corresponding buffers with the following SQL statement:

```
INSERT INTO cust (name, company, address1, address2, city,
state, zip, phone)

VALUES (:NAME, :COMPANY, :BUFFER1, :BUFFER2, :BUFFER3,

                          :BUFFER4, :ZIP, :DIGITS);
```

When you create the QINS cell, you specify the following values in the cell parameters window as shown in Figure 2-9. Note that the "TYPE" changes to "numeric" for the zip value and the phone value.

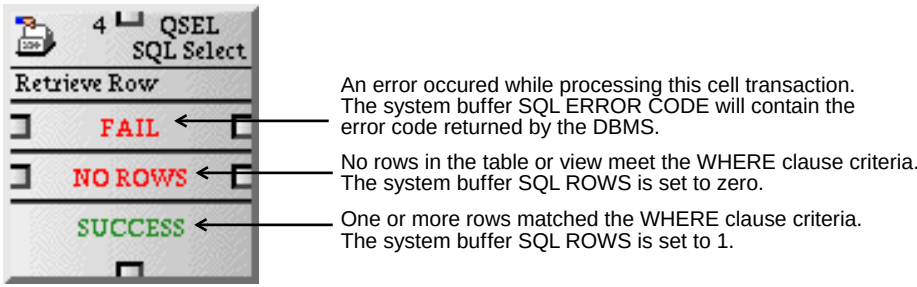
Figure 2-9
QINS parameter



Retrieving information with QSEL (SQL select)

Use the QSEL cell to retrieve a single row of data from a table. The QSEL parameters identify the table or view to be accessed; the columns whose values should be returned; and the WHERE clause selection criteria. You can specify the next cell based on the number of rows selected, whether an error occurs, or if a row is successfully returned. Figure 2-10 shows a QSEL cell and the possible next cells.

Figure 2-10
QSEL cell



Note: Unlike a SELECT statement that can retrieve numerous rows, QSEL only returns one row.

Entering an SQL statement into a QSEL parameter

Suppose that you want to retrieve a row where "order" is greater than or equal to "1" and where "status" is not "pending." You would code the SQL statement to retrieve a row meeting these characteristics:

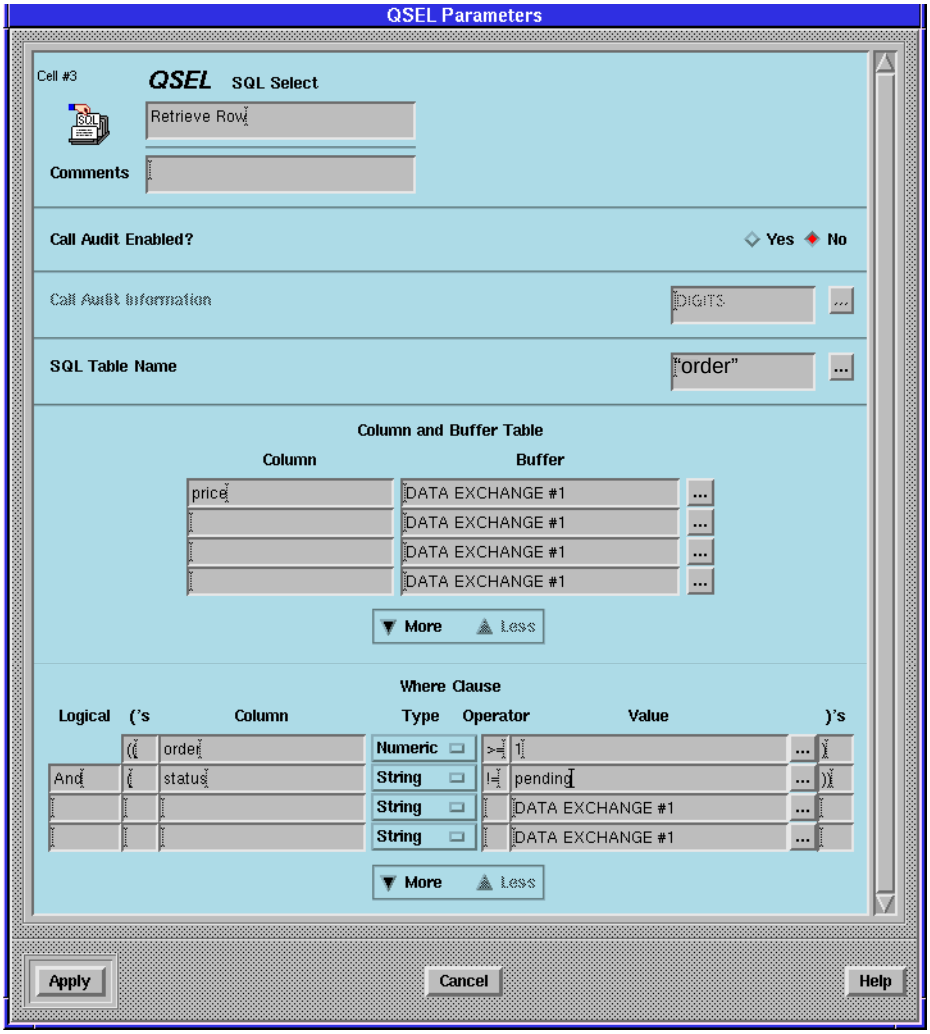
- SELECT PRICE FROM order
- WHERE num > = 1
- AND status ! = pending

Use the values described in Table 2-1. They specify what should be returned when a row is retrieved as shown in Figure 2-11.

Table 2-1
Coding an SQL statement

Screen Location	Field	Specify
	SQL Table Name	order
Under the Column and Buffer Table	Column	price
Under the Where Clause	Column	num
	Operator	>=
	Value (	1
For the “And” clause	Column	status
	Operator	!=
	Value (	pending

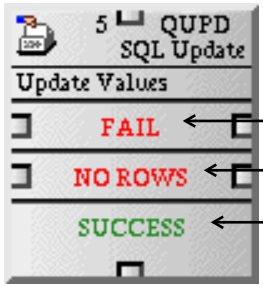
Figure 2-11
QSEL parameters



Updating a row QUPD (SQL updated)

Use a QUPD cell to modify values in existing rows in a database table. QUPD parameters identify the table to be accessed. The rows to be modified are specified by a WHERE clause. Figure 2-12 shows a QUPD cell with the possible next cells. Figure 2-13 shows the QUPD parameters.

Figure 2-12
The QUPD cell



An error occured while processing this cell transaction. The system buffer SQL ERROR CODE will contain the error code returned by the DBMS.

No rows in the table or view meet the WHERE clause criteria. The system buffer SQL ROWS is set to zero.

One or more rows were updated in the specified database table. The system buffer SQL ROWS will contain the number of rows that were modified.

Entering an SQL statement in a QUPD cell

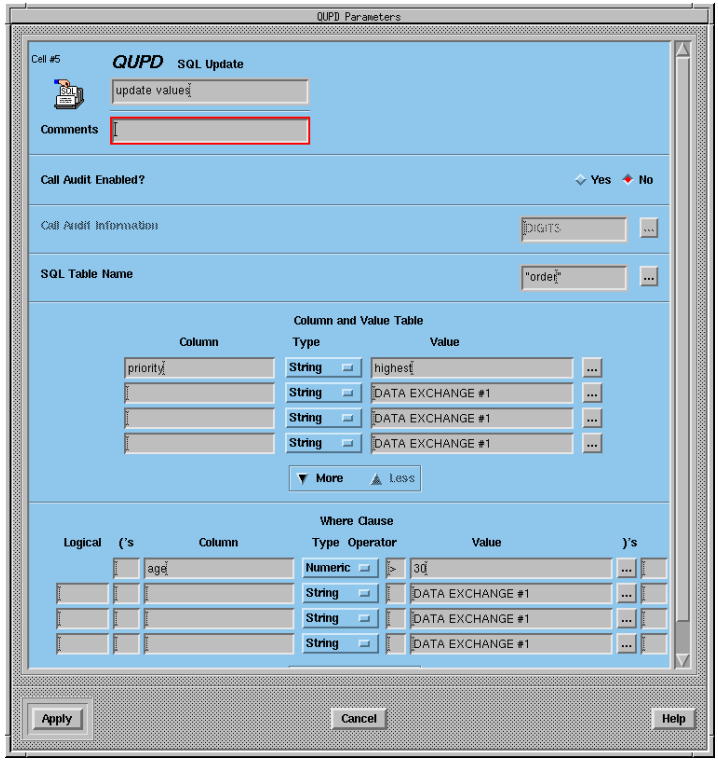
Suppose that you want to upgrade a customer's order priority in the "order" table to "highest" if the customer's order is older than 30 days. If the age of the order has been retrieved (by a previous QSEL cell) into buffer "age," you would code the following SQL statement to upgrade the order:

```
UPDATE order

SET priority = 'highest'

WHERE age > 30 ;
```

When you create the QUPD cell, you specify the following values in the cell parameter window as shown in Figure 2-13:

Screen Location	Field	Specify
	SQL Table Name	order
Under the Column and Value Table	Column	priority
	Value	highest
Under the Where Clause	Column	age
	Operator	>
	Value (	30

Figure 2-13
QUPD parameter window



Chapter 3: An SQL server tutorial

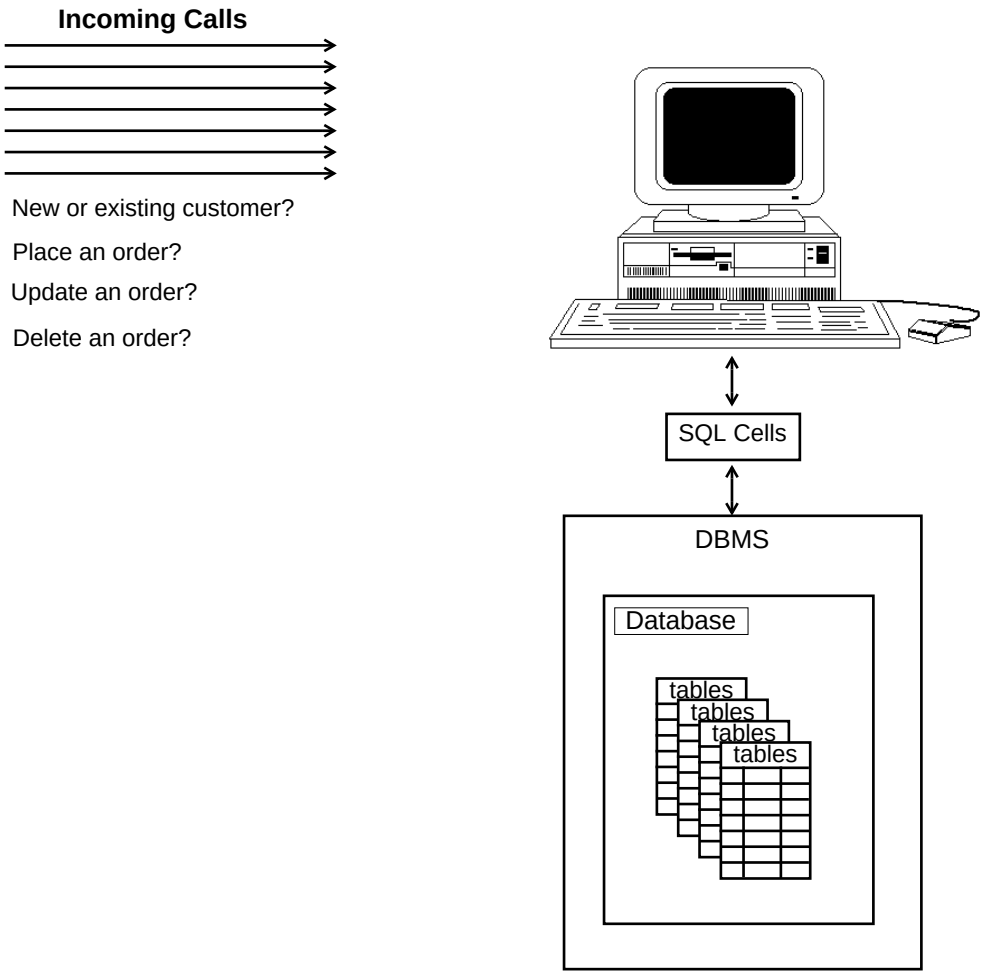
This chapter briefly describes an application that accesses an Ingres database and shows the use of IVR 2.0/I SQL cells.

The following paragraph describes a typical scenario of a distributor who sells clothing by catalog. The application is being implemented to streamline order processing.

Michelle Jordon is an application developer at Kendall and Ives, Inc., a national clothing distributor that uses catalog sales to distribute its products. Michelle is in charge of developing an IVR application to provide high-quality service to customers calling in orders or requesting the status of their orders.

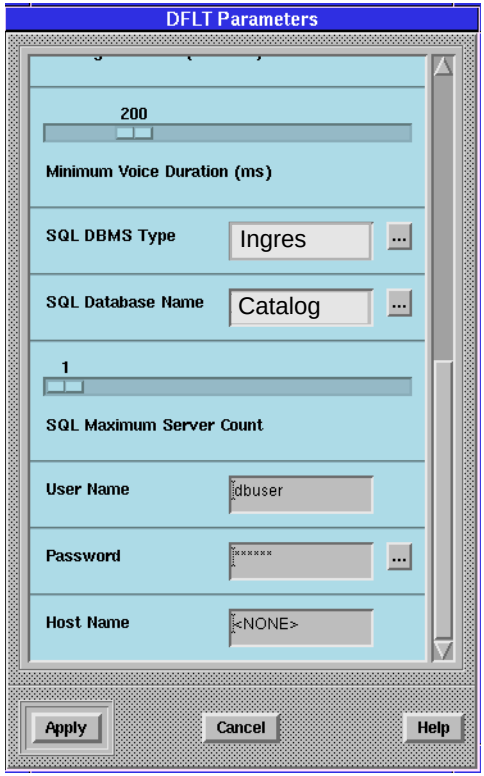
The application will access a database called “catalog,” which contains customer, product, and order information. Customers can call an 800 number to access the company’s Customer Service Division to place an order, check order status, update an order, or delete an order.

Figure 3-1
Sample IVR catalog ordering application



When Michelle defines the default cell parameter, she identifies the SQL DBMS type as “INGRES” and the database name as “catalog”.

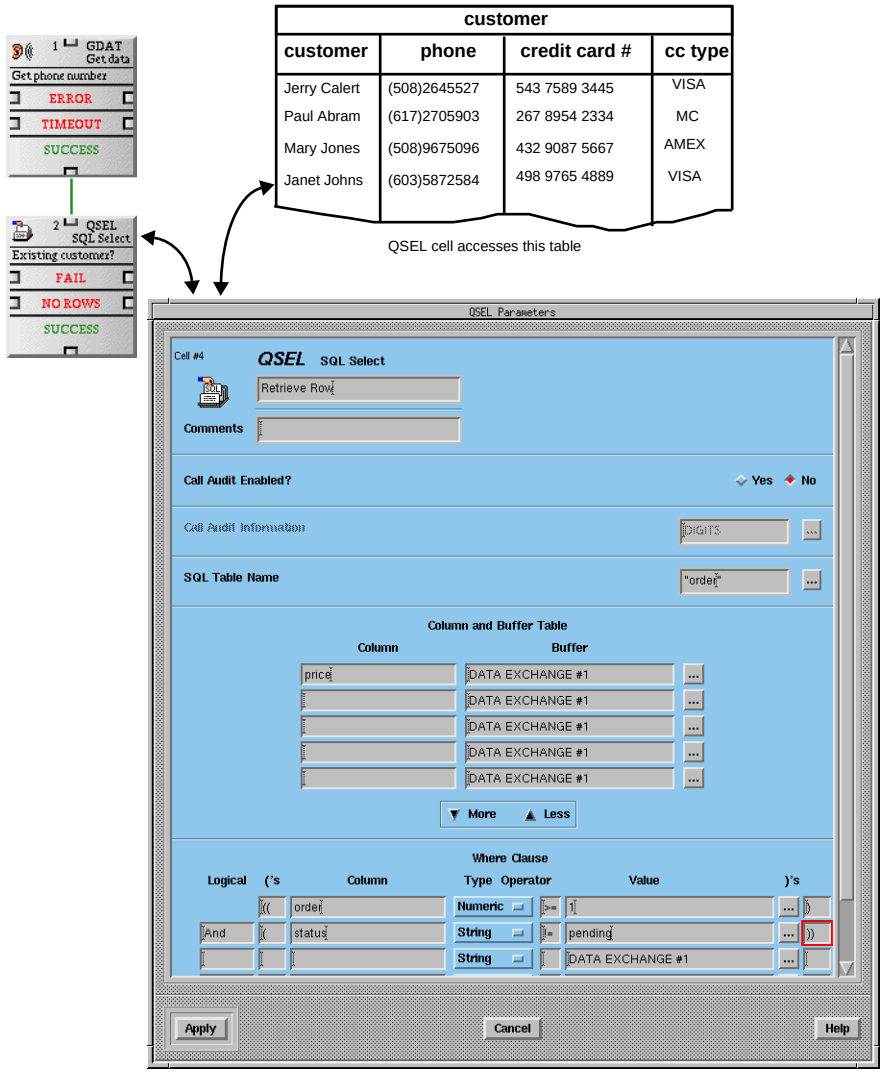
Figure 3-2
DBMS type and database name parameters



One of the first tasks the application will do will be to query the database to see if the caller is an existing customer. To do this, Michelle creates a QSEL cell to search the customer table for all entries where the value of the “phone” field is equal to the telephone number that the caller enters in response to an earlier GDAT cell.

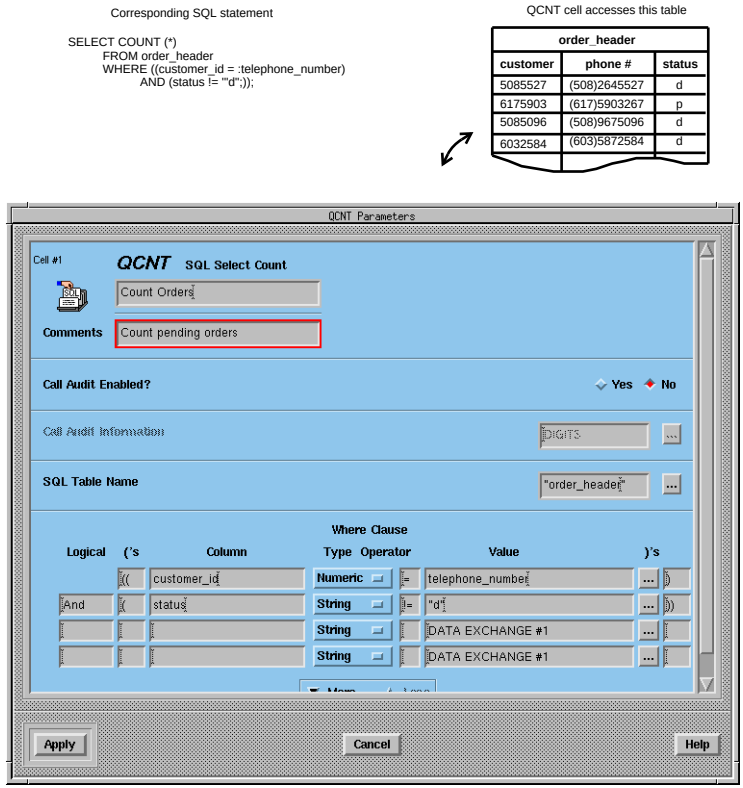
If there are no entries (that is, no rows are selected), the caller is identified as a new customer. If an error occurs while the cell is being processed, the application will play the message and transfer to a live operator. Figure 3-3 shows the QSEL cell created and its parameters, the next cell specified, and the table that is being queried.

Figure 3-3
QSEL Parameters



If the caller is an existing customer, the application checks to see if there are any outstanding orders. For this transaction, Michelle creates a QCNT cell to count the number of rows in the “order_header” table where the “customer_id” field is equal to the customer’s telephone number and the status field is not equal to “d” (for delivered). Figure 3-4 shows the parameters for this cell, the corresponding embedded SQL statement processed, and the table that is queried.

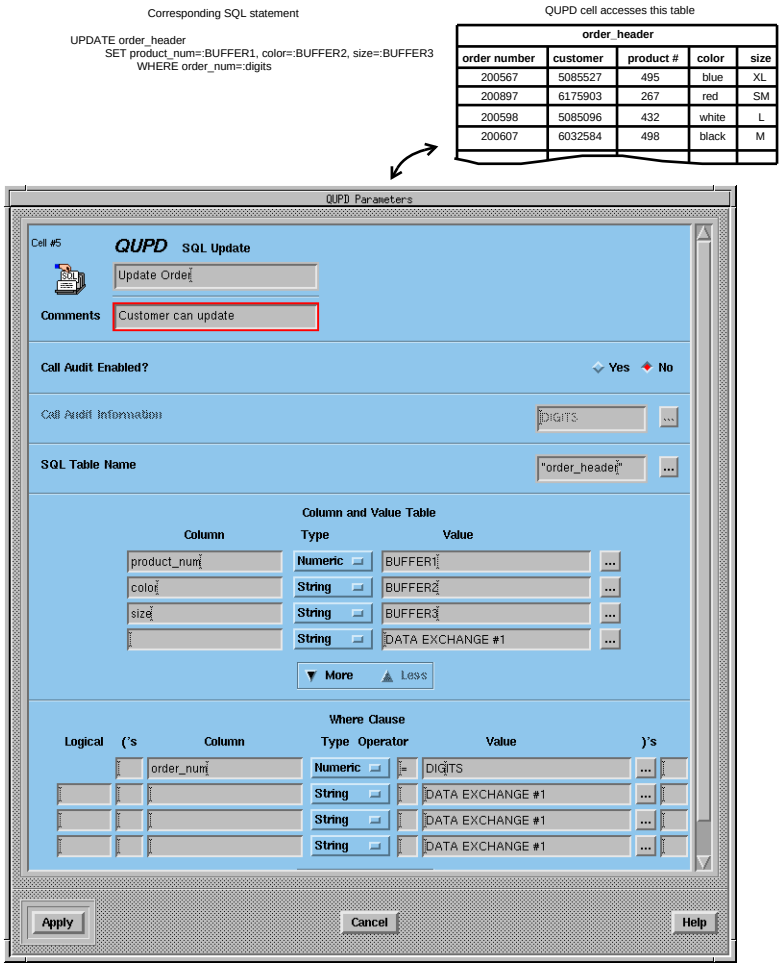
Figure 3-4
QCNT Cell for executing a SELECT COUNT statement



Updating an existing order

To allow a customer to update an existing order, Michelle creates a QUPD cell, as shown in Figure 3-5. This call will execute this embedded SQL UPDATE statement:

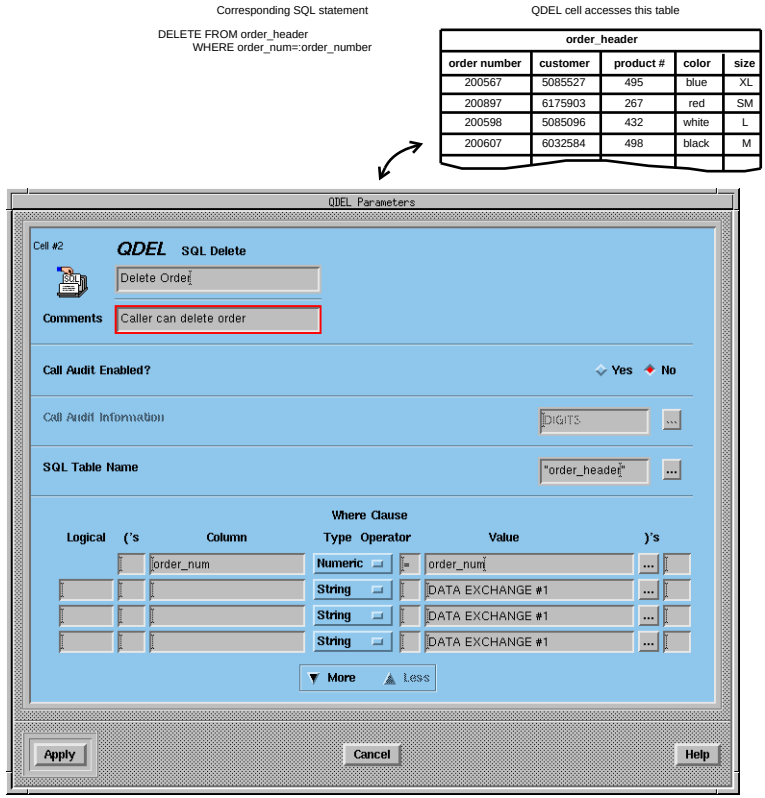
Figure 3-5
Identifying the table to be updated with the QUPD cell



Deleting an order

The application that Michelle is designing with IVR 2.0/I must also permit a caller to delete an order. With the QDEL cell, Michelle can execute a DELETE statement to remove the appropriate row from the order_header table as shown in Figure 3-6. The cell will execute the embedded SQL statement.

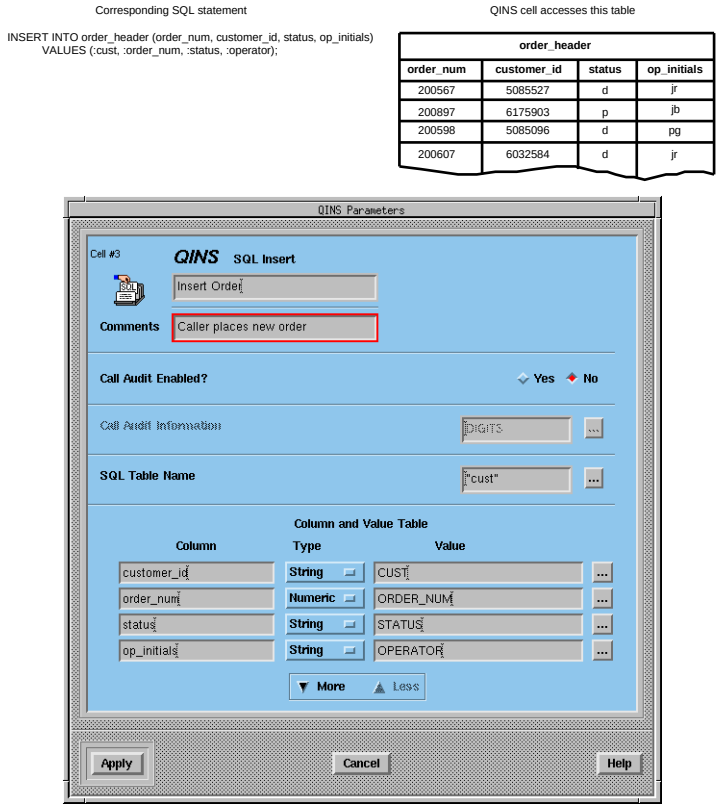
Figure 3-6
Identifying the table to be accessed by the QDEL cell



Inserting information into the database

Throughout the application, Michelle includes the QINS cell to insert data into different tables in the CATALOG database. For example, when the caller decides to place a new order, the QINS cell shown in Figure 3-7 inserts a new row in the order_header table. This call loads different values in the order_num, customer_id, status, and op_initials columns, corresponding to the embedded SQL statement.

Figure 3-7
Identifying the table to be accessed by the QINS cell



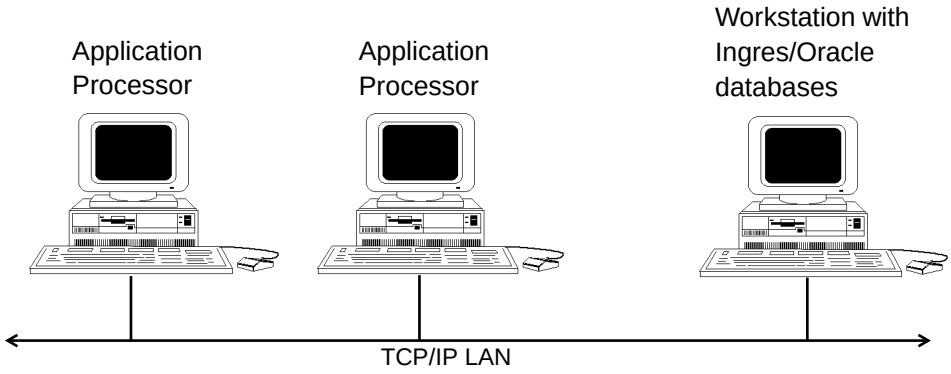
Later, Michelle uses additional SQL cells to insert order detail, update the order total, and update the number of orders associated with a particular telephone order. She can add new fields as needed in a piecemeal fashion without disrupting the existing users as needs change. If you are familiar with standard and embedded SQL statements, you can quickly design IVR 2.0/I applications to access your database.

Chapter 4: Accessing remote databases

Meridian IVR 2.0/I supports remote database access for all supported SQL databases (Ingres, Informix, Oracle and Sybase).

If your SQL databases (Ingres, Informix, Oracle, or Sybase) do not reside on your application processor, Meridian IVR 2.0/I can access them via your TCP/IP LAN. Accessing an SQL database remotely means you do not have to move the databases to your application processor. Figure 4-1 shows two APs and a workstation residing on a TCP/IP LAN. In this situation, the application processor acts as the client and the workstation acts as the server.

Figure 4-1
Typical LAN layout



Ingres

Perform the following to allow your Meridian IVR 2.0/I applications to access remote Ingres databases. Refer to your Ingres documentation for information on configuring your Ingres database with the Ingres utility ***NETU***. Before starting this procedure, you will need:

- the name and TCP/IP address of the remote node
- the Listener Address of the remote node. This is the value set according to the instructions in the Ingres documentation.
- a username and password for Meridian IVR 2.0/I to use to access the data (if different from the default Meridian IVR 2.0/I SQL username and password)

Procedure 4-1
Creating a remote node

1 Login to your application processor as Ingres. Bring up the Ingres utility NETU and create a new remote node with the following settings:

Node type:	Global
Network Type:	TCP/IP
Global/Private Authorization:	GLOBAL

2 Make sure that the username and password created for Meridian IVR 2.0/I provides remote authorization entry and access to the correct databases, and corresponds to the SQL username and password you specify in the Default Cell Parameter Page.

Procedure 4-2
Verify access to the remote database

To verify that you have access to the remote database, quit NETU and perform the following:

1 At the command line, enter:

sql <remote nodename>:<dbname>

where ***<remote nodename>*** is the name of the remote node, and ***<dbname>*** is the name of the database you want to access.

2 The command line prompt will change. At the prompt, enter

***set autocommit on* \q**

3 At the prompt, enter

select * from <table>

where ***<table>*** is the name of one of the tables in the database.

4 The system should respond with remote database information.

If you are able to access the database, bring up Meridian IVR 2.0/I and try running an application. If you are not able to access the database, see the following section on Troubleshooting.

Troubleshooting a remote Ingres connection

If you are unable to access the remote database after performing the steps outlined in the previous section, try the following:

1 Start with the basics. Make sure both the client and server can see each other on the network.

2 Make sure that Ingres is running on both the client and server.

3 Make sure that the Ingres Communications Server (iigcc) is running on both the client and server. The Communications Server should come up when you bring up Ingres.

4 Stop the Communications Server and bring it back up again. Do this through NETU.

Informix

Perform the following to allow your Meridian IVR 2.0/I applications to access remote Informix databases. Refer to your Informix documentation for information on configuring your Informix database with the Informix utility **DB-Access**. Before starting this procedure, you will need

- the name and TCP/IP address of the remote node
- the path to the database

Make sure your username and password has access to the Informix database.

Install the Informix i_net server on the Application Processor.

Change the DBPATH environment variable on the Application Processor to:

```
//machine_name/directory
```

where **machine_name** is the name of the machine with the Informix database, and **directory** is the directory containing the Informix database.

Sybase

Perform the following to allow your Meridian IVR 2.0/I applications to access remote Sybase databases. Refer to your Sybase documentation for information on configuring your Sybase database with the Sybase utility **NETU**. Before starting this procedure, you will need

- the name and TCP/IP address of the remote node
- the path to the database

Procedure 4-3
Access a remote Sybase database

1 On the Application Processor, set the environment variable DSQUERY to the value found in $Sybase/interfaces on the Sybase server.

2 On the Sybase server, start the Sybase server software by entering:

$SYBASE/install/startserver -f RUN_server_name

where ***server_name*** is the name of the server.

3 Do the following to verify that you have remote database access. From your Meridian IVR account on your Application Processor, enter:

isql -s *server_name*

where ***server _name*** is the Sybase server's name.

4 At the prompt, type:

`select * from table`

Where ***table*** is the name of the database table you are accessing.

The Sybase server should respond with remote database information.

Chapter 15 of the Sybase System Administration Guide contains detailed information on accessing Sybase databases remotely.

Remote Ingres and ORACLE7 database access

To access databases remotely, enter the **<nodename>** in the DFLT cell. The values follow:

For Ingres: The nodename on which the database exists. For example, if you need to access data from a database that exists on a node called terra, enter terra in the **<nodename>** field of the DFLT cell. Make sure that the name terra is defined as a **V_NODE** while setting up Ingres/Net.

For ORACLE7: The nodename that you entered for the field HOST (under "Editing Addresses") while setting up the ORACLE TNS network. The "Editing Addresses" option is part of the "Editing Listener" option. To verify that you have entered the correct nodename, check

`$ORACLE_HOME/network/admin/tnsnames.ora`.

The nodename should be the first entry in this file.

ORACLE7 searches the **/etc** directory (which is the default) for specific configuration files. If you wish to locate the configuration files in a directory other than the default, you must designate that directory with the environment variable **TNS_ADMIN**.

To do this, edit the <**oracle.conf**> file in the **sys_files** directory, located under the home directory (the home directory is the same one under which Meridian IVR is installed). In this file, find the comment line which specifies the environment variable value for **TNS_ADMIN** and remove the comment symbol. This line will point ORACLE7 to the directory **$ORACLE_HOME/network/admin**.

Oracle database background

There are two possible scenarios that a multiple-instance Oracle installation may need to address. Note that Oracle is an extremely rich RDBMS, and other variations are possible. Any combination of the following two scenarios will be directly addressed in this release.

- Multiple Instance/Databases using the same Oracle product tree
- Multiple Instance/Databases using different Oracle product trees

There are three basic steps that must occur prior to accessing data residing in an Oracle table:

- Create an Oracle Instance
- Mount the Database
- Open the Database

Creating an Oracle Instance involves allocation of an Oracle SGA (a shared memory region) and starting a set of Oracle background processes.

Mounting an Oracle database associates a physical database with a running instance. For our discussions there is a one-to-one correspondence between Instances and databases, both identified by the same UNIX environment variable **ORACLE_SID**.

You must open the database before you can access it.

Oracle instance/database identification

The Instance is the set of processes and SGA, and the database is the physical manifestation of data. An Oracle Instance and an Oracle Database can be considered identical. Both of these are identified by the single UNIX environment variable **ORACLE_SID**. Prior to connecting to an Oracle Instance, you must set this variable to the correct value.

Oracle

Perform the following activity to allow your Meridian IVR 2.0/I applications to access remote Oracle databases. Refer to your Oracle documentation for information on configuring your Oracle database with the Oracle utility ***net_conf***. Before starting this procedure, you will need

- SQLNet V2 for installation on both Client and Server
- A new net_conf password
- Remote node name and SID address
- Network Name
- Community Name
- Listener Address
- Address Summary

You can obtain this information from the party responsible for setting up or maintaining your Oracle database.

Procedure 4-4
Access a remote Oracle database

1 Install SQLNet V2 on both the local and remote systems.

2 Login to the application processor as ORACLE.

3 Change the password for net_conf. Changing the password for net_conf ensures that your system will remain secure.

4 Use net_conf to provide the following information about the remote node:

- Network Name
- Community Name
- Listener Address (should include the node name and SID address of the remote system)
- Address Summary (should include the node name and the location of ORACLE_HOME on the remote node)
- Terminal Type

5 Call up tnsnames.ora from the client to verify that it includes the following code:

```
#########################
# FILENAME: tnsnames.ora
# NETWORK.: SCO_NET
# SERVICE.: NA
#########################
scodevil (This is the service name)
=
  (DESCRIPTION =
    (ADDRESS_LIST =
        (ADDRESS =
          (COMMUNITY = SCO_LIST)
            (PROTOCOL = TCP)
            (HOST = hostname)
         (PORT = 1521)
          (PORT=1521)
        )
    )
    (CONNECT_DATA =
      (SID = ora)
    )
  )
```

This line must contain the nodename of the server as specified in the net_conf

6 To verify that you have access to the remote database, go to the command line and enter

sqlplus username/passwd@<nodename>

where ***<nodename>*** is the node where the database exists.

If you are able to access the database, bring up Meridian IVR 2.0/I and try running an application. If you are not able to access the database, see the following section on Troubleshooting.

Troubleshooting a remote ORACLE connection

If you are unable to access the remote database after performing the steps outlined in the previous section, try the following:

1 Start with the basics. Make sure the client and server can see each other on the network.

2 Verify that ORACLE is running on both systems.

3 Verify that the username and the password are correct.

4 Make sure that the LISTENER process is running on the server.

5 Make sure that ORACLE_HOME is set properly in the listener.ora file. It should be set to the same value as ORACLE_HOME on the server.

6 Make sure that the nodename specified in tnsnames.ora matches the nodename specified in net_conf, and is the nodename of the workstation with the databases.

7 Bring down the LISTENER process and bring it back up again.

Meridian IVR

SQL Server User's Guide

Nortel
Customer Documentation
522 University Avenue, 14th Floor
Toronto, Ontario, Canada
M5G 1W7

Publication number: 555-9001-314
Product release: 2.0/I
Document release: Standard 1.0
Date: February 1996

Printed in the United States of America

NORTEL
NORTHERN TELECOM

Appendix A: SQL enhancements

For Meridian IVR/I 2.0/I, SQL functionality has been expanded in the following ways:

- Oracle database access through unique SQL keys
- Remote database access now includes all supported database types

Oracle database access through unique SQL keys

Five of the six SQL fields in the Default Cell parameter page now work together to specify a set of Generic Data Server (GDS) processes for your Meridian IVR 2.0/I applications to communicate with. See the Meridian IVR 2.0/I Application Development Guide for information on editing the Default Cell parameters.

Variables for database options

Refer to the following sections for information on setting environment variables for each of the database options.

Informix

Edit the **sys_files/informix.conf** file so that the environment variables DBPATH, INFORMIXDIR, and SQLEXEC are set to the values recommended by your Informix installation notes.

Ingres

Edit the **sys_files/ingres.conf** file so that the environment variable II_SYSTEM is set to the value recommended by your Ingres installation notes.

Oracle 7

Edit the **sys_files/oracle.conf** file so that the environment variables ORACLE_HOME and ORACLE_SID are set to the values recommended by your Oracle 7 installation notes.

Sybase

Edit the **sys_file/sybase.conf** file so that the environment variable DSQUERY, SYSBASE, and SYBPLATFORM are set to the values recommended by your Sybase installation notes.

Remote database access

Your Meridian IVR 2.0/I applications can now access all four types of RDBMS (Ingres, Informix, Sybase, and Oracle) remotely. This means your database servers no longer need to reside on your application processor for your applications to access their data. The following database software packages are required on the Meridian IVR application processor to provide this service

RDBMS	Supported Release	Description
Informix	5.0 (or newer)	This Ingres software package must be for Intel Pentium processor, SCO Open Desktop Lite R3.0 on 5-1/4 inch cartridge tapes or 3-1/2 inch diskettes. The users (VADs or end customers) must purchase this package directly from Ingres Inc.
Oracle	7.0 (or newer) or 7.1 (or newer)	This Oracle software package must be for Intel Pentium processor, SCO Open Desktop Lite R3.0 on 5-1/4 inch cartridge tapes or 3-1/2 inch diskettes. The users (VADs or end customers) must purchase this package directly from Oracle Corp.
Ingres	6.4 (or newer)	This Informix software package must be for Intel Pentium processor, SCO Open Desktop Lite R3.0 on 5-1/4 inch cartridge tapes or 3-1/2 inch diskettes. The users (VADs or end customers) must purchase this package directly from Informix Software Inc.
Sybase	10.0 (or newer)	This Sybase software package must be for Intel Pentium processor, SCO Open Desktop Lite R3.0 on 5-1/4 inch cartridge tapes or 3-1/2 inch diskettes. The users (VADs or end customers) must purchase this package directly from Sybase Inc.

Accessing data residing in an Oracle Database

There are two possible scenarios that a multiple-instance Oracle installation may need to address. Note that Oracle is an extremely rich RDBMS, and other variations are possible. Any combination of the following two scenarios will be directly addressed in this release:

- Multiple Instance/Databases using the same Oracle product tree
- Multiple Instance/Databases using different Oracle product trees

There are three basic steps that must occur prior to accessing data residing in an Oracle table:

- Create an Oracle Instance
- Mount the Database
- Open the Database

Creating an Oracle Instance involves allocation of an Oracle SGA (a shared memory region) and starting a set of Oracle background processes.

Mounting an Oracle database associates a physical database with a running instance. For our discussions there is a one-to-one correspondence between Instances and databases, both identified by the same UNIX environment variable ORACLE_SID.

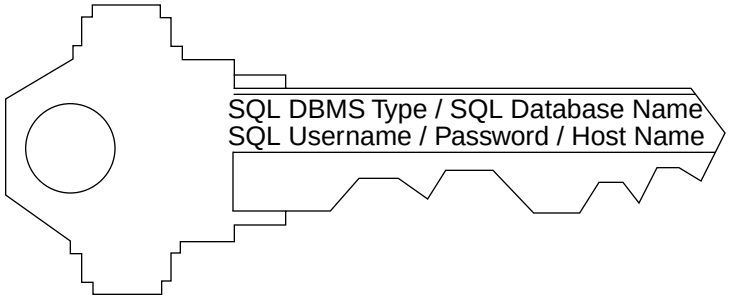
You must open the database before you can access it.

Oracle Instance/Database identification

The Instance is the set of processes and SGA, and the database is the physical manifestation of data. An Oracle Instance and an Oracle Database can be considered identical. Both of these are identified by the single UNIX environment variable ORACLE_SID. Prior to connecting to an Oracle Instance, you must set this variable to the correct value. Additionally, you must set the UNIX environment variable ORACLE_HOME to the correct product tree for a given Instance/Database. You can look up this value in the ORATAB file.

Generic data server key

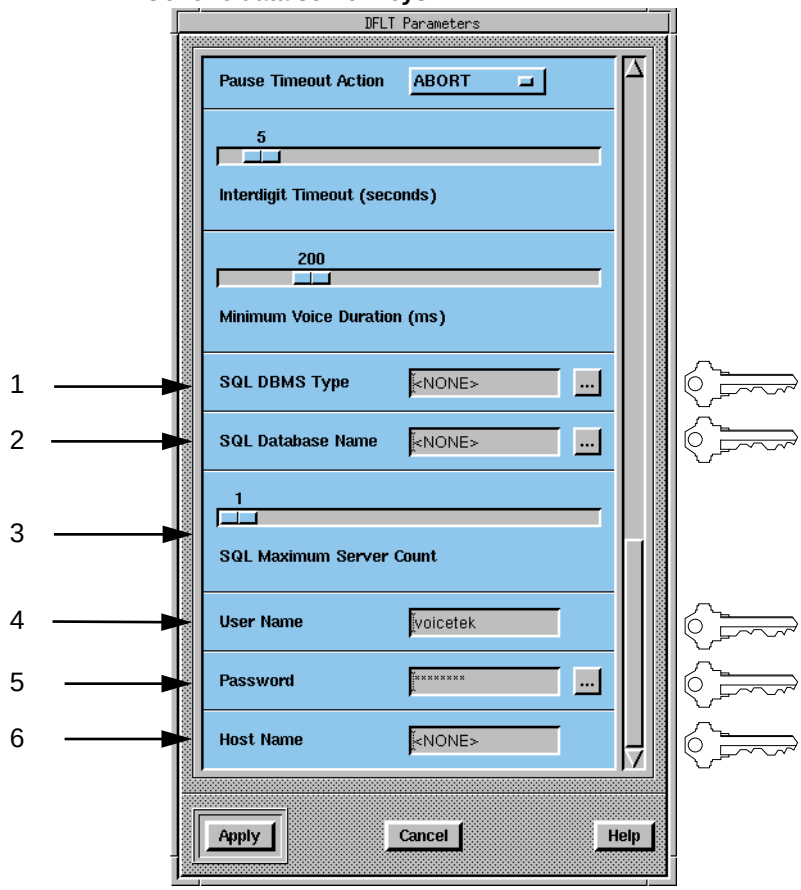
In this release of Meridian IVR 2.0/I, the SQL DBMS Type, SQL Database Name, SQL Username, Password, and Host Name work together to form a unique five-part key that specifies a set of Generic Data Server (GDS) processes for an application to communicate with.



This key lets a specific Meridian IVR 2.0/I application communicate with a specific database, and eliminates any ambiguity as to which database your applications will access.

You create this key through the Default Cell parameters. There are six parameters in the Default cell that pertain to SQL, five of which form the key. Look at Figure A-1— each parameter is clearly marked. Key symbols to the right of the illustration show which parameters are used to form the key.

Figure A-1
Generic data server keys



SQL DBMS type

This specifies the type of SQL database that the application will be using. The current choices are:

- INGRES
- ORACLE
- SYBASE
- INFORMIX

This field is the first part of the key that determines which set of Generic Data Server (GDS) processes an application should use for database access.

SQL database name

This specifies the name of the database that the application will communicate with. Prior to this release of Meridian IVR 2.0/I, this field was only functional for Ingres and Informix databases. It now functions for Ingres, Informix, and Oracle.

When the SQL DBMS Type is ORACLE, this field will specify the Oracle Database Name. Note that the Oracle database name is commonly specified using the UNIX environment variable ORACLE_SID. This value, specified by the user, as well as the ORATAB file will uniquely identify an Oracle database and its instance. For more complete details regarding the specifics of Oracle Databases and Instances, refer to the *Oracle7 Server Concepts Manual*.

This field is the second part of the key that determines which set of GDS processes an application should use for database access.

SQL maximum server count

This field specifies the number of GDS processes that applications with the same key will have. The first application you load containing an SQL key determines the number of GDSs for that key. All subsequent applications with that key use the number of GDSs determined by the first application loaded. This value can range from 1 to 99.

User name

This field specifies the username used for connection to an SQL database. Each type of database has a slightly different approach to this functionality.

This field is the third part of the key that determines which set of GDSs an application should use for database access.

Ingres and Informix

Ingres and Informix set database permissions based on the Operating System credentials of the user who opens the specified database. Therefore, prior to connection, these databases switch to the specified user, with password validation, using appropriate UNIX system calls.

Oracle and Sybase

Oracle and Sybase set database permissions as part of the SQL command used to connect to the database.

Password

This field is used for validation for all four RDBMSs.

This is the fourth part of the key.

Host name

This specifies the remote host identifier used to open/connect to SQL database servers not running locally (on the same machine as the GDSs). For Informix, the remote system//directory_path will be limited to 31 characters in length.

This field is the fifth and last part of the key.

Meridian IVR

SQL Server User's Guide

Nortel
Customer Documentation
522 University Avenue, 14th Floor
Toronto, Ontario, Canada
M5G 1W7

Publication number: 555-9001-314
Product release: 2.0/I
Document release: Standard 1.0
Date: February 1996

Printed in the United States of America

NORTEL
NORTHERN TELECOM