Informix Product Family
Informix
Version 12.10

IBM Informix DB-Access User's Guide

IBM

Informix Product Family
Informix
Version 12.10

IBM Informix DB-Access User's Guide

IBM

Contents

Introduction

This introduction provides an overview of the information in this publication and describes the conventions it uses.

About this publication

This publication describes how to use the DB-Access utility to access, modify, and retrieve information from IBM® Informix® database servers. Chapter 1, "Getting started with DB-Access," on page 1-1 explains how to create and work with the demonstration databases provided with your Informix database server.

Important: Use DB-Access with the current version of an Informix database server. If you use DB-Access with a database server from a different version, you might obtain inconsistent results, such as when you use a version that does not support long identifiers with a version that does.

Types of users

This publication is written for the following users:
- Database users
- Database administrators
- Database-application programmers

This publication assumes that you have the following background:
- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming

If you have limited experience with relational databases, SQL, or your operating system, see the following IBM Informix manuals:
- *IBM Informix GLS User's Guide*
- *IBM Informix Guide to SQL: Reference*
- *IBM Informix Guide to SQL: Syntax*
- *IBM Informix Guide to SQL: Tutorial*

Software dependencies

This publication assumes that you are using one of the IBM Informix, Version 12.10 database servers.

Assumptions about your locale

IBM Informix products can support many languages, cultures, and code sets. All the information related to character set, collation and representation of numeric data, currency, date, and time that is used by a language within a given territory and encoding is brought together in a single environment, called a Global Language Support (GLS) locale.

The IBM Informix OLE DB Provider follows the ISO string formats for date, time, and money, as defined by the Microsoft OLE DB standards. You can override that default by setting an Informix environment variable or registry entry, such as **DBDATE**.

If you use Simple Network Management Protocol (SNMP) in your Informix environment, note that the protocols (SNMPv1 and SNMPv2) recognize only English code sets. For more information, see the topic about GLS and SNMP in the *IBM Informix SNMP Subagent Guide*.

The examples in this publication are written with the assumption that you are using one of these locales: en_us.8859-1 (ISO 8859-1) on UNIX platforms or en_us.1252 (Microsoft 1252) in Windows environments. These locales support U.S. English format conventions for displaying and entering date, time, number, and currency values. They also support the ISO 8859-1 code set (on UNIX and Linux) or the Microsoft 1252 code set (on Windows), which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

You can specify another locale if you plan to use characters from other locales in your data or your SQL identifiers, or if you want to conform to other collation rules for character data.

For instructions about how to specify locales, additional syntax, and other considerations related to GLS locales, see the *IBM Informix GLS User's Guide*.

Example code conventions

Examples of SQL code occur throughout this publication. Except as noted, the code is not specific to any single IBM Informix application development tool.

If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```

To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement. If you are using DB–Access, you must delimit multiple statements with semicolons.

Tip: Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.

For detailed directions on using SQL statements for a particular application development tool or SQL API, see the documentation for your product.

Additional documentation

Documentation about this release of IBM Informix products is available in various formats.

You can access Informix technical information such as information centers, technotes, white papers, and IBM Redbooks® publications online at http://www.ibm.com/software/data/sw-library/.

Compliance with industry standards

IBM Informix products are compliant with various standards.

IBM Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of IBM Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL Common Applications Environment (CAE) standards.

The IBM Informix Geodetic DataBlade® Module supports a subset of the data types from the *Spatial Data Transfer Standard (SDTS)—Federal Information Processing Standard 173*, as referenced by the document *Content Standard for Geospatial Metadata*, Federal Geographic Data Committee, June 8, 1994 (FGDC Metadata Standard).
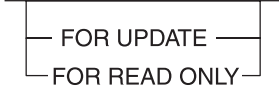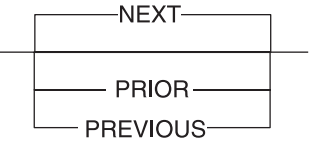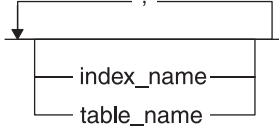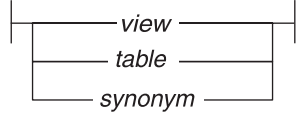
Syntax diagrams

Syntax diagrams use special components to describe the syntax for statements and commands.

Table 1. Syntax Diagram Components

Component represented in PDF	Component represented in HTML	Meaning
▶▶───────────	`>>----------------------`	Statement begins.
───────────▶	`---------------------->`	Statement continues on next line.
▶───────────	`>----------------------`	Statement continues from previous line.
──────────◀▶	`---------------------->< `	Statement ends.
────── SELECT ──────	`--------SELECT----------`	Required item.
┌─── LOCAL ───┐	`--+-----------------+---` `  '------LOCAL------'`	Optional item.

┌─── ALL ───┐
 ├─ DISTINCT ─┤
 └─ UNIQUE ───┘
	`---+-----ALL-------+---`
	 ` +--DISTINCT-----+ `
	 ` '---UNIQUE------'`
		Required item with choice. Only one item must be present.

Table 1. Syntax Diagram Components (continued)

Component represented in PDF	Component represented in HTML	Meaning

FOR UPDATE
FOR READ ONLY
	``---+------------------+---
	   +--FOR UPDATE-----+
	   '--FOR READ ONLY--'```
		Optional items with choice are shown below the main line, one of which you might specify.

NEXT
PRIOR
PREVIOUS
	``.---NEXT---------.
	----+---------------+---
	   +---PRIOR--------+
	   '---PREVIOUS-----'```
		The values below the main line are optional, one of which you might specify. If you do not specify an item, the value above the line is used by default.

index_name
table_name
	``.-------,-----------.
	 V |
	---+---------------+---
	   +---index_name---+
	   '---table_name---'```
		Optional items. Several items are allowed; a comma must precede each repetition.

Table Reference	```>>-| Table Reference |-><```	Reference to a syntax segment.

Table Reference
 	
view
table
synonym
	``Table Reference
	|--+-----view--------+--|
	   +------table------+
	   '----synonym------'```
		Syntax segment.

How to read a command-line syntax diagram

Command-line syntax diagrams use similar elements to those of other syntax diagrams.

Some of the elements are listed in the table in Syntax Diagrams.

Creating a no-conversion job

```
►►—onpladm create job—job——————————————— -n— -d—device— -D—database———————►
                         └─ -p—project—┘
```

```
►— -t—table———————————————————————————————————————————————————————————►
```

```
►—┬─────────────────────────────────────────────────┬—————————————————►◄
  │                                           (1)    │
  └─ -S—server─┘ └─ -T—target─┘ Setting the Run Mode ─┘
```
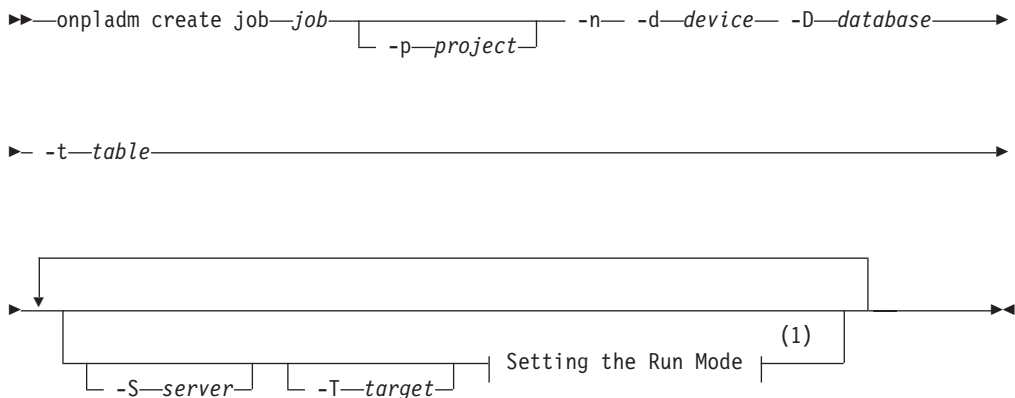
Notes:

1   See page Z-1

This diagram has a segment named "Setting the Run Mode," which according to the diagram footnote is on page Z-1. If this was an actual cross-reference, you

would find this segment on the first page of Appendix Z. Instead, this segment is shown in the following segment diagram. Notice that the diagram uses segment start and end components.

Setting the run mode:
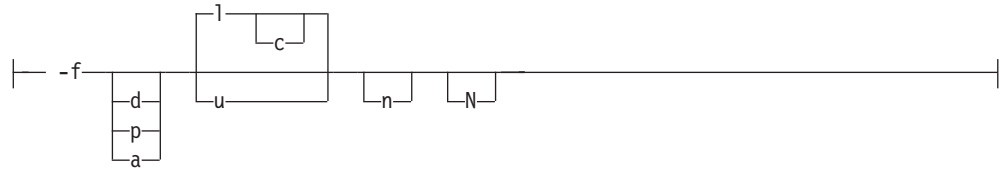
To see how to construct a command correctly, start at the upper left of the main diagram. Follow the diagram to the right, including the elements that you want. The elements in this diagram are case-sensitive because they illustrate utility syntax. Other types of syntax, such as SQL, are not case-sensitive.

The Creating a No-Conversion Job diagram illustrates the following steps:

1. Type **onpladm create job** and then the name of the job.
2. Optionally, type **-p** and then the name of the project.
3. Type the following required elements:
   - **-n**
   - **-d** and the name of the device
   - **-D** and the name of the database
   - **-t** and the name of the table
4. Optionally, you can choose one or more of the following elements and repeat them an arbitrary number of times:
   - **-S** and the server name
   - **-T** and the target server name
   - The run mode. To set the run mode, follow the Setting the Run Mode segment diagram to type **-f**, optionally type **d**, **p**, or **a**, and then optionally type **l** or **u**.
5. Follow the diagram to the terminator.

Keywords and punctuation

Keywords are words reserved for statements and all commands except system-level commands.

When a keyword appears in a syntax diagram, it is shown in uppercase letters. When you use a keyword in a command, you can write it in uppercase or lowercase letters, but you must spell the keyword exactly as it appears in the syntax diagram.

You must also use any punctuation in your statements and commands exactly as shown in the syntax diagrams.

Identifiers and names

Variables serve as placeholders for identifiers and names in the syntax diagrams and examples.

You can replace a variable with an arbitrary name, identifier, or literal, depending on the context. Variables are also used to represent complex syntax elements that are expanded in additional syntax diagrams. When a variable appears in a syntax diagram, an example, or text, it is shown in *lowercase italic*.

The following syntax diagram uses variables to illustrate the general form of a simple SELECT statement.

►►─SELECT─*column_name*─FROM─*table_name*─────────────────────────────────►◄

When you write a SELECT statement of this form, you replace the variables *column_name* and *table_name* with the name of a specific column and table.

How to provide documentation feedback

You are encouraged to send your comments about IBM Informix user documentation.

Use one of the following methods:
- Send email to docinf@us.ibm.com.
- In the Informix information center, which is available online at http://www.ibm.com/software/data/sw-library/, open the topic that you want to comment on. Click the feedback link at the bottom of the page, fill out the form, and submit your feedback.
- Add comments to topics directly in the information center and read comments that were added by other users. Share information about the product documentation, participate in discussions with other users, rate topics, and more!

Feedback from all methods is monitored by the team that maintains the user documentation. The feedback methods are reserved for reporting errors and omissions in the documentation. For immediate help with a technical problem, contact IBM Technical Support at http://www.ibm.com/planetwide/.

We appreciate your suggestions.

Chapter 1. Getting started with DB-Access

DB-Access provides a menu-driven interface for entering, running, and debugging Structured Query Language (SQL) statements and Stored Procedure Language (SPL) routines. You can also run DB-Access interactively from the command line.

You use SQL and SPL commands to perform data-definition tasks, such as specifying the number and type of data columns in a table, and data-management tasks, such as storing, viewing, and changing table data.

You can use DB-Access for the following aspects of database processing:

- Running ad hoc queries that you run infrequently
- Connecting to one or more databases, transferring data between the database and external text files, and displaying information about a database
- Displaying system catalog tables and the Information Schema of databases
- Practicing the SQL and SPL statements and examples that are provided in the *IBM Informix Guide to SQL: Tutorial* or the *IBM Informix Database Design and Implementation Guide*
- Testing applications that you intend to store for use in a production environment
- Creating demonstration databases

Important: DB-Access is not intended as an application-development environment. You cannot branch conditionally or loop through SQL statements when you run them within DB-Access.

The DB-Access utility is included with the Informix server and with the Informix Client Software Development Kit.

The IBM OpenAdmin Tool (OAT) for Informix is an alternative to the DB-Access utility.

Requirements for the Informix server DB-Access utility

Before you start DB-Access, prepare the Informix server environment.

Do the following tasks before you start the DB-Access utility that is included with the Informix server:

- Set environment variables
- If you require globalization, set up the Global Language Support (GLS) locale
- Start the database server

To secure DB-Access connections with IBM Informix, you can use the Secure Sockets Layer (SSL) protocol.

Related concepts:

↪ Secure sockets layer protocol (Security Guide)

Environment variables

As part of the installation and setup process, the system or database administrator sets certain environment variables that enable IBM Informix products to work within a particular operating-system environment.

You must have $INFORMIXDIR/bin in your path if you use DB-Access on a UNIX operating system or %INFORMIXDIR%\bin in your path if you use DB-Access on a Windows operating system. Your operating system uses the path to locate the initialization script and the **dbaccess** executable file.

In a UNIX environment, the database server must have the appropriate terminal that is set up from among the terminals that are listed by the **INFORMIXTERM** environment variable.

DB-Access uses the terminal definitions in the terminfo directory unless the **INFORMIXTERM** environment variable is set to the termcap file. If DB-Access fails to initialize the menus that are based on the **INFORMIXTERM** setting, DB-Access tries to use the other setting. For example, if DB-Access fails to initialize the menus using the terminfo directory, DB-Access starts the menus using the termcap file.

You can set the following optional environment variables:

DBACCNOIGN
> Rolls back an incomplete transaction if you run the LOAD command in menu mode.

DBCENTURY
> Sets the appropriate expansion for DATE and DATETIME values that have only a two-digit year, such as 04/15/12.

DBDATE
> Specifies the user formats of DATE values.

DBEDIT
> Sets the default DB-Access text editor without changing the default text editor that is associated with the operating-system shell.
>
> For more information about how DB-Access uses the text editor that you specify as default, see "A system editor" on page 2-2.

DBFLTMASK
> Sets the default floating-point values of data types FLOAT, SMALLFLOAT, and DECIMAL within a 14-character buffer.
>
> The effect of this variable is limited to the DB-Access display size for numbers.

DELIMIDENT
> Causes the database server to interpret double quoted (") text as identifiers rather than strings.

IFX_LONGID
> Determines whether a client application can handle long identifiers.
>
> If you use the **IFX_LONGID** environment variable to support SQL identifiers with up to 128 bytes, some error, warning, or other messages of DB-Access might

truncate database object names that include more than 18 bytes in their identifiers. You can avoid this truncation by not declaring names that have more than 18 bytes.

Related concepts:

➡ Environment variables (SQL Reference)

Requirements for the Informix Client Software Development Kit DB-Access utility

Before you start DB-Access on a client, set up the client environment.

The DB-Access utility on a client can directly access Informix databases with which Client SDK has a client/server connection.

Do the following tasks before you start DB-Access utility that is included with Client SDK:

*   Set the `sqlhosts` information.
*   Set the INFORMIXDIR environment variable to the Client SDK installation directory.
*   Set the INFORMIXSERVER environment variable for a default server name.

Related concepts:

➡ The sqlhosts file and the SQLHOSTS registry key (Administrator's Guide)

Related reference:

➡ INFORMIXDIR environment variable (SQL Reference)

➡ INFORMIXSERVER environment variable (SQL Reference)

Demonstration databases

You can practice using DB-Access with a demonstration database.

If you use an IBM Informix demonstration database, you can add, delete, or change the provided data and scripts. You can restore the database to its original condition.

You can configure the following demonstration databases:

*   The **stores_demo** database illustrates a relational schema with tables about a fictitious wholesale sporting-goods distributor, as well as other tables that are used in examples. Tables containing electricity usage and geographical location data illustrate time series and spatial information. Many examples in IBM Informix manuals are based on the **stores_demo** database.
*   The **superstores_demo** database illustrates an object-relational schema. The **superstores_demo** database contains examples of extended data types, type and table inheritance, and user-defined routines.

The scripts that you use to install the demonstration databases are in the `$INFORMIXDIR/bin` directory on UNIX and in the `%INFORMIXDIR%\bin` directory on Windows.

Some operating systems require that you have execute permissions to run SQL command files, read permissions to open these files or their contents in DB-Access,

or write permissions to save modified or new files. Use the UNIX **chmod** command to enable execution of the SQL files that the initialization script installed.

The demonstration scripts are designed for the default locale. If you use a non-default locale, such as **en_us.utf8**, some features, such as the SET COLLATION statement, might not function correctly.

Related concepts:

⇨ The stores_demo Database (SQL Reference)

⇨ The superstores_demo database (SQL Reference)

Creating a demonstration database

You create demonstration databases by running the **dbaccessdemo** command.

When you create a demonstration database, the script confirms that you want to copy sample SQL command files. Command files that the demonstration database includes have a `.sql` extension and contain sample SQL statements that you can use.

To create a demonstration database:

1. Create a directory.

   You must have UNIX read and execute permissions for each directory in the path name that you create.

2. Change directories to the new directory and run the **dbaccessdemo** or **dbaccessdemo_ud** command.

3. The initialization script displays a series of messages on the screen as the database is created. Press **Y** to copy the command files into the directory that you created. The demonstration database is created. You are the owner and database administrator (DBA) of that database.

If you want to discard changes that you made to your database or to the command files, rerun the **dbaccessdemo** or **dbaccessdemo_ud** command and press **Y** to replace the existing command files with the original versions.

dbaccessdemo command: Create demonstration databases

Use the **dbaccessdemo** or **dbaccessdemo_ud** command to create the demonstration databases.

Syntax for stores_demo

```
►►──dbaccessdemo──────────────────────────────────────────────►
              └─-log─┘  └─dbname─┘  └─-dbspace──dbspace_name─┘

►──────────────────────────────────────────────────────────►◄
   └─-nots─┘
```
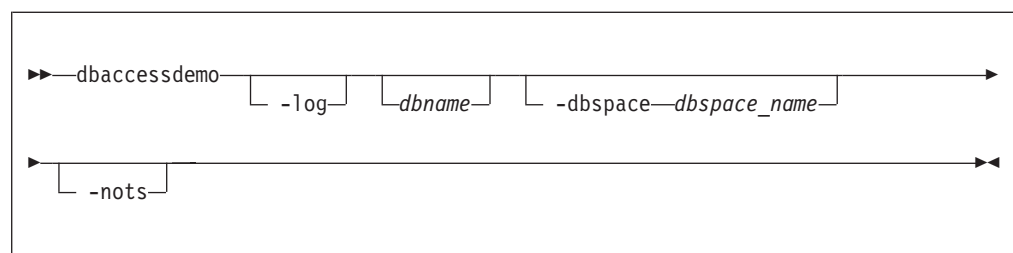
Syntax for superstores_demo

```
>>--dbaccessdemo_ud--------------------------------------------------->◄
                       └--log-┘  └--dbname-┘  └--dbspace--dbspace_name-┘
```
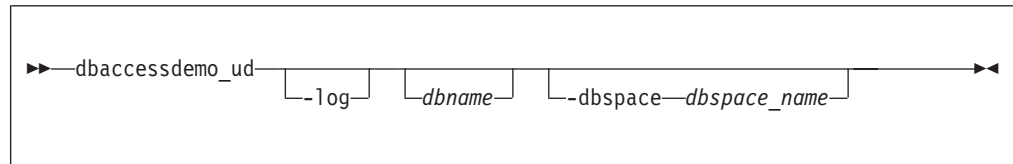
-log   Requests transaction logging for the demonstration database.

dbname
>   Substitutes for the default database name. Must follow Identifier naming guidelines.

-dbspace
>   Requests a particular dbspace location for the demonstration database.

dbspace_name
>   Houses the demonstration database. If you do not specify a dbspace name, by default, the data for the database is put in the root dbspace. To create a dbspace, use the **onspaces** utility.

-nots   Prevents the creation of the time series tables in the **stores_demo** database.

Examples

The following command creates a database that is named **stores_demo**:

```
dbaccessdemo
```

The following example creates an instance of the **stores_demo** database named **demo_db**:

```
dbaccessdemo demo_db
```

The following command initializes the **stores_demo** database and also initiates log transactions:

```
dbaccessdemo -log
```

The following command creates an instance of the **stores_demo** database named **demo_db** in **dbspace_2**:

```
dbaccessdemo demo_db -dbspace dbspace_2
```

The following command creates a database that is named **superstores_demo**:

```
dbaccessdemo_ud
```

Related concepts:

⇨ The stores_demo Database (SQL Reference)

⇨ The superstores_demo database (SQL Reference)

Related reference:

⇨ Identifier (SQL Syntax)

Start DB-Access

You start DB-Access by running the **dbaccess** command from the command line. You can choose whether to use the DB-Access menu interface or the command-line interface.

You can start and use DB-Access in the following ways:

- Start DB-Access at the main menu.
- Start DB-Access from a specific menu or screen.
- Run a file that contains SQL statements without showing the DB-Access menus.
- Start and run DB-Access interactively at the command line, without the menu interface.

On Windows, you can set up the DB-Access program icon to run any of the **dbaccess** commands.

If the `TERM`, `TERMCAP`, or `TERMINFO` environment variables on UNIX do not enable DB-Access to recognize the type of terminal you use, the main menu does not show. Instead, a message similar to the following text is displayed:

```
Unknown terminal type.
```

If you use a window interface on a UNIX terminal, issue the **dbaccess** command from a nonscrolling console window.

If you use a Windows terminal to run DB-Access on a UNIX database server, the terminal-emulation window must emulate a terminal type that DB-Access can recognize, or the database server shows an unknown terminal-type message in the terminal-emulation window.

Tip: If your operating system cannot find **dbaccess**, include the full path before the program name, as follows:

```
$INFORMIXDIR/bin/dbaccess
```

Related concepts:

⇨ Environment variables (SQL Reference)

dbaccess command: Start DB-Access

Use the **dbaccess** command to start DB-Access. Include options to specify the database, command files, or to go to a specific menu screen.

Syntax

```
>>--dbaccess------------------------------------------------------------------------------------------------------------------------------>
              └─-ansi─┘   └─-X─┘

>--database------------------------------------------------------------------------------------------------------------------>
          └─@─server─┘   ├─-q─┬─────────────────────────────────┬───┬─filename─┬──┤   └─-a─┘
          │              │    └─| QUERY-LANGUAGE menu options |──┘(1)└─table────┘  │
          │              ├─-t─────────────────────────────────────┬────────┬─┤
          │              │      └─| TABLE menu options |────┘(2)  └─table─┘
          │              ├─-d─────────────────────────────────────┤
          │              │    └─| DATABASE menu options |──┘(3)
          │              ├─-c─────────────────────────────────────┤
          │              │    └─| CONNECTION menu options |──┘(4)
          │              └─-s─────────────────────────────────────┤
          └─┬─-e─┬──┬─database─┬──filename─────────────────────┘
            └─-m─┘  └────-─────┘

                   (5)
>──────────────────────────────────────────────────────────────────────────><
    ├─--version─┤
    └─-V────────┘
```
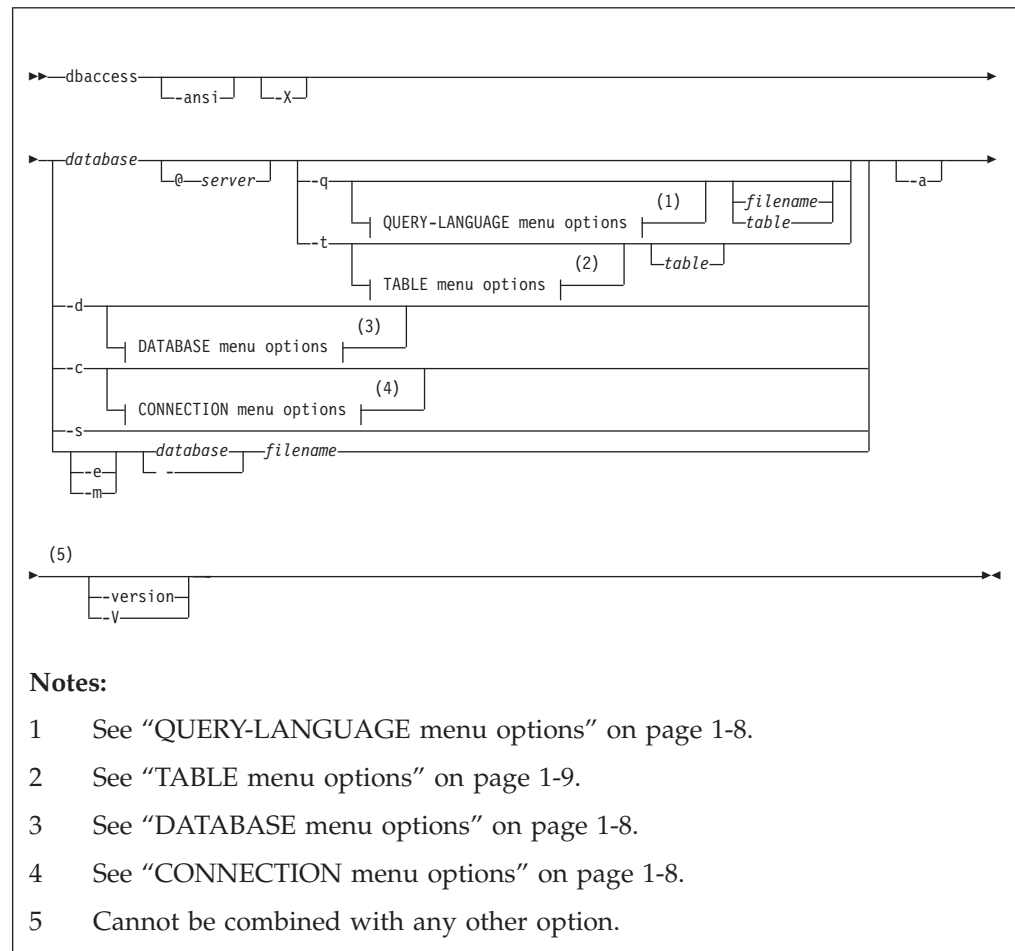
Notes:

1 See "QUERY-LANGUAGE menu options" on page 1-8.

2 See "TABLE menu options" on page 1-9.

3 See "DATABASE menu options" on page 1-8.

4 See "CONNECTION menu options" on page 1-8.

5 Cannot be combined with any other option.

The **dbaccess** command without options starts the main menu with no database selected and no options activated. You select submenus from the main menu.

-ansi Causes DB-Access to generate a warning whenever it encounters an IBM Informix extension to ANSI-compliant syntax. For more information, see "Example: Check for ANSI compliance" on page 1-11.

-a Stops a process directly after the first error is encountered. Stopping a process from continuing after the first error can ensure greater database consistency.

-c Starts with the CONNECTION menu as the top-level menu.

-d Starts with the DATABASE menu as the top-level menu.

-e Echoes each line from a command file designated by *filename*.

-m Displays all error messages generated by multiple levels of the server that pertain to an SQL statement in command files.

-q Starts at the query-language menu (SQL-menu) as the top-level menu.

-s Connects you to the main DB-Access menu and displays information about the current session.

   This information includes database server name, database server type, the host computer, server capabilities, and other settings.

-t   Starts at the TABLE menu as the top-level menu.

-V   Displays the version number and serial number for DB-Access without launching the application. You cannot use any other options with **-V**.

-version
             Displays the version number and build information for DB-Access, including the GLS library version, without launching the application. You cannot use any other options with **-version**.

-X   Activates the hexadecimal format for LOAD and UNLOAD statements.

database
             Name of the database that you want DB-Access to connect to at the startup of your current session. A hyphen (-) indicates that the database is specified in a DATABASE statement in a command file.

filename
             Names a command file to load with the SQL menu.

server   Name of the database server.

table   Specifies a table in the database.

If you exit from a submenu or option that you specified from the command line, you will exit directly to the operating-system command line.

CONNECTION menu options

The CONNECTION menu options for the **dbaccess** command represent short cut keys for the CONNECTION menu.

-cc   Chooses the Connect option on the CONNECTION menu.

-cd   Chooses the Disconnect option on the CONNECTION menu.

DATABASE menu options

The DATABASE menu options for the **dbaccess** command represent short cut keys for the DATABASE menu.

-dc   Chooses the Create option on the DATABASE menu.

      -dcl   Takes you to the LOG option on the CREATE DATABASE menu

-dd   Chooses the Drop option on the DATABASE menu.

-di   Chooses the Info option on the DATABASE menu. With this option, you can add another letter as follows to go to the next menu level and view:

      -dib   The dbspaces information for the current database

      -din   The NLS information for the current database

      -dip   Stored procedures in the current database

      If you do not include a database name before any **-di** option, you must choose a current database from the SELECT DATABASE screen.

-dl   Chooses the CLose option on the DATABASE menu.

-ds   Chooses the Select option on the DATABASE menu.

QUERY-LANGUAGE menu options

The QUERY-LANGUAGE menu options for the **dbaccess** command represent short cut keys for the QUERY-LANGUAGE menu.

-qc   Chooses the Choose option on the SQL menu.

-qd   Chooses the Drop option on the SQL menu.

-qi   Chooses the Info option on the SQL menu. With this option, you can add another letter as shown in the following list (and specify a table) to go to the next menu level and view:

> **-qic**   Columns in the table
>
> **-qif**   Information about fragmentation strategy for the table
>
> **-qig**   Information about triggers in the table
>
> **-qii**   Indexes on the table
>
> **-qio**   Table constraints
>
> **-qip**   Access privileges on the table
>
> **-qir**   Table-level references privilege on the table
>
> **-qis**   Table status information
>
> If you do not include a table name with the **-qi** option, you must choose one from the INFO FOR TABLE screen.

-qm   Chooses the Modify option on the SQL menu.

-qn   Chooses the New option on the SQL menu.

-qs   Chooses the Save option on the SQL menu.

-qu   Chooses the Use-editor option on the SQL menu.

> If you do not include a database name before a **-q** option, you must choose a current database from the SELECT DATABASE screen.

When you select the Modify option on the QUERY-LANGUAGE menu, you must first select a command file to modify from the CHOOSE menu. The MODIFY screen is then displayed and shows the text.

Restriction: You cannot go directly to the Run or Output option on the SQL menu. Trying to do so results in an error message.

TABLE menu options

The TABLE menu options for the **dbaccess** command represent short cut keys for the TABLE menu.

-ta   Chooses the Alter option on the TABLE menu.

-tc   Chooses the Create option on the TABLE menu.

-td   Chooses the Drop option on the TABLE menu.

-ti   Chooses the Info option on the TABLE menu. With this option, you can add another letter as shown in the following list (and specify a table) to go to the next menu level and view:

> **-tic**   Columns in the table
>
> **-tif**   Information about fragmentation strategy for the table
>
> **-tig**   Information about triggers in the table
>
> **-tii**   Indexes on the table
>
> **-tio**   Table constraints
>
> **-tip**   Access privileges on the table

-tir	Table-level references privilege on the table
-tis	Table status information

If you do not include a table name with the **-ti** option, you must choose one from the INFO FOR TABLE screen.

If you do not include a database name before a **-t** option, you must choose a current database from the SELECT DATABASE screen.

Example: Start DB-Access for a database

This example shows how to start DB-Access and specify a database to which to connect.

Assume that the database server that you have online contains a database named **mystores**. To make the **mystores** database the current database, start DB-Access with the following command:

```
dbaccess mystores
```

You can specify a database on a database server that is not online. For example, either of the following commands selects the **newstores** database on the **xyz** database server:

```
dbaccess newstores@xyz
dbaccess //xyz/newstores
```

When DB-Access starts, the database and database server name that you specify are displayed on the dashed line, as the following figure shows.

```
DB-Access:   Query-language  Connection  Database  Table  Session  Exit


---------------- newstores@xyz --------------------Press CTRL-W for Help ---
```

Figure 1-1. The DB-Access main menu with database and database server name

Example: Run a command file

This example shows how to start DB-Access and run a command file that contains SQL statements.

The following sample command runs the SQL statements in a file named `sel_stock.sql` on the **mystores** database:

```
dbaccess mystores sel_stock
```

The following sample command runs the SQL statements in the `sel_all.sql` file on the database that file specifies:

```
dbaccess - sel_all.sql
```

Some operating systems require that you have execute permissions to run SQL command files, read permissions to open these files or their contents in DB-Access, or write permissions to save modified or new files.

Use the UNIX **chmod** command to enable execution of the SQL files that the initialization script installed.

Example: View the Information Schema

This example shows how to start DB-Access and view the Information Schema for the specified database.

The xpg4_is.sql file in the $INFORMIXDIR/etc directory creates the Information Schema and installs the views for a specified database. The following command creates the Information Schema for database **mystores**:

```
dbaccess mystores $INFORMIXDIR/etc/xpg4_is.sql
```

The Information Schema adds to the database four information-only views that conform to X/Open XPG4 with IBM Informix extensions. After you run xpg4_is.sql, use DB-Access to retrieve information about the tables and columns that you have access to in the specified database.

Tip: Do not install XPG4-compliant views on an ANSI database, because the format of XPG4-compliant views differs considerably from the format of the ANSI-compliant Information Schema views that are defined by the SQL standards committee.

Related concepts:

➡ Information Schema (SQL Reference)

Example: Check for ANSI compliance

This example shows how to start DB-Access and check whether a database is ANSI-compliant.

To check your SQL statements for compliance with ANSI standards, include the **-ansi** option or set the **DBANSIWARN** environment variable. Use the **-ansi** option with other **dbaccess** options such as **-dc** (to create a database), **-tc** or **-ta** (to create or alter a table), or **-qc** *filename* (to choose a command file). The following command checks for ANSI compliance while DB-Access creates the database **research**:

```
dbaccess -ansi -dc research
```

You do not need to specify the **-ansi** option on the command line if the **DBANSIWARN** environment variable is set.

DB-Access displays the SQLSTATE value with the warning under the following circumstances:

- You include the **-ansi** option or set the **DBANSIWARN** environment variable.
- You access or create an ANSI database.
- You run DB-Access in command-line mode or specify a .sql input file.
- Running an SQL statement generates a warning rather than an error.

Related concepts:

➡ ANSI-Compliant Databases (SQL Syntax)

Related reference:

➡ DECIMAL(p) Floating Point (SQL Reference)

➡ GET DIAGNOSTICS statement (SQL Syntax)

➡ DBANSIWARN environment variable (SQL Reference)

Example: Show nonprintable characters in hexadecimal

This example starts DB-Access and activates the hexadecimal load and unload format (XLUF) so that the LOAD and UNLOAD SQL statements can format nonprintable ASCII signs in hexadecimal format.

The following command activates the XLUF format for the **mystores** database:

```
dbaccess -X mystores
```

A .unl file that the UNLOAD statement produces contains the hexadecimal format changes.

Related concepts:

⬐ The LOAD and UNLOAD statements (Migration Guide)

Run DB-Access in interactive mode without menus

If you do not want to use the menus and do not have a prepared SQL file, use your keyboard or standard input device to enter SQL statements from the command line.

When you start DB-Access without a menu argument and with a hyphen as the final argument, DB-Access processes commands from the standard input device (on UNIX) or the keyboard (on Windows). DB-Access reads what you type until you indicate that the input is completed. Then DB-Access processes your input and writes the results to the standard output device (on UNIX), or the command window (on Windows).

DB-Access reads and runs SQL statements from the terminal keyboard interactively. While DB-Access runs interactively, the greater than (>) prompt marks the line where you type your next SQL statement.

When you type a semicolon (;) to end a single SQL statement, DB-Access processes that statement. When you press **CTRL-D** to end the interactive session, DB-Access stops running. The following example shows user input and results in an interactive session:

```
dbaccess - -
>database stores_demo;

Database selected.

>select count(*) from systables;

(count(*))

      21

1 row(s) retrieved.

>^D

dbaccess - -
>database stores_demo;

Database selected.

>select count(*) from systables;

(count(*))

      21

1 row(s) retrieved.

>^D
```

Batch command input (UNIX)

You can use an inline shell script to supply one or more SQL statements. For
example, you can use the UNIX C, Bourne, or Korn shell with inline standard
input files:

```
dbaccess mystores- <<EOT!
select avg(customer_num) from customer
where fname matches '[A-G]*';
EOT!
```

You can use a pipe to supply SQL statements, as in this UNIX example:

```
echo 'select count(*) from systables' | dbaccess mystores
```

DB-Access interprets any line that begins with an exclamation mark (!) as a shell
command. You can mix shell escape lines with SQL statements and put them in
SQL statements, as follows:

```
dbaccess mystores -
>select
!echo hello
>hello
count(*) from systables;
>
(count(*))

     21

1 row(s) retrieved.
>
```

Connect to a database environment in interactive mode

You can use the CONNECT . . . USER syntax in SQL statements that you issue in
interactive mode. However, DB-Access does not support the USER clause of the
CONNECT statement when you connect to a default database server.

When you include the USER '*user identifier*' clause in a CONNECT statement in
interactive mode, DB-Access prompts you to enter a password.

The following two command examples show how to connect to a database server
in interactive mode. The first example uses the CONNECT statement without
specifying a user identifier.

```
dbaccess -nohistory- -

> connect to '@starfish';

Connected.
```

If you include the USER clause in a CONNECT statement, as the second example
shows, DB-Access uses echo suppression to prompt you for a password:

```
> connect to '@starfish' user 'marae';

ENTER PASSWORD:

Connected.
```

Restriction: For security reasons, do not enter the password on the screen where it
can be seen. Also, do not include the USING *password* clause in a CONNECT
statement when you use DB-Access interactively. If you are in interactive mode
and attempt to enter a password before the prompt, an error message is displayed.

You can run the USER clause of a CONNECT statement in a DB-Access file that includes the USER clause. The following example uses a command file that contains a CONNECT statement with a USING clause to connect to a database server:

```
dbaccess - connfile.sql
```

Important: An SQL command file that contains the following statement is protected from access by anyone other than the **user_id** that the USER clause identifies:

```
CONNECT TO '@dbserver' USER 'user_id' USING password
```

For UNIX, the following example uses a shell file to connect to a database server. DB-Access prompts you for a password.

```
dbaccess - - <<\!
connect to '@starfish' user 'marae';
!

ENTER PASSWORD:
```

Here the delimiting quotation marks preserve letter case in the database server name and in the authorization identifier of the user.

Chapter 2. The full-screen menu interface

The DB-Access full-screen menu interface guides you through running SQL statements.

The DB-Access user interface combines the following features:
- A hierarchy of menus
- Screens that prompt you for brief responses and choices from selection lists
- Contextual HELP screens
- The interactive Schema Editor that helps you structure tables
- An SQL programmer environment, which includes the following features:
  - The built-in SQL editor where you enter and modify SQL and SPL statements
  - An option to use another editor of your choice
  - The database server syntax checker and runtime debugger
  - Storage, retrieval, and execution of SQL and SPL routines
- A choice of output for database queries and reports

The Query-language option

Use the Query-language option to enter, modify, save, retrieve, and run SQL statements. DB-Access retains the statements, if any, in the editor. These statements are called the *current* statements.

Use the Query-language option to:
- Learn SQL and SPL.

  For example, use the Query-language option to practice the examples in the *IBM Informix Guide to SQL: Tutorial*.
- Create and alter table structures as an alternative to the DB-Access Schema Editor.
- Select, display, add, update, and delete data.

The SQL menu has the following options.

Option	Purpose
New	Clears current statements and positions cursor in SQL editor.
Run	Runs current SQL statements. A message is displayed or the data that is retrieved by a query is displayed with the number of rows retrieved.
Modify	Allows you to modify current SQL statements in SQL editor.
Use-editor	Starts a system editor so that you can modify current statements or create new statements. Use-editor is interchangeable with New and Modify.
Output	Redirects Run-option output to a file, printer, or system pipe.
Choose	Lists SQL command files so that you can choose a file to run or modify.
Save	Saves current SQL statements in a file for later use.

Option	Purpose
Info	Shows table information, such as columns, indexes, privileges, constraints, triggers, status, and fragmentation strategy.
Drop	Deletes a specified SQL command file.
Exit	Returns to main menu.

SQL editor

When you choose the New of Modify option, you see the SQL editor. You can type as many lines of text as you need. You are not limited by the size of the screen, although you might be limited by the memory constraints of your system. If you do not use the Save option to save your typed statements, they are deleted when you select an option that clears the SQL editor (such as New or Choose).

The SQL editor does not display more than 80 characters on a line and does not wrap lines.
- If you choose an existing command file in which the characters in a line extend beyond the 80th column, DB-Access displays a percent sign (%) in the 80th column to indicate an overflow. You cannot see all the characters beyond the percent sign, but the statement runs correctly.
- If you type characters in a new command file so that a line extends beyond the 80th column, DB-Access overwrites all the characters in the 80th column. You cannot see the overflow, and the statement does *not* run correctly.

To make the full text show on the screen, press **Enter** at a logical place in the first 80 characters of each line.

If you must type a quoted character string that exceeds 80 characters, such as an insert into a long CHAR column, use a system editor instead of the SQL editor.

If you want to include comments in the text:
- Use double minus signs for ANSI-compliant databases.
- Preface each comment line with a double minus sign (--) comment indicator. The comment indicator spans the entire line.
- Use braces ({ }) for databases that are not ANSI-compliant. Enclose the entire comment indicator between the braces.

A system editor

When you want to enter or modify a long SQL statement or series of statements, you might prefer the flexibility and familiarity of a system editor to the SQL editor. Select the Use-editor option from the SQL menu to use the system editor.

If you have not set the **DBEDIT** environment variable, you must select a text editor to use for the session. If you select Use-editor, DB-Access prompts you to accept or override the default system editor once each session.

The default editor that DB-Access displays depends on the preference that you establish for your operating system:
- Common UNIX system editors are **vi** and **ex**.
- If you use a text editor as the system default, you must save the `.sql` files as text.

Press **RETURN** to select the default editor you named after the USE-EDITOR prompt. To use a different editor, type the name of that editor and press **RETURN**.

Statements that the Run option supports

After you exit the editor screen, the SQL menu reopens with the Run option highlighted and the statement text is displayed in the bottom of the screen. You can run most SQL statements with the Run option.

To run statements that are not listed, use the SQL menu options New (or Use-editor) and Save to enter and save them, and then run the saved file from the command line.

The following is a list of SQL statements that you can run with the Run option.
- ALLOCATE COLLECTION
- ALLOCATE DESCRIPTOR
- ALLOCATE ROW
- ALTER ACCESS_METHOD
- ALTER FRAGMENT
- ALTER FUNCTION
- ALTER INDEX
- ALTER PROCEDURE
- ALTER ROUTINE
- ALTER SECURITY LABEL COMPONENT
- ALTER SEQUENCE
- ALTER TABLE
- BEGIN WORK
- CLOSE
- CLOSE DATABASE
- COMMIT WORK
- CONNECT
- CREATE ACCESS_METHOD
- CREATE AGGREGATE
- CREATE CAST
- CREATE DATABASE
- CREATE DISTINCT TYPE
- CREATE EXTERNAL TABLE
- CREATE FUNCTION
- CREATE FUNCTION FROM
- CREATE INDEX
- CREATE OPAQUE TYPE
- CREATE OPCLASS
- CREATE PROCEDURE
- CREATE ROLE
- CREATE ROUTINE FROM
- CREATE ROW TYPE
- CREATE SCHEMA
- CREATE SECURITY LABEL COMPONENT

- CREATE SECURITY LABEL
- CREATE SECURITY POLICY
- CREATE SEQUENCE
- CREATE SYNONYM
- CREATE TABLE
- CREATE TRIGGER
- CREATE VIEW
- CREATE XADATASOURCE
- CREATE XADATASOURCE TYPE
- DATABASE
- DEALLOCATE COLLECTION
- DEALLOCATE DESCRIPTOR
- DEALLOCATE ROW
- DECLARE
- DELETE
- DESCRIBE
- DESCRIBE INPUT
- DISCONNECT
- DROP ACCESS METHOD
- DROP AGGREGATE
- DROP CAST
- DROP DATABASE
- DROP FUNCTION
- DROP INDEX
- DROP OPAQUE TYPE
- DROP OPCLASS
- DROP PROCEDURE
- DROP ROLE
- DROP ROW TYPE
- DROP SECURITY LABEL COMPONENT/POLICY/LABEL
- DROP SEQUENCE
- DROP SYNONYM
- DROP TABLE
- DROP TRIGGER
- DROP TYPE
- DROP VIEW
- DROP XADATASOURCE
- DROP XADATASOURCE TYPE
- EXECUTE
- EXECUTE FUNCTION
- EXECUTE IMMEDIATE
- EXECUTE PROCEDURE
- FETCH
- FLUSH
- FREE

- GET DESCRIPTOR
- GET DIAGNOSTICS
- GRANT
- GRANT DBSECADM
- GRANT DEFAULT ROLE
- GRANT EXEMPTION
- GRANT FRAGMENT
- GRANT SECURITY LABEL
- INFO
- INSERT
- LOAD
- LOCK TABLE
- MERGE
- OPEN
- OUTPUT
- PREPARE
- PUT
- RENAME COLUMN
- RENAME DATABASE
- RENAME INDEX
- RENAME SEQUENCE
- RENAME TABLE
- REVOKE
- REVOKE DBSECADM
- REVOKE DEFAULT ROLE
- REVOKE EXEMPTION
- REVOKE FRAGMENT
- REVOKE SECURITY LABEL
- ROLLBACK WORK
- SAVE EXTERNAL DIRECTIVES
- SELECT
- SET AUTOFREE
- SET COLLATION
- SET CONNECTION
- SET CONSTRAINTS
- SET DATASKIP
- SET DEBUG FILE TO
- SET DEFERRED PREPARE
- SET DESCRIPTOR
- SET ENCRYPTION PASSWORD
- SET ENVIRONMENT
- SET EXPLAIN
- SET ISOLATION
- SET LOCK MODE
- SET LOG

- SET OPTIMIZATION
- SET PDQPRIORITY
- SET ROLE
- SET SESSION AUTHORIZATION
- SET STATEMENT CACHE
- SET TRANSACTION
- START VIOLATIONS TABLE
- STOP VIOLATIONS TABLE
- TRUNCATE
- UNLOAD
- UNLOCK TABLE
- UPDATE
- UPDATE STATISTICS
- WHENEVER

Redirect query results

The output from a SELECT statement is normally displayed on the screen. You can use the Output option on the SQL menu to route query results to the printer, store them in a system file, or pipe them to a program. The Output option has the same result as the OUTPUT statement of SQL.

The SELECT statement must be on the screen as the current statement. Select the Output option from the SQL menu, which displays the OUTPUT menu.

You have the following output options:
- Send your query results directly to a printer. DB-Access sends the results to your default printer and displays a message on the bottom of the screen that indicates how many rows were retrieved. The query results do not show on the screen. You can set the **DBPRINT** environment variable to specify a default printer.
- Write query results to a new file or append the results to an existing file. If you do not specify a path when DB-Access prompts you for a file name, the file is stored in the directory that you were in when you started DB-Access.
- Send query results to a pipe. Specify a target program, such as **more**, through which to pipe output. DB-Access sends the results to that pipe.

  On UNIX systems, you must have permission to run the target program.

  On Windows systems, the **cat** utility can serve as a target program through which to pipe output.

Choose an existing SQL statement

When you save SQL statements in a command file, you can retrieve the command file and run or edit the SQL statements at any time.

Select the Choose option on the SQL menu to display the CHOOSE screen with a list of the command files that you can access. These files have the extension .sql, although the extension is not shown. For example, the following figure lists the command files that are included in the demonstration database.

```
CHOOSE >>
Choose a command file with the Arrow Keys, or enter a name, then press Return.

---------------- mystores@dbserver1 ----------- Press CTRL-W for Help -------

   alt_cat            c_state           d_trig            sel_ojoin1

 c_calls            c_stock           d_view            sel_ojoin2

 c_cat              c_stores          del_stock         sel_ojoin3

 c_custom           c_table           ins_table         sel_ojoin4

 c_index            c_trig            opt_disk          sel_order

 c_items            c_type            sel_agg           sel_sub

 c_manuf            c_view1           sel_all           sel_union

 c_orders           c_view2           sel_group         upd_table

 c_proc             d_proc            sel_join
```

Figure 2-1. The CHOOSE screen listing current `.sql` *files*

If no current database exists, the list includes all the command files in the current directory and in any directories that the **DBPATH** environment variable specifies.

Important: This list includes only those file names that have the `.sql` extension. If you create an SQL file outside of DB-Access and save it without the `.sql` extension, the files does not show in the list of files to choose.
DB-Access can only recognize files that are stored in the directory from which you started DB-Access. If the Choose command results in an empty list, and you know that you have command files, exit DB-Access, change directories to the directory that contains your `.sql` files, and restart DB-Access.

Save the current SQL statement

You can save SQL statements in a file for later use, such as to run the statements from the command line or retrieve the saved statements with the Choose option on the SQL menu.

To save the current SQL statement or statements in a file, select the Save option on the SQL menu. Enter a name for the command file:
* Use 1 - 10 characters. Start with a letter, then use any combination of letters, numbers, and underscores (_). Press **Enter** to save the file.
* UNIX: File names are case-sensitive. The file `orders` is not the same as `Orders` or `ORDERS`.

DB-Access appends the extension `.sql` to the file name. For example, if you name your file `cust1`, DB-Access stores the file with the name **cust1.sql**. The CHOOSE screen still lists `cust1`, but the operating system identifies the same file as `cust1.sql` if you list the directory files from the command line.

Support for SPL Routines

You can create and run routines that are written in SPL from the SQL menu.

You can store the SPL routine in a separate command file and then call it from an application or run it as a stand-alone program. After you create the SPL routine, you can run it within DB-Access with the appropriate SQL statement. The following example details the steps.

To create and run an SQL routine

1. To create the text of the routine, type directly in the NEW screen or the Use-editor screen. Enter the SPL and SQL statements for your routine in the statement block of a CREATE PROCEDURE statement.

   Use the CREATE FUNCTION statement if the routine returns values.

2. Use the Run option to create the routine and register it in the **sysprocedures** system catalog table.

3. Use the NEW screen to enter an EXECUTE PROCEDURE statement that names the routine that you want to run.

   If you use IBM Informix and created your routine with the CREATE FUNCTION statement, enter an EXECUTE FUNCTION statement to run the function.

4. Use the Run option to run the routine and display the results.

For example, the `c_proc.sql` command file, which is supplied with the demonstration database, contains an SPL. Before you can run the routine, change the word *procedure* in the `c_proc.sql` file to *function* because the routine returns a value. Then use the Choose option and select **c_proc**.

First you must register the routine in the database. Select the Run option, as the following figure shows.

```
SQL:   New  Run   Modify  Use-editor  Output  Choose  Save  Info  Drop  Exit

Run the current SQL statements.

---------------------- mydata@mynewdb ------- Press CTRL-W for Help --------

create function read_address (lastname char(15))

        returning char(15), char(15), char(20), char(15),char(2), char(5);
        define p_fname, p_city char(15);
        define p_add char(20);
        define p_state char(2);
        define p_zip char(5);
        select fname,  address1, city, state, zipcode
               into p_fname,  p_add, p_city, p_state, p_zip
               from customer
               where lname = lastname;

        return p_fname, lastname, p_add, p_city, p_state, p_zip;

end procedure;
```

Figure 2-2. Displaying the text of an SPL routine on the SQL menu

DB-Access displays a message to indicate that the database server created the routine. To run the routine, select New from the SQL menu and then enter the appropriate EXECUTE statement. In the following example, the user requests the address of a customer whose surname is Pauli:

```
EXECUTE PROCEDURE read_address ("Pauli")
```

After you enter the EXECUTE PROCEDURE or EXECUTE FUNCTION statement on the NEW screen, press **Esc** to return to the SQL menu. Select Run from the SQL menu to run the routine. The following figure shows the result of running the routine.

```
SQL:    New   Run   Modify  Use-editor  Output  Choose  Save  Info  Drop  Exit

Run the current SQL statements.

--------------------- mydata@mynewdb ------- Press CTRL-W for Help --------

Ludwig
Pauli
213 Erstwild Court
Sunnyvale
CA
94086
```

Figure 2-3. Result of running an SPL routine on the SQL menu

Tip: SPL routines are stored in the system catalog tables in executable format. Use the Routines option on the DATABASE INFO menu to display a list of the routines in the current database or to display the text of a specified routine.

What happens when errors occur

If you make any syntax or typing mistakes in an SQL statement, DB-Access does not process the statement. Instead, it continues to display the text of the statement with a message that describes the error.

If an execution or runtime error occurs, DB-Access continues to process the statement and returns an error message. For example, if you try to create a table that was already created, the following message is displayed at the bottom of the screen:

```
310: Table (mavis.mystock) already exists in database.
```

If you try to run a statement that contains more than one SQL statement, you might not see an error message immediately. If, for example, the first statement is a SELECT statement that runs correctly and the next statement contains a typing error, the data that the first statement retrieved shows on the screen before the error message is displayed for the second statement.

When DB-Access detects an error, processing stops and the Modify option on the SQL menu is highlighted. Select one of the following methods to correct the statement:
- Press **Enter** to choose Modify, which returns you to the SQL editor.
- Select the Use-editor option to use the default editor of your choice.

The Database option

Use the Database option to work with databases and transactions.

Use the Database option to:
- Create a database or select a database.

  The database that you work with is called the *current* database.
- Retrieve and display information about a database, such as available dbspaces and the text of routines.
- Delete an existing database or close the current database.
- Commit or rollback transactions.

You can access only databases that are on the current database server. To select a database server as current, you can specify a database server when you start

DB-Access, you can use the Connection menu, or you can run a CONNECT statement from the SQL menu. If you do not explicitly select a database server, DB-Access uses the default database server that the **$INFORMIXSERVER** environment variable specifies as the current database.

If you select or create a database when another database is already open, DB-Access closes that database before it makes your selection the current or new database.

The DATABASE menu displays the following options.

Option	Purpose
Select	Makes a database the current database
Create	Builds a new database and makes that database the current database
Info	Displays information about the current database
Drop	Removes a database from the system. You cannot delete the current database.
cLose	Closes the current database
Exit	Exits the DATABASE menu and returns you to the main menu

List of available databases

When you choose the Select option, the SELECT DATABASE screen opens. The first database in the list of available databases is highlighted, accompanied by the names of database servers.

The list is organized alphabetically by database server and then by database for each database server. You can display a maximum of 512 database names on the SELECT DATABASE screen.

Important: In the SELECT DATABASE screen, the names of databases are limited to 18 characters. If a database name is longer than 18 characters, you see the first 17 characters of the name followed by a '+' sign. Enter a '+' sign to display the complete long name in **vi**. To exit from **vi**, press ESC ZZ.

The list of available databases that is displayed depends on two factors:
- The settings of certain environment variables.
  - If you use one database server, DB-Access displays the names of all databases on the current database server and in your **DBPATH** setting.
  - If you use multiple database servers, the **ONCONFIG** environment variable determines the current database server.
- The current connection. For example:
  - If no explicit connection exists, DB-Access displays the databases in the **DBPATH** setting.
  - If a current explicit connection exists, all databases in the **DBPATH** setting that pertain to the current database server are displayed.

Retrieve nondefault locale information

Globalization affects the order in which lists are displayed in DB-Access. Globalization enables the display and appropriate ordering of

non-English-language data. Informix supports globalization with Global Language Support (GLS) locales. Earlier database server versions used Native Language Support (NLS) for this purpose.

If the current database supports globalization, you can select the Nls option on the DATABASE INFO menu to display information about collating sequence and C CType (character classification type), as the following figure shows.

```
DATABASE INFO:  dBspace   Nls   Routine  Databases  Exit
Display NLS information for a database.

-------------- - stores_demo ------------ Press CTRL-W for Help --------


fr_fr.8859-1 Collating Sequence
CType
```

Figure 2-4. The DATABASE INFO menu with globalization information displayed

DB-Access does not provide an option on the DATABASE INFO menu for displaying the GLS collating sequence and character classification type. To obtain information about the GLS locale that is enabled for your database server, enter the following query with the SQL editor:

```
SELECT tabname, site FROM systables
WHERE tabid = 90 OR tabid = 91
```

The row with tabid 90 stores the COLLATION category of the database locale. The row with tabid 91 stores the CTYPE category of the database locale. The following figure shows the result of the preceding query for the default US English locale.

```
SQL:   New  Run   Modify  Use-editor  Output  Choose  Save  Info  Drop  Exit
Run the current SQL statements.

------- mydata@mynewdb ------ Press CTRL-W for Help ---

tabname          GL_COLLATE
site             en_US.819

tabname          GL_CTYPE
site             en_US.819

2 row(s) retrieved
```

Figure 2-5. Retrieving GLS information

For further information about the COLLATION and CTYPE categories in a GLS locale file, see the *IBM Informix GLS User's Guide*.

Close a database

To close the current database, use the cLose option from the DATABASE menu.

If you begin a transaction but do not commit it or roll it back, and then you try to close a database with transactions, the TRANSACTION menu opens. The TRANSACTION menu ensures that you either commit or roll back an active transaction before you close the current database.

Important: Select an option carefully. You might commit transactions that you do not want if you select Commit, and you do lose any new transactions if you select Rollback.

The TRANSACTION menu also opens whenever you attempt to open a new database or try to leave the DB-Access menu system without first terminating a transaction.

Important: If you begin a transaction in an ANSI-compliant database but do not issue a COMMIT statement or ROLLBACK statement, then try to close the database using a non-menu mode, DB-Access commits the transaction for you. If you do not want to commit the transaction, issue both a ROLLBACK statement and a CLOSE DATABASE statement from the command line.

The Table option

Use the Table option to work with tables.

Use the Table option to perform any of the following table management tasks without SQL programming:
* Create a new table
* Define fragmentation strategy for a new or existing table
* Alter, delete, or display information about an existing table

Use the TABLE menu options as the following table shows.

Option	Purpose
Create	Enables you to define the structure of a new table. The CREATE TABLE menu provides data type options for built-in data types. To define a column with one of the extended data types, such as smart large objects, user-defined (opaque) data types, or a collection data type, use the SQL menu to enter and run a CREATE TABLE statement. DB-Access can construct only a nonclustered, ascending B-tree column index. If you want hash or hybrid fragmentation, use the SQL menu to enter and run the CREATE TABLE or ALTER TABLE statement.
Alter	Enables you to alter the structure of an existing table, including columns, fragmentation, and constraints. You must have the Alter privilege to successfully alter a table. To use the LOAD statement to insert data into a table, you must have both Insert and Select privileges for the table.
Info	Displays information about the structure of a table
Drop	Deletes a table from the database
Move	Moves a table from the current database to another database.
Exit	Returns to the DB-Access main menu

Both the CREATE TABLE and ALTER TABLE menus have the same options, which are described in the following table.

Option	Purpose
Add	Displays the Schema Editor, from which you can add a new column to the table
Modify	Displays the columns that you defined with the Add option so that you can modify the column structure before building the table
Drop	Drops an existing column from the table
Screen	Displays the next screen of column definitions in the Schema Editor

Option	Purpose
Table_options	Enables you to display and select storage spaces for a new table. Displays choices from which to set a fragmentation strategy for a new table. Enables you to set extent sizes and lock mode for a new table. Adds or deletes rowids for an existing fragmented table.
Constraints	Enables you to define primary-key, foreign-key, check, and unique constraints, and to set default column values
Exit	Builds, rebuilds, or discards the schema and structure that you specified with the other options, and then returns to the TABLE menu

Important: You must use the **SPACEBAR** to move between menu options, because the arrow keys control cursor movement in the Schema Editor.

Display table information

Use the Info option on the TABLE menu to display information about the structure of a table.

Note the following items:
- If you are not the table owner, the table name is prefixed by the owner name, as in **june.clients**.
- If the list of tables does not fit on one screen, the last entry is an ellipsis (...). Use the arrow keys to highlight the ellipsis, and the next page of table names are displayed.
- If globalization is enabled, the list of table names is sorted according to the database collation rules defined when the database was created. Thus, different users using different collating sequences for DB-Access see the table names in the database listed in the same order.

To request information about tables on a different database server, use the format *database@server:table* or *database@server:owner.table* at the prompt. The following example requests information about the **customer** table that **dba** created in the **accounts** database on the database server **topend**:

```
INFO FOR TABLE >> accounts@topend:dba.customer
```

The INFO menu has the following options.

Option	Purpose
Columns	Lists data type by column name and indicates which columns can contain a null value
Indexes	Describes each index that is defined for a specified table
Privileges	Lists the users who have Select, Update, Insert, Delete, Index, or Alter privileges for the specified table
References	Lists the users who have the table-level References privilege for the specified table and the names of the columns they can reference
Status	Lists the table name, owner, row size, number of rows and columns, and creation date of the current table
cOnstraints	Displays the referential, primary, unique, and check constraints, and the default values for the columns in the specified table

Option	Purpose
triGgers	Displays header and body information for a specified trigger
Table	Redisplays the INFO FOR TABLE menu so that you can select a different table for examination
Fragments	Lists fragmented dbspaces assigned to the table and, for expression-based fragmentation, displays the expression that is assigned to each dbspace
Exit	Returns to the TABLE menu

Tip: From the CREATE TABLE menu, use Table-options to view extent and lock mode information, or issue a SELECT statement to list the table description in the **systables** system catalog table.

The Connection and Session options

Use the Connection option if you want to connect to a specific database server and database or explicitly disconnect from the current database environment. Use the Session option to display information about the current DB-Access session.

For the globalization considerations that apply to establishing a connection between a client application, such as DB-Access, and a database, see the *IBM Informix GLS User's Guide*. The database server examines the client locale information passed by the client, verifies the database locale, and determines the server-processing locale for transferring data between the client and the database.

You can use the Secure Sockets Layer (SSL) protocol, a communication protocol that ensures privacy and integrity of data that is transmitted over the network, for DB-Access connections with IBM Informix. For information about the SSL protocol, see the "Secure Sockets Layer Communication Protocol" section of the *IBM Informix Security Guide*.

On Windows, if you specify a user identifier but no domain name for a connection to a machine that expects both a domain name and a user name (*domain\user*), DB-Access checks only the local machine and the primary domain for the user account. If you explicitly specify a domain name, that domain is used to search for the user account. The attempted connection fails with error -951 if no matching *domain\user* account is found on the local machine.

The CONNECTION menu displays the following options.

Option	Purpose
Connect	Connects to a database environment. To access a specific database, you must have permission.
Disconnect	Disconnects from the current database environment
Exit	Returns to the DB-Access main menu

When you use the Connect option, the SELECT DATABASE screen alphabetically lists all available databases on the specified database server. The database list on the SELECT DATABASE screen depends on the current connection. For example:

* If no current connection exists or the current connection is an implicit default connection, all the databases that are listed in the **DBPATH** environment variable setting are displayed.

- If a current explicit connection exists, all the databases in the **DBPATH** that pertain to the current server are displayed.

Implicit closures

DB-Access closes any open connections or databases when you connect to a new environment.

DB-Access closes any open connections or databases in the following situations:
- When you connect to a new database environment without explicitly disconnecting from the current one, DB-Access performs an implicit disconnect and the database closes.
- When you connect to a *database@server* and then close the database, the database server remains connected.
- When you connect to a database server, open a database, and then close the database, the database server remains connected.
- If you open a database and then try to connect to a database server, DB-Access performs an implicit disconnect and closes the database.

  Only one connection is allowed. You must disconnect from the database server associated with the open database or close the database before you can connect to another database server.

If DB-Access must close a database that still has outstanding transactions, it prompts you to commit or roll back those transactions.

Appendix A. How to read online help for SQL statements

Specific conventions are used to represent the syntax of SQL statements in DB-Access online help screens.

You can request online help for SQL statements in either of the following ways:
* Highlight the New, Modify, or Use-editor options on the SQL menu and press **CTRL-W**.
* Press **CTRL-W** while you are on the NEW or MODIFY screens of the SQL menu.

The form of the syntax diagrams that shows when you request online Help for SQL statements in DB-Access is different from the syntax diagrams in the *IBM Informix Guide to SQL: Syntax*.

The conventions and rules governing SQL statement syntax in DB-Access online help screens are described in the following list.

ABC   Any term in an SQL statement that is displayed in uppercase letters is a keyword. Type keywords exactly, disregarding case, as shown in the following example:

    CREATE SYNONYM synonym-name

This syntax indicates you must type the keywords CREATE SYNONYM or create synonym without adding or deleting spaces or letters.

abc   Substitute a value for any term that is displayed in lowercase letters. In the previous example, substitute a value for synonym-name.

( )   Type any parentheses as shown. They are part of the syntax of an SQL statement and are not special symbols.

[ ]   Do not type brackets as part of a statement. They surround any part of a statement that is optional. For example:

    CREATE [TEMP] TABLE

This syntax indicates that you can type either CREATE TABLE or CREATE TEMP TABLE.

| The vertical bar indicates a choice among several options. For example:

    [VANILLA | CHOCOLATE [MINT] | STRAWBERRY]

This syntax indicates that you can enter either VANILLA, CHOCOLATE, or STRAWBERRY and that, if you enter CHOCOLATE, you can also enter MINT.

{ }   When you must choose only one of several options, the options are enclosed in braces and are separated by vertical bars. For example:

    {GUAVA | MANGO | PASSIONFRUIT}

This syntax indicates that you must enter either GUAVA, MANGO, or PASSIONFRUIT, but you cannot enter more than one choice.

...   An ellipsis indicates that you can enter an indefinite number of additional items, such as the one immediately preceding the ellipsis. For example:

```
old-column-name
...
```

This syntax indicates that you can enter a series of existing column names after the first one.

The *IBM Informix Guide to SQL: Syntax* contains more detailed syntax diagrams and instructions for interpreting the diagram format that is used in the publication.

Appendix B. Demonstration SQL

Various command files that are available with DB-Access.

The command files all have the extension `.sql` when displayed from the command line but are displayed without the extension on the SQL CHOOSE menu.

Keywords in these command files are shown in uppercase letters to make the SQL statements easier to read. Keywords in the actual command files are lowercase.

Important: Although the command files are listed alphabetically in this appendix, you cannot run the command files that create tables in that order without causing errors. The order in which the tables are created is important because of the referential constraints that link those tables.

When you select the Choose option on the SQL menu, the CHOOSE screen opens. The screen shows a list of the command files that you can access, similar to the display that the following figure shows. These files are included with the **stores_demo** database. Other `.sql` files are described later in this appendix.

```
CHOOSE >> ▌▌
Choose a command file with the Arrow Keys, or enter a name, then press Return.

----------------- stores_demo @dbserver1 -------------- Press CTRL-W for Help ------

alt_cat          c_state          d_trig          sel_ojoin1

c_calls          c_stock          d_view          sel_ojoin2

c_cat            c_stores_demo    del_stock       sel_ojoin3

c_custom         c_table          ins_table       sel_ojoin4

c_index          c_trig           opt_disk        sel_order

c_items          c_type           sel_agg         sel_sub

c_manuf          c_view1          sel_all         sel_union

c_orders         c_view2          sel_group       upd_table

c_proc           d_proc           sel_join
```

Figure B-1. Command files listed on the CHOOSE screen

If you do not see the command files included with your demonstration database, check the following:
- Did you copy the demonstration SQL command files to your current directory when you ran the demonstration database initialization script? If not, you can rerun the initialization script to copy them.
- Did you start DB-Access from the directory in which you installed the demonstration SQL command files? If not, exit DB-Access, change to the appropriate directory, and start DB-Access again.

For instructions about running the initialization script, copying command files, and starting DB-Access, see"dbaccess command: Start DB-Access" on page 1-6.

Use these command files with DB-Access for practice with SQL and the demonstration database. You can rerun the demonstration database initialization script whenever you want to refresh the database tables and SQL files.

SQL files for the relational database model

You can run sample SQL command files on the **stores_demo** demonstration database.

Related concepts:

➦ The stores_demo Database (SQL Reference)

The alt_cat.sql command file

The following command file alters the **catalog** table. It drops the existing constraint **aa** on the **catalog** table and adds a new constraint, **ab**, which specifies cascading deletes. You can use this command file and then the del_stock.sql command file for practice with cascading deletes on a database with logging.

```
ALTER TABLE catalog DROP CONSTRAINT aa;

ALTER TABLE catalog ADD CONSTRAINT
   (FOREIGN KEY (stock_num, manu_code) REFERENCES stock
    ON DELETE CASCADE CONSTRAINT ab);
```

The c_calls.sql command file

The following command file creates the **cust_calls** table:

```
CREATE TABLE cust_calls
   (
   customer_num        INTEGER,
   call_dtime          DATETIME YEAR TO MINUTE,
   user_id             CHAR(18) DEFAULT USER,
   call_code           CHAR(1),
   call_descr          CHAR(240),
   res_dtime           DATETIME YEAR TO MINUTE,
   res_descr           CHAR(240),
   PRIMARY KEY (customer_num, call_dtime),
   FOREIGN KEY (customer_num) REFERENCES customer (customer_num),
   FOREIGN KEY (call_code) REFERENCES call_type (call_code)
   );
```

The c_cat.sql command file

The following command file creates the **catalog** table. It contains a constraint, **aa**, which allows you to practice with cascading deletes by running the SQL statements in the alt_cat.sql and del_stock.sql command files on a database with logging.

```
CREATE TABLE catalog
   (
   catalog_num      SERIAL(10001),
   stock_num        SMALLINT NOT NULL,
   manu_code        CHAR(3) NOT NULL,
   cat_descr        TEXT,
   cat_picture      BYTE,
   cat_advert       VARCHAR(255, 65),
   PRIMARY KEY (catalog_num),
   FOREIGN KEY (stock_num, manu_code) REFERENCES stock
      CONSTRAINT aa
   );
```

The c_custom.sql command file

The following command file creates the **customer** table:

```
CREATE TABLE customer
   (
   customer_num     SERIAL(101),
   fname            CHAR(15),
   lname            CHAR(15),
   company          CHAR(20),
   address1         CHAR(20),
   address2         CHAR(20),
   city             CHAR(15),
   state            CHAR(2),
   zipcode          CHAR(5),
   phone            CHAR(18),
   PRIMARY KEY (customer_num)
   );
```

The c_index.sql command file

The following command file creates an index on the **zipcode** column of the
customer table:

```
CREATE INDEX zip_ix ON customer (zipcode);
```

The c_items.sql command file

The following command file creates the **items** table:

```
CREATE TABLE items
   (
   item_num         SMALLINT,
   order_num        INTEGER,
   stock_num        SMALLINT NOT NULL,
   manu_code        CHAR(3) NOT NULL,
   quantity         SMALLINT CHECK (quantity >= 1),
   total_price      MONEY(8),
   PRIMARY KEY (item_num, order_num),
   FOREIGN KEY (order_num) REFERENCES orders (order_num),
   FOREIGN KEY (stock_num, manu_code) REFERENCES stock
      (stock_num, manu_code)
   );
```

The c_manuf.sql command file

The following command file creates the **manufact** table:

```
CREATE TABLE manufact
   (
   manu_code            CHAR(3),
   manu_name            CHAR(15),
   lead_time            INTERVAL DAY(3) TO DAY,
   PRIMARY KEY (manu_code)
   );
```

The c_orders.sql file

The following command file creates the **orders** table:

```
CREATE TABLE orders
   (
   order_num            SERIAL(1001),
   order_date           DATE,
   customer_num         INTEGER NOT NULL,
   ship_instruct        CHAR(40),
   backlog              CHAR(1),
   po_num               CHAR(10),
   ship_date            DATE,
```

```
ship_weight          DECIMAL(8,2),
ship_charge          MONEY(6),
paid_date            DATE,
PRIMARY KEY (order_num),
FOREIGN KEY (customer_num) REFERENCES customer (customer_num)
);
```

The c_proc.sql command file

The following command file creates an SPL routine. It reads the full name and address of a customer and takes a last name as its only argument.

This routine shows the legacy use of CREATE PROCEDURE.

To conform with the SQL standard preferred with IBM Informix, define a *function* if you want to return values from a routine.

```
CREATE PROCEDURE read_address (lastname CHAR(15))
   RETURNING CHAR(15), CHAR(15), CHAR(20), CHAR(15), CHAR(2), CHAR(5);
   DEFINE p_fname, p_city CHAR(15);
   DEFINE p_add CHAR(20);
   DEFINE p_state CHAR(2);
   DEFINE p_zip CHAR(5);
   SELECT fname, address1, city, state, zipcode
      INTO p_fname,  p_add, p_city, p_state, p_zip
      FROM customer
      WHERE lname = lastname;

   RETURN p_fname, lastname, p_add, p_city, p_state, p_zip;

END PROCEDURE;
```

The c_state command file

The following command file creates the **state** table:

```
CREATE TABLE state
   (
   code                 CHAR(2),
   sname                CHAR(15),
   PRIMARY KEY (code)
   );
```

The c_stock.sql command file

The following command file creates the **stock** table:

```
CREATE TABLE stock
   (
   stock_num            SMALLINT,
   manu_code            CHAR(3),
   description          CHAR(15),
   unit_price           MONEY(6),
   unit                 CHAR(4),
   unit_descr           CHAR(15),
   PRIMARY KEY (stock_num, manu_code),
   FOREIGN KEY (manu_code) REFERENCES manufact
   );
```

The c_stores.sql command file

The following command file creates the **stores_demo** database:

```
CREATE DATABASE stores_demo;
```

The c_table.sql command file

The following command file creates a database named **restock** and then creates a custom table named **sports** in that database:

```
CREATE DATABASE restock;

CREATE TABLE sports
   (
   catalog_no          SERIAL UNIQUE,
   stock_no            SMALLINT,
   mfg_code            CHAR(5),
   mfg_name            CHAR(20),
   phone               CHAR(18),
   descript            VARCHAR(255)
   );
```

The c_trig.sql command file

The following command file creates a table named **log_record** and then creates a trigger named **upqty_i**, which updates it:

```
CREATE TABLE log_record
   (item_num    SMALLINT,
   ord_num      INTEGER,
   username     CHARACTER(8),
   update_time  DATETIME YEAR TO MINUTE,
   old_qty      SMALLINT,
   new_qty      SMALLINT);

CREATE TRIGGER upqty_i
UPDATE OF quantity ON items
REFERENCING OLD AS pre_upd
       NEW AS post_upd
FOR EACH ROW(INSERT INTO log_record
   VALUES (pre_upd.item_num, pre_upd.order_num, USER, CURRENT,
       pre_upd.quantity, post_upd.quantity));
```

The c_type.sql command file

The following command file creates the **call_type** table:

```
CREATE TABLE call_type
   (
   call_code               CHAR(1),
   code_descr              CHAR(30),
   PRIMARY KEY (call_code)
   );
```

The c_view1.sql command file

The following command file creates a view called **custview** on a single table and grants privileges on the view to **public**. It includes the WITH CHECK OPTION keywords to verify that any changes made to underlying tables through the view do not violate the definition of the view.

```
CREATE VIEW custview (firstname, lastname, company, city) AS
   SELECT fname, lname, company, city
      FROM customer
      WHERE city = 'Redwood City'
      WITH CHECK OPTION;

GRANT DELETE, INSERT, SELECT, UPDATE
   ON custview
   TO public;
```

The c_view2.sql command file

The following command file creates a view on the **orders** and **items** tables:

```
CREATE VIEW someorders (custnum,ocustnum,newprice) AS
   SELECT orders.order_num, items.order_num,
        items.total_price*1.5
      FROM orders, items
      WHERE orders.order_num = items.order_num
      AND items.total_price > 100.00;
```

The d_proc.sql command file

The following command file drops the SPL routine that the c_proc.sql command file created:

```
DROP PROCEDURE read_address;
```

The d_trig.sql command file

The following command file drops the trigger that the c_trig.sql command file created:

```
DROP TRIGGER upqty_i;
```

The d_view.sql command file

The following command file drops the view named **custview** that the c_view1.sql command file created:

```
DROP VIEW custview;
```

The del_stock.sql command file

The following command file deletes rows from the **stock** table where the stock number is 102. This delete will cascade to the **catalog** table (although the related manufacturer codes will remain in the **manufact** table). The del_stock.sql command file can be used following the alt_cat.sql command file for practice with cascading deletes on a database with logging.

```
DELETE FROM stock WHERE stock_num = 102;
```

After running the SQL statements in the alt_cat.sql and del_stock.sql command files, issue the following query on the **catalog** table to verify that the rows were deleted:

```
SELECT * FROM catalog WHERE stock_num = 102;
```

The **stores_demo** database has been changed. You might want to rerun the **dbaccessdemo** script to rebuild the original database.

The ins_table.sql command file

The following command file inserts one row into the **sports** table that the c_table.sql command file created:

```
INSERT INTO sports
   VALUES (0,18,'PARKR', 'Parker Products', '503-555-1212',
   'Heavy-weight cotton canvas gi, designed for aikido or
   judo but suitable for karate. Quilted top with side ties,
   drawstring waist on pants. White with white belt.
   Pre-washed for minimum shrinkage. Sizes 3-6.');
```

The sel_agg.sql command file

The SELECT statement in the following command file queries on table data using aggregate functions. It combines the aggregate functions MAX and MIN in a single statement.

```
SELECT MAX (ship_charge), MIN (ship_charge)
   FROM orders;
```

The sel_all.sql command file

The following example command file contains all seven SELECT statement clauses that you can use in the IBM Informix implementation of interactive SQL. This SELECT statement joins the **orders** and **items** tables. It also uses display labels, table aliases, and integers as column indicators; groups and orders the data; and puts the results into a temporary table.

```
SELECT o.order_num, SUM (i.total_price) price,
    paid_date - order_date span
   FROM orders o, items i
   WHERE o.order_date > '01/01/90'
      AND o.customer_num > 110
      AND o.order_num = i.order_num
   GROUP BY 1, 3
   HAVING COUNT (*) < 5
   ORDER BY 3
   INTO TEMP temptab1;
```

The sel_group.sql command file

The following example command file includes the GROUP BY and HAVING clauses. The HAVING clause usually complements a GROUP BY clause by applying one or more qualifying conditions to groups after they are formed, which is similar to the way the WHERE clause qualifies individual rows. (One advantage to using a HAVING clause is that you can include aggregates in the search condition; you cannot include aggregates in the search condition of a WHERE clause.)

Each HAVING clause compares one column or aggregate expression of the group with another aggregate expression of the group or with a constant. You can use the HAVING clause to place conditions on both column values and aggregate values in the group list.

```
SELECT order_num, COUNT(*) number, AVG (total_price) average
   FROM items
   GROUP BY order_num
   HAVING COUNT(*) > 2;
```

The sel_join.sql command file

The following example command file uses a simple join on the **customer** and **cust_calls** tables. This query returns only those rows that show the customer has made a call to customer service.

```
SELECT c.customer_num, c.lname, c.company,
    c.phone, u.call_dtime, u.call_descr
   FROM customer c, cust_calls u
   WHERE c.customer_num = u.customer_num;
```

The sel_ojoin1.sql command file

The following example command file uses a simple outer join on two tables. The use of the keyword OUTER in front of the **cust_calls** table makes it the subservient table. An outer join causes the query to return information about all customers,

even if they do not make calls to customer service. All rows from the dominant **customer** table are retrieved, and null values are assigned to corresponding rows from the subservient **cust_calls** table.

```
SELECT c.customer_num, c.lname, c.company,
    c.phone, u.call_dtime, u.call_descr
  FROM customer c, OUTER cust_calls u
  WHERE c.customer_num = u.customer_num;
```

The sel_ojoin2.sql command file

The following example command file creates an outer join, which is the result of a simple join to a third table. This second type of outer join is called a *nested simple join*.

This query first performs a simple join on the **orders** and **items** tables, retrieving information about all orders for items with a **manu_code** of KAR or SHM. It then performs an outer join, which combines this information with data from the dominant **customer** table. An optional ORDER BY clause reorganizes the data.

```
SELECT c.customer_num, c.lname, o.order_num,
    i.stock_num, i.manu_code, i.quantity
  FROM customer c, OUTER (orders o, items i)
  WHERE c.customer_num = o.customer_num
    AND o.order_num = i.order_num
    AND manu_code IN ('KAR', 'SHM')
  ORDER BY lname;
```

The sel_ojoin3.sql command file

The following example SELECT statement is the third type of outer join, known as a *nested outer join*. It queries on table data by creating an outer join, which is the result of an outer join to a third table.

This query first performs an outer join on the **orders** and **items** tables, retrieving information about all orders for items with a **manu_code** of KAR or SHM. It then performs an outer join, which combines this information with data from the dominant **customer** table. This query preserves order numbers that the previous example eliminated, returning rows for orders that do not contain items with either manufacturer code. An optional ORDER BY clause reorganizes the data.

```
SELECT c.customer_num, lname, o.order_num,
    stock_num, manu_code, quantity
  FROM customer c, OUTER (orders o, OUTER items i)
  WHERE c.customer_num = o.customer_num
    AND o.order_num = i.order_num
    AND manu_code IN ('KAR', 'SHM')
  ORDER BY lname;
```

The sel_ojoin4.sql command file

The following example queries on table data using the fourth type of outer join. This query shows an outer join, which is the result of an outer join of each of two tables to a third table. In this type of outer join, join relationships are possible *only* between the dominant table and subservient tables.

This query individually joins the subservient tables **orders** and **cust_calls** to the dominant **customer** table but does not join the two subservient tables. (An INTO TEMP clause selects the results into a temporary table.)

```
SELECT c.customer_num, lname, o.order_num,
    order_date, call_dtime
  FROM customer c, OUTER orders o, OUTER cust_calls x
  WHERE c.customer_num = o.customer_num
    AND c.customer_num = x.customer_num
  INTO temp service;
```

The sel_order.sql command file

The following example uses the ORDER BY and WHERE clauses to query. In this SELECT statement, the comparison 'bicycle%' (LIKE condition, or 'bicycle*' for a MATCHES condition) specifies the letters bicycle followed by any sequence of zero or more characters. It narrows the search further by adding another comparison condition that excludes a **manu_code** of PRC.

```
SELECT * FROM stock
  WHERE description LIKE 'bicycle%'
    AND manu_code NOT LIKE 'PRC'
  ORDER BY description, manu_code;
```

The sel_sub.sql command file

The following example uses a subquery to query. This self-join uses a correlated subquery to retrieve and list the 10 highest-priced items ordered.

```
SELECT order_num, total_price
  FROM items a
  WHERE 10 >
    (SELECT COUNT (*)
       FROM items b
       WHERE b.total_price < a.total_price)
  ORDER BY total_price;
```

The sel_union.sql command file

The following example uses the UNION clause to query on data in two tables. The compound query performs a union on the **stock_num** and **manu_code** columns in the **stock** and **items** tables. The statement selects items that have a unit price of less than $25.00 or that have been ordered in quantities greater than three, and it lists their **stock_num** and **manu_code**.

```
SELECT DISTINCT stock_num, manu_code
  FROM stock
  WHERE unit_price < 25.00

UNION

SELECT stock_num, manu_code
  FROM items
  WHERE quantity > 3;
```

The upd_table.sql command file

The following example updates the **sports** table that the c_table.sql command file created:

```
UPDATE sports
  SET phone = '808-555-1212'
  WHERE mfg_code = 'PARKR';
```

SQL files for the Dimensional Database Model

You can implement a dimensional database for data-warehousing applications by running scripts that create the **sales_demo** database.

The **sales_demo** database is based on the **stores_demo** schema and data.

To create the **sales_demo** database:

1. Create a **stores_demo** database with the following command:

   ```
   dbaccessdemo -log
   ```

2. Make sure that the `createdw.sql` and `loaddw.sql` files are in the same directory as the files with extension `.unl` that the `loaddw.sql` uses.

3. Run the `createdw.sql` file.

4. Run the `loaddw.sql` file.

The createdw.sql file

This file creates the new **sales_demo** database with logging and then creates tables within that database. It contains the following statements:

```
create database sales_demo with log;

create table product (
   product_code integer,
   product_name char(31),
   vendor_code char(3),
   vendor_name char(15),
   product_line_code smallint,
   product_line_name char(15));

create table customer (
   customer_code integer,
   customer_name char(31),
   company_name char(20));


create table sales (
   customer_code integer,
   district_code smallint,
   time_code integer,
   product_code integer,
   units_sold smallint,
   revenue money (8,2),
   cost money (8,2),
   net_profit money(8,2));

create table time
   (
   time_code int,
   order_date date,
   month_code smallint,
   month_name char(10),
   quarter_code smallint,
   quarter_name char(10),
   year integer
   );

create table geography (
    district_code serial,
   district_name char(15),
   state_code char(2),
   state_name char(18),
   region smallint);
```

The loaddw.sql file

This file contains the commands necessary to load data from two sources:

- The files with the extension .unl in your demonstration directory
- Data selected from the **stores_demo** database

These SQL statements in loaddw.sql accomplish these actions:

```
connect to "stores_demo ";
load from "add_orders.unl"
   insert into stores_demo :orders;
load from 'add_items.unl'
   insert into stores_demo :items;

connect to "sales_demo";
load from 'costs.unl'
   insert into cost;
load from 'time.unl'
   insert into time;

insert into geography(district_name, state_code, state_name)
   select distinct c.city, s.code, s.sname
from stores_demo :customer c, stores_demo :state s
   where c.state = s.code;
update geography      -- converts state_code values to region values
   set region = 1
   where state_code = "CA";
update geography
   set region = 2
   where state_code <> "CA";

insert into customer (customer_code, customer_name, company_name)
   select c.customer_num, trim(c.fname) || " " || c.lname, c.company
   from stores_demo :customer c;

insert into product (product_code, product_name, vendor_code,
   vendor_name, product_line_code, product_line_name)
 select a.catalog_num,
     trim(m.manu_name) || " "|| s.description,
     m.manu_code, m.manu_name, s.stock_num, s.description
   from stores_demo :catalog a, stores_demo :manufact m,
     stores_demo :stock s
   where a.stock_num = s.stock_num and
     a.manu_code = s.manu_code and
      s.manu_code = m.manu_code;
insert into sales (customer_code, district_code,
   time_code, product_code,
   units_sold, revenue, cost, net_profit)
   select c.customer_num, g.district_code, t.time_code, p.product_code,
   SUM(i.quantity), SUM(i.total_price),
   SUM(i.quantity * x.cost),
   SUM(i.total_price) - SUM(i.quantity * x.cost)
   from stores_demo :customer c, geography g, time t,
     product p,
     stores_demo :items i, stores_demo :orders o, cost x
    where c.customer_num = o.customer_num and
      o.order_num = i.order_num and
      p.product_line_code = i.stock_num and
     p.vendor_code = i.manu_code and
     t.order_date = o.order_date and
     p.product_code = x.product_code and
     c.city = g.district_name
     GROUP BY 1,2,3,4;

connect to "stores_demo ";
load from 'add_orders.unl'
```

```
                 insert into stores_demo :orders;
      load from 'add_items.unl'
         insert into stores_demo :items;

      connect to "sales_demo";
      load from 'costs.unl'
         insert into cost;
      load from 'time.unl'
         insert into time;

      insert into geography(district_name, state_code, state_name)
         select distinct c.city, s.code, s.sname
      from stores_demo :customer c, stores_demo :state s
         where c.state = s.code;
      update geography       -- converts state_code values to region values
         set region = 1
         where state_code = "CA";
      update geography
         set region = 2
         where state_code <> "CA";

      insert into customer (customer_code, customer_name, company_name)
         select c.customer_num, trim(c.fname) || " " || c.lname, c.company
         from stores_demo :customer c;

      insert into product (product_code, product_name, vendor_code,
         vendor_name, product_line_code, product_line_name)
       select a.catalog_num,
           trim(m.manu_name) || " " || s.description,
           m.manu_code, m.manu_name, s.stock_num, s.description
         from stores_demo :catalog a, stores_demo :manufact m,
           stores_demo :stock s
         where a.stock_num = s.stock_num and
           a.manu_code = s.manu_code and
            s.manu_code = m.manu_code;

      insert into sales (customer_code, district_code,
         time_code, product_code,
         units_sold, revenue, cost, net_profit)
         select c.customer_num, g.district_code, t.time_code, p.product_code,
         SUM(i.quantity), SUM(i.total_price),
         SUM(i.quantity * x.cost),
         SUM(i.total_price) - SUM(i.quantity * x.cost)
         from stores_demo :customer c, geography g, time t, product p,
           stores_demo :items i, stores_demo :orders o, cost x
          where c.customer_num = o.customer_num and
            o.order_num = i.order_num and
            p.product_line_code = i.stock_num and
            p.vendor_code = i.manu_code and
            t.order_date = o.order_date and
            p.product_code = x.product_code and
            c.city = g.district_name
            GROUP BY 1,2,3,4;
```

User-defined routines for the object-relational database model

You can run sample user-defined routines on the **superstores_demo** database.

The **superstores_demo** database does not replace the **stores_demo** database. Both databases are available. The **superstores_demo** database schema is not compatible with earlier versions with **stores_demo**. In many cases, you cannot use test queries developed for **stores_demo** against the tables of **superstores_demo** because the tables differ.

No SQL command files are associated specifically with **superstores_demo**. However, there are user-defined routines that you can run in the SQL editor or a system editor.

The **superstores_demo** database includes examples of the following features:
- Collection types: SET, LIST
- Named row types: location_t, loc_us_t, loc_non_us_t
- Unnamed row types
- Type and table inheritance
- Built-in data types: BOOLEAN, SERIAL8, INT8
- Distinct data type: percent
- Smart large objects: BLOB and CLOB

The **superstores_demo** database has row types and tables to support the following table-inheritance hierarchies:
- customer/retail_customer
- customer/whlsale_customer
- location/location_us
- location/location_non_us

For more information about user-defined routines, see *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Related concepts:

The superstores_demo database (SQL Reference)

Appendix C. Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

Accessibility features for IBM Informix products

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in IBM Informix products. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers.
- The attachment of alternative input and output devices.

Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

Related accessibility information

IBM is committed to making our documentation accessible to persons with disabilities. Our publications are available in HTML format so that they can be accessed with assistive technology such as screen reader software.

IBM and accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the IBM commitment to accessibility.

Dotted decimal syntax diagrams

The syntax diagrams in our publications are available in dotted decimal format, which is an accessible format that is available only if you are using a screen reader.

In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), the elements can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read punctuation. All syntax elements that have the same dotted decimal number (for example, all syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, the word or symbol is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is read as 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol that provides information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, that element is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 refers to a separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

? Specifies an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element (for example, 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! Specifies a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

* Specifies a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be

repeated. For example, if you hear the line `5.1* data-area`, you know that you can include more than one data area or you can include none. If you hear the lines `3*`, `3 HOST`, and `3 STATE`, you know that you can include `HOST`, `STATE`, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.

2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write `HOST STATE`, but you cannot write `HOST HOST`.

3. The `*` symbol is equivalent to a loop-back line in a railroad syntax diagram.

+   Specifies a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times. For example, if you hear the line `6.1+ data-area`, you must include at least one data area. If you hear the lines `2+`, `2 HOST`, and `2 STATE`, you know that you must include `HOST`, `STATE`, or both. As for the * symbol, you can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy,

modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Index

Special characters

A

C

D

IBM ®

Printed in USA