

ESKER *Tun*[®] *Plus*

Tun SQL – Data Access



Tun Plus 2006
Issued April 2006

Copyright © 1999-2006 Esker S.A. All rights reserved.

© 1998-2003 Neil Hodgson [neilh@scintilla.org]; © 1998-2002 The OpenSSL Project; © 1996 Wolfgang Platzer [wplatzer@iaik.tu-graz.ac.at]; © 1994-2003 Sun Microsystems, Inc.; © 1995-1998 Eric Young [eay@cryptsoft.com]. All rights reserved. Tun contains components which are derived in part from OpenSSH software. See the copyright.txt file on the Tun CD for additional copyright notices, conditions of use and disclaimers. Use and duplicate only in accordance with the terms of the Software License Agreement - Tun Products.

North and South American distributions of this manual are printed in the U.S.A. All other distributions are printed in France. Information in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form or by any means without the prior written consent of Esker S.A..



Esker S.A., 10 rue des Émeraudes, 69006 Lyon, France
Tel: +33 (0)4.72.83.46.46 ♦ Fax: +33 (0)4.72.83.46.40 ♦ info@esker.fr ♦ www.esker.fr

Esker, Inc., 1212 Deming Way, Suite 350, Madison, WI 53717 USA
Tel: +1.608.828.6000 ♦ Fax: +1.608.828.6001 ♦ info@esker.com ♦ www.esker.com

Esker Australia Pty Ltd. (Lane Cove - NSW) ♦ Tel: +61 (0)2 8596 5100 ♦ info@esker.com.au ♦ www.esker.com.au

Esker GmbH (München) ♦ Tel: +49 (0) 89 700 887 0 ♦ info@esker.de ♦ www.esker.de

Esker Italia SRL (Milano) ♦ Tel: +39 02 89 20 03 03 ♦ info@esker.it ♦ www.esker.it

Esker Ibérica, S.L. (Las Rozas) ♦ Tel: +34 91 552 9265 ♦ info@esker.es ♦ www.esker.es

Esker UK Ltd. (Derby) ♦ Tel: +44 1332 54 8181 ♦ info@esker.co.uk ♦ www.esker.co.uk

Esker, the Esker logo, Esker Pro, Extending the Reach of Information, Tun, and Tun Emul are trademarks, registered trademarks or service marks of Esker S.A. in the U.S., France and other countries.

The following are trademarks of their respective owners in the United States and other countries: Microsoft, Windows, Back-Office, MS-DOS, XENIX are registered trademarks of Microsoft Corp. Netscape and Netscape Navigator are registered trademarks of Netscape Communications Corp. IBM, AS/400, and AIX are registered trademarks of IBM Corp. SCO is a registered trademark of Caldera International, Inc. NetWare is a registered trademark of Novell, Inc. Sun, Sun Microsystems and Java are trademarks of Sun Microsystems, Inc. Oracle is a registered trademark of Oracle Corp. Informix is a registered trademark of Informix Software Inc. Sybase is a registered trademark of Sybase, Inc. Progress is a registered trademark of Progress Software Corp. All other trademarks mentioned are the property of their respective owners.

PREFACE

Tun SQL - Data Access is an applications and server suite that enables PCs to work in client/server mode with remote databases (Informix, Oracle, Sybase, DB2, Progress and C-ISAM). **Tun SQL** uses the ODBC architecture defined by Microsoft.

Tun SQL runs on the following platforms: Windows 3.x, Windows 95, Windows 98, Windows NT 3.51 and Windows NT 4.0, Windows 2000, Citrix WinFrame, Citrix MetaFrame and Windows NT TSE.

Tun SQL is part of the **Tun** software product range, as shown below:

	Windows Version (excluding Citrix/ Microsoft NT TSE)	Components in a multi- user environment
Esker TCP/IP Stack	TCP/IP communication stack for Windows 3.x (DLL)	N/A
Network resource access (Tun NET)	TCP/IP applications (NIS, NFS Client and Server, PING, Printer redirection and sharing, FTP Client and Server, TELNET, RSH Client and Server, TAR, WALL, TFTP, TIME)	TCP/IP applications (NIS, NFS Client and Server, PING, Printer redirection and sharing, FTP Client and Server, TELNET VT320, RSH Client, TAR, WALL)
Applications access (Tun EMUL)	Terminal emulator (asynchronous, IBM3270 and IBM5250 emulation, 3287/3812 printer)	Terminal emulator (asynchronous, IBM3270 and IBM5250 emulation, 3287/3812 printer)
Data access (Tun SQL)	ODBC drivers for TCP/IP Client/Server mode (Oracle, Informix, Sybase, DB2, Progress and C-ISAM DBMSs) and database revamping tool	ODBC drivers for TCP/IP Client/Server mode (Oracle, Informix, Sybase, DB2, Progress and C-ISAM DBMSs) and database revamping tool
TCP/IP Network Services	NIS Browser, printer redirection and sharing	Printer redirection and sharing

Most of the functionalities and procedures described in this manual apply equally to Windows 3.x, Windows 95, Windows 98, Windows NT 3.51, Windows NT 4.0, Windows 2000 or Citrix/Windows NT TSE. However, some functionalities and procedures apply to only one or more of these platforms. In this case, the paragraph or section in question is indicated as follows:



Win 3.x

Windows 3.x



Win 95

Windows 95 and Windows 98



Win NT
2000

Windows NT (Windows NT 3.51 and Windows NT 4.0, including multi-user environment if no indication) and Windows 2000



NT 4.0
2000

Windows NT 4.0, including Citrix/Windows NT TSE if no indication



NT 3.51

Windows NT 3.51, including multi-user environment



Win 32

32-bit Windows (Windows 95, Windows NT 3.51 and Windows NT 4.0, including multi-user environment if no indication and Windows 2000)



Multi-user environment



Excluding multi-user environment

Tun SQL for Windows is also delivered with **Tun PLUS** which includes all the modules mentioned above. The **Tun PLUS** installation procedure proposes to install **Tun SQL**.

Except for the **Tun PLUS** for Multi-User Windows version, you can install **Tun SQL** independently of **Tun PLUS**.

Note:

In the entire document, Windows 95 features correspond to Windows 98 features.

TABLE OF CONTENTS

PART 1 PRESENTATION AND USE

CHAPTER 1 - Introduction to Tun SQL.....	1-9
The ODBC mechanism.....	1-9
The Client/Server model.....	1-11
ODBC and the SQL Client/Server model.....	1-13
Tun SQL.....	1-13
CHAPTER 2 - Configuration and use in Windows.....	2-17
Verifying the functioning of Tun SQL.....	2-17
Creating a database.....	2-21
Creating a data source.....	2-21
Transferring the demonstration database.....	2-29
Creating a virtual data source.....	2-31
Character conversion tables.....	2-33
CHAPTER 3 - C-ISAM.....	3-37
Introduction to C-ISAM.....	3-37
Using sqltools.....	3-39

PART 2 DATABASE REVAMPING

CHAPTER 4 - Revamping.....	4-51
Virtual databases.....	4-51
Revamping in Tun SQL.....	4-53
CHAPTER 5 - Tun DB Revamp general use.....	5-57
General options.....	5-57
Importing data source environments.....	5-59
Creating an environment.....	5-60
Creating a virtual table.....	5-60
Creating a field.....	5-61
Assigning field filters.....	5-64
Inter-table links.....	5-66
Querying real and virtual databases.....	5-68
Validating an environment.....	5-70
Exporting data source environments.....	5-71
Updating a virtual data source.....	5-72
Creating a virtual data source.....	5-73
Displaying warnings.....	5-73

Local revamped data source management	5-74
Field identification.....	5-75

PART 3 APPENDICES

APPENDIX A - Reference.....	A-79
-----------------------------	------

APPENDIX B - SQL statements used in C-ISAM.....	B-95
Principle instructions	B-95
SQL statement syntax	B-96
Data types	B-130

INDEX.....	I-137
-------------------	--------------

PART 1
PRESENTATION AND USE

INTRODUCTION TO TUN SQL

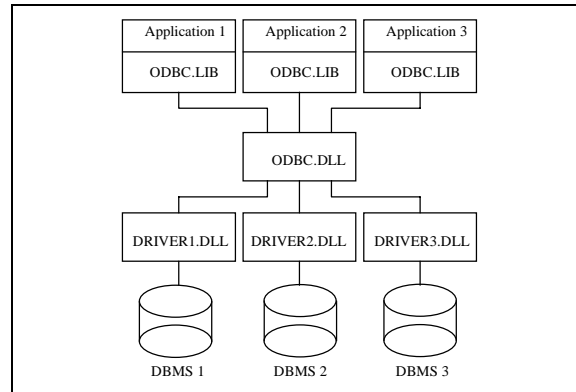
The ODBC mechanism

In the world of databases, programmers traditionally use a mechanism called "Embedded SQL" to provide an interface between their applications and a specific database. Embedded SQL lets you insert SQL requests into programs written in COBOL or C. It has the advantage of making applications more widely portable on different machines.

Embedded SQL, however, has several disadvantages:

- There are as many Embedded SQLs as there are DBMS engines on the market. Applications using SQL can only interface with one DBMS at a time. They must be rewritten or at least modified if they're to interface with other databases. For applications that need to interface with all the databases on the market, it's inconceivable to use the Embedded SQL mechanism.
- Embedded SQL is relatively undeveloped, quite restrictive and difficult to use. It doesn't let databases be used to full advantage, and it's sometimes preferable to use the API provided by the DBMS directly.

To compensate for the disadvantages of Embedded SQL, Microsoft conceived a new approach based on the ODBC mechanism (Open Database Connectivity). ODBC is a further development of WOSA (Windows Open System Architecture).



ODBC is a well-defined set of C functions that makes it possible to retrieve data from, or update data in, a DBMS. These functions have been assembled in a DLL (Dynamic Link Library) that can be used by any Windows application. The functions of the ODBC DLL (ODBC.DLL) analyze SQL requests. They then pass them to ODBC drivers whose job it is to convert the calls to suit the particular API of the DBMS you're using. An ODBC driver lets you view the DBMS interface. You can then enable your application to use it just like any other ODBC-compliant DBMS.

Microsoft supplies the ODBC.DLL library and all the tools required to use it. They don't, however, supply ODBC drivers for all the DBMSs on the market. Microsoft is happy supplying drivers for its proprietary storage systems (Excel, Word, Access...).

ODBC drivers for specific databases are supplied directly by the DBMS publisher or by third parties who specialize in this domain (Esker).



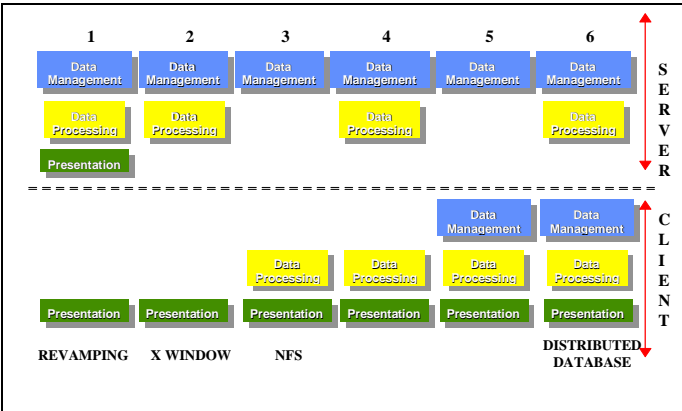
Finally, ODBC provides maximum interoperability. A simple Windows application can access different management systems even if it wasn't specifically designed to do so. ODBC lets developers program, compile and deliver programs without having to worry about the DBMS the program will be used with. The users only have to provide the right driver so the application on which they're working cooperates with their DBMS.

ODBC is a valuable mechanism for multi-domain applications such as spreadsheets, word processing applications, and development tools that manipulate information from DBMSs without knowing *a priori* which DBMS is being used.

The Client/Server model

For some years, the term Client/Server has been on the lips of everyone in the computing industry. In its widest sense, the Client/Server model is a computing model in which at least two units cooperate to supply a particular service.

The "Gartner Group" has identified six principal types of Client/Server mode applications which they've classified according to the number of functions performed by the client or the server.



The applications that make the least demands on the "client" are "revamping" applications or X-WINDOWS servers. The applications that make the most demands on the "client" are those that use distributed databases.

When the computing industry speaks about the Client/Server model, it often means applications concerned with distributed databases.

The most common illustration of this mode is shown in the following schematic:

Erreur! Des objets ne peuvent pas être créés à partir des codes de champs de mise en forme.

A "client" PC has a conventional management system that works in graphic mode. The data is stored centrally on an UNIX server and managed by a Relational Database Management System (RDBMS). To retrieve or update data, the client sends an SQL request to the server. The server executes the request and returns the reply to the client over the network.

The principal advantages of this architecture are the following:

- The end-user has a graphic, user-friendly Man/Machine interface on his Windows PC.
- The PC can be used for tasks other than normal applications (office applications, calculations, personal applications...).
- From his machine, the user can access centralized data on a server at the same time as other users.
- The server is relieved of applications tasks and all its power can be concentrated on serving data. There is a balanced distribution of the workload between the client and the server.
- The network is only used to send essential application data: It isn't overloaded with presentation information.

SQL Client/Server mode as described above is a modernized version of the transactional mode as is still encountered in a Mainframe-synchronous terminal environment.



ODBC and the SQL Client/Server model

As well as facilitating access to all data management systems, ODBC can also be used on a remote database. In this context, ODBC enables a Windows application to use any sort of DBMS. It also enables multi-domain applications such as "Excel", "Word" or "Access" to use the company's centralized information. In a Client/Server context, the ODBC mechanism lets you increase the number of applications which can use the company's data.

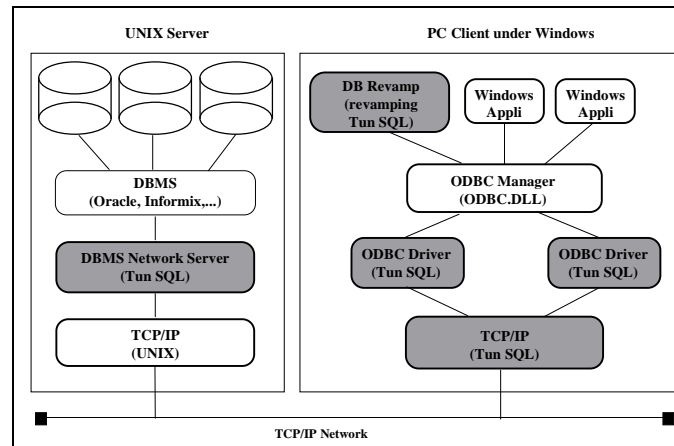
The implementation of this type of architecture at the present time is no simple matter. Users who already have a local PC network, a UNIX server and a DBMS must also acquire the following components to implement an SQL Client/Server architecture with ODBC:

- The network server part of the DBMS (Informix Net, Sql Net...).
- The PC client part of the DBMS (Informix Net PC, Sql Net PC...).
- The appropriate ODBC driver.
- A **Winsock**-compatible TCP/IP stack for the PC.

DBMS software houses always supply the first two components. This isn't always the case with the ODBC driver and is never the case for the TCP/IP stack. The last two components must be obtained elsewhere, with particular attention being paid to the question of compatibility.

Tun SQL

Tun SQL was designed to supply a definitive solution to the above problem. It integrates all the components mentioned above, as well as powerful database revamping functionality and a virtual ODBC driver for access to the revamped databases, into one homogenous software package. Even the network components of the DBMS (client and server) are included in **Tun SQL**. This represents a considerable saving when it comes to equipping a large PC installation. It especially makes things very simple when it comes to implementing SQL Client/Server architecture.



➤ **A single ODBC driver for most of the DBMSs on the market**

To limit the number of software products required for the PC client, **Tun SQL** features a single ODBC driver to access the following DBMSs indifferently:

- Oracle version 7.
- Informix versions 5 and 7.
- Sybase version 10.
- DB2 version 2.
- Progress versions 6, 7 and 8
- C-ISAM files (versions 4 through 7).

➤ **Database revamping**

Tun SQL, with the help of an integrated user-friendly application, lets you redefine the tables in a database and make them more accessible to the end-user (through customized table reorganization, modification of table and field names, and preset functions).

➤ **A virtual ODBC driver for revamped databases**

To use a database that's been revamped (redefined), **Tun SQL** includes a virtual ODBC driver which translates requests made to virtual tables into requests that a normal ODBC driver can handle.



➤ The server part of the DBMS is included in Tun SQL

The server part of each of the DBMSs is installed on the UNIX or NT machine and is supplied as standard with **Tun SQL** for the following operating systems:

- ScoUnix 3.2x v.4.2 and 5.0
- SunOs 4.1.3
- Solaris 2.5
- AIX 3.2 and 4.1
- HP-UX 9.x and 10.x
- OSF1 v.3.2
- Windows NT
- IBM MVS

This feature saves **Tun SQL** users the cost of the server side of their DBMSs.

➤ TCP/IP stack delivered as standard (16-bit Windows)



Win 3.x

Like all the software in the **Tun** range, **Tun SQL** is delivered with **Esker TCP/IP Stack** as standard. The stack has an excellent performance record and has been tested with all of **Tun SQL**'s components. The inclusion of the stack in the **Tun SQL** package saves you the trouble of having to obtain a stack elsewhere.

➤ Simple installation and administration

The objective of **Tun SQL** is to facilitate the implementation of a Client/Server architecture based on Windows and UNIX. Consequently, **Tun SQL** has a simple installation procedure for UNIX and Windows and comes with complete documentation.

In addition to the ODBC driver, two Windows applications are supplied for testing and implementing the Client/Server architecture:

- **Tun DB Show** tests the Client/Server connection from end to end. The application can query the UNIX server to find out which DBMSs are installed: It can also query some DBMSs to find out which databases they manage.
- **Tun DB Script** executes SQL batch files on the Windows machine to create databases on the remote DBMS.

In addition to the security offered by UNIX and the different DBMSs, **Tun SQL** includes a feature that denies certain Windows applications access to sensitive databases.



Lastly, the integration of the NIS (Network Information Service) into **Tun SQL** permits the centralized management of network resources and facilitates access to remote resources. For more information on the NIS, please consult the **TCP/IP Network Services** user guide or the **Tun NET** user guide.

➤ **Tun SQL driver conformity**

The **Tun SQL** driver supports all the level 1 functions and some of the level 2 functions. The "Microsoft ODBC Cursor Library" is supplied with the driver. Although this library only supports static and "forward only" cursors, this is sufficient for many applications.



CONFIGURATION AND USE IN WINDOWS

Verifying the functioning of Tun SQL

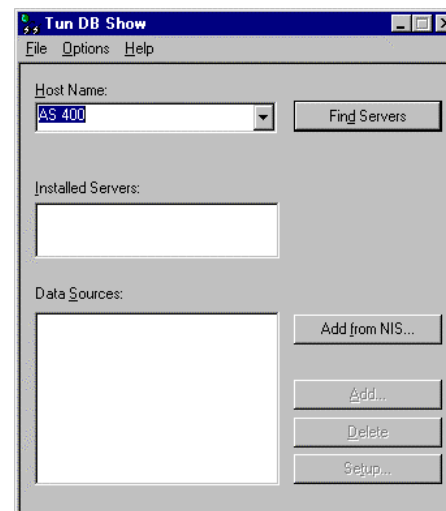
➤ Executing Tun DB Show

When you've installed and configured **Tun SQL** on both the Windows and UNIX machines, you must check everything's running correctly. You can do this by running **Tun DB Show**.



Run the program by clicking the **Tun DB Show** icon in the **Data Access** group (menu **Start, Programs, Esker Tun** under Windows 95/98/2000 and Windows NT).

The following window appears:



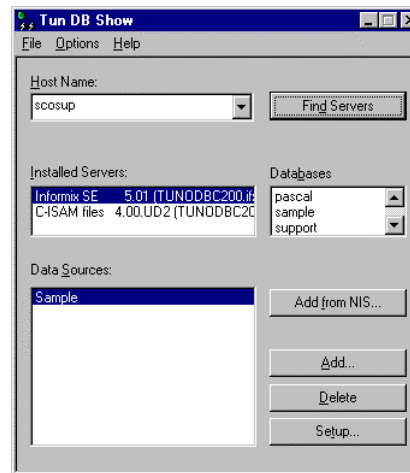
You use this utility to query a host on the network to see if one or more **Tun SQL** servers are present. Enter the name or IP address of the server in the **Host Name** field or select the server you want from the drop-down list (the list shows the servers declared in the **hosts** file, and on the NIS server).



To configure **Tun NIS**, please refer to the manual **TCP/IP Network Services** or **Tun NET**.

Click the **Find Servers** button.

If one or more **Tun SQL** servers are correctly installed on the remote machine, the **Installed Servers** list appears as follows (there's at least one line):



Each line contains the following information:

- The name of the DBMS that the **Tun SQL** server interfaces with.
- The version number of the DBMS.
- The name of the executable file that functions as the server (**tunodbc200.ora**, for example).

Note:

With some DBMSs (Informix On-Line, for example), selecting the server in the list displays the list of databases managed by the DBMS in the appropriate column (Databases).



If there are no **Tun SQL** servers in the list, it means that a problem occurred during installation. In this case, you must carry out all the operations and checks described in the previous chapters again.

➤ Parameters

Use the **Options** menu in **Tun DB Show** to:

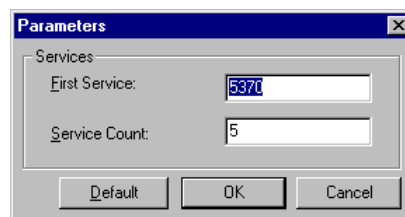
- Manage services.
- Enter the proxy server settings.

Managing services

A service number from 5370 is associated with each **Tun SQL** server process. For **Tun SQL** to function correctly, you must define the **First Service** and set the number of services possible. This enables **Tun SQL** to detect the different database systems accessible to a **Tun SQL** server. The default **First Service** is 5370 and the **Service Count** default is 5. The list of available services is given below:

- Oracle 5370
- Informix 5371
- Sybase 5372
- DB2/RS6000 5373
- Progress 6 5374
- Progress 7 5375
- C-ISAM 5376
- DB2 for MVS 5377
- Progress 8 5378

Choose **Options**→**Parameters...** from the main menu. The following dialog box opens:

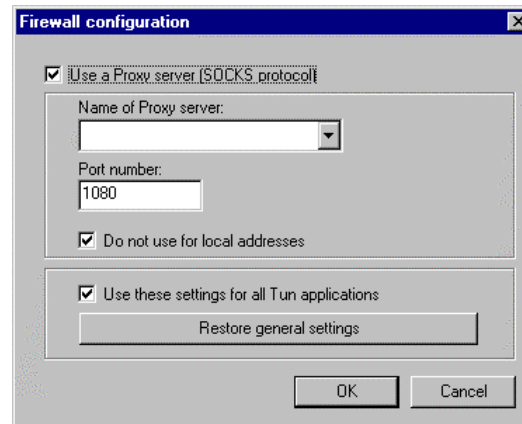


Using a proxy server firewall

When you configure a proxy server in **Tun DBRevamp**, access to an outside server goes through a proxy gateway machine.

To set the firewall parameters (IP address, port number, etc.), choose **Options**→**Firewall** from the main menu.

The following dialog box opens:



Select the **Use a Proxy server** check box.

Enter the name or IP address of the server. Only enter a name if you use a DNS. You can also choose one from the drop-down list (click the down arrow to the right of the field). The list contains the names of the servers listed in the server table (hoststab) and on the NIS server (NIS resources have yellow icons).

Also enter the SOCKS port number (usually the default value 1080).

To avoid using the firewall for local connections, select **Do not use for local addresses**.

The firewall configuration can be applied to all the **Tun** applications: To do that, select the **Use these settings for all Tun applications** check box. To apply the general configuration to all the **Tun** applications in use (after using a specific **Tun DBShow** configuration, for example), click **Restore general settings**.



Creating a database

To familiarize users with ODBC functionality in a client/server environment, **Tun SQL** is supplied with practical examples.

To use the examples supplied with the **Tun SQL** software package, a specific database must be created in one of the available DBMSs. You do this using the tools that come with the DBMS in question.

Given the difficulty inherent in creating a database in some DBMSs (Oracle), you can use an existing database provided that it doesn't contain sensitive data. The newly created database should be called **tunsqldemo** for a better understanding of the next part of the manual.

Creating a data source

➤ Introduction to data sources

ODBC-compliant applications have to recognize a data source before they can use a particular driver and database. **Data source** is a key term which refers to the name of the ODBC driver used (for example, **tunodb32.dll**) and the information required to make it function. This information is as follows:

- The name or IP address of the remote host.
- The type of DBMS (Oracle, Informix, Sybase, DB2, Progress, C-ISAM).
- The name of the database.
- Comments.
- Optional supplementary information.

➤ Creating a data source

To familiarize users with ODBC functionality in a client/server environment, **Tun SQL** is supplied with practical examples. The examples use the same data source (except the examples concerning database revamping).

You must create this data source before you can use the examples (see the manual "**Getting started with Tun**").

To create a data source, you can use:

- **Tun DB Show**.
- **ODBC DataSource Administrator** (Windows utility).

Creating a data source with Tun DB Show

Start **Tun DB Show** again and do the following:

- Enter the name or IP address of the server with the database for which you want to create a data source.
- Click **Find Servers** to display the **Tun SQL** servers installed on the host.
- Select the server for the DBMS with the database for which you want to create a data source.
- Click **Add...**



If **Tun NIS** is installed on the PC and the network administrator has configured the NIS tables, you can use the **Add from NIS...** button to access the data sources on the network. See the **TCP/IP Network Services** or **Tun NET** user guide for instructions on how to configure **Tun NIS**.

Now please read the section "**Configuring the data source**".

Creating a data source with ODBC Administrator

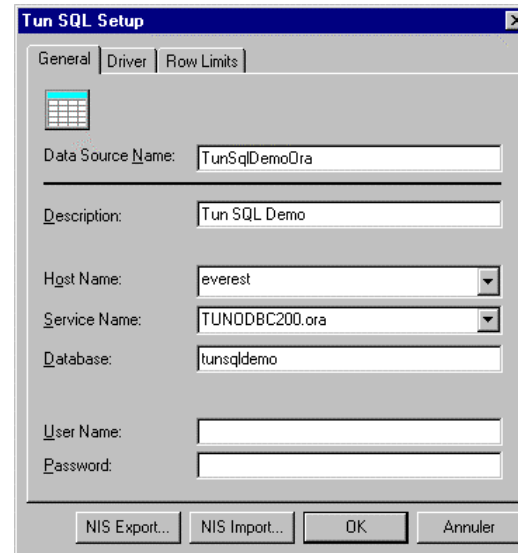
Open the **Control Panel** and click the **ODBC** icon (32-bit ODBC in Windows 95). Click the **Add** button in the dialog box that opens.

Select the ODBC driver **Tun32 Driver**.



Configuring the data source


Click the **Add** button in **Tun DB Show** or the **ODBC Administrator** to display the following dialog box:



➤ General tab

The first tab contains the following fields:

Data Source Name

The icon  represents the field containing the name of the data source as used by ODBC-compliant applications. For you to be able to connect to the sample database, it's imperative that the data source is called:

- **TunSqlDemoIfx** For Informix On-Line
- **TunSqlDemoIse** For Informix SE
- **TunSqlDemoOra** For Oracle
- **TunSqlDemoSyb** For Sybase
- **TunSqlDemoDB2** For DB2
- **TunSqlDemoPro** For Progress

Description

Enter an optional comment to associate with the data source.

Host Name

Enter the IP address or the name of the host on which the database you want to use is installed.

Service Name

Enter the name of the **Tun SQL** server process associated with the DBMS in which the required database has been created (for example, **tunodbc200.ora**).

If you're using a different TCP/IP stack from **TCP/IP Stack**, you should complete the **services** file in the TCP/IP software used with the following values:

tunodbc200.ora	5370/tcp	# Tun-SQL ORACLE
tunodbc200.ifx	5371/tcp	# Tun-SQL INFORMIX
tunodbc200.syb	5372/tcp	# Tun-SQL SYBASE
tunodbc200.db2	5373/tcp	# Tun-SQL DB2
tunodbc200.pro	5374/tcp	# Tun-SQL PROGRESS

The other services are given below (not used in the example):

tunodbc200.pro7	5375/tcp	# Tun-SQL PROGRESS7
tunodbc200.ism	5376/tcp	# Tun-SQL C-ISAM
tunodbc200.mvs	5377/tcp	# Tun-SQL DB2/MVS
tunodbc200.pro8	5378/tcp	# Tun-SQL PROGRESS8

Database

Enter the name of the database you want to use. To use the sample database, enter **tunsqldemo** (the sample database supplied with **Tun SQL**).

User Name

Enter the name of a user authorized to access the database.

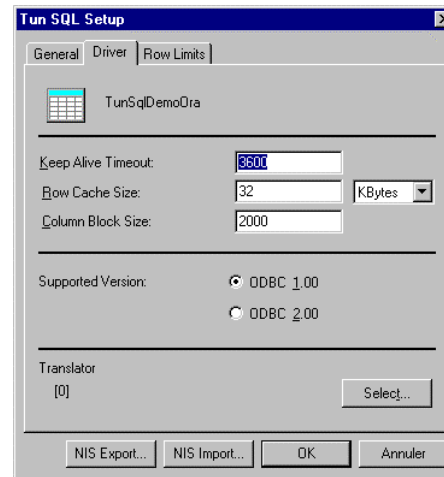
Password

Enter the password associated with the user.



➤ Driver tab

Click the **Driver** tab to display the ODBC driver configuration window:



Keep Alive Timeout

Since the PC is liable to software and hardware failures, the **Tun SQL** server has to regularly check that the PC is still under power. To do this, it regularly sends packets to the PC. If the PC doesn't reply in a time of **n** seconds, the process stops. Enter the timeout value (the default is 1 hour).

Row Cache Size

Indicates the size of the data packets extracted from a table during an SQL "select" operation. The value can be expressed in kilobytes or number of lines.

If the value equals 1, then one TCP packet per row is retrieved. If the value equals 100 then the rows are grouped in packets of 100 up to the number of rows actually retrieved. This field lets you optimize exchanges on the network. The optimum value is between 50 and 150. The default value is 32 Kb.

Column Block Size

Indicates the fragmentation unit to be used when very wide columns have to be retrieved from the database (large amounts of text or images). If this value is too low, the number of exchanges on the network are considerably increased.

Supported version

Currently, there are two versions of the ODBC API, numbered 1.00 and 2.00. Some applications are only compatible with ODBC version 1.00 (Microsoft Access) and don't work with drivers from a higher version. The **Tun SQL** ODBC driver, which is compatible with version 2.00, can emulate version 1.00 to allow these applications to run.

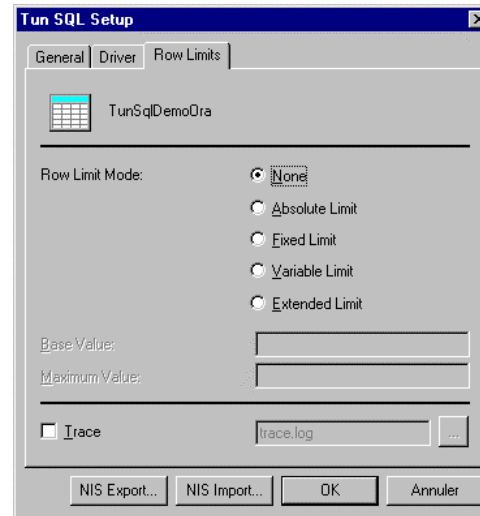
Select the appropriate check box to indicate to the ODBC driver the level of compatibility required for a given application.

Translator

Due to the differences in accented character notation in the Windows (CP850) and UNIX (ISO8859) environments, it's sometimes necessary to set up character conversion tables for the ODBC driver. The **Select...** button enables the user to choose the required conversion table. You can ignore this field in the sample database since the text it contains is in English.

<p>Note: You can use Tun DB Map to create or edit the conversion tables.</p>
--

Click the **Row Limits** tab to display the limits dialog box:



Row Limit Mode

Some office applications let you compose your own SQL requests. In other cases, applications read the contents of an entire table before displaying them on the screen. This doesn't pose any particular problems when they're dealing with small tables in a local database.

On the other hand, insurmountable problems can arise when it's a question of very large tables in a centralized, remote database. In this case, there's a considerable number of exchanges on the network and there may be insufficient memory in the PC to store the received data. The PC needs to be frequently rebooted after such a "select" query.

To compensate for this problem, the **Tun SQL** ODBC driver incorporates the notion of limits. You define the limits on the **Row Limits** tab.

Five types of limits are recognized by the **Tun SQL** ODBC driver:

No Limit	No limit is imposed by the ODBC driver.
Absolute Limit	The ODBC driver refuses to load more than n rows during one select request. No messages are displayed to inform the user.
Fixed Limit	The ODBC driver refuses to load more than n rows during one select request. A message will be displayed informing the user.
Variable Limit	The ODBC driver refuses to load more than n rows during one select request. However, it displays a message proposing to load more, within the limit of the Maximum Value .
Extended Limit	The ODBC driver refuses to load more than n rows during one select request. However, it displays a message proposing to load more, with no Maximum Value. In this case, the message displayed is really only a warning.

Trace

You can select the **Trace** check box to save a record of your SQL queries in a ".log" file that you can later consult. This option lets you see what the ODBC driver does when you pass it an SQL query.

Click the button ... to choose the directory where you want to put this file.

NIS



If **Tun NIS** is installed on the PC and the network administrator has configured the NIS tables, you can click the **Import NIS...** button to access the data sources on the network. See the manual **TCP/IP Network Services** (if you don't have **Tun NET**) for instructions on how to configure **Tun NIS**, or the chapter "**The NIS Browser**" in the **Tun NET** or **TCP/IP Network Services** manual



Notes:

In this chapter you were asked to create a data source called **tunsqldemoXXX** which is used by the examples supplied with **Tun SQL**.

To use **Tun SQL** with other applications and databases, you must create the required data source for each case. Typically, there must be an individual data source for each application and database used.

You can create a data source directly using the Windows ODBC application from the Control Panel.

Transferring the demonstration database

Tun SQL is delivered with a test database for use with the examples included. This database must be downloaded from the PC to the data source **tunsqldemoXXX** that you were asked to create in the preceding sections.

Note:

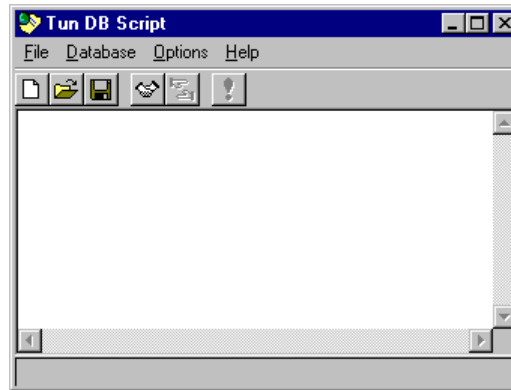
The variable **XXX** in the name of the data source may take one of the following values:

- **Ifx** For Informix On-Line
- **Ise** For Informix SE
- **Ora** For Oracle
- **Syb** For Sybase




To perform the downloading, run the program by clicking the **Tun DB Script** icon in the **Data Access** group (menu **Start, Programs, Esker Tun** under Windows 95/98/2000 and Windows NT).

When the application starts, the following window appears:




➤ Loading the SQL batch file to create the database

You can load the database of your choice by choosing **File→Open** or clicking the button .

To download the example database, you must load the file **\\Demo\Db\XXXCREAT.SQL** from the **Tun SQL** installation directory.

➤ Connecting with the data source


Before you can run the SQL batch file, you must establish a connection with a data source. To do this, click the button  or choose **Database→Connect**.

This operation displays a dialog box that asks for the name of the data source to use. If the connection can be established, the batch file is run.


To download the example database, select the data source **TunSqlDemoXXX** which was defined earlier.



➤ Execution

To execute the SQL batch file, choose **Database→Execute** in the main menu or click the button . **Tun DB Script** then submits the SQL commands consecutively to the database which corresponds to the selected data source. If there's an error, **Tun DB Script** stops running and displays an error message.

➤ Disconnecting the data source

After execution, you must disconnect the data source by clicking the button  or by choosing **Database→Disconnect** from the main menu. A disconnection of the data source also occurs when you quit the application.

Note:

Although you can use **Tun DB Script** to download the **Tun SQL** example database, it also has other uses. In fact, **Tun DB Script** can execute whole lists of SQL commands to create other databases or update or completely clear extremely large tables.

Creating a virtual data source

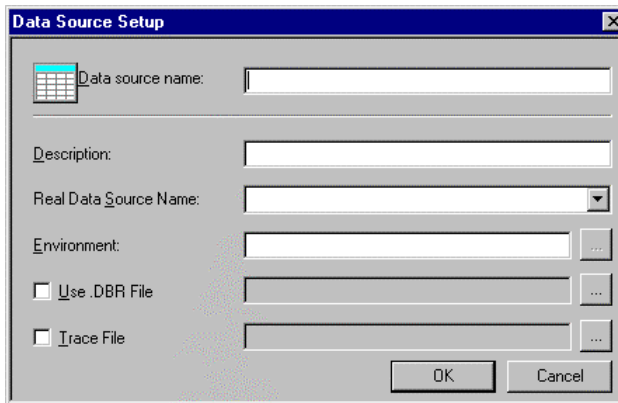
Tun DB Revamp can link virtual tables, adapted to the end user's environment, to a real database. See the section "**Database Revamping**" for more information on that application.

When you create a virtual (revamped) database, you can create data sources for it, just as if it were a real database. Users can then use the virtual ODBC driver supplied with **Tun SQL** to access the virtual database you've created especially for them.

A virtual data source can be considered as the association of a real data source and an environment.

To create a virtual data source, access the configuration of the virtual data source in **ODBC Administrator**: Choose the **Tunmap32** driver. See "**Creating a data source**".

The following dialog box opens:



Data source

Enter the name of the revamped data source.


Description

Enter a description of the data source for identification purposes.

Real Data Source Name

Enter the name of the data source for the real database the virtual database is derived from.


Environment

Enter the name of the environment for which you're creating a virtual data source. An environment is the set of virtual tables: Each revamped database can have one or more environments. Click the button  to choose the environment from the list of environments defined in the database.

Local .DBR file


Instead of entering a real data source name, you can select the environment from a local **.dbr** file. See the section "**Database Revamping**" for information on the use of this type of file.



Select the **Local .DBR file** check box. Enter the full path name of the file or click the browse button  to select the ".dbr" file. Then choose the environment (**Virtual source** field).

Trace File

To keep a trace of your SQL queries, select the **Trace File** check box. A record is saved in a log file that you can consult with a text editor. This lets you see what the ODBC driver does when you pass an SQL query to it.

Click the browse button  to choose the directory where you want to save the log file and also name this file.

Character conversion tables

This section can be skipped on the first reading.

➤ Difference in notation between the different computing systems

Although all the characters used in English have been perfectly codified in the ASCII table (0 to 127), this isn't the case for the special characters or accents used by other languages (for example, French, German, Spanish, Italian, etc). Despite the fact that norms exist (for example, ISO 8859), they're not always implemented in every computing system.

As **Tun SQL** enables a computing system (the PC) to access data located in another computing system (a DBMS operating on the UNIX platform), Esker has included in **Tun SQL** a function to handle the differences in notation between the national characters in different systems. This function enables the PC to display an accented character (an é for example) even if it is coded differently in the UNIX DBMS.

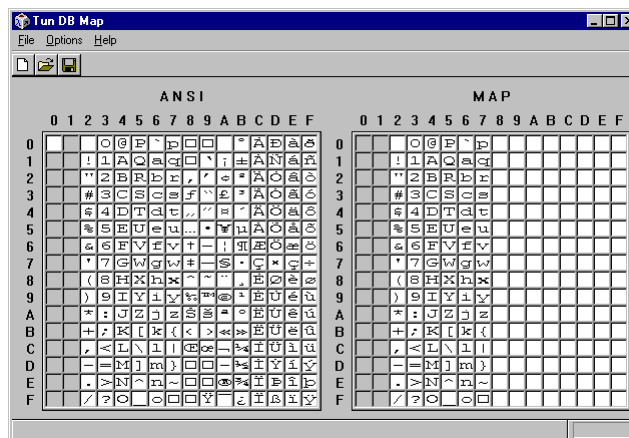
➤ Creation of conversion tables

The conversion function supplied with **Tun SQL** uses "conversion tables" that can be created or updated with the application **Tun DB Map**.



Run the program by clicking the **Tun DB Map** icon in the **Data Access** group (menu **Start, Programs, Esker Tun** under Windows 95/98/2000 and Windows NT).

The following screen is displayed:



The table on the left shows all the characters available on the PC (ASCII and extended ASCII). The table on the right shows the same characters but in positions that correspond to the notation used by the DBMS on the UNIX machine. The first 128 positions in the table on the right are already filled since they are the same in both systems.

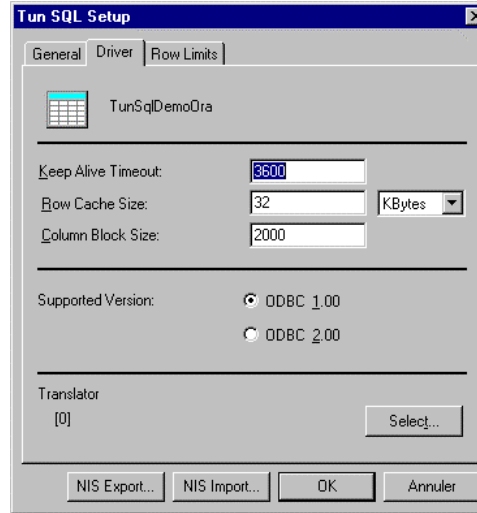
To assign a character to one of the positions in the right-hand table, select a character in the left-hand table and "drag" it across to one of the squares in the right-hand table, with the mouse button pressed.

You can create as many conversion tables as you need. Choose **File→Save** from the general menu. Save the files with the extension ".tnt".

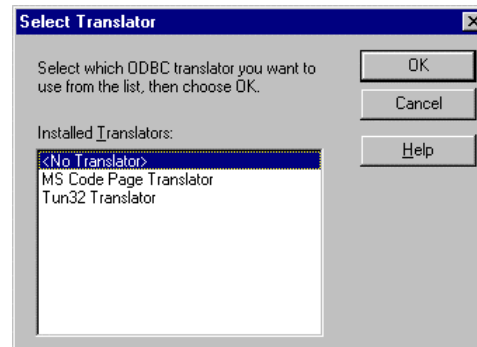


➤ **Implementation of conversion tables**

For conversion tables to be taken into account, they have to be associated with a data source using the dialog box displayed by **Tun DB Show** to this effect. Enter the details in the **Translator** section:



Click **Select...** to display the following dialog box:



Select the ODBC translator you want to use and click **OK**.

C-ISAM

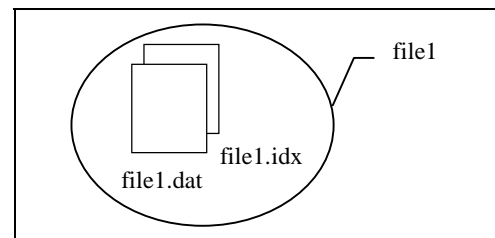
Introduction to C-ISAM

➤ The C-ISAM file system

C-ISAM (Indexed Sequential Access Method) is a library of C functions developed by Informix. It allows the management of indexed sequential files (file creation, and insert, delete and read operations). C-ISAM includes other features like file locking and transaction support which ensure data integrity. These features guarantee that data is accessible, valid and correctly used.

C-ISAM uses data types similar to those used in C. Since C-ISAM implements these types independently of the UNIX system used, the way it stores the data can be different than the way the data is represented at run time. C-ISAM includes conversion functions to convert run-time data formats to storage data formats.

A C-ISAM file is really a combination of two files: One contains data (file.dat) and the other an index for finding the data in the data file (file.idx). The two files are always used together as one logical C-ISAM file.



➤ RDBMSs and C-ISAM

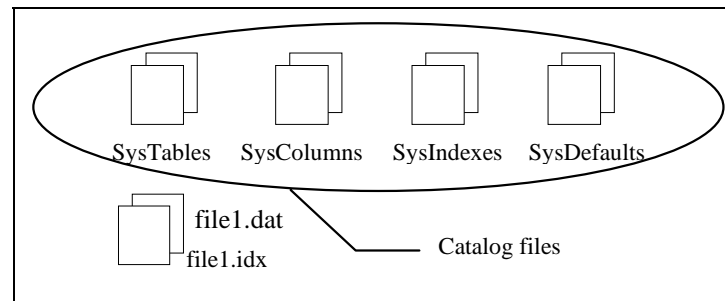
As C-ISAM files use indexed sequential access, developers must understand the file structure and use the index files to access data.

Building a database system using C-ISAM files frees developers from these constraints by incorporating an indexed file structure as tables, columns, and indexes in a catalog.

➤ Tun SQL C-ISAM

C-ISAM depends on C functions to query and update sequential files. The C-ISAM driver shipped with **Tun SQL** lets you view sequential files as a standard relational database with tables, fields and keys. You can then use standard SQL instructions to query or update a database created with C-ISAM files. The C-ISAM driver C-ISAM translates the instructions into C functions that perform the necessary operations on the sequential files.

To go from a sequential data view to a relational database view, the C-ISAM driver adds descriptive database files known as the **catalog** to the standard C-ISAM data and index files. These are C-ISAM type files (**.dat** data file and **.idx** index file): SysTables, SysColumns, SysIndexes and SysDefaults.



sqltools is a UNIX tool for creating and managing databases built on C-ISAM. It works on the same principle: SQL instructions are used to construct the database and are translated into C functions before they can be read by the C-ISAM file system.



➤ Installing the C-ISAM driver

To install the **Tun SQL C-ISAM**, refer to the "**Tun Installation Guide**".

Using sqltools

➤ Using sqltools

You can use **sqltools** on-line or from a semi-graphical interface (windows, menus).

Connect to the UNIX server containing the C-ISAM files. We recommend you create a user login ID to access the C-ISAM files.

Change to the **sqltools** installation directory and run the application by entering the following command:

```
sqltools  
to run the application on-line,  
or  
sqltools -v  
to use the graphical interface
```

➤ Creating the database

The first step is to create the database containing the data from the C-ISAM files. To do that, use one of the following methods:

- Choose **Database→Create** and enter the name of the database you want to create. .
- Enter the following statement in the lower pane (Input window):

```
create database "databasename";
```

This statement creates a directory with the name of the database and the extension **.ism** in the directory indicated by the variable **ISAM-PATH**. See "**DBMS Installation Settings**" in the "**Tun Installation Guide**" for more information on the **ISAM-PATH** variable. "

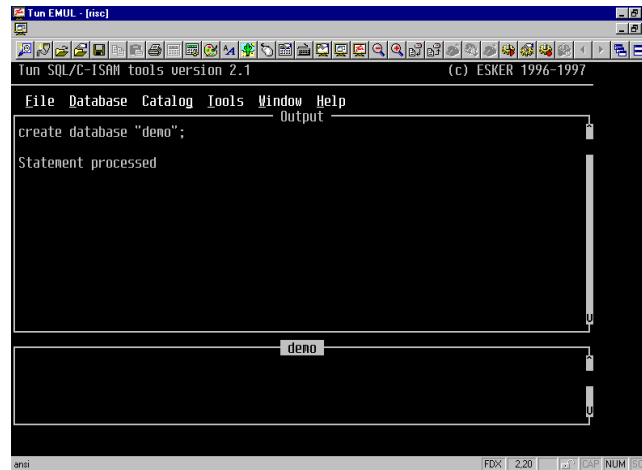
Example:

create database "demo";

*creates the directory demo.ism in the directory /TunSql/bases if ISAM-
PATH=/TunSql/bases.*

This directory contains the C-ISAM .dat and .idx files that describe the database: SysTables, SysColumns, SysIndexes and SysDefaults, a total of 4 logical C-ISAM files and 8 operating system files.

The lower window (Input window) takes the name of the database:



Note:

You can run shell commands directly from **sqltools**. Enter the command preceded by an exclamation mark and followed by a semi-colon in the lower pane (Input window).

Example:

!ls -a ;

To enter character strings in uppercase, use quotes.

Example:

!ls "/TunSql/locisam";

➤ **Connecting to an existing database**

If you're using an existing database, you can perform operations on the tables after connection.



To do that, choose **Database→Connect** and enter the name of the database. The lower pane (Input window) takes the name of the database you connect to.

➤ Creating tables

Once the database is created (the **.ism** directory contains the table's descriptive files), you can then create tables in it. .

You can perform this step using a pair of existing C-ISAM files (the **.dat** data file and the **.idx** index file), or create new ones. The statement used in the second case is different, as are the precautions you must take: Use the **create table** statement to create the pair of C-ISAM files at the same time as the table, and the **define table** statement to create a table from a pair of existing C-ISAM files.

Creating tables from scratch

To create the pair of C-ISAM files at the same time as the table, enter the following statement in the lower pane (Input window), which now has the database name as a title:

```
create table tablename (field1 type1, field2  
type2,..., primary key(field1));
```

This statement creates a table with the name "tablename" in the database. The table contains the fields field1, field2, etc. with the types type1, type2, etc. See the list of types.

The C-ISAM data and index file pair are created in the database directory (**databasename.ism**). The names of these files use the first seven characters of the table name and a unique identification number that's attributed automatically: The extension **.dat** is added for the data file and **.idx** for the index file. If the table name is less than seven characters, these file names are padded out with underline characters ("_").

Example:

```
create table table1 (field1 longint, field2 char(25),  
filler char (30), primary key(field1));
```

*creates the files **table1_100.dat** and **table1_100.idx**. The table contains the field **field1** containing a **longint** type, the field **field2** containing a maximum of 25 **char** types, and the field **filler** containing a maximum of 30 **char** types. The primary key for this table is **field1**.*

Creating tables from an existing pair of C-ISAM files

To create a table in a database for which the C-ISAM **.dat** and **.idx** file pair already exists, enter the following statement in the lower pane (Input window) which has the database name as its title:

```
define table tablename file is filename (field1  
type1, field2 type2,..., primary key(field1));
```

This statement creates a table with the name "tablename" from the existing files **filename.dat** and **filename.idx**.

Important note:

Before using the table (for example, before you use **select** statements on the table), you must copy the files specified by the **file is** keywords into the database directory.

The **define** statement, however, can be executed even if the files haven't yet been copied to this directory. It's even recommended **not** to copy the files until you execute the **create index** statement, if you want to create an index for the table.

➤ Creating an index

To create an index for one through eight columns in a table, enter the following statement in the lower pane (Input window), which has the database name as a title:

```
create unique index indexname on tablename (field1,  
field3);
```

This statement creates the index "indexname" for columns **field1** and **field3** in the table "tablename".

➤ C Structures

Each statement passed to **sqltools** is translated into C code for the C-ISAM file system. To view the corresponding C structure for a table, choose **Catalog**➔**GetCStruct**.



For example, the statement that creates **table1** produces the following C structure:

```
struct root_table1
/* file "table1_100" */
{long          lint_field1; /* field1 longint */
 char          chr_field2[25]; /* field2 char(25)*/
 char          chr_filler[30]; /* filler char(30)
 */
 unsigned char null_flags[1];
 /* reserved */
};
```

In this example, there's a reserved field (an array of **unsigned chars**) one byte long. This field is specific to the C-ISAM management system. When a table is created with the statement **create table**, **sqltools** automatically adds this reserved field to the table.

➤ Validating a table created from existing C-ISAM files

When you create a table from an existing pair of C-ISAM files (with the **define** statement), you must be careful that the table structure you create conforms to that of the C-ISAM files.

For example, you create the table "table2" based on the C-ISAM files **filename.dat** and **filename.idx**, written in C. These files have the following record structure:

- A field composed of a longint variable,
- A field composed of an array of 25 char variables,
- A field composed of an array of 30 char variables.

If you define table "table2" as follows:

```
define table table2 file is table1_100 (field1
longint, field2 char(25), filler char(25));
```

you create a table whose records have the structure:

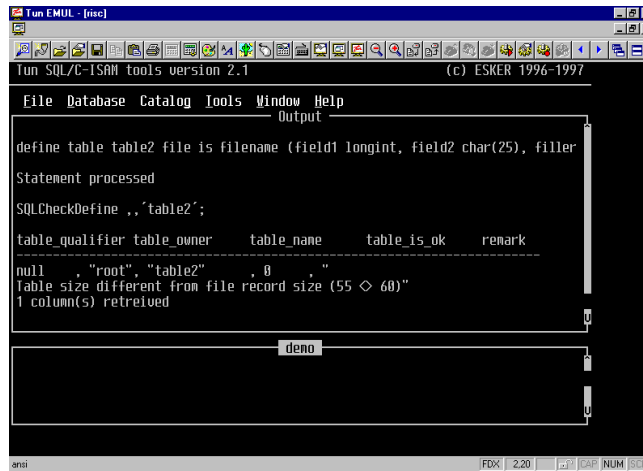
- A longint type field,
- Two char array fields of length 25,

which doesn't correspond to the structure generated by the C-ISAM files the table is based on.

In this case, there's a mismatch between the created table and the C-ISAM files it's based on.

To verify that a table's data structure corresponds to the original C-ISAM files, choose **Catalog** → **Check Define**. To verify all the tables in the database, choose **Tools** → **Check Catalog**.

In our example, the Output window shows the following results:



```
Tun SQL/C-ISAM tools version 2.1 (c) ESKER 1996-1997
File Database Catalog Tools Window Help
Output
define table table2 file is filename (field1 longint, field2 char(25), filler
Statement processed
SQLCheckDefine ,, 'table2';
table_qualifier table_owner table_name table_is_ok remark
-----
null "root" "table2" 0 "
Table size different from file record size (55 <> 60)"
1 column(s) retrieved
deno
FDX: 2:20 SCAN NUM
```

The following message is displayed:

```
Table size different from file record size (55 <>
60)"
```

The message warns that table **table2** was defined differently than the C-ISAM "filename" files it's based on. The **define** statement should be as follows:

```
define table table2 file is filename (field1 longint,
field2 char(25), filler char(30));
```



➤ Viewing a table's C structure

You can view a table's C structure, that is, the different columns created (name, data type and length), as well as the information concerning data management (the primary keys, for example).

To do that, choose **Catalog**➔**GetCStruct** and enter the name of the table whose C structure you want to view. To view the C structure of all the tables in the database, choose **Tools**➔**List Structures**.

The example below shows the result of this command for a table created with a primary key:

The following statement creates the table:

```
create table customer
(cust_number longint,
 cust_name char(20),
 cust_address1 char(20),
 cust_address2 char(20),
 filler char(20),
 primary key (cust_number)
);
```

The C structure generated is:

```
struct doc_customer          /* file "custome110" */
{long   lint_cust_number;   /* cust_number longint */
 char   chr_cust_name[20];  /* cust_name char(20) */
 char   chr_cust_address1[20];/* cust_address1 char(20) */
 char   chr_cust_address2[20];/* cust_address2 char(20) */
 char   chr_filler[20];     /* filler char(20) */
 unsigned char null_flags[1]; /* reserved */
};

struct keydesc idx_customer_1;
idx_customer_1.k_flags = ISNODUPS;
idx_customer_1.k_nparts = 1;
idx_customer_1.k_part[0].kp_start = 0;
idx_customer_1.k_part[0].kp_leng = 4;
idx_customer_1.k_part[0].kp_type = LONGTYPE;
```

The first piece of code shows the structure of the table (the fields), the second shows the primary key and its various assignments.

➤ Catalog information

You can obtain information on the catalog using the **Catalog** menu. The menu options provide information on:

- **TypeInfo:** Each data type is identified by a number, as defined by the ODBC standard. Enter the number for the data type you want to verify, or enter 0 to view the data type list.
- **Tables:** You can query the catalog for information on its tables, using the name of the user who created the table, the table name itself, or the table type (system table, synonym, etc.). Enter the % character to include all the tables or table types in your query.
- **Columns:** You can query the catalog for information on its columns, using the name of the user who created the table, the table name itself, or the column name. Enter the % character to include all the tables or columns in your query.
- **Statistics:** You can obtain statistics on the data in the catalog.
- **PrimaryKeys:** You can query the catalog for a table's primary keys, using the name of the user who created the table or the table name itself. Enter the % character to include all the tables in your query.

For more information on the **Catalog** menu options, refer to the catalog system section in the ODBC standard reference manual.

➤ Editing a table

The length of records in a table is set when the table's created. Consequently, you can't add or delete records if the total length is affected.

If you want to change the structure of a table, you must first delete the table respecting the precautions described in "Deleting a table".

➤ Deleting a table

There are two ways to delete a table from the catalog:

You created the table using the statement **create table**: Use the statement **drop table** to delete it. **Note:** This statement removes all references to the table from the catalog files and deletes the pair of C-ISAM files linked with the table.



You defined the table using the statement **define table**: Use the statement **undefine table** to cancel the **define** statement. This statement only removes all references to the table from the catalog files.

Note:

You can use the **drop table** statement for a table defined by **define table**. However, the **drop table** statement deletes the pair of C-ISAM files specified by the **file is** keywords if they're in the database directory. You should, therefore, be careful using the **drop table** statement if you created the table from existing C-ISAM files.

If you want to keep the pair of C-ISAM files associated with a table you delete using the **drop table** statement, you must first copy these files to a different directory. That way, after deleting the table, you can use these backup copies.

➤ Maintaining the C-ISAM files

When you create tables from existing C-ISAM files (using the **define table**), you must copy these files into the database directory if you want to use the tables.

However, if these files are used and updated in other applications, it's useful to be able to use the updates in your C-ISAM database. To do that, you must create symbolic links with the existing C-ISAM files from the database, rather than make copies.

Example:

*You create the database **dbtest** in the directory **/TunSql**. For the tables in this database, you use the files **filename.dat** and **filename.idx**, which are in the directory **/data** and are used by other applications.*

*You create symbolic links to these files in the directory **/TunSql/dbtest.ism** (the database directory) with the following command:*

```
ln -s /data/filename.* /TunSql/dbtest.ism
```

➤ Deleting a database

To remove (delete) a database, choose **Database→Drop** and enter the name of the database to remove, or enter the following command in the lower pane (Input window):

```
drop database databasename
```

➤ Saving results

You can save the results displayed in the upper pane (Output window) to a text file with the extension **.res**.

To do that, choose **File→Save as...** and select the location and the save file name.

➤ Running a script

You can use the **File→Execute** option to execute an SQL script on condition that you use the SQL statements supported by **sqltools**.



PART 2
DATABASE REVAMPING

REVAMPING

Virtual databases

Most of today's structured data storage facilities consist of Relational Database Management Systems (RDBMS). Databases make it possible to store the corporation's data resources which can then be updated using special applications. The mass of data assembled in this way is also of interest to a large number of users who need to extract information for their work (performance indicators, statistics, expert systems). The SQL language is used to update and query databases.

However, database structure, an essential part of the information system, can complicate access to database information throughout the corporation:

- There's a considerable number of tables and records in databases, too many in fact for the average end-user who is often only interested in a part of the data.
- Database structure is always complex and the user needs a lot of experience to find his way around.
- The computing environment of databases isn't very user-friendly. For example, the names of the tables and records are seldom expressed in straightforward terms.
- Data manipulation requires prior knowledge of the SQL language to query databases and obtain the desired results.

Several breakthroughs have been made which reduce these obstacles and facilitate access to databases (for example, the inclusion of graphical interfaces in database query tools).

The next step is to free the end-user totally from the pre-requisite of technical database knowledge by presenting him only the information he needs in the most suitable form for his work environment.

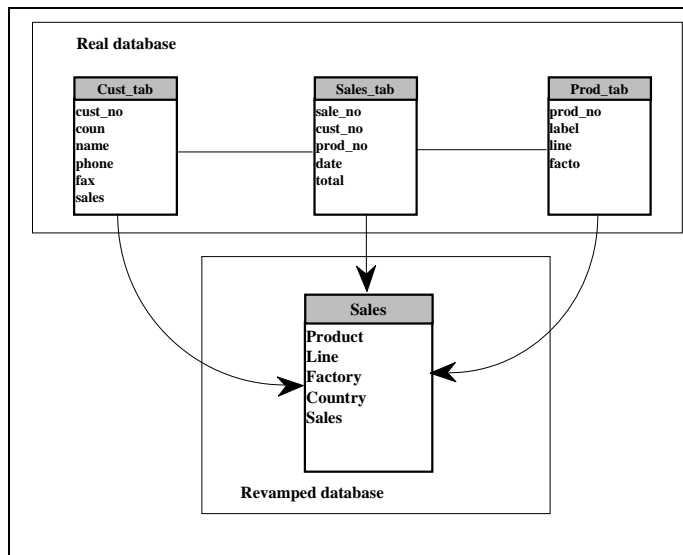
The consequences of this are:

- Improved productivity: The end-user becomes autonomous in his use of data, and analysis and decision-making take less time because they're more simple.
- More relevant information: Using only the data he needs, the user sharpens his capacity for analysis and synthesis, and refines the results.

➤ Revamping

Revamping consists of constructing a virtual database that's adapted to the user's environment from the existing database. Although it doesn't exist as a real database, the new structure is seen by the user as a normal database whose tables and fields correspond exactly to his needs: The database only contains the information the user really needs for his analyses, in a form which suits his requirements (understandable data names, predefined functions).

The redefined database is perfected by an administrator who adapts the tables and fields from real databases. For example:



In the above example, the real database contains three tables: "Cust_tab" (customer table), "Sales_tab" (sales table) and "Prod_tab" (product table).

The administrator defines a virtual table that presents the results of sales per product, per product line, per production site and per country.

The virtual table entitled "Sales" contains the following fields:

- Product (real field: "prod_tab.label»)
- Line (real field: "prod_tab.line")
- Factory (real field: "prod_tab.fact")
- Country (real field: "cust_tab.coun")
- Sales (real field: "sales_tab.total")

The virtual table constructs a join between the "Prod_tab" and "Sales_tab" tables using the common field "prod_no", and a join between the "Sales_tab" and "Cust_tab" tables using the common field "cust_no".

Revamping in Tun SQL

Tun SQL uses two components to administrate and use virtual databases:

- The database administrator **Tun DB Revamp** which manages the revamping.
- The virtual ODBC driver which allows the user to access the revamped database.

➤ The DB Revamp administrator

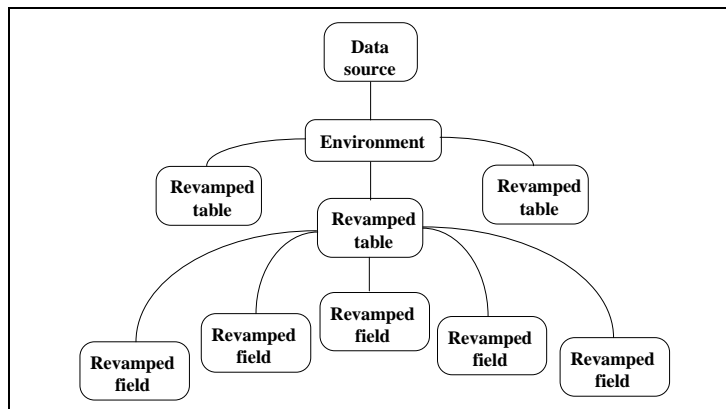
The objective of the **Tun SQL** virtual database is to offer the end-user contextually redefined information for a particular "environment".

With the intuitive graphic interface, the administrator can define as many "environments" for different users or types of user as he wants. The environment is based on "occupation": An accountant need only see the tables related to accountancy, sales staff the tables related to their jobs.

Each environment can be viewed as a special data source by the ODBC front-end which makes the query (refer to the architecture diagram in the section "**Virtual ODBC driver**").

The virtual database model is as follows:

- One or more environments, defined from a real data source, which contain a selection of tables from the real database. The selection depends on the end-user's needs.
- The tables defined in an environment are either native tables from the real database or the result of joins between two or more tables (concept of view).
- Each table only contains the fields required by the end-user.
- The revamped fields are either existing fields from real tables or recalculated fields that simplify the end use of the database.
- The revamped tables and fields can be given names that are clearer for the end-user (for example, "Cust_tab" can be renamed "Customer Table" and "Cust_no" "Client Number").



The tables redefined in an environment don't physically exist in the database. However, the revamped database is stored in an indexed form in three supplementary tables created in the real database:

- An environment table containing a list of environments in the form "environment name" and "description".
- A table containing a list of redefined tables. Each of the redefined tables is indexed by a name, a description and the name of the environment to which it belongs.



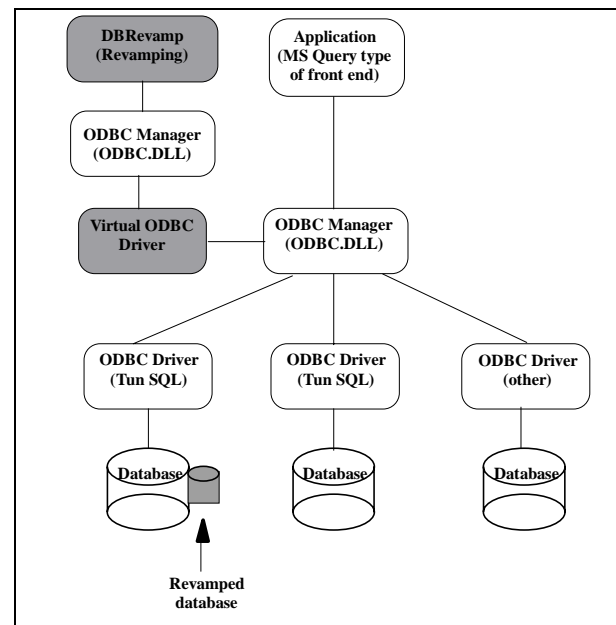
- A table of redefined fields. Each of the redefined fields is indexed by a name, a description, an origin (existing field, processed data, data concatenation) and the name of the virtual table to which it belongs.

After a revamping operation, a database always contains these three extra tables.

➤ Virtual ODBC Driver

For the end-user, querying a virtual database is similar to querying a real database in read-only mode. This openness is due to the inclusion of a special virtual database ODBC driver in **Tun SQL**.

When the ODBC manager (ODBC.DLL) receives queries from an environment, it transmits them to the virtual ODBC driver. The virtual ODBC driver then translates them into appropriate queries for the real database. Next, the virtual ODBC driver passes the translated queries back to the ODBC manager which directs them to the real database's normal ODBC driver.



TUN DB REVAMP GENERAL USE

General options

➤ Choosing the working language

To choose the interface language, choose **?→Language**" and select the language you want to work in.

➤ Modifying the display



To modify the controls displayed in the main **Tun DB Revamp** window:

- Select or cancel the **View→Toolbar** option to display or hide the toolbar.
- Select or cancel the **View→Status Bar** option to display or hide the status bar.
- Select or cancel the **View→Property Box** option to display or hide the object properties bar.

➤ Copying an object


Use one of the following methods to copy an object:

1. To use the **"drag and drop"** method, select the object to be copied, and drag it to the place you want to copy it to, with the mouse button held down.
2. Choose **Edit→Copy** from the main menu to copy the selected object, and then **Edit→Paste** to paste it in the desired location.
3. Choose **Copy** and **Paste** from the context menu (displayed by clicking the right mouse button) to copy the selected and object and paste it in the desired location.

4. Use the keyboard keys **Ctrl-C** (copy) and **Ctrl-V** (paste) to perform the operation.
5. Use the toolbar buttons,  (copy) and  (paste).

➤ Deleting an object

To delete an object, select it by clicking it and then do one of the following:

1. Choose **Edit→Delete** from the main menu.
2. Choose **Delete** from the context menu.
3. Use one of the keyboard's **Del** keys.
4. Click the button  in the toolbar.

➤ Renaming an object


To rename an object, first select it and then use one of the following methods:

1. Use the **General** tab in the property box.
2. Use the **F2** function-key on the keyboard and replace the old name with the new one.
3. Click the object again and proceed as for method 2.

➤ Saving changes

To save changes made to property values, press **Enter** with the cursor situated in the relevant dialog box, or click **Apply**.

➤ Obtaining help


To access the on-line help or obtain more information about **Tun DB Revamp**, choose **?→About DBRevamp**, or use the toolbar button .

➤ Quitting Tun DB Revamp

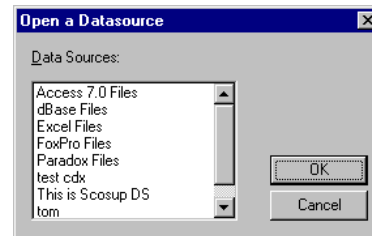
To quit the application, choose **File→Exit**.



Importing data source environments

To redefine (revamp) a real database, you have to select a corresponding data source. You do this by choosing **File→Import...** or else by clicking the button  in the toolbar.

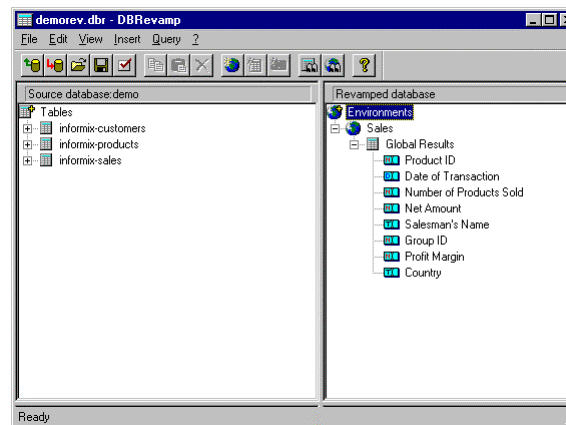
The following dialog box opens:



It shows a list of the data sources declared on the PC. To create a data source, see "**Creating a data source**". Since virtual data sources, obtained by revamping a real database, can't be redefined, they don't appear in this list. They do, however, appear in the list of data sources made available to the end-user in any Windows application that uses them (for example, **Microsoft Query**).

Select the data source you want to use.

A **Tun DB Revamp** window similar to the following appears:




The real database tables appear in the left pane of the window.

If the database in question hasn't been revamped with **Tun DB Revamp**, the right pane contains an empty environment named "New Environment". This is the first environment you can configure.

On the other hand, if you've already revamped the database with **Tun DB Revamp**, (in which case, it's a question of updating the virtual database), the right pane contains a list of the environments already created and their contents.

Creating an environment




To define a new environment for the selected data source, select the environment root (called "Environments") and choose **Insert→New Environment** from the main menu. You can also click the toolbar button .

Enter a name and, optionally, a description for this environment.

Creating a virtual table



To create a virtual table in an environment, select the environment and do the following:

- Choose **Insert→New Table** from the main menu or choose **New Table** from the environment's context menu. You can also click the toolbar button .
- Choose **View→Property Box** to display the property box of the newly created table (if it's not already displayed).
- On the **General** tab in the **Property Box**, enter a name and optional description for the table. You can also use the F2 function-key to rename a selected table.



If you want the virtual table to contain all or part of a real table, you can copy the real table into the environment of your choice: All the fields in the real table are also copied. To do this:

- Use one of the methods described in the introduction (**drag 'n drop**, **Copy/Paste**, keyboard shortcut or toolbar button) to select the real table in the source database and copy it to the target environment.
- Delete the fields you don't require from the virtual database or change them as described in the section "**Creating a field**".
- If you want to, you can change the names of the objects copied (tables and fields), and add a description to them on the corresponding **General** tab.

Creating a field



In a virtual table, you can:

- Insert an existing field from a real database, without changing its definition.
- Create a new virtual field from the real database fields.

➤ Existing fields


You can copy an existing field from a table in the real database directly into the virtual table. To do this:

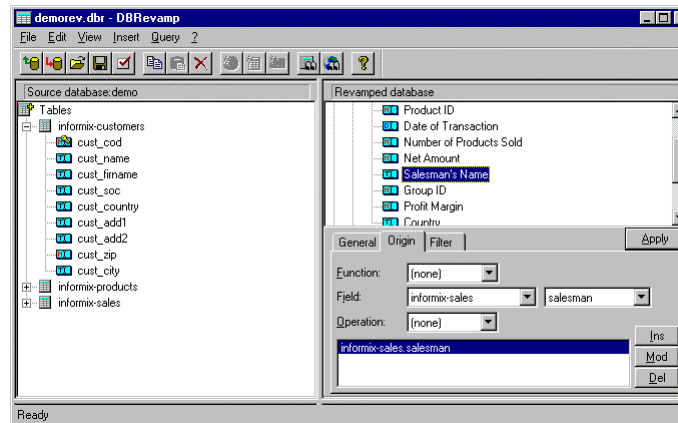
- Use one of the methods described in the introduction (**drag 'n drop**, **Copy/Paste**, keyboard accelerator or toolbar button) to select the field in the real database table and copy it to the virtual table of the redefined database.
- If you want, you can change the name of the field and give it a description on the corresponding **General** tab (or else use the **F2** function-key).



➤ New field

To define a new field in a virtual table, select the virtual table and do the following:

- Choose **Insert**➔**New Field** from the main menu or choose **New Field** from the table's context menu. You can also click the toolbar button .
- Choose **View**➔**Property Box** to display the property box of the newly created field (if it's not already displayed).
- Enter a name and, optionally, a description on the **General** tab. Click the **Origin** tab. You can then:
 - Add a function to the field: Select the function from the **Function** list box. The available functions are: **Sum**, **Min**, **Max**, **Number**, **Average**, and **None**.
 - Add a value from an existing field in a real database to the new field or the function selected above: Select the real table and field you want from the two **Field** list boxes.
 - Add an operation to the field selected above: Choose the operator you want from the **Operation** list box. The available operations are: +, -, *, /, and none. The operator + can be used to concatenate characters.
- Then click the button **Ins** to add these options to the field definition.



Example 1:

You have access to a real table "res_tab" which contains four fields res1, res2, res3 and res4 corresponding to the quarterly results in a particular year. You want to define the "Result" field in a table in your virtual database that contains the sum of the four real fields.

On the **Origin** tab, for the field "Result":

- Select the "res_tab" table and the "res1" field in the **Field** option's list boxes.
- Select the operator + in the list box of the **Operation** option.
- Click the **Mod** button to replace the default entry. The new entry is "res_tab.res1 +".
- Next select the "res_tab" table and the "res2" field in the list boxes of the **Field** option.
- Select the operator + from the **Operation** list box again.
- Click the button **Ins** to add the newly created entry "res_tab.res2 +".
- Do the same for the field "res3".
- For the field "res4", select the **None** operator instead of +.
- The **Result** field is finally defined from the following list:

```
res_tab.res1+  
res_tab.res2+  
res_tab.res3+  
res_tab.res4
```

which means that the **Result** field contains the sum of the four fields "res1", "res2", "res3" et "res4".

Example 2:

You want the Results field to show the sum of the year's sales. To do that, find the sum of all the res1 fields, res2 fields,... and then find the sum of these four results.

On the **Origin** tab of the **Result** field:

- Select the **Sum** function from the **Function** field list box.
- Select the res_table table and the res1 field from each of the **Field** option's list boxes.
- Select the + operator from the **Operation** option's list box.
- Click the **Mod** button to replace the default entry. The new entry is "res_tab.res1 +".

- Do the same for the fields "res2" and "res3". For the field "res4", select the operator "none" instead of the **Operation** option's "+" operator.

You can change an item in a field definition using the **Mod** button (the highlighted element is replaced by the values selected above). Click the **Delete** button to delete the highlighted item.

You can change an item in a field definition using the **Mod** button (the highlighted element is replaced by the values selected above). Click the **Delete** button to delete the highlighted item.

After defining the field, click **Apply** for the new options to take effect.

As soon as you define a new virtual field, remember to define the joins between the table(s) used to create the virtual table, if there are any. See the section "**Inter-table links**".

To check that the calculation you assigned to the created field is what you want, use **Tun DB Revamp**'s query function. See "**Querying real and virtual databases**".

Assigning field filters

You can complete the definition of a virtual field with a filter, that is, you can define a condition for the calculation of the field value. The filter corresponds to limiting conditions in the query (as in **MS Query**).

Example:

You want to obtain the sum of the fields "res1" when the "res2" field is greater than a particular value. The condition on "res2" is a filter.

Tun DB Revamp lets you attribute a filter to virtual fields that is used when the end user utilizes the field. A filter can be:

- Static: The filter value is fixed.
- Dynamic: Users enter their own values when they make a query.



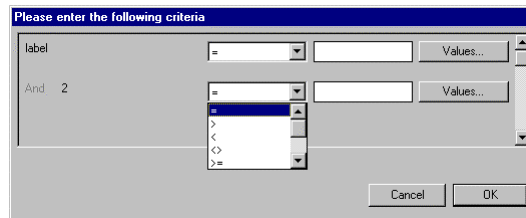
To assign a filter to a virtual field, select the field and then click the **Filter** tab in the **Property Box**. Then do the following:

- Enter a label for the filter in the **Label** field. The label is optional for a static filter. For a dynamic filter, the label must indicate the purpose of the filter for which the user is asked to supply a value.
- Select the table and field to apply the filter to from the **Field** drop-down list boxes.
- Select the comparison operator from the **Comp** drop-down list box.
- For static filters, enter the filter value in the **Value** field. For dynamic filters, enter a question mark (?).
- Click **Ins** to insert the defined criterion.

You can create a set of conditions or criteria: Define the criteria as described above. Select **And** or **Or** to add the extra criteria.

To check that the filter you assigned to the created field is what you want, use **Tun DB Revamp's** query function. See "**Querying real and virtual databases**".

If the filter is dynamic, queries to the virtual field display a window like the following:



Enter the value requested to apply the filter to the virtual field. Click the **Values...** button to display the list of possible values for field in question.

Inter-table links

The fields defined in a virtual table are obtained from one or more tables in a real database.

For each virtual table, it's essential to define the links between the real tables from which its constituent fields are extracted. Defining these links makes it possible to create the joins between the real tables when the end-user queries the virtual database. The links can be direct or indirect (that is, links between one table and another or between several tables via intermediary tables).

➤ Defining links

The simplest way is to define the links at the same time as you define the fields in the virtual table. The real tables used to define the fields must be linked directly or indirectly to the other real tables used by the virtual table.

To define links between real tables for the same virtual table, select the virtual table and do the following:

- Click the **Links** tab for the virtual table.
- For each real table, select the real table name and the field to use as a join with the other table. To do this, use the list box **Key1** for the first real table, and **Key2** for the second real table.

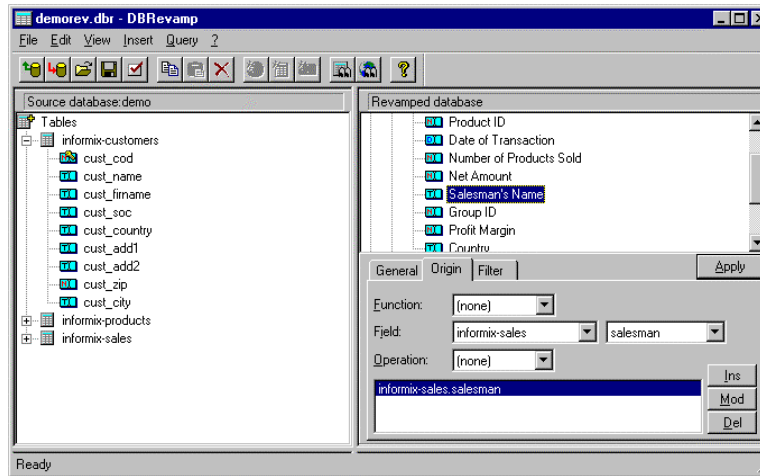
Note:

The names of the two fields linking the tables can be different, even if they represent the same data.

- Select a comparison operator in the **Comp** list box.



- Click the **Ins** button to add the link to the list of links in the virtual table.




You can change a link with the **Mod** button after changing the values of the selected item. Click **Del** to delete the selected item.

Click **Apply** to validate the list of links you've defined.

➤ Checking links

Tun DB Revamp provides a function for verifying the links you define between the real tables to construct the virtual table.

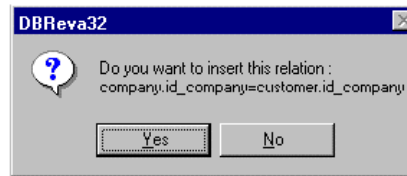
For each virtual table, you can easily check if the real tables used are linked and if the defined links form a coherent whole.

To do this, click the "magic wand" button  on the **Links** tab.

Tun DB Revamp then examines all the links you've defined and detects direct or indirect links that isolate particular tables from the rest.

When a link between two tables is missing, **Tun DB Revamp** tries to link them using two fields of the same name.

If the two fields exist, **Tun DB Revamp** proposes to define a link between them as follows:



In most cases, the proposed link is the right one. If, however, you think that the two fields proposed by **Tun DB Revamp** mustn't form the link between the tables, define the link manually as described in the section "**Defining links**".

If two unlinked tables don't have fields of the same name, **Tun DB Revamp** displays a list of the unlinked tables:




In this case, define the link manually as described in the section "**Defining links**".

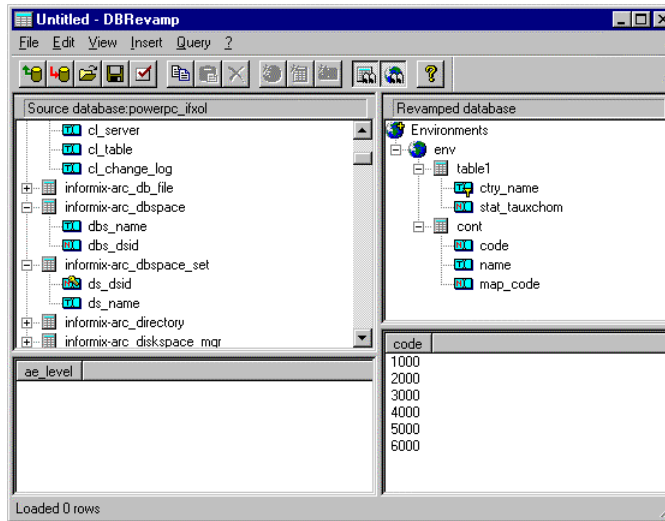
Querying real and virtual databases

Tun DB Revamp includes a query function for real and virtual database tables and fields.

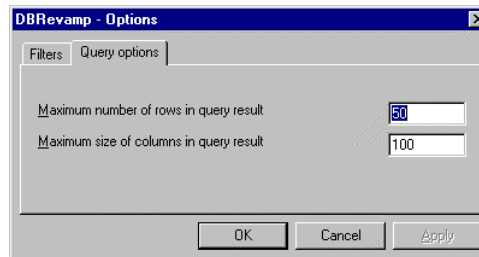
You can use this function to view a real or virtual database table or field directly from **Tun DB Revamp** without using a query tool like **MS Query**.

To query a table (or a field) in the real database, choose **Query→Source** (or **Query→Environment**) from the main menu, or click the toolbar button .

A pane opens below the real or virtual database pane depending on the option you chose. You can choose both options at the same time.



To limit the number of records and the column width displayed when you query a table or a field, choose **View→Options...** from the main menu and click the **Query options** tab:



Enter the maximum number of records to display in the field **Maximum number of rows in query result**.

Enter the maximum column width to display in the field **Maximum size of columns in query result**.

Note:

If limits were defined when the data source was created (see "**Creating a data source**"), they have a higher priority than the **Query options**.

Example:


Say a variable limit of 11 through 15 lines was set when the real data source was configured.

If you query a real database containing more than 11 records, a warning message is displayed, informing you of the limit. The exact warning message depends on the type of limit used.

Validating an environment

Tun DB Revamp includes a function for verifying the consistency between the contents of the environments you create (as administrator) and the contents of the real database used. This is especially useful when changes have been made to the structure of the real database (for example, a field has been changed or deleted), that you haven't accounted for in the virtual database.

To use this function, you must validate the environment before you export it to avoid any possible inconsistency. Choose **File→Validate**

Environments from the main menu or click the button  in the toolbar.

Example:

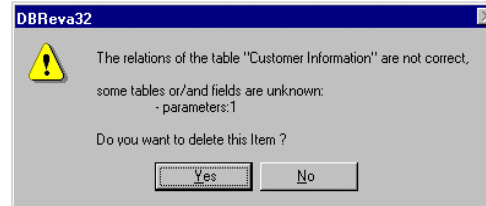
Take, for example, the case where the "parameters:1" table and its fields have been copied to the environment "Marketing". Each field in the virtual table thus created has the "parameters:1" table as its point of origin.

If this logical table is consequently deleted in the real database, the fields in the virtual table will no longer have a point of origin. Similarly, all the tables with a reference to the "parameters:1" table in their links will become incoherent.

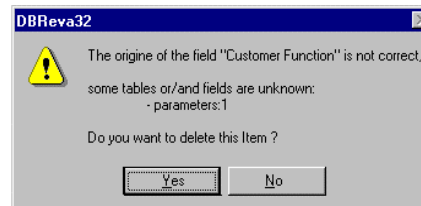


On a request for environment validation, **Tun DB Revamp**:

- Identifies the virtual tables with links to the deleted table and offers to delete them. In this case, it's better not to delete them: Delete the fields that are copied from fields in the deleted table.



- Identifies the virtual fields copied from the deleted table and proposes to delete them.




If no inconsistency is detected, the following window is displayed:



Exporting data source environments

To make the database redefined by its environments available to users, you must export it from the PC to the server.

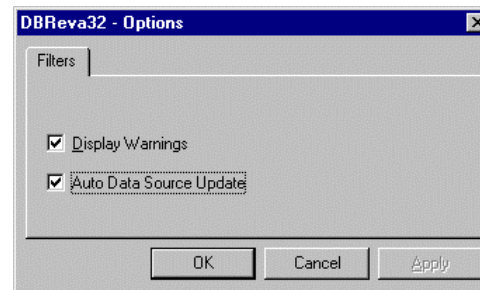
To export data source environments from the PC to the server, choose **File**→**Export...** from the main menu or click the toolbar button .

This operation creates or updates the three tables containing the revamped database information.

Updating a virtual data source

When you change the name of an environment, you can update the corresponding virtual data source. This functionality is automatic if you selected the **Auto Data Source Update** check box (choose **View**→**Options**).

This opens the following dialog box:



If the option is selected, there are two possibilities:

- If the **Display Warnings** check box is selected, **Tun DB Revamp** requests confirmation before performing an automatic update.
- If this check box isn't selected, the update is carried out automatically without confirmation.

If the **Auto Data Source Update** check box isn't selected, choose **Insert**→**Create/Update Data Source** from the main menu or **Update Data Source** from the context menu for the current environment to update the corresponding data source.

If you want to cut the link temporarily between an environment and its data source, choose **Insert**→**Remove Data Source** or **Remove Data Source** from the context menu. You can later recreate the link by choosing **Update Data Source**.

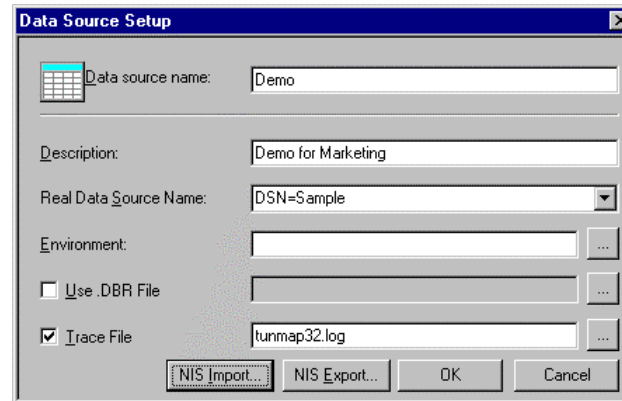


Creating a virtual data source

The chapter "**Configuration and use**" describes how to create a virtual data source with **ODBC Administrator**.

You can also do this with **Tun DB Revamp**. Choose **Create associated data source...** from the environment's context menu.

The following dialog box opens:

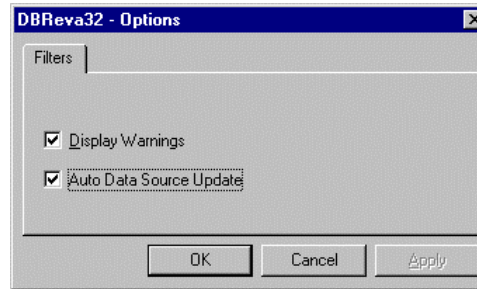


Enter a description of the data source in the **Description** field. Select the **Trace File** check box and specify the log file if you want to keep an activity trace for the virtual data source. For more information on this dialog box, see the section "**Creating a virtual data source**" in the chapter "**Configuration and use**".

Displaying warnings

You can choose to show or hide warnings while you're using **Tun DB Revamp**. These warnings supply information, for example, on inconsistencies which arise during the revamping of the database and missing elements in the new structure.


Tun DB Revamp displays these warnings by default. However, you can switch off the display of message boxes by clearing the **Display Warnings** check box (**View**→**Options**).



Local revamped data source management


You can save and open locally the description of the revamped database. This can be useful if you don't want to export the revamped database immediately, or if you want to keep earlier versions. This description is saved in a file with the extension **".dbr"**.

➤ Saving locally

To save the description of a revamped database produced with **Tun DB Revamp** locally, choose **File**→**Save** (or **File**→**Save As...** to save it under a different name), or click the button  in the toolbar.

The path for the real data source is also saved. This means you can later export the data source and its environments without modifying the path.

➤ Opening a local data source

To open a revamped data source that's saved in a **".dbr"** file on the local machine, choose **File**→**Open** from the main menu or click the toolbar button .

Select the revamped data source you want (a **".dbr"** file).

You can open the most recently used data sources from the **File** menu.



➤ Reloading the database structure






When you open a ".dbr" saved locally, you can update the database structure from which the environments were created. This is useful when the real database has been changed since the last time the ".dbr" file was saved. To do that, choose **File→Reload database structure** from the main menu.

After this operation, it's recommended that you validate the environments created previously, especially if the real fields used to define the virtual fields have been moved or deleted. See "Validating an environment" for more details.

Field identification

➤ Field icons

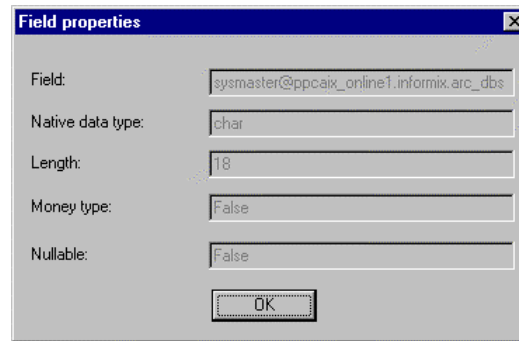
Tun DB Revamp uses field icons to make it easier to read real or virtual tables.

Icons	Meaning
	character field
	date field
	numeric field
	binary field
	table primary key

➤ Real field properties

You can choose **Properties** from a real table field's context menu (left pane) for additional information on the field.

The **Field properties** dialog box opens:



Field properties

Field: sysmaster@ppcaix_online1.informix.arc_dbs

Native data type: char

Length: 18

Money type: False

Nullable: False

OK

Note:

The type of field referred to here corresponds to the native type and so depends on the DBMS used.



PART 3
APPENDICES

REFERENCE

Index

Note:

xxx stands for the relative file extension for a specific database. The file extensions are as follows:

- **ifx** Informix
- **ora** Oracle
- **syb** Sybase
- **db2** DB2
- **pro** Progress
- **pro7** Progress7
- **pro8** Progress8
- **ism** C-ISAM
- **mvs** DB2 for MVS

CONFIG.XXX	File containing the working and security parameters of the Tun SQL UNIX server
DBMAP	Windows application to create or edit character conversion tables.
DBSCRIPT	Windows application which interprets and executes SQL batch files
DBSHOW	Windows application for testing and configuration
PARAM.XXX	File containing the setup parameters of the Tun SQL UNIX server
TUNODBC200.XXX	Tun SQL UNIX server

CONFIG.XXX

Contains the working and security parameters of the **Tun SQL** server.

➤ Description

The **config.xxx** files supply a certain number of parameters to the **Tun SQL** server. Unlike the **param.xxx** files, the parameters don't concern the overall functioning of the server but refer to a particular database. For example, a **config** file looks like this:

```
#Optional declaration for databases
#Example :
#[base_name]
#Define=ENV_VARIABLE:value
#RowLimitMode=None|Absolute|Fixed|Variable|Extended|1
|2|3|4|5
#RowLimitValue=value
#RowLimitMax=value
#DbmsName=DatabaseName
#Version=DatabaseVersion

# In this section, list allowed configuration
(base,user,product)
# Base_Name|*,User_Name|*,Product_Name|*
[Allowed]

# In this section, list denied configuration
(base,user,product)
# Base_Name|*,User_Name|*,Product_Name|*
[Denied]
```

This file can contain as many sections as there are databases managed by the DBMS associated with the **Tun SQL** UNIX server. You don't have to include all the databases if you have no special parameters to set for them.

The title of each section is the name of the corresponding database enclosed in square brackets (for example, **[tunsqldemo]**). You can define the following parameters for each section:



Define=END_VARIABLE:value

Assigns a value to the environment variable **ENV_VARIABLE** before the opening of the database (database installation directory, date format...). Include this option for each database in the **config** file.

RowLimitMode=None|Absolute|Fixed|Variable|Extended|1|2|3|4|5

RowLimitValue=value

RowLimitMax=value

Some office applications let you compose your own SQL requests. In other cases, applications tend to read a table in its entirety before displaying it on the screen. This doesn't pose any real problem if it's a question of small tables contained in a local database. However, insurmountable problems can arise when you're dealing with very large tables in a centralized, remote database. In this case, there's a considerable number of exchanges on the network and there is insufficient memory in the PC to store the received data. The PC has to be frequently rebooted after this type of **select** request. To compensate for this problem, the **Tun SQL** ODBC driver incorporates limits that you define with the **RowLimitMode** parameter. If this parameter is defined, it takes priority over the values defined in the data source on the PC.

The parameter can take five different values:

None	No limit is imposed by the ODBC driver
Absolute	The ODBC driver doesn't load more than RowLimitValue rows in one select request. No messages are displayed to inform the user.
Fixed	The ODBC driver doesn't load more than RowLimitValue rows in one select request. A message is displayed informing the user.
Variable	The ODBC driver doesn't load more than RowLimitValue rows in one select request. However, it displays a message proposing to load more within the limit of the maximum value (RowLimitMax).
Extended Limit	The ODBC driver doesn't load more than RowLimitValue rows in one select request. However, it displays a message proposing to load more without a maximum value. In this case, the message displayed is really only a warning.

DbmsName=DatabaseName

Sets or changes the name of the database to transmit to the ODBC driver.

Version=DatabaseVersion

Sets or changes the version number of the database to transmit to the ODBC driver.

In addition to the sections corresponding to each database, the **config** file can include **[Allowed]** and **[Denied]** sections. You can use these sections to secure access to certain databases. These sections function as follows:

[Allowed]

This section must contain a series of triple terms such as:

base_name, user_name, product_name

where:

- **base_name** is the name of a database
- **user_name** is the name of a user of the database
- **product_name** is the name of a Windows application that uses the server.

Each set of triplets indicates if a user (**user_name**) is authorized to use the database (**base_name**) with the Windows application (**product_name**). Each parameter can be replaced by the generic character *.

For example, the triplet **tunsql Demp,*, excel** means that all the users can use the base **tunsqldemo** from the application EXCEL except for those whose names are contained in a similar triplet in the **[Denied]** section.

[Denied]

This section contains a series of triple terms similar to those in the **[Allowed]** section.



Each set of triple terms indicates if a user (**user_name**) is not authorized to use the database (**base_name**) with a Windows application (**product_name**). Each term can be replaced by the generic character *****.

For example, the triple term ***john,excel** means that the user **john** cannot use any database from the application **excel** except for those whose names are contained in a similar, contradictory triple term in the **[Allowed]** section.

Notes:

For a **Tun SQL** server to take account of a config file, you must use the command line option **-c**.

Two different Informix database engines may coexist (Informix versions 5 and 7). For example, one database could be accessed by database engine 1 from the directory `/u/informix1` and a second by database engine 2 from the directory `/u/informix2`. In this case, the file `config.ifx` contains:

```
[database1]
Define=INFORMIXDIR:/u/informix1
Define=DBPATH:/u/database1
Version=5.01
[database2]
Define=INFORMIXDIR:/u/informix2
Define=DBPATH:/u/database2
Version=7.01
```

➤ **See also**

`param.xxx`, `tunodbc200.xxx`

DBMAP

Windows application for creating or editing conversion tables.

➤ Syntax

```
DBMAP [-ffile_name]
```

➤ Description

You can use **Tun DB Map** to create or edit character translation tables. The translation tables lets you avoid problems caused by the differences in the codification of accented characters on the PC and the remote DBMS. For the translation tables to take effect, you must declare them when you define the data source.

-ffile_name

Replace "file_name" with the name of the existing translation table you want to use.



DBSCRIPT

Windows application for executing SQL batch files.

➤ Syntax

```
DBSCRIPT [-ddata_source] [-ffile_name]
```

➤ Description

With **Tun DB Script**, you can execute a whole list of SQL requests in a single operation. It interprets the SQL commands one by one and stops when it encounters an error. You can also use **Tun DB Script** to change and save SQL batch files.

Tun DB Script is useful for downloading the contents of a database from a Windows PC to a remote DBMS. You can also use it to update or clear large databases.

-ffile_name

Replace "file_name" with the name of the file containing the SQL commands. This file is loaded when the application starts running.

-ddata_source

Replace "data_source" with the name of the data source that the SQL batch file will use. **Tun DB Script** doesn't automatically create the connection with the data source. You must do this subsequently "by hand".

DBSHOW

Windows application for testing and configuring.

➤ Syntax

```
DBSHOW [-hhost_name]
```

➤ Description

You use **Tun SQL** to query a remote host to see if there are any **Tun SQL** servers present.

Enter the name of the host in the **Host** field, then click **Find Servers** to obtain this information. **Tun DB Show** returns the names of any **Tun SQL** servers correctly installed on the remote machine and also the names of the DBMSs they interface with.

This application is particularly useful for checking the conformity of the installation.

-hhost_name

Replace "host_name" with the name of the host you want to query.



PARAM.XXX

File containing the setup parameters of the **Tun SQL** server.

➤ Description

Rather than running **Tun SQL** servers with a large number of command line options, it's better to save the options, each on a separate line, in a file and transmit the name of the file to the host using the **-f** option.

The **Tun SQL** installation procedure uses this mechanism and writes the options to **param.xxx** files (where **xxx** stands for the abbreviated name of the DBMS (**param.ora**, **param.syb**, **param.ifx**, etc.)).

Here's an example:

```
-output=/dev/null
-output2=/dev/null
-DORACLE_HOME=/home3/oracle/7.1.4
-DORACLE_SID=odbc
-config=/usr/tunsql/config.ora
```

The meaning of the different options is explained in the section **tunodbc200.xxx**.

➤ Progress

Some Progress fields may contain several values: These are known as "array fields". To view these values from applications like MS Query and MS Access, the following option must appear in the file **param.proX** (where **X** is the number of the Progress version used): :

```
-arrayfields=*
```

where ***** stands for one of the following characters:

\$, &, #, %, -, _.

The default character is **_**.

The columns required to view the different array field values will then be named:

columnname*n*,

where * is the character chosen in the file param.proX (by default, "_") and n is the position of the value in the table.

Example:

The second value in the array field appears in column col_2_.

If using the character "_" causes a problem when the column is generated (other columns may have a similar name), you must choose one of the other four characters.

➤ **See also**

config.xxx, tunodbc200.xxx

TUNODBC200.XXX

Tun SQL server.

Note:

The different **Tun SQL** servers have a number of options in common. You can obtain a list of these options by running the executable **tunodbc200.xxx** with the option **-a[ll]**, where xxx stands for the database extension.

➤ Syntax

```
tunodbc200.XXX
-a[ll]
    -c[onfig]=config_file
    -Dname=value
    -db[ms]=DBMS_name
    -de[bug]
    -f[ile]=param_file
    -h[old]
    -i[nter]
-l[owercase]
-n[opassword]
    -nor[owcount]
    -o[utput]=file_name
    -o[utput]2=file_name
    -ow[ner]
    -p[rogress]=XX
    -s[electby]
    -sv[archar]
    -sy[scolumns]
    -t[imer]=xx
    -u=user1,user2...
    -v[ersion]=DBMS_version_number
-x=user1,user2...
```

➤ Common options

-a

Lists all the options supported by **tunodbc200.xxx**.

-c=config_file

Associates a configuration file (**config.xxx**) with the server (Cf. config.xxx).

-db=name

Associates a DBMS name with the **Tun SQL** server. This is the name that's displayed by **Tun DB Show** when you click **Find Servers**.

-de

Tells the server to function in trace mode. By default, the messages are displayed on the device **/dev/console**.

-Dname=value

Sets the environment variable "**name**" to the value "**value**" before the server is started (database installation directory, date format, etc.). Use this option in the command line as many times as required. It's indispensable to define specific variables for the **Tun SQL** server to function with some DBMSs. This definition is performed by the installation procedure. For reference purposes, these variables are needed for the following DBMSs:

Oracle

ORACLE_HOME: Oracle installation directory.

ORACLE_SID: Default database.

Informix

INFORMIX_DIR: Informix installation directory.

DBPATH: Directory containing the database (only SE).

Sybase

SYBASE: Sybase installation directory.

SYBSERVNAME: Identifier of the server setup file (optional). This value is equivalent to the variable DSQUERY, defined and used by SYBASE.

-f=param_file

Replace "param_file" with the name of the file containing the program options. Use this option to run the program without having to enter multiple options.



-i

Uses interactive test mode to check the server is working properly.

-o=file

Replace "file" with the name of the file or device that the server and access controller (Watchdog) trace messages are written to. Works only in **debug** mode.

-o2=file

Replace "file" with the name of the file or the device that only the Watchdog trace messages (and not the server's) are written to. Works only in **debug** mode.

-t=xx

Assigns a timeout value. If no reply is received from the PC after this time, the **Tun SQL** server thinks the PC has stopped. The server therefore also stops. This value is of lesser priority than that assigned on the PC when the data source is defined.

-u

Used with the **-x** parameter to define authorized users ("*" for everyone). The default value is *. For example:

```
-x = *  
-u = bill                (only "bill" is authorized)
```

-v=XX

Associates a DBMS version number with the **Tun SQL** server. This value is displayed by **Tun DB Show**.

-x

Used to define users who are denied access ("*" for everyone). For example:

```
-u = *  
-x = bill                (only "bill" is denied access)
```

➤ Informix options

-h

By default, Informix doesn't maintain the cursors open during the execution of the **commit** and **rollback** commands. The **-h** option ensures the cursors are kept open after one of these commands if the applications using this type of server can't do it.

-n

Applies only to SCO UNIX 5. If the system SCO UNIX 3.2 version 5 can't check user passwords correctly, this option cancels password checking.

-s

By default, Informix can't execute **select** commands with a sort option (**group by** or **order by**) on a column which isn't included in the **select** command. Use the **-s** option to compensate when applications make no provision for this.

➤ Oracle options

-l

If tables, columns, indexes or views are created in the catalog with lowercase characters, the **-l** option ensures that the catalog functions return data enclosed in quotation marks. Applications using this type of server create queries with names in quotation marks.

➤ Progress options

-n

Applies only to SCO UNIX 5. If SCO UNIX 3.2 version 5 can't check user passwords correctly, this option cancels password checking.



-nor

Progress can't tell how many lines have been modified or deleted when an **update** or **delete** command is executed. By default, the server compensates for this lack. However, there's a time penalty each time an **update** or **delete** command is executed. If applications using the server don't need to know the number of modified lines, the **-nor** option cancels server compensation, thus saving time.

-ow

By default, the server doesn't support the notion of object owner in the database. Some applications try to add an owner prefix when owners are returned with the objects, provoking errors during execution. This option, **add owners**, is useful when the application needs to know the owner and the owners are correctly handled.

-p=XX

If you must use options that are specific to Progress (for example, the **-Q** option which forces the database to respect the ANSI norm), set the parameter as **-p=XX**, where **XX** stands for the string containing the required options in quotation marks.

-sv

Progress uses only one type of character string. By default, all the strings are considered to be of a fixed length (**SQL_CHAR** in ODBC). If this option is used, the strings are treated as though they were of variable length (**SQL_VARCHAR** in ODBC).

-sy

Progress can define system or hidden columns that aren't returned when a search of all the columns are carried out using a wildcard character. This is why, by default, the server doesn't describe the columns in its catalog. If, however, the hidden columns are required, this option enforces their return.

➤ See also

param.xxx, config.xxx

SQL STATEMENTS USED IN C-ISAM

Principle instructions

CREATE DATABASE	97
CREATE TABLE	98
DEFINE TABLE.....	99
COLUMN DEFINITION OPTION.....	100
DEFAULT CLAUSE	101
NOT NULL CLAUSE.....	102
CONSTRAINT DEFINITION SUBSET.....	103
CONSTRAINT DEFINITION OPTION.....	104
FILE IS OPTION	105
CREATE INDEX.....	106
CREATE SYNONYM	107
COMMENT	108
DROP DATABASE.....	109
CONNECT DATABASE.....	110
DISCONNECT DATABASE	111
DROP INDEX.....	112
DROP TABLE	113
DROP SYNONYM.....	114

UNDEFINE TABLE	115
SELECT	116
SELECT CLAUSE	117
EXPRESSION	118
FROM CLAUSE	119
WHERE CLAUSE	120
GROUP BY CLAUSE.....	121
HAVING CLAUSE	122
ORDER BY CLAUSE.....	123
DELETE.....	124
INSERT.....	125
VALUES CLAUSE.....	126
UPDATE.....	127
SET CLAUSE.....	128
AGGREGATE EXPRESSION.....	129

SQL statement syntax

The SQL statements are presented using the following syntax:

- Reserved names are in uppercase characters (INSERT, UNIQUE, etc.). You can, however, use lowercase characters to enter them on the command line.
- Variable names are in italic (*Databasename, etc.*).
- Square brackets indicate optional parameters or entries ([optional]).
- Braces and the expression **xor** indicate an exclusive choice ({A xor B xor C}).
- The exponent n (ⁿ) indicates a sequence that can be repeated from 0 to n times (sequenceⁿ).

Punctuation marks and parentheses are literal symbols that must be entered exactly as they appear.



CREATE DATABASE

➤ Purpose

Creates a new database.

➤ Syntax

```
CREATE DATABASE Basename
```

Note:

The database name must be less than 18 characters.

➤ Use

The database created becomes the current database.

This statement can only be used with **sqltools** (**Tun SQL** tool).

After the creation of the database, a directory named *Databasename.ism* is created. This directory contains the extra C-ISAM files composing the catalog (*SysTables*, *SysColumns*, *SysIndexes*, *SysDefaults*). It is a subdirectory of the directory designated by the *ISAM-PATH* environment variable if set, or of the current directory.

➤ Example

```
CREATE DATABASE TEST;
```

Creates the directory *test.ism* containing the files *SysTables.dat*, *SysTables.idx*, *SysColumns.dat*, *SysColumns.idx*, *SysIndexes.dat*, *SysIndexes.idx*, *SysDefaults.dat*, and *SysDefaults.idx*.

CREATE TABLE

➤ Purpose

Creates a new table in the current database and places data integrity constraints on its columns or a group of columns.

➤ Syntax

```
CREATE TABLE tablename (Column definition [,Column definition]n)  
[,Constraint definition]n
```

Note:

The table name must be less than 18 characters.

➤ Use

Table names in the same database must be unique. Each column in the same table must have a different name.

The table name can be prefixed with the name of a UNIX user who becomes the owner of the table. If no name is specified, the current login ID is used.

➤ Example

```
CREATE TABLE TABLE_TEST (c1 char);
```

This statement creates table *table1* in the relational database by creating the associated C-ISAM files (*table1_100.dat* and *table1_100.idx*, for example).

The C-ISAM file names are created by taking the first seven characters of the table name and adding a unique value to them. If the table name has less than seven characters, the new file names are padded out with underline characters (). The unique value is 100 for the first table created: This value is then incremented by 1 for each new table.



DEFINE TABLE

➤ Purpose

Defines a new table in the current database, with data integrity constraints on its columns or a sequence of its columns, and conditional options for the existence of files.

➤ Syntax

```
DEFINE TABLE tablename [File is option] (Column definition  
[,Column definition]n) [,Constraint definition]n
```

Note:

The table name must be less than 18 characters.

➤ Use

Table names in the same database must be unique. Column names in the same table must all be different.

The table name can be prefixed with the name of a UNIX user who becomes the owner of the table. If no name is specified, the current login ID is used.

➤ Example

```
DEFINE TABLE TABLE1 file is file_1 (c1 char);
```

In this example, the *file_1* C-ISAM files (*file_1.idx* and *file_1.dat*) already exist: Only the table must be defined.



COLUMN DEFINITION OPTION

Used in the statements **CREATE TABLE** and **DEFINE TABLE**.

➤ Purpose

The "column definition" option in the **DEFINE TABLE (CREATE TABLE)** statement lists the name, type, default value and constraint for a single column.

➤ Syntax

Columnname Data type [Default clause] [Not null clause] [Constraint definition subset]

➤ Example

```
CREATE TABLE PETS  
(name char (20),
```

```
sex char(1));
```

This table is composed of the columns *name*, *race* and *sex*.



DEFAULT CLAUSE

Used in the option **COLUMN DEFINITION**.

➤ Syntax

DEFAULT [{Literal xor NULL xor Current xor Today xor User}]

➤ Use

The default value is inserted into the column when an explicit value is not specified. If a default isn't specified and the column allows nulls, the default is NULL.

LITERAL	String of characters or numeric constant characters defined by the user
NULL	Null value
CURRENT	Current date and time (only usable with TIMESTAMP type)
TODAY	Current date (only usable with DATE type)
USER	Name of the current user (only usable with VAR or VARCHAR type)

➤ Example

```
CREATE TABLE PETS
(name char (20),
 race char(25),
 sex char(1) DEFAULT 'M')
```

In this table, the default value for *sex* is a string literal: 'M'.

NOT NULL CLAUSE

Used in the option **COLUMN DEFINITION**.

➤ Purpose

If you don't indicate a default value for a column, the default is null unless you include the NOT NULL keywords after the data type of the column. In this case, there is no default value for the column.

➤ Syntax

NOT NULL

➤ Example

```
CREATE TABLE INVOICE
(invoice_id longint NOT NULL,
 customer_name char (30))
```

If the column is designated as NOT NULL (and no default value is specified), you need to enter a value in this column when inserting a row or updating that column in a row. If not, the database server returns an error.



CONSTRAINT DEFINITION SUBSET

Used in the option **COLUMN DEFINITION**.

➤ Purpose

The constraint definition subset lets you create constraints for a single column.

➤ Syntax

{UNIQUE *xor* PRIMARY KEY} [CONSTRAINT *Constraint name*]

Note:

The constraint name mustn't be longer than 18 characters and must be unique in the database.

➤ Use

UNIQUE	Constrains the field to be unique
PRIMARY KEY	Constrains the field to be unique and to be the primary key of the table

If the name of the constraint isn't specified, a default name is assigned.

➤ Example

```
CREATE TABLE INVOICE
(invoice_number longint UNIQUE CONSTRAINT un_invoice,
 customer_name char (30))
```

The unicity constraint of the invoice number is called *un_invoice*.



CONSTRAINT DEFINITION OPTION

Used in the statements **CREATE TABLE** and **DEFINE TABLE**.

➤ Purpose

The constraint definition option lets you create constraints for a set of columns (from 1 through 8 columns).

➤ Syntax

```
{UNIQUE      xor      PRIMARY      KEY}      (Columnname  
[,Columnname]n)[CONSTRAINT Constraint name]
```

➤ Use

UNIQUE	Constrains the field to be unique for the set of columns
PRIMARY KEY	Constrains the field to be unique and to be the primary key for the set of columns

If the name of the constraint isn't specified, a default name is assigned to the constraint.

Each column named in a constraint must be a column in the table and can't appear in the constraint list more than once.

➤ Example

```
CREATE TABLE FAMILY  
(name char (20),  
surname char (20),  
birth_date date,  
PRIMARY KEY (name, surname) CONSTRAINT pk_family)
```

The primary key constraint *pk_family* affects the *name* and *surname* fields.



FILE IS OPTION

Used in the statement **DEFINE TABLE**.

➤ Purpose

Defines the use of the table according to the existence or non-existence of a file.

➤ Syntax

FILE IS *filename*

➤ Use

If this option is used, no C-ISAM files are created. **Tun SQL** must find the files *filename.dat* and *filename.idx* in the directory of the current database before it can use the table.

If the option isn't used, a default name is assigned to the files created in the current database directory.

➤ Examples

```
CREATE DATABASE TEST;  
DEFINE TABLE TABLE1 FILE IS FIC1 (C1 CHAR);
```

In this example, no files are created. The files *foo1.dat* and *foo1.idx* must be added to the directory *test.ism*.

```
DEFINE TABLE TABLE1 (C1 CHAR);
```

In this example, the files *table_100.dat* and *table_100.idx* (default names) are created in the directory *test.ism*.

CREATE INDEX

➤ Purpose

Creates an index for one or more columns in a table (from 1 through 8 columns).

➤ Syntax

```
CREATE {UNIQUE xor DISTINCT} INDEX indexname ON  
tablename (Columnname [,Columnname]n)
```

Notes:

The index name mustn't be longer than 18 characters.
The number of columns can vary from 1 through 8.

➤ Use

UNIQUE	Constrains the index to be unique
DISTINCT	Synonym for UNIQUE

For tables defined with the FILE IS option, the CREATE INDEX statement must be executed before copying the files filename.dat and filename.idx.

➤ Example

```
CREATE DISTINCT INDEX ix_name ON Table1 (name,  
birth_date) ;
```

This statement creates the index *ix_name* for the *name* and *birth_date* columns in the table *Table1*.



CREATE SYNONYM

➤ Purpose

Attributes a synonym to a table.

➤ Syntax

```
CREATE SYNONYM synonymname FOR tablename
```

Notes:

The name of the synonym must not be longer than 18 characters.
--

➤ Use

The name of a synonym must be preceded by the name of a UNIX user who then becomes the owner of the synonym. If no name is given, the current login ID is used by default.

➤ Example

```
CREATE SYNONYM employee FOR Table1;
```

This statement creates the synonym *employee* for the table *Table1*.

COMMENT

➤ Purpose

Attributes a comment to a table or a synonym.

➤ Syntax

```
COMMENT ON {tablename xor synonymname} IS 'comment string'
```

➤ Example

```
COMMENT on TABLE1 IS 'Employees' Table'
```



DROP DATABASE

➤ Purpose

Removes (deletes) an entire database, including all catalogs, indexes and data.

➤ Syntax

DROP DATABASE *basename*

<p>Note: The database name must be less than 18 characters.</p>
--

➤ Use

A database that's being used by another user can't be dropped.

The DROP DATABASE statement doesn't remove the database directory if it contains any files other than those created for the database's tables and indexes.

➤ Example

```
DROP DATABASE DBTEST;
```

Deletes the database *DBTEST*.

CONNECT DATABASE

➤ Purpose

Connects to a different database. You can't perform operations on a database if you're not connected to it.

➤ Syntax

```
CONNECT DATABASE basename
```

➤ Example

```
CONNECT DATABASE DBTEST2;
```

Connects to the database *DBTEST2*.



DISCONNECT DATABASE

➤ Purpose

Disconnects from the current database.

➤ Syntax

DISCONNECT DATABASE *basename*

➤ Example

```
DISCONNECT DATABASE DBTEST2;
```

Disconnects from the database *DBTEST2*.

DROP INDEX

➤ Purpose

Deletes an index.

➤ Syntax

```
DROP INDEX indexname
```

➤ Example

```
DROP INDEX ix_name ;
```

Deletes the index *ix_name*.



DROP TABLE

➤ Purpose

Deletes a table, along with its associated indexes and data.

➤ Syntax

```
DROP TABLE {tablename xor synonymname}
```

➤ Use

If a synonym is deleted by the statement **DROP TABLE**, the table is also deleted.

If the statement **DROP TABLE** applies to a table, the table's synonyms will only be deleted if the statement **DROP SYNONYM** is used.

➤ Example

```
DROP TABLE TABLE1;
```

Deletes the table TABLE1, and its indexes and data.

DROP SYNONYM

➤ Purpose

Deletes a previously defined synonym.

➤ Syntax

```
DROP SYNONYM synonymname
```

➤ Use

If a table is dropped, the synonym remains in place until you explicitly drop it using the DROP SYNONYM statement.

➤ Example

```
DROP SYNONYM employee;
```

Deletes the synonym *employee* attributed to the table *Table1*. The table is not deleted with this statement.



UNDEFINE TABLE

➤ Purpose

Deletes a table defined by a DEFINE statement but doesn't delete the associated data and index files.

➤ Syntax

```
UNDEFINE TABLE {tablename xor synonymname}
```

➤ Example

```
UNDEFINE TABLE TABLE1;
```

Deletes the table TABLE1 created by the statement DEFINE TABLE TABLE1.

SELECT

➤ Purpose

Queries a database.

➤ Syntax

SELECT Select clause From clause [Where clause] [Group by clause]
[Having clause] [Order by clause]

➤ Use

You can query the tables in the current or a different database.

➤ Example

```
SELECT customer_name  
FROM customers  
WHERE turn_over > 250  
ORDERBY country ;
```

This query reads the customers with an annual turnover of over 250 from the *customers* table and returns their names listed by country.



SELECT CLAUSE

Used by the statements **SELECT** and **INSERT**.

➤ Syntax

[[**ALL** xor **DISTINCT** xor **UNIQUE**]] {Expression [[**AS**] *Display label*] [,Expression [[**AS**] *Display label*]]ⁿ

In the **SELECT** clause, you specify exactly which data to select, and if you want to omit duplicate values.

➤ Use

ALL	Specifies that all the selected values are returned, even if they're duplicates.
DISTINCT	Removes duplicate rows from the query results.
UNIQUE	Synonym for DISTINCT .

➤ Example

```
SELECT customer_name  
FROM customers  
WHERE turn_over > 250  
ORDERBY country ;
```

This query reads the customers with an annual turnover of over 250 from the *customers* table and returns their names listed by country.

```
SELECT order_date, COUNT(*), paid_date - order_date  
FROM orders  
GROUP BY 1, 3
```

This query returns the order date, the number of orders and the difference between the payment and order dates, grouped by order date and time lapse (difference between the dates).

Used by the clause **SELECT**.

➤ Syntax

```
{ [{tablename xor synonymname xor tablealias}.] columnname  
xor NULL  
xor Literal number  
xor Quoted string  
xor User  
xor Aggregate expression}
```

➤ Example

```
'Cordwainer'
```

The character string '*Cordwainer*' is a subexpression.



FROM CLAUSE

Used by the statements **SELECT** and **DELETE**.

➤ Purpose

Lists the table or tables from which data are selected.

➤ Syntax

```
FROM {Table name xor Synonym name} [[AS] Table alias]  
    [{Table name xor Synonym name} [[AS] Table alias]]n
```

➤ Example

```
SELECT customer_name, order_num  
FROM customers c, orders o  
WHERE c.customer_num = o.customer_num ;
```

In this statement, data is extracted from the tables *customers* and *orders* and the tables are given aliases.

WHERE CLAUSE

Used by the statements **SELECT**, **DELETE** and **UPDATE**.

➤ Purpose

Specifies search and join conditions for the data that are selected.

➤ Syntax

WHERE Condition [AND Condition]¹

➤ Example

```
SELECT customer_name
FROM customers
WHERE last_order_date < '28/07/1993'
ORDERBY country ;
```

In this example, the search condition is the last order date.



GROUP BY CLAUSE

Used by the statement **SELECT**.

➤ Purpose

Produces a single row of results for each group.

➤ Syntax

```
GROUP BY {Table name xor Synonym name } . Column name xor  
Select number  
[, {Table name xor Synonym name} . Column name xor Select  
number}]n
```

➤ Use

A group is a set of rows that have the same values for each column listed.

The "select number" variable is an integer that represents the position of a column in the SELECT clause.

➤ Example

```
SELECT order_date, COUNT(*), paid_date - order_date  
FROM orders  
GROUPBY order_date, 3 ;
```

The results are grouped by *order_date* and *paid_date - order_date*.



HAVING CLAUSE

Used by the statement **SELECT**.

➤ Purpose

Applies one or more conditions to groups.

➤ Syntax

HAVING Condition

➤ Example

```
SELECT customer_num, call_dtime, call_code
FROM cust_calls
GROUP BY call_code, 2 , 1
HAVING customer_num < 42 ;
```

This query returns the call_code, call_dtime and customer_num tables and groups them by call_code for all the calls from customers whose customer number is less than 42.



ORDER BY CLAUSE

Used by the statement **SELECT**.

➤ Purpose

Sorts query results by the values contained in one or more columns.

➤ Syntax

```
ORDER BY {Table name . xor Synonym name .} Column name xor  
Select number xor Display label} [, {Table name . xor Synonym name .}  
Column name xor Select number xor Display label}]n
```

➤ Use

The "select number" variable is an integer that represents the position of a column in the SELECT clause.

➤ Example

```
SELECT customer_name  
FROM customers  
WHERE turn_over > 250  
ORDERBY country ;
```

This query returns its results ordered by country.

DELETE

➤ Purpose

Deletes one or more rows from a table.

➤ Syntax

```
DELETE FROM {Table name xor Synonym name}
           [WHERE {Condition xor CURRENT OF Cursor name}]
```

➤ Example

```
DELETE FROM customers WHERE last_order_date<1992 ;
```

This statement deletes the rows from the *customers* table where the last order date is earlier than 1992.



INSERT

➤ Purpose

Inserts one or more rows into a table.

➤ Syntax

```
INSERT INTO {Table name xor Synonym name} [(Column name  
[,Column name]n)] {Values clause xor Select clause}
```

➤ Example

```
INSERT INTO Pets VALUES ('Socks', 'Cat', 'M') ;
```

This statement inserts the values *'Socks'*, *'Cat'* and *'M'* into the *Pets* table.

VALUES CLAUSE

Used by the statement **INSERT**.

➤ Syntax

VALUES ({*Variable name* : *Indicator variable* xor NULL xor Literal number xor Quoted string xor Literal Timestamp xor Literal date xor Literal time} [, {*Variable name* : *Indicator variable* xor NULL xor Literal number xor Quoted string xor Literal Timestamp xor Literal date xor Literal time}]ⁿ)

➤ Use

When you use the VALUES clause, you can only insert one row at a time. Each value that follows the VALUES keyword is assigned to the corresponding column listed in the INSERT INTO clause (or to the columns in order if a list of columns isn't specified).

➤ Example

```
INSERT INTO Pets VALUES ('Socks', , 'M') ;
```

This statement inserts the value *'Socks'* in the first column and *'M'* in the third column of the table *Pets*. The second column is untouched.



UPDATE

➤ Purpose

Changes the value of one or more columns or one or more rows in a table.

➤ Syntax

```
UPDATE {Table name xor Synonym name} SET Set clause [WHERE  
{Condition xor CURRENT OF Cursor name}]
```

➤ Example

```
UPDATE Catalog SET item_price = 10 WHERE item_type =  
'L' ;
```

This statement changes the price of all the '*L*' type articles in the table *Catalog* to 10.

SET CLAUSE

Used by the statement **UPDATE**.

➤ Syntax

```
{Column name = {Constant xor (Select statement) xor NULL}  
[,Column name = {Constant xor (Select statement) xor NULL}]n  
xor {(Column name [,Column name]n xor *} = (Select statement)
```

➤ Example

```
UPDATE Catalog SET item_price = 10
```

In this statement, the **SET** clause changes the price *item_price* to 10..



AGGREGATE EXPRESSION

Used by the statement **SELECT** or in an expression.

➤ Syntax

```
{COUNT(*)  
xor  
{MIN xor MAX xor SUM xor AVG xor COUNT} ({DISTINCT xor  
UNIQUE}) {Table name xor Synonym name xor Table alias} . Column  
name)  
}
```

➤ Example

```
SELECT COUNT (DISTINCT item_type) FROM Catalog ;
```

This statement returns the number of different types of article in the table *Catalog*.

Data types

This section describes the data types supported by the SQL language with their equivalent C language definitions. The descriptions correspond to the SQL types used by CREATE TABLE. They are native mode equivalents. For the importation of files created outside the C-ISAM database with the command **DEFINE TABLE**, the differences are indicated.

In the following table, the types in italics in the column "C language type" only apply to the use of CREATE TABLE.

SQL type in the database	SQL type in ODBC	C language type
bit	SQL_BIT	<i>unsigned char notnull_data;</i> unsigned char data;
byte	SQL_TINYINT	<i>unsigned char notnull_data;</i> unsigned char data;
char(maxlength) 1 <= maxlength <= 32511	SQL_CHAR	char data[maxlength];
varchar(maxlength) 1 <= maxlength <= 32511	SQL_VARCHAR	char data[maxlength];
binary(maxlength) 1 <= maxlength <= 32511	SQL_BINARY	<i>unsigned char notnull_data;</i> unsigned char data[maxlength];
varbinary(maxlength) 1 <= maxlength <= 32511	SQL_VARBINARY	<i>unsigned char size_data[2];</i> <i>unsigned char data[maxlength];</i>
smallint	SQL_SMALLINT	short data; /* 2 bytes */
longint	SQL_INTEGER	long data; /* 4 bytes */
real	SQL_FLOAT	float data; /* 4 bytes */
double	SQL_DOUBLE	double data; /* 8 bytes */
decimal(prectot, precdec) 1 <= prectot <= 32 et 0 <= precdec <= prectot	SQL_DECIMAL	char data[(prectot+1)/2+1];
date	SQL_DATE	unsigned long data;
time	SQL_TIME	unsigned long data;
timestamp	SQL_TIMESTAMP	unsigned long data;

➤ The bit type

This type corresponds to the SQL_BIT type in ODBC. It stores the binary values 0 and 1, or the value null.

If this data type is used with CREATE TABLE, a two-byte block is reserved in the created file to differentiate between the null value and 0 or 1. If such a field is used in the definition of a key, the two bytes are used in the key.



If the bit data type is used with DEFINE TABLE, only one byte is reserved. The value 0 is no longer differentiated from the value null. In DEFINE TABLE therefore this type of field can't be null.

The following table shows the values stored. The data in italic only applies to CREATE TABLE.

Bit type field	<i>notnull_data</i>	Data value
0	<i>1</i>	0
1	<i>1</i>	1
<i>Null</i>	<i>0</i>	<i>0</i>

➤ **The byte type**

This type corresponds to the SQL_TINYINT type in ODBC. It stores values from 0 to 255, or null.

If this data type is used with CREATE TABLE a two-byte block is reserved (as with the **bit** type) to differentiate the value null from the other values. If this type of field is used in the definition of a key, the two bytes are used in the key.

If the data type is used with DEFINE TABLE, only one byte is reserved. The value 0 therefore can't be differentiated from null. In DEFINE TABLE this type of field can't be null.

The following table shows the values stored. The data in italic only applies to CREATE TABLE.

byte type field	<i>notnull_data</i>	data value
0	<i>1</i>	0
1	<i>1</i>	1
...	<i>1</i>	...
255	<i>1</i>	255
<i>null</i>	<i>0</i>	<i>0</i>

➤ **The char type**

This type corresponds to the SQL_CHAR type in ODBC. It can stock from 1 to 32511 characters.

When this type of data is inserted in a field, all the non-significant spaces at the end of the string are deleted and the corresponding bytes in the file are set to '\0' (ASCII code 0). When data is read from these fields, the string read is automatically completed with spaces (ASCII code 32) to its maximum size. If an empty string is stored in a field of type **char**, a single space is written to the field to distinguish it from null.

char(10) type field	data value
''	0x32000000000000000000
' '	0x32000000000000000000
'A '	0x41000000000000000000
'AaBb'	0x41614262000000000000
null	0x00000000000000000000

➤ The varchar type

This type corresponds to the SQL_VARCHAR type in ODBC. It's used in practically the same way as the **char** type. It also stores from 1 to 32511 characters.

It differs from the **char** type in that when data is inserted into this type of field, none of the non-significant spaces at the end of the string are deleted but the corresponding bytes in the file are filled with the character '\0'. When data is read from these fields, ODBC retrieves the unmodified string. As for the **char** type, an empty string is stored as a single space character to distinguish it from null.

varchar(10) type field	data value
''	0x32000000000000000000
' '	0x32000000000000000000
'A '	0x41202020202000000000
'AaBb'	0x41614262000000000000
null	0x00000000000000000000

➤ The binary type

This type corresponds to the SQL_BINARY type in ODBC. It can store from 1 to 32511 characters.



When this type of data is used with CREATE TABLE, an extra byte (nonnull_data) is used. This byte can be set at 0 or 1 to distinguish the value null from an empty string. If a **binary** type field is used in the definition of a key, the nonnull_data byte is included with the data bytes in the key.

No extra byte is added when the binary type is used with DEFINE TABLE. Only the significant bytes are stored. The value 0 can't be distinguished then from the value null. For this reason, this type of field can't be null.

When data is inserted into this type of field, the corresponding bytes in the file are completed with the '\0' character. When this type of field is read, all the bytes are retrieved by ODBC.

The following table shows the stored values. The data in italics only apply to CREATE TABLE.

binary(10) type field	<i>nonnull_data</i>	data value
0x	<i>1</i>	0x00000000000000000000
0x00	<i>1</i>	0x00000000000000000000
0x1234	<i>1</i>	0x12340000000000000000
<i>null</i>	<i>0</i>	<i>0x00000000000000000000</i>

➤ The varbinary type

This type corresponds to the SQL_VARBINARY type in ODBC. It can store from 1 to 32511 characters.

This type can only be used with the CREATE TABLE command. No **varbinary** type fields can be used with DEFINE TABLE. Two extra bytes are used with this data type. The bytes store the length of the binary data plus one as a short integer. The value 0 corresponds to a null binary and the value 1 to a binary of zero length.

Like the **binary** type, when data is inserted into this type of field, the corresponding block in the file is completed with the character '\0'. When this type of field is read, only the bytes with the binary length are retrieved by ODBC. If a **varbinary** type of field is used in the definition of a key, the size_data bytes aren't included with the data bytes in the key.

varbinary(10) type field	size_data	data value
0x	0x0001	0x00000000000000000000
0x00	0x0002	0x00000000000000000000
0x1234	0x0003	0x12340000000000000000
null	0x0000	0x00000000000000000000

➤ The smallint type

This type corresponds to the SQL_SMALLINT type in ODBC. It stores integers in the range -32767 to 32767 in 2 bytes. The value -32768 is reserved for a null field.

smallint type field	data value
-32767	-32767
0	0
30000	30000
null	-32768

➤ The longint type

This type corresponds to the SQL_INTEGER type in ODBC. It stores integer values in the range -134217727 to 134217727 in 4 bytes. The value -134217728 is reserved for a null field.

longint type field	data value
-32767	-32767
0	0
134217	134217
null	-134217728

➤ The real type

This type corresponds to the SQL_REAL and SQL_FLOAT types in ODBC. It stores floating point numbers in machine format in 4 bytes.

real type field	data value
-25	(float)-25.0
0	(float)0.0
3.1415	(float)3.1415
null	non-significant value



➤ The double type

This type corresponds to the SQL_DOUBLE type in ODBC. It stores floating point numbers in machine format in 8 bytes.

double type field	data value
-25	(double)-25.0
0	(double)0.0
3.1415	(double)3.1415
null	non-significant value

➤ The decimal type

This type corresponds to the SQL_DECIMAL type in ODBC. It stores fixed point numbers.

The decimal type must be followed by two parameters in brackets, "decimal(n, m)", where n is the total number of digits and m the number of decimals. If m isn't specified, a default value of 0 is used.

The C-ISAM functions stdecimal() and lddecimal() are used to read/write fields of this type.

➤ The date type

This type corresponds to the SQL_DATE type in ODBC. It stores a date as the number of days since the 1st January in the year 0 as a 4-byte integer. To insert or test a date in SQL commands, use ODBC notation ({ d 'AAAA-MM-JJ' }).

date type field	data value
{ d '0000-01-01' }	0
{ d '1997-02-17' }	729438
null	-134217728

➤ The time type

This type corresponds to the SQL_TIME type in ODBC. It stores the time as the number of seconds in the day in a 4-byte integer. To insert or test a time in SQL commands, use ODBC notation ({ t 'hh:mm:ss' }).

time type field	data value
{ t '00 :00 :00' }	0
{ t '13 :40 :10' }	49210
null	-134217728

➤ The timestamp type

This type corresponds to the SQL_TIMESTAMP type in ODBC. It stores the time as the number of seconds from the 1st January 1970 (similar to the time() function in C). The maximum value corresponds to the date 5 February 2036, 00.00 h. To insert or test a timestamp in SQL commands, use ODBC notation ({ ts 'AAAA-MM-JJ hh :mm :ss' }).

timestamp type field	data value
{ ts '1970-01-01 00 :00 :00' }	0
{ ts '1997-02-17 13 :40 :10' }	856186810
null	-134217728



INDEX

A

Allowed,82
Arrayfields (Progress),87

B

BackOffice,2
binary,130, 132
bit,130
byte,130, 131

C

char,130, 131
C-ISAM,37
 Create database,39
 ISAM-PATH,39
 sqltools,38
 SysColumns,38
 SysDefaults,38
 SysIndexes,38
 SysTables,38
Client/Server model,11
COLUMN DEFINITION,100
COMMENT,108
config.xxx,80
CONNECT DATABASE,110
CONSTRAINT DEFINITION,103, 104
Conversion tables,33
Create database,39
CREATE DATABASE,97
CREATE INDEX,106
CREATE SYNONYM,107
CREATE TABLE,98, 130

D

Data source,21
Database revamping,14
date,130, 135
DB Script,15
DB Show,15, 17
DB2,14
DBMAP.EXE,84

DBSCRIPT.EXE,85
DBSHOW.EXE,86
Debug,91
decimal,130, 135
DEFAULT,101
DEFINE TABLE,99, 130
DELETE,124
Denied,82
DISCONNECT DATABASE,111
double,130, 135
DROP DATABASE,109
DROP INDEX,112
DROP SYNONYM,114
DROP TABLE,113, 115

E

Embedded SQL,9
Environment,53, 60
Exporting data source environments,71

F

FILE IS,105
Files
 .dat files,37, 38, 40, 41
 .idx files,37, 38, 40, 41
 C-ISAM,37
FROM,119

G

GROUP BY,121

H

HAVING,122

I

Importing a data source,59
Index (C-ISAM),42
Informix,14, 90
INSERT,125
Inter-table links,66

ISAM-PATH,39

J

Joins,53

L

Limits,27
longint,130, 134

N

NOT NULL,102

O

ODBC,9, 26
ODBC.DLL,10
Oracle,14, 90
ORDER BY,123
Ordres SQL/C-ISAM
 COMMENT,108
 CREATE SYNONYM,107

P

PARAM.XXX,87
Progress,14

R

RDBMS,12
real,130, 134
Revamping,53

S

Script (running a script from sqltools),48
Security,82
SELECT,116
SELECT CLAUSE,117
Sequential files,38
SET,128
smallint,130, 134
Special characters,33
SQL,12
SQL/C-ISAM clauses
 DEFAULT,101
 FROM,119

GROUP BY,121
HAVING,122
NOT NULL,102
ORDER BY,123
SELECT CLAUSE,117
SET,128
VALUES,126
WHERE,120
SQL/C-ISAM options
 COLUMN DEFINITION,100
 CONSTRAINT DEFINITION,104
 FILE IS,105
SQL/C-ISAM statements
 CONNECT DATABASE,110
 CREATE DATABASE,97
 CREATE INDEX,106
 CREATE TABLE,98
 DEFINE TABLE,99
 DISCONNECT DATABASE,111
 DROP DATABASE,109
 DROP INDEX,112
 DROP SYNONYM,114
 DROP TABLE,113, 115
 INSERT,125
 UPDATE,127
SQL/C-ISAM subset
 CONSTRAINT DEFINITION,103
SQL_BINARY,130, 132
SQL_BIT,130
SQL_CHAR,130, 131
SQL_DATE,130, 135
SQL_DECIMAL,130, 135
SQL_DOUBLE,130, 135
SQL_FLOAT,130, 134
SQL_INTEGER,130, 134
SQL_REAL,134
SQL_SMALLINT,130, 134
SQL_TIME,130, 135
SQL_TIMESTAMP,130, 136
SQL_TINYINT,130, 131
SQL_VARBINARY,130, 133
SQL_VARCHAR,130, 132
sqltools,38
Statements
 DELETE,124
 SELECT,116
Sybase,14, 90
SysColumns,38
SysDefaults,38
SysIndexes,38
SysTables,38



T

Tables (C-ISAM),41
time,130, 135
timestamp,130, 136
Trace,90
Translator,26
TUNODBC.XXX,89
Type
 binary,130, 132
 bit,130
 byte,130, 131
 char,130, 131
 date,130, 135
 decimal,130, 135
 double,130, 135
 longint,130, 134
 real,130, 134
 smallint,130, 134
 time,130, 135
 timestamp,130, 136
 varbinary,130, 133
 varchar,130, 132

U

UPDATE,127

V

Validating environments,70
VALUES,126
varbinary,130, 133
varchar,130, 132
Virtual data source,31
Virtual databases,51
Virtual field,54, 61
Virtual ODBC driver,13, 14, 55
Virtual table,54
Virtual table,60

W

Warnings,73
WHERE,120
WOSA,10