# FrontBase®

# Users Guide

Updated 2002-10-08 TLH
Because FrontBase is continually being improved, some of the information in this manual may be inaccurate. Please read the Release Notes on the FrontBase distribution for the latest up-to-date information.

## How to Contact FrontBase:

| | |
|---|---|
| **U.S.A. and international** | FrontBase Inc.<br>27042 Towne Center Drive<br>Suite 265<br>Foothill Ranch, CA 92610<br><br>FrontBase Inc.<br>Datavej 52<br>DK-3460 Birkerød<br>Denmark |
| **Ordering** | Voice: +1 949 455 2913<br>Fax: +1 949 457 0966<br><br>Voice: +45 4582 6262<br>Fax: +45 4582 0816 |
| **World Wide Web** | `http://www.frontbase.com` |
| **Technical support** | `support@frontbase.com` |
| **Information** | `info@frontbase.com` |
| **Sales,marketing** | `sales@frontbase.com` |
| **Licensing** | `license@frontbase.com` |

# Table of Contents

# 1 Document Guide

The FrontBase User’s Guide is a document that is designed for ease of use as well as providing operation guidelines.

This document contains the following chapters:

2

# Foreward

FrontBase is a scalable relational database server. A few general concepts will help explain what that means.

## Architecture

### Client/server architectures

These terms should be familiar to anyone who has used the world wide web. With the web, your browser (the client) makes a request for a page from some web site (the server). The web site processes the request and delivers a result (the web page). The FrontBase server is similar in some ways to a web server. It listens for requests from FrontBase clients, processes requests, and returns results.

### Relational Database server

However, FrontBase is a relational database server. While a web server typically serves web pages for display in a browser, FrontBase handles requests to store and retrieve data. By implementing the "relational" model, FrontBase stores data in application-defined tables, among which, application-defined relations may exist. Tables are defined by their columns. For example, a phone book table might have the following columns: name, phone_number. The actual data is entered in rows of a table. A row in the phone book table might have "John Doe" in the name column and "555-1212" in the phone_number column.

### Highly Scalable

FrontBase is highly scalable, which means it can handle very large data sets and high request loads. The size of a data set is typically described by the number of rows in a table. Using its column indexing features, FrontBase can efficiently insert and lookup data in tables with millions of rows. In applications with high request loads, FrontBase's replication and clustering features can be exploited to run the same database on multiple servers.

## SQL 92 query language

FrontBase implements the industry-standard SQL 92 query language. FrontBase clients use this language to store and retrieve data on the server. They also use the language to manage users, tune performance, and do other administrative tasks. FrontBase implements the SQL 92 standard quite strictly, but also has extensions to handle FrontBase-specific issues. FrontBase uses other standards, such as TCP / IP for client / server communication and Unicode for character value storage.

## Inherent Interfaces

With its features, scalability, and adherence to standards, FrontBase is the ideal database for today's custom client / server applications and for building dynamic web sites. Application developers can use FrontBase's FBCAccess C library, JDBC, ODBC, Tcl drivers, and other interfaces to develop custom applications. They can use FrontBase's array of adaptors to develop dynamic websites. Currently there are adaptors for PHP, Perl, and WebObjects / EOF, a REALBasic plugin and an Omnis Studio DAM.

# 3

# Introduction

FrontBase is a high performance relational database engine conforming to the standards and demands of today's quality minded developers and users.

This chapter contains the following sections:

## Overview

The engine is written in pure ANSI C and benefits from 15+ years of experience with compiler and run-time systems, embedded systems, object-oriented programming, database systems, and command-and-control systems.

The topics in this section are:

### Supported Platforms

– “Unix” on page 57.

## Designed with forethought

FrontBase provides high performance and conformance to both international and de facto standards such as:

SQL 92: FrontBase is the first industrial strength database engine in compliance with this important international standard. This includes full integrity constraint checking built right into the engine.

Unicode: FrontBase uses Unicode 2.0 exclusively for handling all CHARACTER data, while conserving space by using the UTF-8 standard for representation. This provides easy support for mixed client environments and their varying character sets.

Communication: FrontBase uses sockets for communicating with clients, making it easy for developers to support a wide variety of client platforms.

Administration: FrontBase databases can be administrated from any computer on the internet - a normal Web browser is all that is needed.

## Solid Foundation

The underlying truly relational oriented engine provides a solid foundation for dealing with databases very efficiently and without limitations:

– Stringent transaction control
– 100% resiliency against crashes
– Super-fast start-up times
– Terabyte size databases
– Gigabyte size CHARACTER / VARCHAR strings and BLOBs
– Multi-column optimized B-tree indexing with very low overhead
– Host OS filesystem independency

- In-memory caching of tables
- Serializable isolation level with versioned reads
- Row-level locking facilities
- Read-only databases

## Standards Compliance

This section describes international standards to which FrontBase adheres. These include SQL 92, Unicode, TCP/IP, and ANSI C.

FrontBase adheres to several international standards, ensuring that you can leverage these standards when developing and deploying your application with FrontBase.

This section will discuss the following:

- "Full SQL 92 Compliance" on page 22.
- "Unicode Character Representation" on page 23.
- "TCP/IP Client/Server Interaction" on page 23.
- "ANSI C Codebase" on page 23.

### Full SQL 92 Compliance

FrontBase implements the full SQL 92 standard. Great care has been taken to implement all features defined by the standard and implement them as defined by the standard. This document provides several tutorial examples of using SQL 92 with FrontBase but for a comprehensive listing please see our SQL reference document. The ultimate guide to SQL 92 is the standard itself. You can obtain the standard "SQL 92 Standard at ANSI" from ANSI for a fee at:

```
http://webstore.ansi.org/
```

An excellent book by renowned database experts C. J. Date and Hugh Darwen is:

A Guide to the SQL Standard, Fourth Edition

```
http://www.amazon.com/exec/obidos/ASIN/0201964260
```

While it offers an academic approach, it is a very amusing book. Date and Darwen are not fans of many of the decisions that resulted in the final SQL 92 standard.

## Unicode Character Representation

FrontBase stores all character data (including CLOBs) using Unicode. Combined with FrontBase's support for COLLATIONs, this ensures that server-side string comparisons work with character sets other than standard ASCII.

FrontBase's support for Unicode works seamlessly across client platforms. Character data is converted to Unicode on the client side, using the client operating system's native Unicode library. Character data in Unicode format is then passed to the FrontBase server, where it is stored. When a client extracts character data from the server, the client then converts from Unicode format to a format suitable for use on the client platform.

## TCP/IP Client/Server Interaction

FrontBase clients and servers use the standard TCP / IP Internet protocol to communicate. This allows tremendous flexibility in how you design and deploy systems that incorporate FrontBase servers. For example, in a WebObjects deployment, you may connect several application servers to a cluster of FrontBase servers in a server area network (SAN). Or, if you don't require such performance and / or redundancy, you can run WebObjects, Apache, and FrontBase on a single FrontBase server.

## ANSI C Codebase

FrontBase is written in ANSI C. This ensures a stable cross-platform codebase and allows us to move FrontBase to new UNIX-based platforms with minimal effort. This offers FrontBase customers flexibility in development and deployment platforms.

The FBCAccess client library is also written in ANSI C. Source code for FBCAccess is available by special request, so if you need to deploy FrontBase clients on platforms (such as PalmOS or PocketPC) where FrontBase servers have not been ported, you can.

## Key Features

This section introduces key features of FrontBase which make it the ideal database for Internet applications. FrontBase supports most of the important features of "the big boys" and is more standards-compliant than free and open source solutions.

This section will discuss the following:

- Terabyte Databases, Gigabyte Column Values
- FrontBase Security
- Transactions
- Row Level Privileges
- Live Backup
- Caching
- Multi-server Deployment

### Terabyte Databases, Gigabyte Column Values

FrontBase supports terabyte-sized databases, gigabyte-sized character column values, and gigabyte-sized Binary Large OBjects (BLOBs) and Character Large OBjects (CLOBs).

FrontBase implements its own file system within the files used to store the database. The file system in FrontBase consists of up to 2^32 512-byte blocks, yielding over 2 terabytes of usable space. FrontBase 3.x uses a block-size of 2048 bytes for low-level disk access, which, in most cases, reduces the disk accesses with a factor of four . This also enables up to 4 terabyte databases. Future versions of FrontBase will use 64 bit addressing capabilities - the FrontBase development team is currently enhancing the addressing capabilities of the low level disk access to handle peta- to exa-byte databases spread over several physical devices. As character column values, BLOB, or CLOB can occupy up to 2^32 bytes. In practice, large objects will be much smaller so that one does not consume the entire addressable space for the database. Thus, we advertise gigabyte-sized column values.

On platforms where the logical file size cannot exceed the gigabyte range, FrontBase will, when necessary, break its storage for a database into several files that fit within the file system's limits. Since FrontBase maintains its own file system within these files, this partitioning does not impose any limits on sizes of character column values, BLOBs, or CLOBS.

## FrontBase Security

FrontBase uses passwords, encryption, and client IP address checks for security.

### Passwords

FrontBase provides two layers of passwords for protecting access to databases: database passwords and user passwords.

1. Database password

   If the database password is set, a client must send the database password to the server as part of the connection protocol. If the server cannot verify the password, the client connection is closed immediately.

2. User password

   Each database user can have a password. The password is verified by the server when a session is created for that user. If the verification fails, the session is not created. When a session is successfully created, the protection defined by the SQL 92 standard takes over.

3. Password handling in general

   Passwords may be of any length. Passwords are never exposed outside the client software and they are not even in the database. As soon as an application sends a password to the FrontBase client library, a one-way function is applied to generate a password digest. The function will throw away parts of the password so that it is impossible to deduce the password from the digest. The user name is part of the digest, so two users with the same password will not have the same digest. The password digest is transmitted to the server and used for verification in place of the password.

### Encryption

Encryption is used to protect communication channels and data storage. When you create a FrontBase database, you may optionally specify that data stored on the disk should be encrypted. You may also optionally specify that communication channels between the server and its clients must be secure. You must provide an encryption key for each option specified.

1. Data Encryption

   Data stored on the disk is encrypted using a triple DES in cipher block chaining mode on 512 byte blocks. The data store itself is block-oriented with 512 bytes / block, so this effectively encrypts all data, including table definitions, table contents, character data, and BLOBs. The initialization vector depends on the logical position of the block within the system, thus blocks with the same contents will never generate the same cipher text blocks. The key used for encryption of data is a 64 bit initialization vector, and 3x56 bits for the DES encryption.

2. Communication Encryption

   A client and the server are able to establish a secure channel. When a client connects to the server, it receives a public RSA key from the server. The client then generates a set of random session keys: one for outgoing data and one for incoming data. It encrypts those session keys with the public RSA key and sends the results to the server. The server decrypts the session keys sent by the client using its private key. Thus, the client and the server have established a common set of secret keys.

   The algorithm used for encryption of communication data is a triple DES in byte stream mode with cipher text and clear text feed back. The clear text feedback ensures that an error will propagate to all bytes following the error. This ensures simple detection of errors and introduces only a small amount of redundancy.

### IP Address Checks

FrontBase implements black and white lists (also known as the "whiskey list") for determining which clients may connect.

When a client connects to the FrontBase server, the IP address of the client is checked against a black and white list. If the IP address is on the black list, the connection is refused. If the IP address is on the white list, the connection is accepted.

If an IP address is on the white list, you can specify if a secure communication channel is required for that address. In most cases, it will be ok to allow local connections to run without encryption.

FrontBase can also run in a mode where it accepts only local connections. This may be useful when backing up a WebObjects or PHP powered web server, as it ensures that outsiders cannot connect directly to the database.

## Transactions

The “Transaction Logging” on page 93. section describes FrontBase's transaction support.

## Row Level Privileges

FrontBase offers a unique feature called “Row Level Privileges” on page 295. which allows you to specify access privileges for individual rows. Each row is said to be owned by a specific user and belonging to a specific group. Access privileges (SELECT, UPDATE, and DELETE) for a row can be specified for the owner, the group, and the world.

## Live Backup

The “Backup and Restore” on page 112. section describes FrontBase's versioning system which allows it to perform backups of live databases (i.e. while clients continue to access and modify the database).

## Caching

The “Tuning FrontBase” on page 127. section describes FrontBase's caching schemes.

## Multi-Server Deployment

The section “Replication” on page 99. describes FrontBase's replication features and “Clustering” on page 107. for multi-server deployment for both redundancy and enhanced performance.

# Drivers, Adaptors and Plugins

The following is a brief overview of the drivers and adaptors that make FrontBase an extremely flexible database for application development. All of the following are available to download from our 'Downloads' section at www.frontbase.com along with associated documentation.

## ODBC

The ODBC driver is for use on WinNT/2000/ME/98 with any ODBC compliant application. Details about establishing a connection with each driver can be found in the readme files supplied with the drivers.

## JDBC

The JDBC driver provides general connectivity to FrontBase from applications such as Java programs.

## WebOjects 5 plugin

A WebObjects plugin enables connection to FrontBase on any supported operating system. There are separate plugins for Win32 and MacOS X.

## PHP3 and PHP4 adaptors

The API for these adaptors is based on the respective adaptors for MYSQL. Where functions begin with mysql_ in the MYSQL adaptors they begin with fbsql_ in the FrontBase adaptors. The most obvious advantage of this similarity is that porting a PHP application to FrontBase is relatively simple. The PHP 4 driver supports php4.0.6 (release) and php4.0.7-dev (current dev version).

## Perl adaptor

The Perl adaptor is a Perl Database Interface (DBI). This abstraction layer should make porting Perl applications to FrontBase from other databases fairly easy.

## Omnis Studio DAM

A release DAM (Data Access Module) for Omnis Studio 3.x is available from our download page. This DAM provides developers with a rich RAD environment for development with FrontBase. The DAM is currently available for MacOS Classic (8/9) MacOS X and Win32. Documentation is included in the download package.

## REALBasic

These plugins are for REALbasic 3.0 PPC running on MacOS Classic (8/9) and REALbasic 3.0 Carbon running on MacOS X.

## Tcl driver

FrontBase now supports development with Tool Command Language (Tcl), the simple, open source scripting language available cross-platform.

## EOF adaptor

The EOF adaptor allows FrontBase to work with Apple's WebObjects 4.5.

## Migration

FrontBase has tools for importing databases from FileMaker and MySQL.

### FileMaker

"FileMaker" on page 146, migration is a two step process. The tables of a FileMaker database are exported from FileMaker. The exported files are moved to MacOS X, where an application imports them into FrontBase.

### MySQL

The "MySQL" on page 147, migration tool uses JDBC to extract table data from a MySQL database and import it into a FrontBase database.

Other import mechanisms are available via the enhanced import facility "Enhanced Flat-File Import and Export Functions" on page 115.

## FrontBase-Related Processes

There are three main process that run in a FrontBase installation: FBExec, FrontBase, and FBWebEnabler. This section describes these processes, how to determine if they are running, and how to start them.

### FBExec

FBExec acts as a broker between FrontBase databases running on your computer and client software running on your computer or over the network.

When you install FrontBase, your computer will be set up so that FBExec is launched at start-up. In Mac OS X this is achieved via the StartupItems folder of the Library folder. To make sure that FBExec is running on a UNIX-based installation, enter the following in a terminal session:

```
ps axc | grep FBExec
```

If FBExec is running, the system should reply with something like:

```
374 ? S 0:00 FBExec
```

If FBExec is not running, you should start it as follows (assuming you have added <install dir>/FrontBase/bin to your $PATH):

```
FBExec &
```

Be sure to include the ampersand ("&") at the end so that the task doesn't end with your terminal session.

If, however, you are running FrontBase on Windows NT, you can use the Task Manager to check whether FBExec is running. If it is not running, go to the Service Manager and start it. FBExec is installed as a service so that it will start on system startup. If, however, you are running FrontBase on Mac OS X, you can use the Process Viewer utility provided by Apple to check whether FBExec is running.

#### Options

To provide control over the FBExec logging mechanism the FBExec has the invocation options:

-nolog
-log

FrontBase version 3.6 will, by default, not use a log file.

### FrontBase

One instance of FrontBase will be running for each database that is running on your computer. Each FrontBase instance is started directly from the command-line (UNIX installations), by the Service

Manager (Windows NT), or using the FrontBaseManager and FBWebManager tools.

The command-line options for FrontBase, which affect the behavior of a server during execution are described in the“Invocation Options Available” on page 268,. These options can also be set when using the “FrontBaseManager” on page 153, and “FBWebManager” on page 221, tools to start and create FrontBase databases.

## FBWebEnabler

The FBWebEnabler process maintains persistent connections for the FBWebManager system, allowing you to perform administrative functions on your FrontBase server from any web browser, either locally or from another computer.

When you install FrontBase, your computer will be set up so that FBWebEnabler is launched at start-up. In Mac OS X this is achieved by way of the StartupItems folder in /Library. If it stops running, you will likely find out when you attempt to access cgi-bin/FBWebManager through a web browser. “FBWebManager” on page 221, will then indicate that it cannot connect to FBWebEnabler.

If FBWebEnabler is not running, you should start it as follows (assuming you have added <install dir>/FrontBase/bin to your $PATH):

```
FBWebEnabler &
```

Be sure to include the ampersand ("&") at the end so that the task doesn't end with your terminal session.

If, however, you are running FrontBase on Windows NT, you can go to the Service Manager and start FBWebEnabler.

Like FBExec, FBWebEnabler is installed as a service so that it will start on system startup.

## Process IDentification

In order to determine the PID of a FrontBase server process at startup time on MacOS there is a file providing the information. The .pid file is created in the Databases directory:

```
/Library/FrontBase/Databases/<database name>.fb.pid
```

4

# Installation

We offer a different installation of FrontBase for each supported server platform. Each installation contains the FrontBase server and the server administration tools and client libraries available for the platform. This chapter contains the following sections:

## Downloading FrontBase

The following platforms are supported:

– “MacOS” on page 36.

– “Windows” on page 39.

– “Linux” on page 41.

– “Unix” on page 57.

# Installing FrontBase

## MacOS

The Following Administrative Tools are provided:

- "FrontBaseManager" on page 153.
- "FBWebManager" on page 221.
- "sql92" on page 251.

The following Client Libraries are available:

- FBAccess
- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 2.0
- EOF Adaptor

The following Operating Systems are supported:

- "MacOS X and MacOS X Server 10.x" on page 36.

### MacOS X and MacOS X Server 10.x

Following the instructions below, you should be able to install FrontBase in a few minutes. After installing FrontBase, you can read about your new FrontBase server. You may find it useful to print this section before continuing.

1. Log into your MacOS X computer. FrontBase can be installed and run by any user
2. To download FrontBase, please visit www.frontbase.com.

   The archive weighs in at less than 7 MB and should require only a few minutes to download on a 56K modem.

3. If you have a previous version of FrontBase running on your server, the install script will stop FrontBase related processes automatically. Refer to “Administration Tools” on page 92. if you wish to stop things manually.

   If you are also running client software (such as WebObjects, PHP, etc.), you should disable it so it does not access FrontBase during the upgrade process. Some client software may detect that FrontBase is not running and attempt to restart it while you are upgrading! Your installation should only be offline for a few minutes while you upgrade.

4. Once you have downloaded the .dmg file, locate the file according to the preferences of your browser. The .dmg file is expanded with the disk utility to a .pkg file

5. Select the .dmg file and double click. To install the .pgk, select the file and double click and the installer will be launched ready to install FrontBase. The installer will provide you with further instructions.

   By default, the installer will install FrontBase into the /Library/FrontBase directory.

   The installer will attempt to install FBWebManager files into appropriate directories for use by the Apache web server.

6. Verify that FrontBase has been successfully installed and started by entering the following in a terminal window:

```
ps axc | grep FB
```

If the FBExec process (a key FrontBase component) and the FBWebEnabler process (web administration tool process) are running, the system should reply with something like:

```
374 ? S 0:00 FBExec
375 ? S 0:00 FBWebEnabler
```

You can also check these processes on MacOS X by running the ProcessViewer.

If one or both are not running, try to launch them from the command-line:

```
cd /Local/Library/FrontBase
```

(default install location for FrontBase)

```
./bin/FBExec &
./bin/FBWebEnabler &
```

Please note that the FBExec is installed to be automatically started when the computer is restarted. The FBExec is also started as part of the installation process so there is no need to restart the computer after installation. If launching FBExec or FBWebEnabler results in an error, please send e-mail to support@FrontBase.com. We will be happy to help you.

7. If you are upgrading from a previous version of FrontBase and had databases or client software (e.g. WebObjects, PHP, etc.) running prior to starting your upgrade, you should restart those now.

   First, start your databases as described in“Administration Tools” on page 92. Then restart your client software.

8. Adjusting the search path to include the /Library/FrontBase/bin directory will make your life on the command-line simpler.

9. Congratulations, you have successfully downloaded and installed FrontBase! You do not need to restart your computer.

## Windows

The Following Administrative Tools are provided:

- “FrontBaseManager” on page 153.
- “FBWebManager” on page 221.
- “sql92” on page 251.

The following Client Libraries are available:

- FBAccess
- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 2.0
- EOF Adaptor

The following Operating Systems are supported:

- “Windows 2000 / NT” on page 39.

### Windows 2000 / NT

Following the instructions below, you should be able to install FrontBase in a few minutes. After installing FrontBase, you can read about your new FrontBase server. You may find it useful to print this section before continuing.

1. Log into your Windows NT computer as "administrator". FrontBase currently needs to be installed by administrator so that it can run like IIS and similar services. It makes no use of special ports nor does other things that may cause security concerns.
2. To download FrontBase, please visit www.frontbase.com.

   The archive weighs in at less than 7 MB and should require only a few minutes to download on a 56K modem.
3. Use your favorite zip file utility (such as PKZip) to unzip the downloaded archive. Unzipping will yield three executable (.exe) files. Run each of them and follow the installation in-

structions provided. Also included is an additional executable WebObjects plugin which is required for those using WebObjects 5 with FrontBase on Windows 2000/NT.

FrontBase will be installed into the <drive>:/usr/FrontBase directory, where <drive>: is the drive onto which you've installed FrontBase.

Windows NT specific components will be installed into the <drive>:/Program Files/FrontBase Tools and <drive>:/Apple/Library/Frameworks directories.

4. By default, FrontBase expects to be installed on the C: drive. If you have installed it on another drive (e.g. F:), you'll need to add a system environment variable to NT.

   Go to Start —> Settings —> Control Panel, double-click the System icon, and add the FB_HOME_DRIVE environment variable, setting it to the letter of the drive onto which you installed FrontBase (e.g. F:).

5. The installation process will automatically define FBExec and FBWebEnabler as NT services and start them. However, should you ever need to perform this task manually for some reason it is relatively simple. For example, to install and start FBExec you would need to define the FBExec as an NT service. Bring up a shell (e.g. a Bourne or DOS shell) and enter the following command:

```
<drive>:/usr/FrontBase/bin/FBExec -install
```

   Start the FBExec service using the Service Control Manager (Start —> Settings —> Control Panel, double-click the Services icon). Using the service Control Manager, you can also specify that the FBExec service should be started automatically whenever the computer is restarted.

   You can verify that FBExec now is running by launching the "Windows NT Task Manager" (Ctrl-Alt-Del, click on Task Manager).

6. Congratulations, you have successfully downloaded and installed FrontBase! You do not need to restart your computer. You should now spend some time reading the documentation which accompanies your FrontBase server.

## Linux

The Following Administrative Tools are provided:

- “FBWebManager” on page 221.
- “sql92” on page 251.

The following Client Libraries are available:

- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 2.0

The following Operating Systems are supported:

- “RedHat 7.x Linux (x86)” on page 42.
- “SuSE 7.x Linux (x86)” on page 45.
- “YellowDog Linux (PPC)” on page 48.
- “Debian Linux (x86)” on page 51.
- “Mandrake Linux (x86)” on page 54.
- If you are interested in FrontBase for Linux for IBM S390, please contact us at info@frontbase.com.

### RedHat 7.x Linux (x86)

Following the instructions below, you should be able to install FrontBase in a few minutes. After installing FrontBase, you can read about your new FrontBase server. You may find it useful to print this section before continuing.

**NOTE:** RedHat-based Linux installs simply with RPM.

1. Log into your RedHat Linux computer as "root". FrontBase currently needs to be installed by root so that it can run like Apache and other root level services. It makes no use of special ports nor does other things that may cause security concerns.
2. To download FrontBase, please visit www.frontbase.com.

   The archive weighs in at less than 7 MB and should require only a few minutes to download on a 56K modem.
3. If you have a previous version of FrontBase running on your server, the install script (which you'll run in the next step) will stop all FrontBase related processes. You should usually let the script stop these processes for you. Refer to Basic Administration if you wish to stop things manually.

   If you are also running client software (such as WebObjects, PHP, etc.), you should disable it so it does not access FrontBase during the upgrade process. Some client software may detect that FrontBase is not running and attempt to restart it while you are upgrading! Your installation should only be offline for a few minutes while you upgrade.
4. From the terminal, expand FrontBase and run the installation script as follows (note that the actual version number may be higher). If you are installing FrontBase for the first time:

```
rpm -i FrontBase-<version-number>.rpm
```

If you are updating FrontBase, use:

```
rpm -U FrontBase-<version-number>.rpm
```

Please note that in addition to -U, one could use --force as an option to force installation disregarding any errors.

By default, the script will install FrontBase into the /usr/local/FrontBase/bin directory. The script will attempt to install FBWebManager related files into the relevant Apache directories.

5. Verify that FrontBase has been successfully installed and started by entering the following in a terminal window:

```
ps axc | grep FB
```

If the FBExec process (a key FrontBase component) and the FBWebEnabler process (web administration tool process) are running, the system should reply with something like:

```
374 ? S 0:00 FBExec
375 ? S 0:00 FBWebEnabler
```

If one or both are not running, try to launch them from the command-line:

```
cd /usr/local/FrontBase
```

(default install location for FrontBase)

```
./bin/FBExec &
./bin/FBWebEnabler &
```

If launching FBExec or FBWebEnabler results in an error, please send e-mail to support@FrontBase.com. We will be happy to help you.

6. If you are upgrading from a previous version of FrontBase and had databases or client software (e.g. WebObjects, PHP, etc.) running prior to starting your upgrade, you should restart those now.

First, start your databases as described in “Administration Tools” on page 92. Then restart your client software.

7. Adjusting the search path to include the /usr/local/FrontBase/bin directory will make your life on the command-line simpler.
8. Congratulations, you have successfully downloaded and installed FrontBase! You do not need to restart your computer. You should now spend some time reading the documentation which accompanies your FrontBase server.

### SuSE 7.x Linux (x86)

Following the instructions below, you should be able to install FrontBase in a few minutes. After installing FrontBase, you can read about your new FrontBase server. You may find it useful to print this section before continuing.

**NOTE:** SuSE Linux installs simply with RPM.

1. Log into your SuSE Linux computer as "root". FrontBase currently needs to be installed by root so that it can run like Apache and other root level services. It makes no use of special ports nor does other things that may cause security concerns.
2. To download FrontBase, please visit www.frontbase.com.

   The archive weighs in at less than 7 MB and should require only a few minutes to download on a 56K modem.
3. If you have a previous version of FrontBase running on your server, the install script (which you'll run in the next step) will stop all FrontBase related processes. You should usually let the script stop these processes for you. Refer to Basic Administration if you wish to stop things manually.

   If you are also running client software (such as WebObjects, PHP, etc.), you should disable it so it does not access FrontBase during the upgrade process. Some client software may detect that FrontBase is not running and attempt to restart it while you are upgrading! Your installation should only be offline for a few minutes while you upgrade.
4. From the terminal, expand FrontBase and run the installation script as follows (note that the actual version number may be higher). If you are installing FrontBase for the first time:

```
rpm -i FrontBase-<version-number>.rpm
```

If you are updating FrontBase, use:

```
rpm -U FrontBase-<version-number>.rpm
```

Please note that in addition to -U, one could use --force as an option to force installation disregarding any errors.

By default, the script will install FrontBase into the /opt/FrontBase directory. The script will attempt to install FBWebManager related files into the relevant Apache directories.

5. Verify that FrontBase has been successfully installed and started by entering the following in a terminal window:

```
ps axc | grep FB
```

If the FBExec process (a key FrontBase component) and the FBWebEnabler process (web administration tool process) are running, the system should reply with something like:

```
374 ? S 0:00 FBExec
375 ? S 0:00 FBWebEnabler
```

If one or both are not running, try to launch them from the command-line:

```
cd /opt/FrontBase
```

(default install location for FrontBase)

```
./bin/FBExec &
./bin/FBWebEnabler &
```

If launching FBExec or FBWebEnabler results in an error, please send e-mail to support@FrontBase.com. We will be happy to help you.

6. If you are upgrading from a previous version of FrontBase and had databases or client software (e.g. WebObjects, PHP, etc.) running prior to starting your upgrade, you should restart those now.

   First, start your databases as described in “Administration Tools” on page 92. Then restart your client software.

7. Adjusting the search path to include the /opt/FrontBase/bin directory will make your life on the command-line simpler.

8. Congratulations, you have successfully downloaded and installed FrontBase! You do not need to restart your computer. You should now spend some time reading the documentation which accompanies your FrontBase server.

### YellowDog Linux (PPC)

Following the instructions below, you should be able to install FrontBase in a few minutes. After installing FrontBase, you can read about your new FrontBase server. You may find it useful to print this section before continuing.

**NOTE:** YellowDog Linux installs simply with RPM.

1. Log into your YellowDog Linux computer as "root". FrontBase currently needs to be installed by root so that it can run like Apache and other root level services. It makes no use of special ports nor does other things that may cause security concerns.
2. To download FrontBase, please visit www.frontbase.com.

   The archive weighs in at less than 7 MB and should require only a few minutes to download on a 56K modem.
3. If you have a previous version of FrontBase running on your server, the install script (which you'll run in the next step) will stop all FrontBase related processes. You should usually let the script stop these processes for you. Refer to “Administration Tools” on page 92. if you wish to stop things manually.

   If you are also running client software (such as WebObjects, PHP, etc.), you should disable it so it does not access FrontBase during the upgrade process. Some client software may detect that FrontBase is not running and attempt to restart it while you are upgrading! Your installation should only be offline for a few minutes while you upgrade.
4. From the terminal, expand FrontBase and run the installation script as follows (note that the actual version number may be higher). If you are installing FrontBase for the first time:

```
rpm -i FrontBase-<version-number>.rpm
```

   If you are updating FrontBase, use:

```
rpm -U FrontBase-<version-number>.rpm
```

Please note that in addition to -U, one could use --force as an option to force installation disregarding any errors.

By default, the script will install FrontBase into the /opt/FrontBase directory. The script will attempt to install FBWebManager related files into the relevant Apache directories.

5. Verify that FrontBase has been successfully installed and started by entering the following in a terminal window:

```
ps axc | grep FB
```

If the FBExec process (a key FrontBase component) and the FBWebEnabler process (web administration tool process) are running, the system should reply with something like:

```
374 ? S 0:00 FBExec
375 ? S 0:00 FBWebEnabler
```

If one or both are not running, try to launch them from the command-line:

```
cd /opt/FrontBase
```

(default install location for FrontBase)

```
./bin/FBExec &
./bin/FBWebEnabler &
```

If launching FBExec or FBWebEnabler results in an error, please send e-mail to support@FrontBase.com. We will be happy to help you.

6. If you are upgrading from a previous version of FrontBase and had databases or client software (e.g. WebObjects, PHP, etc.) running prior to starting your upgrade, you should restart those now.

First, start your databases as described in “Administration Tools” on page 92. Then restart your client software.

7. Adjusting the search path to include the /opt/FrontBase/bin directory will make your life on the command-line simpler.
8. Congratulations, you have successfully downloaded and installed FrontBase! You do not need to restart your computer. You should now spend some time reading the documentation which accompanies your FrontBase server.

### Debian Linux (x86)

Following the instructions below, you should be able to install FrontBase in a few minutes. After installing FrontBase, you can read about your new FrontBase server. You may find it useful to print this section before continuing.

**NOTE:** Debian Linux installs with a shell script.

1. Log into your Debian Linux computer as "root". FrontBase currently needs to be installed by root so that it can run like Apache and other root level services. It makes no use of special ports nor does other things that may cause security concerns.
2. To download FrontBase, please visit www.frontbase.com.

   The .deb archive file weighs in at less than 7 MB and should require only a few minutes to download on a 56K modem.
3. If you have a previous version of FrontBase running on your server, the install script (which you'll run in the next step) will ask whether you wish to stop FrontBase related processes. You should usually let the script stop these processes for you. Refer to "Administration Tools" on page 92. if you wish to stop things manually.

   If you are also running client software (such as WebObjects, PHP, etc.), you should disable it so it does not access FrontBase during the upgrade process. Some client software may detect that FrontBase is not running and attempt to restart it while you are upgrading! Your installation should only be offline for a few minutes while you upgrade.
4. Once you have downloaded the .deb file, install it as follows (terminal window, logged in as root):

```
dpkg -i FrontBase-3.3.deb
```

The FBWebManager executable is attempted to be installed in the cgi-bin directory (/usr/lib/cgi-bin) of the Apache installation. Images and other files needed by the FBWebManager are installed into /var/www/FBWebManager. If

Apache on your computer isn't installed into /usr/lib/cgi-bin and /var/www, please copy/move the FBWebManager and the associated files to the appropriate places.

5. Verify that FrontBase has been successfully installed and started by entering the following in a terminal window:

```
ps axc | grep FB
```

If the FBExec process (a key FrontBase component) and the FBWebEnabler process (web administration tool process) are running, the system should reply with something like:

```
374 ? S 0:00 FBExec
375 ? S 0:00 FBWebEnabler
```

If one or both are not running, try to launch them from the command-line:

```
cd /usr/lib/FrontBase
```

(default install location for FrontBase)

```
./bin/FBExec &
./bin/FBWebEnabler &
```

Please note that the FBExec is installed to be automatically started when the computer is booted. The FBExec is also started as part of the installation process, i.e. there is no need to restart the computer after installation.

If launching FBExec or FBWebEnabler results in an error, please send e-mail to support@FrontBase.com. We will be happy to help you.

6. If you are upgrading from a previous version of FrontBase and had databases or client software (e.g. WebObjects, PHP, etc.) running prior to starting your upgrade, you should re-start those now.

   First, start your databases as described in “Administration Tools” on page 92. Then restart your client software.

7. Adjusting the search path to include the /usr/lib/FrontBase/bin directory will make your life on the command-line simpler.
8. Congratulations, you have successfully downloaded and installed FrontBase! You do not need to restart your computer. You should now spend some time reading the documentation which accompanies your FrontBase server.

### Mandrake Linux (x86)

Following the instructions below, you should be able to install FrontBase in a few minutes. After installing FrontBase, you can read about your new FrontBase server. You may find it useful to print this section before continuing.

**NOTE:** Mandrake Linux installs simply with RPM.

1. Log into your Mandrake Linux computer as "root". FrontBase currently needs to be installed by root so that it can run like Apache and other root level services. It makes no use of special ports nor does other things that may cause security concerns.
2. To download FrontBase, please visit www.frontbase.com.

   The archive weighs in at less than 7 MB and should require only a few minutes to download on a 56K modem.
3. If you have a previous version of FrontBase running on your server, the install script (which you'll run in the next step) will stop all FrontBase related processes. You should usually let the script stop these processes for you. Refer to Basic Administration if you wish to stop things manually.

   If you are also running client software (such as WebObjects, PHP, etc.), you should disable it so it does not access FrontBase during the upgrade process. Some client software may detect that FrontBase is not running and attempt to restart it while you are upgrading! Your installation should only be offline for a few minutes while you upgrade.
4. From the terminal, expand FrontBase and run the installation script as follows (note that the actual version number may be higher). If you are installing FrontBase for the first time:

```
rpm -i FrontBase-<version-number>.rpm
```

   If you are updating FrontBase, use:

```
rpm -U FrontBase-<version-number>.rpm
```

Please note that in addition to -U, one could use --force as an option to force installation disregarding any errors.

By default, the script will install FrontBase into the /usr/local/FrontBase directory. The script will attempt to install FBWebManager related files into the relevant Apache directories.

5. Verify that FrontBase has been successfully installed and started by entering the following in a terminal window:

```
ps axc | grep FB
```

If the FBExec process (a key FrontBase component) and the FBWebEnabler process (web administration tool process) are running, the system should reply with something like:

```
374 ? S 0:00 FBExec
375 ? S 0:00 FBWebEnabler
```

If one or both are not running, try to launch them from the command-line:

```
cd /usr/local/FrontBase
```

(default install location for FrontBase)

```
./bin/FBExec &
./bin/FBWebEnabler &
```

If launching FBExec or FBWebEnabler results in an error, please send e-mail to support@FrontBase.com. We will be happy to help you.

6. If you are upgrading from a previous version of FrontBase and had databases or client software (e.g. WebObjects, PHP, etc.) running prior to starting your upgrade, you should restart those now.

First, start your databases as described in "Administration Tools" on page 92. Then restart your client software.

7. Adjusting the search path to include the /usr/local/FrontBase/bin directory will make your life on the command-line simpler.
8. Congratulations, you have successfully downloaded and installed FrontBase! You do not need to restart your computer. You should now spend some time reading the documentation which accompanies your FrontBase server.

## Unix

The Following Administrative Tools are provided:

- "FBWebManager" on page 221.
- "sql92" on page 251.

The following Client Libraries are available:

- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 2.0

The following Operating Systems are supported:

- "Solaris" on page 57.
- "FreeBSD (x86)" on page 60.

### Solaris

Following the instructions below, you should be able to install FrontBase in a few minutes. After installing FrontBase, you can read about your new FrontBase server. You may find it useful to print this section before continuing.

**NOTE:** FBManager is not available on Solaris. FBAccess and EOF Adaptor on Solaris require Apple's WebObjects to be installed. FrontBase for Solaris has been generated using SunOS 5.8. For other versions, please contact us at info@frontbase.com

1. Log into your Solaris computer as "root". FrontBase currently needs to be installed by root so that it can run like Apache and other root level services. It makes no use of special ports nor does other things that may cause security concerns.
2. To download FrontBase, please visit www.frontbase.com.

The archive weighs in at less than 10MB and should require only a few minutes to download on a 56K modem.

3. If you have a previous version of FrontBase running on your server, the install script (which you'll run in the next step) will ask whether you wish to stop FrontBase related processes. You should usually let the script stop these processes for you. Refer to "Administration Tools" on page 92. if you wish to stop things manually.

   If you are also running client software (such as WebObjects, PHP, etc.), you should disable it so it does not access FrontBase during the upgrade process. Some client software may detect that FrontBase is not running and attempt to restart it while you are upgrading! Your installation should only be offline for a few minutes while you upgrade.

4. From the terminal, expand FrontBase and run the installation script as follows (note that the actual version number may be higher):

```
cd FrontBase-<version-number>
sh ./install.sh
```

5. By default, the script will install FrontBase into the /opt/FrontBase directory. The script will attempt to install FBWebManager related files into the relevant Apache directories.

6. Verify that FrontBase has been successfully installed and started by entering the following in a terminal window:

```
ps -e | grep FB
```

If the FBExec process (a key FrontBase component) and the FBWebEnabler process (web administration tool process) are running, the system should reply with something like:

```
374 ? S 0:00 FBExec
375 ? S 0:00 FBWebEnabler
```

If one or both are not running, try to launch them from the command-line:

```
cd /opt/FrontBase
```

(default install location for FrontBase)

```
./bin/FBExec &
./bin/FBWebEnabler &
```

If launching FBExec or FBWebEnabler results in an error, please send e-mail to support@FrontBase.com. We will be happy to help you.

7. If you are upgrading from a previous version of FrontBase and had databases or client software (e.g. WebObjects, PHP, etc.) running prior to starting your upgrade, you should restart those now.

   First, start your databases as described in "Administration Tools" on page 92. Then restart your client software.

8. Adjusting the search path to include the /opt/FrontBase/bin directory will make your life on the command-line simpler.

Congratulations, you have successfully downloaded and installed FrontBase! You do not need to restart your computer. You should now spend some time reading the documentation which accompanies your FrontBase server.

### FreeBSD (x86)

Following the instructions below, you will be able to install Front-Base in a few minutes. After installing FrontBase, you can read about your new FrontBase server. You may find it useful to print section before continuing.

**NOTE:** FreeBSD installs with shell script and requires some manual configuration.

**NOTE:** FrontBase for FreeBSD has been generated using FreeBSD 4.2

1. Log into your FreeBSD computer as "root". FrontBase currently needs to be installed by root so that it can run like Apache and other root level services. It makes no use of special ports nor does other things that may cause security concerns.
2. To download FrontBase, please visit www.frontbase.com.

   The archive weighs in at less than 7 MB and should require only a few minutes to download on a 56K modem.
3. If you have a previous version of FrontBase running on your server, the install script (which you'll run in the next step) will ask whether you wish to stop FrontBase related processes. You should usually let the script stop these processes for you. Refer to “Administration Tools” on page 92. if you wish to stop things manually.

   If you are also running client software (such as WebObjects, PHP, etc.), you should disable it so it does not access Front-Base during the upgrade process. Some client software may detect that FrontBase is not running and attempt to restart it while you are upgrading! Your installation should only be offline for a few minutes while you upgrade.

4. From the terminal, expand FrontBase and run the installation script as follows (note that the actual version number may be higher):

```
tar xvf FrontBase-<version-number>.tar
cd FrontBase-<version-number>
sh install.sh
```

5. By default, the script will install FrontBase into the /usr/local/FrontBase directory. The script will attempt to install FBWebManager related files into the relevant Apache directories.
6. Verify that FrontBase has been successfully installed and started by entering the following in a terminal window:

```
ps axc | grep FB
```

If the FBExec process (a key FrontBase component) and the FBWebEnabler process (web administration tool process) are running, the system should reply with something like:

```
374 ? S 0:00 FBExec
375 ? S 0:00 FBWebEnabler
```

7. If one or both are not running, try to launch them from the command-line:

```
cd /usr/local/FrontBase
```

(default install location for FrontBase)

```
./bin/FBExec &
./bin/FBWebEnabler &
```

If launching FBExec or FBWebEnabler results in an error, please send e-mail to support@FrontBase.com. We will be happy to help you.

8. If you are upgrading from a previous version of FrontBase and had databases or client software (e.g. WebObjects, PHP,

etc.) running prior to starting your upgrade, you should restart those now.

First, start your databases as described in “Administration Tools” on page 92. Then restart your client software.

9. Adjusting the search path to include the /usr/local/FrontBase/bin directory will make your life on the command-line simpler.

10. Congratulations, you have successfully downloaded and installed FrontBase! You do not need to restart your computer. You should now spend some time reading the documentation which accompanies your FrontBase server.

# FrontBase Licenses

## Obtaining a License

You can obtain a license string from the FrontBase.com website by registering and supplying the IP or MAC (Ethernet) address and platform of the server. A "license string" and "license check" for your server will be sent to you via e-mail.

### Available Licenses

**E-Starter**
Non-expiring license with Security Checks and Row Level Privileges. You may use the license for deployment, it allows remote access, but has no support for backup.

**E-Business**
Security checks, export/import, table caching, backup, raw device driver, encryption, replication client. This is the minimal license we recommend for deployment in a serious environment where periodic backups are a must.

**E-Enterprise**
Security checks, export/import, table caching, backup, raw device driver, encryption, replication client, replication master, clustering, cluster monitor. Replication / cluster design and configuration assistance included.

**Developer**
E-Enterprise license - without deployment rights. A 6-month expiration, may be renewed every 6 months.

**Embedded**
For 3rd party companies (ISVs/OEMs) to incorporate FrontBase into their solution. Full details can be found in the section “Embedding FrontBase into your own application or solution” on page 302.

### Obtain a Free License

From the FrontBase.com website, navigate to the ‘Buy’ section and then to ‘License’. In here you will find a link to the free ‘E-Starter’ license for FrontBase.

**NOTE:** The free license **does not** support backup and restore.

You can also apply for a free Developer license (an "Enterprise" license without deployment rights) through this link.

### Purchase a License

From the FrontBase.com website, navigate to the ‘Buy’ section and then to ‘License’. In here you will find a link to the various commercial licenses available for FrontBase.

## Installing a License

There are three ways to install the license string:

1. If you are running FrontBase on a platform that includes FrontBaseManager, you can install it from there. Choose **License management** from the **Tools** menu, select the server, and paste the license string and license check which were e-mailed to you.
2. If you have FBWebManager running on the server, connect to it with your web browser and click **License** in the left column. Paste the license string and license check which were e-mailed to you.
3. Although not recommended, you can create the license file yourself using a text editor or, for example cat>LicenseString from the command line. The file is called **LicenseString** and resides in the main **FrontBase** directory. It's format is as follows:

```
<64 char license string>:<16 char license check>
```

## FrontBase Directory Structure

The FrontBase directory contains all the files required for your FrontBase installation. The files and subdirectories of the FrontBase directory are explained in this section.

### Directory Location

The location of the FrontBase directory depends on the platform on which you install FrontBase

| Platform | Path |
|---|---|
| “MacOS” on page 36. | /Library/FrontBase |
| “Windows” on page 39. | <drive>:/usr/local/FrontBase |
| “RedHat 7.x Linux (x86)” on page 42. | /usr/local/FrontBase |
| “SuSE 7.x Linux (x86)” on page 45. | /opt/FrontBase |
| “YellowDog Linux (PPC)” on page 48. | /opt/FrontBase |
| “Debian Linux (x86)” on page 51. | /usr/lib/FrontBase |
| “Mandrake Linux (x86)” on page 54. | /usr/local/FrontBase |
| “Solaris” on page 57. | /opt/FrontBase |
| “FreeBSD (x86)” on page 60. | /usr/local/FrontBase |

A future version of FrontBase and its platform-specific installers will allow you to install FrontBase anywhere. Currently, the Front-Base directory needs to be owned by "root" or "administrator", but that mostly artificial requirement should also disappear in a future version.

### Directory Contents

The FrontBase directory will normally contain the following subdirectories and files:

1. **Databases** is a directory which holds all databases served from the host. For each database, there are two files: <data-

base-name>.fb and <database-name>.fb.log. The former contains the actual data in the database, while the latter contains miscellaneous status and error messages. You can delete a database by deleting the <database-name>.fb file. Of course, you can also log into the database as _SYSTEM and issue a DELETE DATABASE command. Alternatively you could use the FrontBaseManager to delete your databases. On Windows NT, you should make sure that you've removed the database as a service before deleting it.

2. **Collations** is a directory which holds all defined collations (orderings of Unicode characters which can be instantiated as COLLATIONs in your SCHEMAs).
3. **Translations** is a directory which holds all defined translations.
4. **bin** is a directory which contains FrontBase executables. You may want to add the FrontBase/bin directory to your $PATH environment variable for easy access with command-line tools. All examples in this document assume that you have done so.
5. **FBExec.log** is a file to which the FBExec process appends miscellaneous status and error messages.
6. **FBWebEnablerLog** is a file to which the FBWebEnabler process appends miscellaneous status and error messages.
7. **Library** is a directory containing miscellaneous files required to create a new database.
8. **Backups** is the directory that contains backup files.
9. **TransactionLogs** contains all transaction logs in separate sub-directories.
10. **include** contains header files used for development.
11. **lib** contains FrontBase libraries used for development.
12. **Templates** contains html templates for FBWebManager.

## Removing FrontBase

Should you ever need to completely remove FrontBase from your computer, this section describes how to do that.

Removing or uninstalling a FrontBase installation is very platform dependent, hence a description of the process necessary for each supported platform is given below.

A rule of thumb is that :

- rpm packages are removed by using the rpm command.
- deb packages are removed by using dpkg command.
- tar packages are removed by using the deinstall.sh command in the FrontBase installation directory.

## Mac OS X and MacOS X Server 10.x

1. Login as root or use the ‘sudo’ approach as any user.
2. Stop all FrontBase processes and the FBExec process
3. Remove the FrontBase installation using the remove commands (or use the Finder):

```
rm -r /Applications/FrontBaseManager.app
rm -r /Applications/FBUnicodeManager.app

rm -r /Library/Frameworks/FBAccess.framework
rm -r /Library/Frameworks/FrontBaseEOAdaptor.framework

rm /Library/StartupItems/FrontBase*

rm -r /Library/WebServer/Documents/FBWebManager
rm -r /Library/CGI-Executables/FBWebManager

rm -r /Library/FrontBase
```

**NOTE:** Please note that the last rm in step 3 will irreversible remove all your databases.

## Windows NT/2000

1. Login as administrator
2. Stop the FBExec and FBWebEnabler services using the ServiceControlManager application.
3. Remove the FBExec as a service:

```
<drive>:\usr\FrontBase\bin\FBExec -remove
```

4. Remove the FBWebEnabler as a service:

```
<drive>:\usr\FrontBase\bin\FBWebEnabler -remove
```

5. Remove all FrontBase databases as services:

```
<drive>:\usr\FrontBase\bin\FrontBase -remove <database name>
```

6. Repeat step 4 for each database that has been installed as a service (find the list of installed databases using the Service-ControlManager application).
7. Remove the following folders (using e.g. the Windows NT Explorer):

```
<drive>:\Apple\Local\Library\Frameworks\FBAccess.framework
<drive>:\Apple\Local\Library\Frameworks\FrontbaseEOAdaptor.framewo
rk
<drive>:\Apple\Local\Library\Executables\FBAccess.dll

<drive>:\Apple\Local\Library\Executables\FrontbaseEOAdaptor.dll

<drive>:\Program Files\FrontBaseTools
<drive>:\usr\FrontBase
```

**NOTE:** Add/Remove programs contains entries to remove installed FrontBase components, Server, FrameWorks and Tools. By removing the last folder in step 6, you irreversible remove all your databases.

## Solaris

1. Login as root
2. Change directory to the FrontBase installation directory.

```
cd /opt/FrontBase
```

3. Remove the FrontBase installation.

```
sh ./deinstall
```

4. Go to the patent directory.

```
cd ..
```

5. Remove the remains of the installation directory:

```
rm -r FrontBase
```

**NOTE:** By removing the last directory in step 3, you irreversible remove all your databases.

### FreeBSD

1. Login as root
2. Change directory to the FrontBase installation directory.

```
cd /opt/FrontBase
```

3. Remove thr FrontBase installation.

```
sh ./deinstall
```

4. Go to the patent directory.

```
cd ..
```

5. Remove the remains of the installation directory:

```
rm -r FrontBase
```

**NOTE:** By removing the last directory in step 3, you irreversible remove all your databases.

### RedHat

1. Login as root
2. Remove the FrontBase installation

```
rpm -e FrontBase
```

3. Remove the remains of the installation directory:

```
rm -r /usr/local/FrontBase
```

## Mandrake

1. Login as root
2. Remove the FrontBase installation

```
rpm -e FrontBase
```

3. Remove the remains of the installation directory:

```
rm -r /usr/local/FrontBase
```

### SuSE

1. Login as root
2. Remove the FrontBase installation

```
rpm -e FrontBase
```

3. Remove the remains of the installation directory:

```
rm -r /opt/FrontBase
```

## Debian

1. Login as root
2. Remove the FrontBase installation

```
dpkg -r frontbase
```

3. Remove the remains of the installation directory:

```
rm -r /usr/lib/FrontBase
```

**NOTE:** By removing the last directory in step 3, you irreversible remove all your databases.

## Keeping Current

FrontBase is continually improved. If you have obtained FrontBase from a CD-ROM or bundled with your operating system or computer, you most likely are not running the latest version (or reading the latest documentation!).

### Determining the Latest Version

#### Using sql92

Establish a connection to a FrontBase database, then execute the following SQL statement:

```
VALUES(SERVER_NAME);
```

The version of FrontBase currently running as well as the version of FrontBase used to create the database will be returned.

#### FBWebManager

Establish a connection to a FrontBase database, then execute the following SQL statement:

```
VALUES(SERVER_NAME);
```

The version of FrontBase currently running as well as the version of FrontBase used to create the database will be returned.

#### FrontBaseManager

(MacOS X and Windows NT only) The simplest way is to connect to a database and look at the ‘Database’ pane, or you can execute the following SQL statement:

```
VALUES(SERVER_NAME);
```

The version of FrontBase currently running as well as the version of FrontBase used to create the database will be returned.

### Command Line / Terminal

Starting FrontBase with the -v invocation option also details the version of FrontBase in use:

```
FrontBase -v <database-name>
```

## Upgrading your FrontBase server

When you need to upgrade, proceed to section “Installation” on page 35. Choose your platform and download the FrontBase software.

**NOTE:** If you currently have a previous version of FrontBase installed and running on your server, pay careful attention to the upgrade instructions!.

5

# Basic Concepts

This chapter is divided into the following sections:

## SQL 92 Concepts

SQL 92 is the latest official standard for SQL from the ANSI/ISO bodies and as such represents an amalgamam of many years of experience with the language SQL and the various implementations

FrontBase is the first industrial strength database server that implements virtually all of SQL 92. This may not be important to the work you want to do with a database server, but there are at least a couple of issues that you need to be aware of. The concept of SCHEMAs can sometimes cause confusion with other vendors’ terminology. While many products on the market use the word SCHEMA in their literature, very few implement this concept (at least in the SQL 92 sense).

Actually, there is a layer on top of SCHEMAs called a CATALOG. An SQL 92 database is comprised of a number of CATALOGs, each holding a number of SCHEMAs. A SCHEMA can be viewed as the container for a number of objects: TABLEs, VIEWs, DOMAINs, COLLATIONs, etc.

The topics in this section are:

- “Keywords and Identifiers” on page 84.
- “Learning more about SQL 92” on page 84.

## CATALOGs

Currently, FrontBase offers the support for one CATALOG in a database. This CATALOG inherits the name of the database, e.g. if the database was created with the name Movies.fb, the catalog is named MOVIES. This catalog name is always used as the default catalog and thus you don't really need to worry about CATALOGs.

## SCHEMAs

As mentioned earlier, a catalog can contain many SCHEMAs. A SCHEMA is owned by a user (see further below) and only this user can add or drop objects in the SCHEMA. A SCHEMA object can be a table, a view, a domain, ... By means of the GRANT/REVOKE statements, other users can be granted a number of privileges pertaining to the objects within a schema, e.g. the INSERT privilege on a table.

Objects in a SCHEMA can be referenced via so-called qualified names:

```
[[<catalog name>.] <schema name>.] <object name>
```

SQL 92 offers a rich "default for everything" setup, so whenever you want to reference objects in the current SCHEMA, only the <object name> needs to be given.

When a new database is created, two SCHEMAs are created as well: DEFINITION_SCHEMA and INFORMATION_SCHEMA as required by the SQL 92 standard. The DEFINITION_SCHEMA holds all the objects used to maintain all other SCHEMAs. The INFORMATION_SCHEMA holds various objects, which offer access to the objects in the DEFINITION_SCHEMA, and a number of convenience objects. For example if you want to see which TABLEs have been defined in a database, the following SQL 92 statement could be executed:

```
SELECT * FROM INFORMATION_SCHEMA.TABLES;
```

INFORMATION_SCHEMA.TABLES is actually a VIEW defined like:

```
CREATE VIEW INFORMATION_SCHEMA.TABLES AS
   SELECT * FROM DEFINITION_SCHEMA.TABLES;
```

The VIEWs in INFORMATION_SCHEMA are all non-updateable, i.e. an INSERT like:

```
INSERT INTO INFORMATION_SCHEMA.TABLES ....will fail.
```

The DEFINTION_SCHEMA is maintained exclusively by FrontBase and cannot be accessed or manipulated directly by any user.

To create a new SCHEMA (which the current user will then own):

```
CREATE SCHEMA <schema name>;
```

**NOTE: See the SQL 92 standard for the complete syntax of which the example is only but a tiny fragment.**

To make a schema the current schema:

```
SET SCHEMA '<schema name>';
```

**NOTE: (Please note that the schema name is given using a character string).**

To see the list of defined SCHEMAs:

```
SELECT * FROM INFORMATION_SCHEMA.SCHEMATA;
```

To see what is the current schema, the CURRENT_SCHEMA string function is available. Please note that CURRENT_SCHEMA is a FrontBase extension to SQL 92.

## USERs

The concept of users in SQL 92 is relatively simple, but is tied very closely to the concept of schemas.

To access a database, a user name is required, otherwise access is denied (FrontBase offers password protection as an extension to SQL 92).

When a new database is created, a number of user names are also created, among which are: _SYSTEM and _PUBLIC. Both these user names are considered by SQL 92 as special user names, in fact the leading underscore cannot be used in regular identifiers, and is not to be used.

### To create a new user:

```
CREATE USER <user name> [DEFAULT SCHEMA <schema name>];
```

### To change the default schema:

```
ALTER USER <user name> SET DEFAULT SCHEMA <schema name>;
```

The optional <schema name>, which must exist when the user name is created, will be the default schema for the user whenever the database is accessed. If no default <schema name> is given, a schema with the same spelling as the user name is created and used as default (this will happen the first time the user accesses the database).

### To see who is the current user

The USER and CURRENT_USER string functions can be used (USER is simply a shorthand for CURRENT_USER).

**To make a user name the current user:**

```
SET SESSION AUTHORIZATION <user name>;
```

**To see the list of defined user names:**

```
SELECT * FROM INFORMATION_SCHEMA.USERS;
```

## DATE, TIME and TIMESTAMP

SQL 92 has an elaborate time concept which includes the following datatypes:

- DATE
- TIME
- TIME WITH TIME ZONE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE

DATE holds year, month and day, i.e. NO time components.

TIME holds hour, minute, and second.

TIMESTAMP holds year, month, day, hour, minute, and second.

When a TIME or TIMESTAMP literal is inserted into a database, the server's time zone is added to the literal. Example: If the server is running in Denmark and it is August, the server's time zone is GMT+02:00. If TIMESTAMP '1999-08-02 11:49:00' is inserted, the literal is thus adjusted with +02:00. If, however, TIMESTAMP '1999-01-02 11:49:00' is inserted, the literal is adjusted with +01:00 (because there is no daylight savings in January).

If you want to be in full control over the time zones, you should use the TIMESTAMP WITH TIME ZONE datatype.

```
Example: TIMESTAMP '1999-08-02 11:49:00-08:00'.
```

The same comments apply to TIME and TIME WITH TIME ZONE.

### Keywords and Identifiers

SQL 92 has a very extensive set of keywords and you may run into some surprises when selecting the spelling for an identifier of yours. It may very well collide with the spelling of an SQL 92 keyword. Please also note that an identifier cannot begin with an underscore.

There are a couple of ways to reduce the "collision problems":

- There is no keyword in SQL 92 ending with an underscore, e.g. it will be perfectly legal to use SELECT_ as an identifier.
- By enclosing the identifier in double quotes, essentially any spelling can be used as an identifier, e.g. "SELECT" is a legal identifier.
- You could, for example, avoid collisions by ending table names with "_tbl", and field names with "_fld" or "_col"

SQL 92 is case insensitive, .e.g Movies as an identifier is considered identical to MOVIES.

### Learning more about SQL 92

You can always get hold of a copy of the standard itself from either ANSI or ISO, but the standard is not really aimed at users. A better way to get acquainted with SQL 92 is to buy "A Guide to The SQL Standard, Fourth Edition" by Chris J. Date and Hugh Darwen. This book explains all concepts and constructs of SQL 92, sometimes with quite an academic viewpoint, but nonetheless very complete and absolutely readable and understandable.

The book is published by Addison-Wesley and has ISBN #: 0-201-96426-0. An easy way to order this book is to go to www.amazon.com, but your local bookstore will most likely be able to help you as well.

## Understanding Transactions

This section briefly describes the database concepts of transactions, isolation levels, locking discipline, and updatability; all concepts that are used for controlling simultaneous access to a FrontBase database

## Simultaneous access

One of the basic features of a database server is to provide users with parallel access to shared data, thus the database server must ensure that updates made to a database are performed in an orderly manner such that data is not corrupted or lost.

## Transactions

A transaction is used to control users' access to the database. A user cannot access the database without a transaction, and all operations are performed in the context of a transaction. All the changes made to the database by a user in the context of a transaction are made visible to other users when the transaction is committed. A transaction is, as seen from the outside, one single atomic operation.

During its existence a transaction may fail, and you cannot commit a transaction that has failed, the only action to take is to start all over again (with the hope that the transaction will not fail the next time around). A database server can, in principle, fail transactions at will, but a good server will only fail a transaction for a good reason. The only good reason is an access conflict such as a deadlock.

When a transaction is created it is assigned an isolation level, an updateability, and a locking discipline. The isolation level determines how isolated a transaction is from other transactions, the updateability determines if the access is read only or read write, and the locking discipline determines the type of lock used to synchronize access to the database.

## Updateability

The updateability can be READ ONLY or READ WRITE, a transaction which has the updateability of READ ONLY cannot modify the database. The updateability is quite important because transactions that are READ ONLY does not interfere.

## Isolation level

SQL92 defines 4 isolation levels:

– READ UNCOMMITTED

– READ COMMITTED

– REPEATABLE READ

– SERIALIZABLE

and FrontBase defines one more:

– VERSIONED

Users accessing data in the database may experience the following phenomena:

### Dirty reads

One transaction is writing some data to the database, a second transaction is then reading that data, but the first rolls the transaction back. The second transaction has now read data that not really existed.

### Non-repeatable Read

A transaction read a row. A second transaction updates the values of the row and does a COMMIT. If the first transaction reads the row again it will get a different result.

### Phantom

One transaction selects some data in the database, a second transaction updates or inserts rows that satisfy the predicates that the first transaction used. The second transaction is committed. If the first transaction performs the select again, it would get a different result.

The table below shows which phenomena a given isolation level permits:

| | **Dirty Reads** | **Non-repeatable** | **Phantom** |
|---|---|---|---|
| READ UNCOMMITTED | YES | YES | YES |
| READ COMMITTED | NO | YES | YES |
| REPEATABLE READ | NO | NO | YES |
| SERIALIZABLE | NO | NO | NO |
| VERSIONED | NO | NO | NO |

The amount of data that is locked is reflected by the isolation level. With READ UNCOMMITTED nothing is locked and the isolation level is actually upgraded to READ COMMITTED in FrontBase. With READ COMMITTED nothing is locked, but only data that is committed is read. REPEATABLE READ locks rows as they are selected, in other words, immediately as part of the query execution. SERIALIZABLE locks the whole table. In FrontBase row locks are not escalated to table locks at any point.

The VERSIONED isolation level is only valid for READ ONLY transactions and will keep the current version of the database for the duration of the transaction, in effect creating a snapshot of the database at the time the transaction was created. Other transactions may modify the database, but the changes will not be visible to the VERSIONED transaction. Any number of VERSIONED transactions can be ongoing at the same time, sharing committed versions of the database.

## Locking Discipline

In addition to updateability and isolation level, FrontBase introduces the concept of locking discipline. The locking discipline has the following values:

– PESSIMISTIC

- DEFERRED (also called upgradeable)
- OPTIMISTIC

PESSIMISTIC locking assumes that the given object will be changed, i.e. a transaction must wait until the object is available (unlocked). When a transaction is waiting there is a possibility for deadlocks. Deadlocks are detected and broken by failing one of the transactions causing the deadlock. FrontBase detects a deadlock when two or more connections are competing for exclusive access to the same resourses and when none of them can continue before they have acquired all the requested resourses. The connection that causes the deadlock to become a problem is the transaction that will get rolled back.

OPTIMISTIC locking assumes that a given object isn't changed by other transactions, and any changes are performed without further ado. When the transaction is committed, it is checked that the accessed objects weren't changed during the transaction, if they were changed, the commit fails.

DEFERRED is a version of PESSIMISTIC locking which assumes that objects are only read, initially the lock is a read lock and if the object is updated the lock is upgraded to a write lock.

## Locking and EOF

EOF is using OPTIMISTIC locking, a transaction is started by, for example, a fetch and is terminated when changes are saved. EOF only checks the objects that are going to be updated, which is not entirely correct, all objects that have been accessed should be checked. The user loads a number of rows, does some calculations and stores the result in a row. All the rows used for the calculation may be changed, which is undetected, and the result would be wrong.

FrontBase does implement OPTIMISTIC locking. The limited check problem with EOF (outlined above) can be solved by allowing nested transactions on the client, start a transaction when the user selects objects and commit it when the user saves the changes. Actual ROLLBACK and COMMIT should be made available to the user. If the server implements the locking, the locking in the EOF is redundant, and snapshots etc. may be turned off.

6

# Administration

FrontBase is designed to be "zero-maintenance", and for many applications, starting up a database may be all you need to do. More advanced or critical applications may involve managing users, backing up, keeping your installation current, tuning, and other tasks. These tasks are designed to be simple and straight-forward, and many of them may be carried out from the command line of a terminal window. These are explained in this chapter. FrontBase furthermore offers three primary tools for administering your FrontBase server detailed overleaf. The examples in this chapter assume that you are logged in as the user owning the FrontBase installation and that the FrontBase /bin directory is in your $PATH.

This chapter contains the following sections:

# Administration Tools

The “sql92” command-line tool, the web-based “FBWebManager” tool, and the “FrontBaseManager” application (for MacOS X and MacOS X Server 10.x) are each described in separate chapters as indicated below:

## FrontBaseManager

“FrontBaseManager” on page 153, introduces the application that lets you monitor and administer local and remote database servers. It is available for MacOS X installations of FrontBase. Windows installations use the original FBManager.

## FBWebManager

“FBWebManager” on page 221, introduces the FBWebManager web-based application, which is available for all installations of FrontBase and is accessed through any browser that supports dynamic HTML (aka JavaScript).

## sql92

“sql92” on page 251, introduces the sql92 command-line tool, which is available for all installations of FrontBase.

## FBScriptAgent

FBScriptAgent is a “scriptable” application which forwards commands to the FrontBase database. In addition to sending raw SQL commands, FBScriptAgent can start, stop, create, and delete FrontBase databases. FBScriptAgent exposes a great deal of database functionality that has previously only been available through more complex database client libraries. A user can now access the powerful features of FrontBase simply by keying in a few AppleScript commands.

A clever design aspect of FBScriptAgent is that it allows the scripter to deal with meta-data and data-handlers. For example, if a database fetch were to select 1,000,000 records, this would be difficult for most applications to deal with if all the records were returned. Instead, FBScriptAgent fetches return a meta-data object by which a user can determine how many records were selected, any errors or warnings generated by the select, the execution time, the column names, etc. The meta-data also includes a fetch “handle”, which may used to actually fetch the rows, in batches if required. A hallmark of the intelligent engineering of FBScriptAgent is how easily it allows a user to work with massive amounts of data.

The application, with documentation and examples, is posted in the ‘Download’ section at http://www.frontbase.com

## Transaction Logging

By default, FrontBase (version 3.x and later) maintains a transaction history log. This is a complete list, in the form of SQL statements, of all transactions that have altered data or structure in the database. Such a log therefore provides a complete history of the database development and it is possible to re-create a database from a particular starting point by essentially replaying the SQL statements of the transaction log. Such a particular starting point is either the creation of the database or the point where a backup of the database was created.

The concept of transaction logging in FrontBase serves several purposes:

1. It provides an extra level of security against loss of data
2. It enables the database server to decouple its client interface handling from its handling of disk operations (i.e. the client handling does not have to wait for disk write operations to be completed)
3. It serves as the basis for database clustering and replication.

The second point above implies that the transaction log may be "ahead" of the actual database contents at any given point in time. When FrontBase is started for an existing database, it therefore automatically examines if the transaction log contains transactions that have not been committed to the database and in this case the database is brought up to date.

This mechanism of automatically bringing a database up to date with respect to its transaction log is also useful in connection with clustering and replication. The following sections explain the various elements of transaction logging in FrontBase:

## Implementation

The transaction log of a database is found in the TransactionLogs directory of the FrontBase installation: TransactionLogs / <dbname>.

This directory contains one or more transaction log directories, each of which are named: L_yyyy_mm_dd-hh_mm_ss, identifying the point in time when the directory was created. The names should therefore provide the correct ordering of the transaction log directories when more than one exists.

A transaction log directory contains a file named transactions.log plus possibly files named tttttttt.sql (where tttttttt is a transaction number). The latter files contain particularly large transactions. FrontBase attempts to keep the log of a transaction in memory while it is ongoing, only writing it to the log file when the transaction is committed. However, there is a limit on the size of the in-core transaction (currently 1 MB) which, when exceeded, implies that the transaction is written to a log file of its own.

There is a limit to the (combined) size of files kept in a transaction log directory which, when exceeded, implies that a new transaction log directory is created. This limit is 512 MB by default, but may be changed by the user.

When a transaction log is used to update a database from a particular starting point, it is obviously necessary that the starting point is well defined. It is therefore such that when a backup of a database is created, a change in the transaction log directory will take place indivisibly. The names of the backup file and the new transaction log directory will be identical, except for the two first characters (B_ and L_, respectively).

## Administration

The use of transaction logging is optional, but it is the FrontBase default and recommended for added safekeeping. Notice that transaction logging is required for running FrontBase in replication and clustering contexts.

When transaction logging is enabled, FrontBase permits the user interface to continue even before a transaction has been physically written to the file system. This may mean that the transaction log is "ahead" of the database as found on the disk, and in this situation the transaction log is vital for preserving transactions. The state of the database on the disk is always consistent - i.e. it reflects the latest transaction actually written to the disk - but it may actually lag behind the state of the transaction log.

When the FrontBase server is started on an existing database, it is automatically verified that the database is up to date with respect to the transaction log (if present), and if not, it is brought so. This implies that the latest transaction log directory is definitively always relevant. In general, it is difficult to predict exactly when changes to the transaction log directory (brought about by the log size limitation) takes place, and it is also difficult to predict just how far the transaction log is ahead of the database. Therefore it is safest to state that transaction log files are relevant back to the last well-defined starting point: Database creation or backup point.

It is never necessary to remove transaction log files for any other reason than disk space economy.

The following simple rule dictates the administrative handling of transaction logging:

A database backup indivisibly creates a new transaction log directory and makes all previous transaction log directories obsolete.

This means that with respect to the just created backup, all previous transaction log directories may be deleted. How many generations of backup and accompanying transaction logs an installation wishes to keep before physically deleting them is an individual matter.

## SQL syntax

The following SQL statements have been introduced for handling transaction logging:

```
CREATE TRANSACTION LOG;
```

Enables transaction logging (default). If transaction logging was disabled, a new transaction log directory would be created

```
DROP TRANSACTION LOG;
```

Disables transaction logging (not recommended)

```
SET TRANSACTION LOG LIMIT <integer-expr>;
```

Sets the size of a transaction log directory to n MB, where n is the value specified by <integer-expr>

```
SHOW LOGS [ALL | <integer-expr> ];
```

Shows a summary of the n latest transaction log directories, where n is the value specified by <integer-expr>. A value of 0 (or ALL) implies all existing transaction log directories. The absence of <integer-expr> implies 1 (i.e. the newest)

```
SWITCH TO NEW TRANSACTION LOG;
```

Creates a new transaction log directory

```
WRITE DATA;
```

or for FrontBase 3.5b and later:

```
WRITE BACKUP [ TO <path name expr> ];
```

Creates a backup of the database and indivisibly creates a new transaction log directory. This way, the backup provides a well-defined starting point for the transaction log just initiated.

## Options

Options can be specified when starting FrontBase via the command-line. The following invocation options for the FrontBase server have been introduced:

-tlog=no | yes

Overrides, at the time of starting the server, the transaction logging mode stored with the database; this mode, however, is not permanently changed.

-keeptlog

Prevents the server from clearing the transaction log, if one exists.

Neither of these options are useful during normal (nominal) operation.

## Utilities

The following utilities serve to inspect the state and contents of the transaction log for a given database. They can be found in your /FrontBase/bin directory:

```
FBTLog [-d <transactionlog-directory>] <database-name>
[<transaction-number>]
```

Lists transactions in a transaction log, by default starting from the oldest transaction log directory. If specified, <transaction-number> identifies the oldest transaction number that should be listed, and the -d option identifies the specified transaction log directory as the starting point.

```
FBTLogs [-s <file-count>] <database-name>
```

Shows a summary of the n latest transaction log directories, where n is the value specified by <file-count>. A value of 0 (default) implies all existing transaction log directories.

# Replication

Replication and Clustering are useful for both redundancy (protect from downtime when a server becomes unavailable) and load distribution (better response time in heavy load situations).

## Replication

FrontBase supports replication of one master database to several read-only clients which are updated when the master is updated. The clients have a fairly loose coupling to the master; a client may, for example, be out-of-date when it has been off-line for a while. When a client comes on-line again it will be updated with the transactions that were executed while it was off-line.

As the master database and the clients are only loosely coupled, and the master server does not have any overhead associated with the clients whatsoever, a replication of a master can be used to off-load time consuming backup and report writing tasks from the master, and a client may serve as a backup by itself. Apart from that, replication may be used for the more traditional distribution of readonly copies of a database.

### Introduction

As the clients are all read-only, the migration of changes from the master to the clients does not require elaborate synchronization, such as the two-phase commit protocol required for clustering, but can be controlled simply by counting the transactions that have been executed on the master and on the clients. A transaction can be committed on the master without any synchronization with the clients, thus the actual replication mechanism can be moved outside the server as long as the server provides the necessary serialization of the transactions and also provides a representation of the changes external to the server. This external representation is provided by the database transaction log.

### Overall design

The replication system consists of:

### The Master Database Server

In order to enable replication the FrontBase server must be started with the -rmaster option:

```
FrontBase -rmaster <other-options> <master-database-name>
```

### The Replication Process

The replication of a master database is done by a separate process, the replication daemon, which reads the transaction log that the master maintains. The transaction log contains SQL for all modifications to the master. When a transaction is committed, the SQL for the transaction is written to the transaction log. Only the SQL that actually modifies the database is written to the log, so it is primarily insert and update statements.

The replication daemon monitors the transaction log of the master and whenever a transaction has been written completely to the log file, this SQL is executed by the replication daemon on all the clients that are accessible. The monitoring of the transaction log is basically a polling with a configurable interval which controls the reaction time of the replication daemon. In order not to have any interference among the clients, replication to each client is performed in its own thread of control, such that potential communication delay with one client does not affect any other client.

The replication daemon comes with a small tool for status reporting and client creation, addition and removal, and the transaction log comes with a tool for inspection.

The replication process is started with the command:

```
FBReplicator [-v] [-s] [-i <seconds>] [-d <transactionlog-
directory>] <database-name>;
```

- -v

  Verbose. The replication daemon will log events to stdout.
- -s

  Silent. The replication daemon will not log events.
- -i <seconds>

  Interval. The polling interval determines how often the replication daemon will examine the transaction log of the master. The replication daemon will only sleep the specified amount of time when no transactions are pending replication. The default value is 10 seconds.
- -d <transactionlog-directory>

  The transaction log directory specifies the transaction log directory to be used for replication. In the normal case it is not necessary to specify this option.

The replication daemon must run on the same host as the master database as the replication daemon examines and reads the transaction log of the master.

### The Clients

The replication clients must be started with the -rclient option:

```
FrontBase -rclient <other-options> <client-database-name>
```

which disallows the users of the replication clients to update the database. The client database must have been created in a replication setup with the master, so that the master and the client share a common history. In other words, the client database cannot be created prior to setting up the replication scenario.

### FBTLogs

FBTlogs[ -s <directory-count> ] <database-name>

The FBTlogs tool displays the status of a transaction log for a database. As the tool accesses the log directly, it must be run on the same host as the database server.

**Options**

-s <directory-count>

Specifies the number of transaction log directories to be considered. Default is all.

**Arguments**

<database-name>

Name of the database involved.

### FBTlog

FBTlog[ -d <transactionlog-directory> ]<database-name>[ <transaction-number> ]

The FBTLog tool lists the contents of a transaction log for a database. As the tool accesses the log directly, it must be run on the same host as the database server.

**Options**

-d <transactionlog-directory>

Specifies the first transaction log directory to be considered. Default is the oldest.

**Arguments**

<database-name>

Name of the database involved.

[ <transaction-number> ]

List transactions starting with the specified number.

### The FBRAccess tool

FBRAccess <command> <arguments>

FBRAccess is a command line tool for managing the replication daemon. For this tool, the name of a database may generally be written as: <db>[@<host>], where <db> is the name of the database and <host> is the name of the machine hosting it. The absence of the host specification implies localhost, the current machine. The tool implements the following commands:

```
FBRAccess add <master-host> <master-database> <client-database>[<database-
password>[<system-password]]

FBRAccess create <master-database> <client-database>[<database-password>
[<system-password>]]

FBRAccess remove <master-database> <client-database>

FBRAccess status <master-database>
```

– **Create** command adds a client to the master. If the client database does not exist, it is created and is initialized with a copy of the master. If the client database exists, the replication process will start to update the client server. If you are dealing with large databases and want to create a new client it may be better to provide the copy by other means than letting the replication process perform the task as it may be very time and resource consuming. The optional passwords must be the database password and the password for the _SYSTEM user on the client database, if specified. The replicator needs the passwords to establish the connection to the client, but it is only necessary to specify the passwords if they are different from the current passwords for the master.

– **Add** command works as the Create command except that it will not create a client database that does not exist and it will not start the client if it is not running.

– **Remove** command removes the client from the replication list, the replication process will simply stop sending updates to the client, but the server is left running.

– **Status** command returns status information of the clients. The first line prints the host name and the database name, and the current transaction number for the master. For each client a line is printed with database name, host-name, current transaction number, and status for the client.

### How to create a new client

The simplest way to create a client is to use the FBRAccess create command, but that may not always be desirable as it includes copying the master database to the client host, which may be time consuming for larger databases.

If you are working on hosts of the same endian type, you may copy the database from the master host to the client; make sure, however, that the master database is not running when the file is copied. If you do not want to stop the master database you can create a backup with the WRITE DATA command, move the backup to the client host and restore it there.

If you are working on hosts of different endian type, you must create a flat file version of the database with the WRITE ALL command (see "Enhanced Flat-File Import and Export Functions" on page 115.), which creates an endian neutral ascii representation of the database contents. Move that to the client host and load the flat file into the database.

If you have a complete transaction log (which you would normally only have for young data bases), you can simply start the client database.

When the client database has been established, just add the client to the list of clients using FBRAccess.

### Setting up Replication

A terminal oriented procedure for creating a replication setup for a new master database and an associated client could be the following:

1. Start the master database e.g. masterDB on machine masterhost:

```
   FrontBase -create -rmaster masterDB &
```

2. Commit some transactions using the sql92 tool, e.g:

```
CONNECT TO masterDB USER _SYSTEM;
CREATE USER u1;
CREATE SCHEMA AUTHORISATION u1;
ALTER USER u1 SET DEFAULT SCHEMA u1;
DISCONNECT CURRENT;
QUIT
```

3. Start the replication daemon:

```
   FBReplicator masterDB &
```

On the other machine (e.g. clienthost):

4. Start the client database (e.g. clientDB)

```
   FrontBase -create -rclient clientDB &
```

On the first machine:

5. Establish connection from masterDB to clientDB with the following.

```
   FBRAccess add masterDB clientDB@clienthost
```

The master-client relationship is established. Inspect, for example, the transaction logs of the two databases by doing:

6. On masterhost:

```
   FBTLog masterDB
```

On clienthost:

```
   FBTLog clientDB
```

If, on the other hand, you have an existing master database that you wish to replicate to a fresh client:

- Omit -create from point 1 above

- Omit point 2 from above

### Replication and Passwords

The replication daemon is in all respects an ordinary user for the replication clients, using the standard client library. When it establishes contact to a client, it does that as the _SYSTEM user, thus is required to know the _SYSTEM and database passwords in order to establish a connection. The replication daemon keeps track of the current passwords for a given client, so you may change the passwords on the master even when a client is off-line.

If you create a new client by some sort of copying as described above the replication daemon does not know the passwords for the database, and consequently cannot connect to the client. The FBRAccess tool therefore allows you to specify the current passwords when adding/creating the client.

### Database identification checks

Each FrontBase database has a 48-bit unique identification. The replication daemon requires that the unique identification of the master database is identical to the unique identification of each client. This check ensures that the master and the clients are related, sharing the same history.

### Replication commands

The sql92 command line tool now includes commands for controlling and maintaining a replication master and its clients. The sql92 commands assume that you have a session to the master server, and the commands work on the corresponding replicator. All related commands can be found in the SQL commands section - "Start Replicator" on page 264.

**Example:**

The example create a master database with one client.

```
create database master options -rmaster;
connect to master user _system;
start replicator;
create client client;
show clients;
```

```
table t0 ( c0 integer );
insert into t0 values 1;
table t0;
disconnect all;
connect to client user _system;
table t0;
disconnect all;
```

## Clustering

A replication cluster is a number of database servers which implements one database, a client can connect to any server in the cluster and access and update the database

### Background

The reasons for clustering databases can be best described as follows:

1. Enhanced data security

   Each database server in the cluster holds a complete copy of the database, so as long as one machine remains intact no data will be lost.

2. Load distribution

   A typical client application will execute read transactions with a rate that in orders of magnitude are larger than the rate of read / write transactions, and a read transaction will only access the cluster member it is connected to, thus the load can be distributed.

3. Hot spare

   When a client looses a connection to one database in the cluster, it can simply connect to another, the only data that may be lost is the outstanding transactional data.

## Implementation

1. Two Phase Commit

   When a client modifies a member of the cluster, the other members of the cluster must be updated before the modifications can be committed. When the members of the cluster is updated the transaction is committed with the so-called two phase commit algorithm, which ensures that the transaction is committed on all the cluster members or on none.

2. Cold starting a cluster

   When the first member of a cluster is started, it is assumed that that server has the latest version of the database. As the server is the only one in the cluster it is not necessary to synchronize with the cluster.

3. Adding members to the cluster

   When the remaining members of the cluster are started they will determine the members of the cluster, and roll the new member forward in order to synchronize with the cluster. If it is not possible to synchronize the new member and the cluster the new member is aborted. When the new member is synchronized it will allow clients to connect and it will receive updates from the rest of the cluster members.

4. Removing members from the cluster

   Removing a member from the cluster is simple: just stop / kill that server.

5. Clients connection to the Cluster

   When a client wants to connect to a cluster it can try to connect to one of the members, and if that fails try the second and so forth until either a usable server is found or all the servers have been tried and failed, in which case it is impossible to connect to the cluster.

### Preliminary Checks

Check that the servers / machines to be used in the cluster can be accessed by the network and by the internet. If there is a problem accessing the machines, contact the system administrator, there may be a problem with DNS set up. If you know that there are database clusters already running on the machines to be used, then you should be OK.

In Mac OS X server, go to computer settings and select network. Go to the DNS section and check out the “Search Domains” box. If there is no entry for the domain to be used for the project, then enter the domain you require, for example “hotroof.com”. This is not essential, but may just make life a little easier later on.

### Clustering A New Database

The following will detail the process of establishing clustering for a new database.

1. Start the first Cluster Member on the Primary Machine

Log on to the machine1 and execute the following command line:

```
FrontBase -rcluster=newdb@machine1,newdb@machine2 newdb
```

This should give a response like the following:

```
2002-01-30 12:33:35 FrontBase Server - <version-number>
2002-01-30 12:33:35 Bootstrapping new database: /opt/FrontBase/
Databases/newdb.fb
2002-01-30 12:33:35 Bootstrapping new database
2002-01-30 12:33:35 Bootstrapping DEFINITION_SCHEMA done
2002-01-30 12:33:36 Bootstrapping INFORMATION_SCHEMA done
2002-01-30 12:33:36 Cleared Transaction Log directory: /opt/
FrontBase/TransactionLogs/newdb
Jan 30 12:33:36 2002 [12630] Synchronized at transaction: 0
```

The FrontBase server for newdb on machine1 is now running, and is willing to accept as a cluster member a server for newdb running on machine2.

2. Start the second Cluster Member on the Secondary Machine

Log on to the machine2 and execute the following command line:

```
FrontBase -rcluster=newdb@machine1,newdb@machine2 newdb
```

This should give a response like the one above, with the addition of the following:

```
Jan 30 12:44:12 2002 [12640] Connected to newdb@machine1
Jan 30 12:44:12 2002 [12640] Inherits database ID from
newdb@machine1
```

Also, on machine1, you should get the following output:

```
Jan 30 12:44:12 2002 [12630] Connected to newdb@machine2
```

There are now two members of the cluster, and they are set up to communicate with each other.

If you need more than two members, simply expand the -rcluster option:

```
- rcluster=newdb@machine1,newdb@machine2,newdb@machine3
```

and repeat point 2 on each of the machines involved.

You may also later on add a new cluster member (say, on machine7) to an already existing cluster:

Log on to the machine7 and execute the following command line:

```
FrontBase -rcluster=newdb@machine1 newdb
```

The server will derive the full list of cluster memebers from the specified member of the cluster, so you do not have to specify the complete list.

It is possible at any time to re-start a cluster member, that for one reason or another has been stopped, using the above command lines.

### Clustering an Existing Database

In order to set up clustering for an already existing database, a copy of the database must be first established on the cluster machine. This may be done by any means described in the section “Backup and Restore” on page 112.

Then the procedure described above may be followed.

### Database Identification Checks

Each FrontBase database has a 48-bit unique identification. All members of a database cluster must share the same unique database identifier, signifying that they share a common history. This property is checked whenever a new member is added to a cluster.

## Backup and Restore

FrontBase allows you to backup a complete database to flat files. It allows you to restore content data from ASCII flat files directly into your database. FrontBase can even backup a "live" database.

You should backup critical databases on a regular basis to protect from loss due to hardware failures, lost files, or other catastrophes. You will also need to backup when moving your FrontBase databases to a new server or when sending them to FrontBase for diagnosis.

**NOTE:** We have seen that Mac OS X Server 1.x, when swapping under heavy load, can cause unfortunate problems in the file system. These file system problems can impact the FrontBase database files. We highly recommend upgrading to Mac OS X Server 10.x in order to avoid these potential problems.

### Overview

As the measures you can take to safeguard your databases are identical on all platforms and because it makes sense to protect your data in general, we have produced this write-up of the possible strategies you can pursue.

This section contains the following:

## Copy the Database Files

This strategy is very simple, basic, and certainly better than nothing.

FrontBase stores its database files in <install-path>/FrontBase/Databases. You can backup the contents of this directory by using your preferred backup utility. The tar utility is available on all Linux/Unix systems, it needs to be executed from the command line:

```
tar cvf <destination file> <install path>/FrontBase/Databases
```

Example for Mac OS X Server 10.x (the backup is created in current directory):

```
tar cvf fbbackup /Library/FrontBase/Databases
```

**NOTE:** You need to STOP ALL FrontBase servers before making the backup.

## Backup of A Live Database

With FrontBase you can create a backup of a database while it is running. The backup is non-obtrusive and the database can continue to be updated while the backup progresses.

You need to connect to a database as the user _SYSTEM and issue the following SQL statement:

```
WRITE DATA;
```

**NOTE:** The syntax has been modified for FrontBase 3.5b and later as follows:

```
WRITE BACKUP [ TO <path name expr> ];
```

where <path name expr> is a general expression, of a character type, that is to specify the full pathname of where the backup file should go.

On most systems, the task of starting the backup can be automated. On Linux / Unix systems the cron utility and the sql92 command line tool can, together, automate the process.

## Restoring a Database

**TIP: As an extra safeguard measure, prior to doing a restore from a backup, we strongly recommend you make a copy of the "old" database files in <install path>/FrontBase/ Databases first.**

To restore a database from a backup, you need to do so from a command line (logged in as root):

**Syntax:**

```
FrontBase -create -restore <database name>
```

## Export Into Flat-Files

FrontBase can export an entire database, schema definitions and contents, into flat files in ASCII format from which a database can later be rebuilt.

Two purposes of the flat-file export / import functionality are:

1. Backup and Restore of a database in ASCII form.

2. Porting of a database between platforms of different "endian" type (big-endian / little-endian).

The following SQL 92 statements will respectively export and import a complete database:

```
WRITE ALL OUTPUT(DIR='path to export directory', CONTENT=TRUE);
```

```
SCRIPT <path to export directory>/schema.sql;
```

This approach is described fully in the section “Enhanced Flat-File Import and Export Functions” on page 115.

### Replication

The replication approach is fully described in the section “Replication” on page 99. Replication is a way to have a master database, which receives all updates etc., and N mirrored databases that are read-only copies of the master.

A "suspender and belt" approach could be to have two mirrored databases, one which handles the usual database traffic andone which handles also the backups. This will assure a very high degree of safety, with multiple machines involved (could easily be multi-platform) and multiple safe guard techniques.

## Enhanced Flat-File Import and Export Functions

FrontBase has provided flat-file import/export functionality in the past but we have sought to improve this with some enhancements in FrontBase 3.x. This section seeks to explain and demonstrate these enhancements. The previous export and import SQL statements still function unchanged but are incomplete in functionality and thus de-committed.

## Enhanced Flat-File Export Function

To improve the features, flexibility and robustness of the flat-file export function, we have enhanced the export functionality.

**Syntax:**

```
WRITE ALL OUTPUT(list of <key value pair>);
```

<key value pair> ::= <key> = <value>
<key> ::= RSEP | CSEP | DIR | FOLDER | CONTENT
<value> ::= "general expression"

RSEP - Row SEParator

<value> must be a string expression and it identifies a string that separates the rows in the actual input from each other. Certain control characters can be escaped by using a normal backslash+letter combo:

\n new line
\r carriage return
\t horizontal tab

CSEP - Column SEParator

<value> must be a string expression and it identifies a string that separates the columns in the actual input from each other (within a row). Certain control characters can be escaped by using a normal backslash+letter combo (see RSEP).

DIR or FOLDER

<value> must be a string expression and it identifies a string that denotes the path of the directory (folder) into which the flat-file export will be generated. If the directory doesn't exist, it will be attempted generated.

CONTENT

<value> must be a boolean expression that evaluates to either TRUE or FALSE. TRUE means that also the contents of tables will be exported (one file per table).

## Enhanced Flat-File Import Filter

To improve the features, flexibility and robustness of the flat-file import, we have enhanced the import filter functionality. This functionality in FrontBase is quite general and can cover a very wide range of file formats:

**Syntax:**

```
INSERT INTO <table> [FROM] INPUT(list of <key value pair>) [COMMIT
<count>];
```

<key value pair> ::= <key> = <value>
<key> ::= RSEP | CSEP | FILE | TYPE | COLUMNS | COUNT | SKIP | CHECK | RDQ | STOP | STRIP | RSQ | UNIQUE | STRIP | POSITIONS | <column name>

<value> ::= "general expression"

**TYPE**

<value> must be one of 'FrontBase', 'Access' or 'OpenBase'.

This value provides a hint to the import filter and implies a number of default values:

TYPE = 'FrontBase' =>
RSEP = '§\n', CSEP = '§§', SKIP = 1

This value also implies that the actual input file has a format that is identical to what FrontBase produces when dumping a table into a flat-file (see WRITE TABLE).

TYPE = 'Access' =>
RSEP = '\n', CSEP = ';'

TYPE = 'OpenBase' =>
RSEP = '\n', CSEP = ' , ', RDQ = TRUE, SKIP = 1

The value of TYPE also implies in what format DATE, TIME and TIMESTAMP values are given.

**RSEP** - Row SEParator

<value> must be a string expression and it identifies a string that separates the rows in the actual input from each other. Certain control characters can be escaped by using a normal backslash+letter combo:

\n new line
\r carriage return
\t horizontal tab

Example: RSEP = '~\n'

**CSEP** - Column SEParator

<value> must be a string expression and it identifies a string that separates the columns in the actual input from each other (within a row). Certain control characters can be escaped by using a normal backslash+letter combo (see RSEP).

Example: CSEP = '\t'

**FILE**

<value> must be a string expression and it identifies a string that denotes the path of the actual flat file input.

Example: FILE = '/tmp/db0/3_45'

**COLUMNS**

<value> must be a list of column names each denoting a column of the given <table>. The order in which the column names are given is important and must match the actual input.

Example: COLUMNS = (C0, "TYPE", NULL, C1)

**SKIP**

<value> must be an integer expression and it identifies how many lines of the actual input file that are to be unconditionally skipped before any other lines are read.

Example: SKIP = 2

**COUNT**

<value> must be an integer expression and it identifies for how many columns there will be values in the actual input. COUNT is only necessary if no value for COLUMNS is specified and if TYPE = 'Access' (the column names are assumed given on the first line of the actual input).

Example: COUNT = 8

**CHECK**

<value> must be a boolean expression that evaluates to either TRUE or FALSE. TRUE means that for each row to be inserted, all constraint checks are enforced. If FALSE is specified, no constraint checks are enforced. It is a good idea to make sure that the input is indeed wellformed if CHECK is set to FALSE, e.g. to avoid duplicate PRIMARY KEYs.

Example: CHECK = TRUE

**RDQ** - Remove Double Qoutes

<value> must be a boolean expression that evaluates to either TRUE or FALSE. TRUE means that if a column value is enclosed in double quotes, they will automatically be removed.

Example: RDQ = TRUE

**RSQ** - Remove Single Quote

<value> must be a boolean expression that evaluates to either TRUE or FALSE. TRUE means that if a column value is enclosed in single quotes, they will automatically be removed.

Example:  RSQ = TRUE

**STOP** - Stop after an error has been generated

<value> must be a boolean expression that evaluates to either TRUE or FALSE. TRUE means that an import session will be terminated once an error has been found.

Example:  STOP = FALSE

**POSITIONS**

<value> is one of two types of a list which describes 1) the width of each value in the input or 2) the start position (starting with one) and length of each value in the input. This makes it possible to import from fixed width formats.

```
<value>                   ::= <width list> | <start and width list>
<width list>              ::= <list of integers>
<start and width list>  ::= <list of (<start>, <width>)
```

Examples:  POSITIONS = (2, 4, 4, 10)
POSITIONS = ((1, 2), (3, 4), (7, 4), (11, 10))

**UNIQUE** - Insert a unique value

<value> must designate an integer column of the given file. During import a unique number will be inserted, for the given column, into each new row. Note that this can be applied to multiple columns in a single import.

Example:  UNIQUE = PK_COLUMN

**<column name>**

<value> can be a general expression where the only requirement is that the datatype of the expression must match that of the column.

Examples:  COUNTRY = 'USA'
SKIPPED = 0
SKIPPED = (SELECT SKIPPED FROM ... WHERE ...)

### Example import

The following import routine uses a rich text file. This example demonstrates the flexibility of the ‘Access’ type for text file imports. The column seperator is defined as a comma and all quotes will be removed. In this case the import session will be terminated once an error has been found.

```
INSERT INTO "_SYSTEM"."T0" FROM INPUT(FILE = '/var/root/Desktop/
text.rtf',TYPE = 'Access',RSEP = '',CSEP = ',',SKIP = 0,CHECK =
TRUE,RDQ = TRUE,RSQ = TRUE,STOP = TRUE, COLUMNS = ("C0"));
```

### Bulk Imports

When doing bulk imports, either via regular INSERTs or flat-files it is advisable to follow some general recommendations:

1) Turn auto commit off:

```
SET COMMIT FALSE;
```

2) Switch to SERIALIZABLE, PESSIMISTIC:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE, LOCKING PESSIMISTIC;
```

3) Drop all constraints

4) Drop all explicit indexes

5) Do the bulk import

6) COMMIT;

7) Add constraints

8) Add explicit indexes.

9) COMMIT;

Assuming SERIALIZABLE, PESSIMISTIC, you can import as many rows you want in a single transaction.

NOTE: It is recommended that you try to remove NULL “values” from your source datafile before attempting an import

# Indexing

## Index management

```
   CREATE INDEX  [<index name>] FOR | ON <table name> (<column
name list>);
```

Creates an index for the given table and columns. There is no restriction on the number of indexes that can be created for a table. There is no restriction on how many columns that can be included in an index definition. The <index name> is needed if the index is to be dropped later on. The current user must be the owner of the schema that holds the specified table.

As the index is created while-you-wait, the execution of this statement may take a little time depending on how many rows the table holds.

Please note that indexes created via CREATE INDEX are only used to optimize SELECTs, i.e. these indexes do not imply any integrity constraint checks.

```
   DROP INDEX <index name>;
```

Drops the specified index. The current user must be the owner of the schema that holds the index.

As a convenience to aid in porting existing SQL applications, we have also introduced:

```
   CREATE UNIQUE INDEX [<index name>] FOR | ON <table name>
(<column name list>);
```

This is semantically identical to:

```
   ALTER TABLE <table name> ADD [CONSTRAINT <index name>] UNIQUE
(<column name list>) INITIALLY IMMEDIATE NOT DEFERRABLE;
```

All of the above statements are transaction initiating, i.e. a COMMIT is needed to make the changes visible to other users.

## Strategies

FrontBase offers two strategies for maintaining indices:

1. **PRESERVE SPACE**

   The default strategy, called PRESERVE SPACE, is very space efficient and works well with tables up to a few hundred thousand rows. An index on a table, no matter the number of columns, costs less than 5 bytes per row. If you have a table with 100,000 rows, creating an index on this table will thus increase the disk footprint by less than 500 KB. Memory efficiency is attained since column values are not stored together with the index information (an optimized B-tree). The downside is that rows from the table must be loaded to get column values when using the index. This index mode works well in most cases. Users like the low disk space footprint of a database.

2. **PRESERVE TIME**

   The alternative strategy, called PRESERVE TIME is fast when searching through millions of rows at the expense of higher disk space consumption. The mode scales very well and can easily handle tables with many millions of rows. Column values are copied into the B-tree, which increases the disk space footprint, but speeds up lookups considerably. The actual rows are loaded only when needed. If a given SELECT only fetches column values that are part of an index, the actual rows are not loaded at all. Such a SELECT is very fast.

## Example:

Consider a typical indexing setup with a WORD table, a DOCUMENT table and a RELATION table:

```
CREATE TABLE WORD(
    WORDPK INT PRIMARY KEY,   -- Implies an index
    WORD VARCHAR(64));

CREATE INDEX ON WORD(WORD);
CREATE TABLE DOCUMENT(
    DOCUMENTPK INT PRIMARY KEY,   -- Implies an index
    DOCUMENT CLOB);

CREATE TABLE RELATION(
    WORDFK INT,
    DOCUMENTFK INT,
    PRIMARY KEY(WORDFK, DOCUMENTFK));   -- Implies an index

CREATE INDEX ON RELATION(DOCUMENTFK, WORDFK);

COMMIT;
```

To get a list of DOCUMENTFKs identifying the documents in which a given word is found:

```
SELECT DOCUMENTFK
    FROM RELATION, WORD
    WHERE RELATION.WORDFK = WORD.WORDPK AND WORD.WORD = '<some-word>';
```

To get a list of WORDFKs identifying the words found in a given document:

```
SELECT WORDFK
    FROM RELATION
    WHERE DOCUMENTFK = <some-document-pk>;
```

To find a list of DOCUMENTFKs identifying the documents in which two given words are found:

```
SELECT DOCUMENTFK
    FROM RELATION, WORD
    WHERE RELATION.WORDFK = WORD.WORDPK AND WORD.WORD = '<word_1>'
    INTERSECT
    SELECT DOCUMENTFK
    FROM RELATION, WORD
    WHERE RELATION.WORDFK = WORD.WORDPK AND WORD.WORD = '<word_2>'
```

This could / should be wrapped into a VIEW.

In a reasonable setup with 100,000 documents and 100 words on average per document, the RELATION table holds 10,000,000 rows and is a perfect candidate for PRESERVE TIME. In all of the above SELECT statements, the actual rows would not be loaded if the indices were set to PRESERVE TIME. The indices would hold the actual column values.

The WORD table is a less likely candidate for PRESERVE TIME. It will probably make more sense to use PRESERVE SPACE combined with a proper sized cache for this table.

# Tuning FrontBase

As your FrontBase database gets large or your performance demands increase, you can tune FrontBase to improve overall and specific-query performance. This section describes how to tune FrontBase using the Raw Device Driver feature (overall performance) as well as table and index caches (specific query performance).

## Database server performance

Database server performance depends upon three things: CPU, RAM, and disk subsystem speed.

When your database is small (i.e. up to a hundred thousand rows per table) the performance is mainly dependent on the CPU speed: how fast the server can move data around. As your database grows larger, less of it fits into memory. Many databases are too large to fit in RAM at all times. Caches, caching strategies, and caching options greatly affect perceived performance. Eventually, your database becomes many times larger than the amount of available RAM. At that point, database server speed becomes most dependent on disk speed: how fast the server can get to the data.

Quite often, rather than upgrading your machine, you can install more RAM and upgrade the disk subsystem. You can also give the database server hints about how to cache data so that it can use the available RAM more effectively.

## FrontBase's caching mechanisms

FrontBase offers an elaborate caching strategy with two major components accessible to and tunable by the database developer: table caching and the raw device driver (or global cache). The crucial quality of FrontBase caches is that the integrity offered by transactions is maintained. When data in the cache is updated, the same data is also written to the disk upon executing a COMMIT. The COMMIT succeeds only after the data is successfully written to disk.

## Summary of caches

The FrontBase database server actually has five distinct cache types:

### The RDD cache

The RDD (raw device driver) cache is a write through cache which resides above the file system. The major purpose of the RDD cache is to avoid reads to the underlying storage media, but it also forms a basis for optimization of writes to the storage media. All reads and writes go through the RDD cache. You can use the RDD cache with normal files as with unformatted partitions.

### The descriptor cache

The implementation of the data stored for a table requires a descriptor  to provide fast access to the data. The descriptors are cached in the descriptor cache.

### The row caches

Each table has a cache for the fixed part of the rows in the table. These caches ensure that frequently used tables reside in core, which avoids time consuming reads from the underlying storage media.

### The dope caches

The data for columns of varying size is implemented by a fixed size reference to the actual data for the column, such that all rows in a table have the same fixed size. The data for the variable-sized columns is cached in the dope cache.

### The index caches

The indices for a table are cached, either in a cache shared by all indices for a table, or in a cache per index, depending on the index mode.

The row, dope and index caches are collectively known as table caches.

## When should caching be tuned?

If hitting the database server continually with SELECTs makes the CPU utilization percentage drop significantly, you should tune the caching mechanisms. In this case, the database server is waiting for the disk subsystem to deliver the data. You may be able to increase performance simply by adding RAM to the machine, or may need to tune caching.

FrontBase offers the following caching:

– "The Descriptor cache" on page 129.
– "Table Caching" on page 130.
– "Raw Device Driver (RDD)" on page 140.

## The Descriptor cache

Currently FrontBase does not provide any means for changing the descriptor cache. Approximately 0.2 % of the pages in the FrontBase database file are used for providing access to the data in the file. It is important that these pages are readily available in order to read the actual data, otherwise the system would have first to read the descriptor off the disk and then the data.

The performance of the descriptor cache can be inspected with the sql92 command line tool by issuing the command:

```
SHOW IO DESCRIPTOR;
```

The following headings are listed:

**Cache blocks used** - The number of blocks currently used in the cache

**Cache blocks free** - The number of blocks currently not used

**Descriptors read** - The number of descriptor read requests

**Descriptors hits** - The relative number of descriptor read requests that was read from the cache

**Descriptors write** - The number of descriptors written

**Read count** - The number of data reads

**Read blocks** - The number of data blocks read

**Write count** - The number of writes

**Write blocks** - The number of blocks written

The most important figure for determining the performance of the Descriptor cache is the hit ratio over a certain period. For a newly started database it should grow and stabilize at close to a 100%. The rest of the numbers give a feeling for the distribution between descriptor reads and writes and data reads and writes, and is thus a measure of IO activity.

## Table Caching

Table caching is a powerful feature that allows the developer to balance memory requirements against performance.

There a number of factors which can be detrimental to the performance of a database server, but broadly the most significant can be grouped into two categories; those relating to competition for the CPU, and those relating to accessing the hard drive. Clearly if other applications are competing for the CPU, then in any given period of time the database server is able to do less.  In the time that it takes for a hard drive to complete one revolution, however, a CPU intensive application can perform small miracles, e.g. scan over thousands of rows evaluating a WHERE clause. Anything the system does which results in increased disk access, therefore, is likely to have serious consequences for performance. Conversely, anything which reduces the frequency of disk access, or optimizes disk access, is likely to be of benefit for the database server.

If the combined memory requirements of all running applications are greater than the physical memory available, the operating system will start to swap, i.e. write / read portions of main memory to / from disk to accommodate the combined memory requirements. Some operating systems are better at handling swapping than others, but it is in general something that should be avoided for a production database server. The first line of defense against poor performance is therefore simply to add RAM to the server, to try to ensure that the system never swaps. Ultimately, however, there is a limit — whether due to physical or financial constraints — to the amount to RAM that can be added.

There are two techniques a database server can employ to ensure that the effects of disk access are minimized:

1) Cache data in main memory

2) Optimize disk access

Caching data in main memory is clearly an optimal technique, as most waits for the disk subsystem to deliver data are eliminated. For larger databases, however, it may become impractical to cache all tables, i.e. there might not be enough physical memory in the computer to accommodate 100% caching. In this case caching all the tables may be counter-productive, as it will lead to increased disk access. The goal in this situation should be to cache only that data which will be frequently accessed.

Optimizing disk access is almost a science in itself, but critical to a successful implementation is whether other applications are accessing the same disk subsystem simultaneous with the database server. The "perfect" disk access algorithm can easily be ruined by the unknown, such as some other application reading / writing to / from the disk subsystem at unpredictable times.

For most database applications, the above may sound more serious than it is. Normally, databases are actually fairly small and a database server can live in a happy coexistence with one or more applications, accessing it, on the same computer.

Table caching allows the developer or administrator to adjust how many of the rows of a given table that should be cached:

- Min. row count, the minimum number of rows to be cached.
- Max. row count, the maximum number or rows to be cached.
- Percentage, the percentage of the total row count to be cached.
- Persistent, keep the cache across transactions.
- Preload, cache the table the first time the table is referenced.

The Persistent setting is available (and not always ON) because in some scenarios, you may wish to control the actual memory usage while allowing certain bursts of memory usage to occur. An example is report generation, where each transaction can span many SELECTs and will likely reference some tables that normally aren't in use that much. If Persistent is set to OFF for such tables, the cache will get loaded as used and then flushed when the transaction is COMMITted.

### Implementing table caching

When it is deployed "out of the box", FrontBase doesn't enable any cache settings. This makes building a new database straightforward and works very well for smaller databases and in a development environment. Once a database solution is to be deployed, however, it may make sense to analyze the actual use of the database and tune caches etc. accordingly.

Each table in a FrontBase can be cached on a 100% individual basis tailored to how the table is used. The actual numbers of rows in a given table is of course also important when tuning cache settings.

There are actually a number of caches associated with a table and although they normally are assigned reasonable default settings, it sometimes makes sense to adjust each cache individually:

1) **Row cache**: the static part of rows.

2) **String cache**: variable length strings that aren't stored in the static part of a row.

3) **Index caches**: each index can be cached individually.

### Common settings

Common for all caches associated with a table are two persistent settings:

1) Pre-loading the caches when the server starts.

2) Maintaining the cache content across transactions.

Pre-loading caches is disabled per default, but can be enabled via this SQL statement:

```
ALTER TABLE <table name> SET PREPARE TRUE;
```

Only caches that are specified to be cached 100% will get pre-loaded during startup.

Maintaining the cache content across transaction is disabled per default, but can be enabled via this SQL statement:

```
ALTER TABLE <table name> SET PRESERVE TRUE;
```

Setting PRESERVE FALSE for a table is effectively the same as disabling the cache.

### Row Cache

The row cache is adjusted via this SQL statement:

```
ALTER TABLE <table name> SET CACHE(<min>, <max>, <percent>);
```

The <min> and <max> values are measured in rows.

**Examples:**

Caching the rows of a table 100%, but only up to a maximum of 250,000 rows:

```
  ALTER TABLE <table name> SET CACHE(0, 250000, 100);
```

Caching 1% of the rows in a table that has 3,000,000 rows:

```
    ALTER TABLE <table name> SET CACHE(0, 3000000, 1);
```

The default values are <min> = 2,000, <max> = 20,000, <percent> = 2100 and they will apply if PRESERVE TRUE is specified, but no explicit setting of the row cache.

If the <percent> is less than 100, only the first <percent> of the rows will get pre-loaded.

### String Cache

A string inserted into a table is stored either directly in the static part of the row or indirectly in a so-called string (or spelling) table. The threshold that determines whether a string is stored indirectly, depends on the datatype:

**CHARACTER**

If the max. size given in the datatype specification is less or equal to 64, inserted string values will be stored in the static part of the row. If the max. size is larger than 64, inserted string values are stored indirectly in the string table.

**VARCHAR**

If the max. size given in the datatype specification is less or equal to 16, inserted string values will be stored in the static part of the row. If the max. size is larger than 16, inserted string values are stored indirectly in the string table.

Strings stored indirectly can be normalized, i.e. a given spelling is only stored once:

```
  ALTER TABLE <table name> SET COLUMN [NO] MATCH (<list of column
names>);
```

Storing strings normalized can save space not only in the disk representation of the data, but also — and probably even more importantly — in the string cache. If many identical strings are inserted into a table, which could save a join operation in SELECTs, it will make sense to normalize the strings.

The string cache is adjusted via this SQL statement:

```
  ALTER TABLE <table name> SET STRING CACHE(<min>, <max>,
<percent>);
```

The <min> and <max> values are measured in number of entries in the string table.

**Examples:**

Caching the strings of a table 100%, but only up to a max. of 250,000 strings:

```
ALTER TABLE <table name> SET STRING CACHE(0, 250000, 100);
```

Caching 1% of the strings in a table that has a total of 3,000,000 string table entries:

```
ALTER TABLE <table name> SET STRING CACHE(0, 3000000, 1);
```

The default values are <min> = 2,000, <max> = 20,000, <percent> = 80 and they will apply if PRESERVE TRUE is specified, but no explicit setting of the string cache.

### Index Caches

A table has either one index cache, covering all indexes defined on the table, or one cache per index. If the so-called INDEX PRESERVE SPACE mode is in effect, only one cache for all indexes is created. If the so-called INDEX PRESERVE TIME mode is in effect, one cache for each index is created.

INDEX PRESERVE SPACE is the default and is a very disk space efficient mode, with the overhead being less than 5 bytes per index per row, no matter the number of columns in the index.

INDEX PRESERVE TIME is efficient for larger tables. The overhead depends on the datatypes and actual values of the columns in the index. If all columns in a SELECT are found in an index, the actual rows are NOT loaded in full, which can mean a significant savings in memory requirements and query execution time. Once the SELECT characteristics of a given database is known, dramatic performance gains can be achieved by defining indexes with more columns than are normally used, simply to avoid loading actual rows.

The index mode of a table can be switched via this SQL statement:

```
  ALTER TABLE <table name> SET INDEX PRESERVE TIME | SPACE;
```

If INDEX PRESERVE SPACE is in effect, the single index cache can be adjusted by specifying the name of any index defined on the given table.

An index cache is adjusted via this SQL statement:

```
ALTER INDEX <index name> SET CACHE(<min>, <max>, <percent>);
```

The <min> and <max> values are measured in number of disk blocks (a disk block is currently 2048 bytes);

**Examples:**

Caching an index of a table 100%, but only up to a maximum of 100,000 blocks:

```
ALTER INDEX <index name> SET CACHE(0, 100000, 100);
```

Caching 50% of the blocks of an index, in a table, that occupies 200,000 blocks:

```
ALTER INDEX <index name> SET CACHE(0, 200000, 50);
```

The default value for <percent> is 100. The default values for <min> and <max> will actually vary depending on the size of the index when it is created, i.e. a way to adjust the settings of an index cache optimally will be to drop the index and re-create it.

### Frequently Asked Questions

**Which cache will store BLOB and CLOB values?**
Such values are typically large and are such not cached in a table cache, but is either streamed directly from the disk subsystem or the so-called RDD cache (which is described in a separate document).

**In what cases could it make sense to store only e.g. 10% of the rows in a cache?**
If there are so many rows in the table that a 100% caching isn't practical and if the most "popular" queries only return up to 10% of the rows (over and over again).

**From this very large table, I only use SELECTs that has a value for the PRIMARY KEY in each WHERE clause, should I worry about table caching?**
No, at least not the row cache, fetching a row based on a PRIMARY KEY is typically very fast. You could consider tuning the cache for the index that corresponds to the PRIMARY KEY, but this is seldom needed. If enough memory is not an issue, table caching is always a way to make queries execute that fraction of a second faster.

**Why should I as a developer of a database driven application care about table caching?**
Users of any type application are an impatient bunch, they simply hate to wait for any action to complete. Fetching data from a disk subsystem is inherently slow compared to how many instructions even a slow CPU can perform per millisecond. To optimize the user experience of your application, you need to make sure that waits are as few and short as possible. Table caching is a feature that very eas-

ily can become your friend and help you provide a good user experience.

### Performance indicators

The performance of the table caches can be inspected with the sql92 command line tool by issuing the command:

```
SHOW CACHE[<schema-name>.]<table-name>;
```

**NOTE:** The names may contain wild cards % _.

If <table-name> is omitted then all the preserved tables in the current schema are shown. The SHOW CACHES command will print the following headings:

**Type** - Row caches are marked with an R, dope caches with a D, space optimized index caches with a S and time optimized index caches with a T.

**Item name** - The name of the item. For row and dope caches it is the name of the table, for index caches it is the name of the index. Please note that names beginning with _ (underscore) must be enclosed in " (double quote) when used in other SQL statements.

**Items** - The total number of items in the structure. For a row cache it is the number of rows in the table, for a dope cache it is the number of unique strings, and for an index it is the number of pages used to implement the index.

**Actual** - The number of items currently in the cache.

**Byte size** - The number of bytes of RAM used for the cache.

**Max Items** - The maximum number of items that the cache will hold. This number is not relevant for the dope cache as the size of the individual items vary and it is omitted.

**Byte size -** The estimated number of bytes that a completely filled cache will use. For a dope cache this number is also the limit size of the cache, as such a cache is limited in that way.

**Look ups** - The number of times an item was accessed.

**Hits** - The number of look-ups where the item was found in the cache.

**Hit Rate** - The relative number of look-ups that was found in the cache and thus did not cause one or more read operations. A look-up in the cache which loaded an item into a cache which was not full is counted as a hit. This has the consequence that loading a cache will result in hits until it is full. During loading the hit ratio will be 100%.

**Refs** (dope caches only) - The dope stores unique strings, a string that occurs several time will be stored only once. Refs gives total number of references to strings in the dope cache.

**Ratio** (dope caches only) - The ratio between the refs and the item count giving the average number of references to one unique string. If the ratio is close to 1, it may be worth considering to turn off the mechanism for avoiding duplication of identical strings.

Apart from the listing for each cache, a total line is computed for the actual memory usage, which will give an idea of how much memory the caches are using.

### Adjusting table cache settings

Using the sql92 command-line tool, you can adjust table caches using the following extensions to SQL 92 provided by FrontBase. You can also adjust table cache settings using the FrontBaseManager and FBWebManager tools.

```
ALTER TABLE <table name> SET CACHE ([<lower>], [<upper>], [<percent>]);
```

Sets or adjusts the cache parameters for the given table. <lower> and <upper> are given as absolute row counts. The server will keep a minimum of <lower> rows and a maximum of <upper> rows in memory. Defaults are 2000 for <lower> and 20,000 for <upper>. The <percent> value tells the server to keep a varying number of rows in the cache, while still obeying the <upper> value. The default value for <percent> is 20.

```
ALTER TABLE <table name> SET CACHE PRESERVE FALSE | TRUE;
```

Instructs the server to maintain (TRUE) the cache for the given table even if there are no references to the table. FALSE means that the server will discard the cache when there are no more references to the given table.

```
ALTER TABLE <table name> SET CACHE PREPARE FALSE | TRUE;
```

Instructs the server to load (TRUE) the cache fully for the given table the first time the table is referenced. FALSE means that the cache is loaded as dictated by the actual use of the given table.

The above statements are transaction initiating. A COMMIT is required to make the changes visible to other users.

## Raw Device Driver (RDD)

With today's fast computers, most performance issues in a database server are related to how fast you can get data off the hard disk. One way to increase the performance in this area, is to bypass the host OS filesystem. In FrontBase this is done through the deployment of the Raw Device Driver (RDD) module. RDD even allows you to specify a raw partition to be used as datastore. Additionally, RDD allows you to specify the size of its combined write-through and read cache.

### When should RDD be used?

RDD is typically used in combination with table caching so that smaller tables are cached 100%, while larger tables are cached via the RDD cache. An example is an indexing solution. You would typically have two smaller tables, WORDS and DOCUMENTS, with the third larger table being the relation table, HIT. With FrontBase, you would cache WORDS and DOCUMENTS 100%, while letting the

RDD manage the HIT table. 100-300 MB of RDD cache memory might work well, depending on the size of the HIT table.

### Adjusting RDD settings

RDD settings are specified when the database is created or started. Both the FrontBaseManager and FBWebManager let you specify RDD settings with their respective "Start Advanced" commands. From the command-line, you specify RDD settings when starting a FrontBase process.

### RDD performance indicators

The performance of the RDD cache can be inspected with the sql92 command line tool by issuing the command:

```
SHOW IO;
```

The following headings are listed:

**Used Pages** - The  number of pages currently in use

**Free Pages** - The number of free pages in the cache

**Pending pi** - The number of pages pending input

**Pending po** - The number of pages pending output

**Pending ps** - The number of pending synchronizations.  This number measures the number of commits that have been successful and have been written to the transaction log, but not to the database file itself

**Read count** - The number of read requests

**Read blocks** - The number of blocks read

**Read hits** - The relative number of blocks read which was read from the cache

**Read waits** - The number of reads that had to wait for a block to become available

**Read stalls** - The number of reads that had to wait for a buffer to become available for reading the data off the disk

**Write count** - The number of write requests

**Write blocks** - The number of blocks written

**Write stalls** - The number of times a write had to wait for a buffer to become available

**Device read** - The actual number of reads from the disk

**Device write** - The actual number of writes to the disk

As the cache is statically allocated, the number of pages in the cache is constant and equal to the sum of the free and used pages. The major performance indicator for the performance of the RDD cache is the hit ratio. The higher ratio, the less disk activity. The hit ratio may be improved by allocating more memory to the RDD or by growing the size of the table caches. The stalls should be 0; if they are not it, means that the cache is filled with data pending output and it should be a good idea to make the cache bigger.

## Improving performance

We now have all the numbers for the various cache performances. But what are the indicators of possibilities for improving the performance?

The first thought that springs to mind is to make all caches so large as to accommodate the complete underlying data structure. It can be done on the table cache level and it can be done with the RDD cache. If you start the server with an RDD cache which is bigger than the size of the underlying file, existing data will be read exactly once, and never read again. Modified pages are just written to the file and kept in the cache. You can also set the table caches to a size which will cache the complete table; this is more efficient as the structure for the table is not rebuild each time the table is opened.

The above settings will be efficient for small databases on machines with sufficient RAM, but as soon as the machine starts to page, the performance will drop dramatically and some more elaborate scheme must be used to get the maximum performance of the given hardware configuration. In most cases, FrontBase will perform better than the paging system as it will only have to re-read data while the paging system will have to write a page and read a new one.

You should avoid having table caches for rarely used tables. You can determine this by looking at the number of look-ups in a given cache, and if it appears to be low, turn off preservation of the table. The table data is then cached by the RDD cache.

You could also look for tables with a low hit ratio. A hit ratio less than 100% is only achieved for tables with more items than the cache may accommodate. A very low hit ratio suggests that table scans are performed, either because the complete table is returned, in which case the only thing to do is to increase the size of the caches, or because an index is required for optimization of the queries performed against the table, in which case the cause of the table scan must be isolated. A low hit ratio is typically less than 20%.

When you make the table caches more efficient by improving the hit ratio, you will also make the RDD cache more efficient, as reads satisfied by a table cache do not go to the RDD cache, thus reading of non preserved caches are more efficient. The cost is that the table caches use more RAM than the RDD cache.

## Mac OS X and Raw Devices

FrontBase can operate directly on a raw disk device, thus bypassing the file system and the buffering mechanisms. The use of raw devices enhance the performance and reliability of FrontBase as the overhead of the file system is avoided and finer control over the disk is achieved.

### Creating a Raw Device

Create a partition on the disk with the desired size and with the type FrontBaseFS and the name of the database you want to create. The type must be FrontBaseFS but the name is mainly used by the pdisk and similar programs . To create the partition on the disk used the pdisk utility.

To make the raw data partition available to FrontBase create a symbolic link:

ln -s /dev/disk<x>s<y> /Library/FrontBase/Databases/<database-name>.fb

where <x> is the device number, and <y> is the slice number. The slice number is the partition number as listed by the pdisk L command. The <database-name> is the name that must be used to access the FrontBase database.

### Creating a FrontBase Database on a Raw Device

To initialize a FrontBase database on a raw device the FrontBase server must be started with the -create option which allows FrontBase to overwrite the contents of an existing file. Once the database has been initialized FrontBase may be used in the normal way. However it is suggested that the -rdd option is used to specify a suitable raw device driver cache.

## Memory Usage

There is one more statistical command that you may want to use:

```
SHOW MEMORY [ALL];
```

The following headings are listed:

**The name** - Currently we have two distinct memory zones: One for IO, the IOZone, and one for everything else, the DefaultZone

**VM** - The number of MB allocated to the zone

**Small-Large** - The number of MB allocated for large pieces, and for small pieces of memory

**Used-Free** - The number of MB for used small pieces and free small pieces. Typically the free small pieces will be around 5-10 % of the space allocated for small pieces, when VM allocated to small pieces is more than 10 MB

**Used-Free** (second time) - The number of used and free memory pieces

**Used-Free** (third time) - The average size of used and free small pieces

The total size of the IOZone is close to the size of the RDD. The size of used small pieces is typically close to the total size of the caches within a few MB

If the number of used small pieces is growing, the table caches have typically not been filled yet. If the total allocation is close to physical RAM in the machine, it may be a good idea to review the cache settings, or install more RAM.

# Migration

This section explains how to use FrontBase's import features to migrate databases from other vendors' database servers.

Currently FrontBase has mechanisms for importing from the following:

– “FileMaker” on page 146.

– “MySQL” on page 147.

– Other import mechanisms are available via the enhanced import facility “Enhanced Flat-File Import and Export Functions” on page 115.

## FileMaker

The FileMaker Pro migration tool lets you move a FileMaker Pro database to FrontBase. It has two key restrictions. First, as FrontBase is a relational database server and not a front-end tool, only your FileMaker Pro tables will be migrated. Second, SQL 92 has no provision for calculated columns, so they can't be moved over to FrontBase if you have them in a FileMaker Pro database, however they can be implemented in VIEWS.

Currently, you'll need to download the FileMaker Pro migration tool separately. It is only available for the MacOS X platform. The tool is available from the downloads section of the FrontBase website.

When you download and decompress the FileMaker Pro migration tool, you'll find the following items:

1. FM2FB.app

   This is the main application build for Mac OS X Server. It can be used right away, or it can be moved to one of the Application folders. The application requires the FrontBase client frameworks to run. These frameworks are installed when you install the FrontBase package.

2. MetaDumper.fp3

This FileMaker script extracts information from FileMaker files. This script should be moved to a place where FileMaker can access it. It works with both MacOS and Windows versions of FileMaker. In MacOS, you may not be able just to double-click the script file. Instead, start FileMaker and open it using the "Open..." menu item.

3. README

   An older "read me" document.

To access the documentation on how to use FM2FB and the MetaDumper script, start FM2FB.app and choose "FM2FB Help" from the "Help" menu. This will open the HTML documentation in your web browser. The HTML files are embedded in the application wrapper.

## MySQL

The MySQL migration tool lets you move a MySQL database to FrontBase using JDBC drivers for both databases. It has one key limitation. It cannot handle MySQL with enum or set column types. The MySQL migration tool requires a Java virtual machine version 1.2 or higher.

Currently, you'll need to download the MySQL migration tool separately. The tool is available from the downloads section of the FrontBase website (www.frontbase.com). Eventually, it will be rolled into the FrontBase downloadable for every platform.

You will need to acquire a JDBC driver for MySQL. Simply install the MySQL JDBC driver in the directory containing the rest of the MySQL migration tool.

To run the MySQL migration tool, change your working directory to the directory containing the tool and execute the following from the command line:

```
java -cp mysql_2_comp.jar:frontbasejdbc.jar:MySQL2FB.jar MySQL2FB
```

It will ask you for some information to complete the migration:

### Source

The source field is the location and name of the MySQL database to transfer. Input must have the following format: dbName@host.

### Destination

The destination field is the desired location and name of the Front-Base database. There has to be a running version of FrontBase 2.0 on the specified host. Input must have the following format: db-Name@host.

### User name

The user name is the user name needed to log on to the MySQL database.

### Password

The password is the password for the MySQL user name.

The import filter is very unforgiving about incorrectly formatted input. Please experiment with a test database before working with a production database!

# Troubleshooting

## Logging SQL statements

FrontBase allows you to enable logging of all SQL statements sent to and executed by the server. This in turn allows for not only a powerful stress test tool (ClientSimulator), but also a very important debugging vehicle. Last, but not least, the logging also provides you with a textual representation of the state changes the various clients have imposed on the server.

Prior to FrontBase 3.1, SQL logging had to be turned on when starting the server:

```
<install dir>/FrontBase/bin/FrontBase -logSQL [<other options>]
<database name> &
```

With the release of 3.1 and later versions, the SQL logging can also be controlled in a more dynamic way by means of a regular SQL statement:

```
SET WRITE SQL  { TRUE | FALSE}  [ GLOBAL ];
```

The various combinations of this statement syntax are described below.

```
SET WRITE SQL TRUE GLOBAL;
```

Will turn on logging on a global basis, i.e. create the log file if it doesn't exist and turn on logging for all new agent connections. Please note that existing connections will NOT get logging turned on.

```
SET WRITE SQL FALSE GLOBAL;
```

Will turn off logging for all existing and new agent connections. Please note that the actual log file will not get closed. This implies that if logging is later on turned on again, the log file does NOT change location (e.g. in case the LogFiles directory was created in between turn-off/turn-on).

```
SET WRITE SQL TRUE;
```

Will turn on logging for the executing agent connection, but only so if logging have been turned on first using the GLOBAL option.

```
SET WRITE SQL FALSE;
```

Will turn off logging for the executing agent connection.

Please observe that when enabling the logging, the actual file is not truncated, i.e. new log entries are appended. Turning on logging, even globally, will not span a server stop / restart cycle.

## Understanding the SQL log file

The .sql log file that is maintained by the FrontBase server, is instrumented with certain information that will allow for a realistic replay of the log file. Information pertaining to timing measurements is also included. The instrumentation is in the form of SQL comments that are formatted in a particular way.

There are currently 9 different instrumentation comments in a FrontBase SQL log:

--0 <connection id><timestamp>
Creation of a new connection

--1 <connection id><session id><timestamp> "<user name>"
Creation of a new session in a connection

--2 <connection id><session id><timestamp><no of bytes in the following SQL statement>
Execution of a SQL statement

--3 <connection id><session id><timestamp>
Terminating a session

--5 <connection id> <timestamp>
Termination a connection

--6 <connection id><session id><timestamp>
When the server has sent a reply back to the client (as a consequence of a --2)

--7 <connection id><session id><timestamp>
A client requesting a batch of rows in a result set

--8 <connection id><session id><timestamp> "<number of rows returned>"
When the server has sent back the batch to the requesting client

## Location of the SQL log file

The log file will per default get created as

<install dir> / FrontBase / Databases <database name>.fb.sql

This location can be changed by creating a directory called LogFiles in the FrontBase installation directory:

mkdir <install dir> / FrontBase / LogFiles

Any servers with an open log file will NOT pick up this change of location until they have been stopped and restarted.

Please note that the LogFiles directory can actually be a soft (symbolic) link to a directory residing somewhere else in the file system.

## New SQL log file

SQL log files can get pretty large over time and it can be beneficial to split the log over a number of files. The server can be directed to 1) rename the existing log file and 2) create a new log file:

```
SWITCH TO NEW SQL LOG;
```

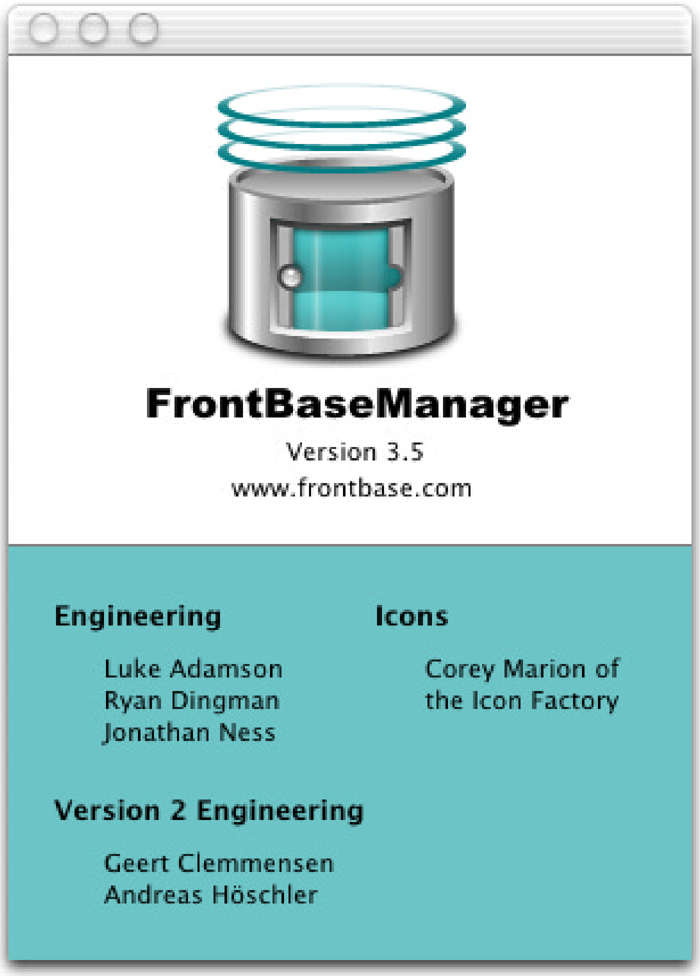
The name of the renamed log file will stay the same except that a timestamp is added as a suffix:

<database name>.fb.sql.yyyymmddhhmmss

The name of the new log file will be the same as before the switch was made, i.e. no change of directory.

# FrontBaseManager

FrontBaseManager is an application that lets you start, stop, monitor, create, remove, and generally manage every aspect of your FrontBase database on the MacOS. FrontBaseManager gives you a clean graphical interface to perform all your database maintenance, create tables and views, retrieve data from your database and upload data to your database, all without the need for SQL programming. FrontBaseManager for WinNT/2000 is included with v3.5.



FrontBaseManager
Version 3.5
www.frontbase.com

Engineering
Luke Adamson
Ryan Dingman
Jonathan Ness

Icons
Corey Marion of
the Icon Factory

Version 2 Engineering
Geert Clemmensen
Andreas Höschler

This chapter has the following sections:

## Changes since FBManager

Welcome to the latest revision of the FrontBase database management application! This section will give the experienced FrontBase developer an overview of the changes from the previous database management application (known as FBManager) and will give you an overview of FrontBaseManager's features:

- **New Documentation**
  New application documentation is available from the FrontBaseManager Help menu.

- **Multithreading**
  FrontBaseManager is now multithreaded, allowing multiple, concurrent connections to the same database, without the possibility of deadlock. This also prevents the main application thread from being bound when the user encounters a table or row lock during a fetch.

- **Aqua User Interface**
  The interface has been redesigned to fit within Aqua human interface guidelines. Plus, we have some beautiful new icons to spice up the app!

- **Enhanced Connection Panel**
  The new, integrated database connection panel supports connection by both database name, and TCP port.

- **Transaction Status**
  Transaction information is now available at all times on the database connection window. This includes whether or not a transaction is in progress, as well as the SQL 92 transaction settings such as Isolation Level, Locking Discipline, Updatability (or Access Mode), and Commit Mode.

- **Enhanced SQL Interpreter**
  The SQL interpreter now provides helpful keyword highlighting, supports Copy (Command-c) in the history list, and supports file drags into the SQL text area. Context menus, available by right clicking in the SQL Interpreter text area, provide rapid access to actions such as changing isolation level, locking discipline, toggling auto commit, etc.

- **Easy Database Information Access**
  The database pane readily provides basic information about your database, such as the database version, the up time, the size of the database on disk, the mode the database is in, etc.

- **SQL 92 Session Customization**
  Using the Session pane, you can now configure the

transaction settings associated with your session. For instance, you can make your isolation level and locking discipline more permissive, or switch to Read Only mode, enable/disable autocommit, etc.

- **Enhanced Usage Information**
  The usage pane now provides much more information about connected users, such as their SQL 92 transaction settings, their connection time, their transaction duration, etc.

- **Licensing Demystified**
  The licensing pane displays detailed information about your database license. As an added bonus, you can now remotely license database servers.

- **Schema Objects Browser**
  Using the Schema Objects Browser pane, you can traverse all of the objects within your database, including Tables, Views, Procedures, Functions, Collations, and so on. At any point, you can easily create a new schema object, or browse the content or definition of an existing schema object. Using Right-Click on the mouse will bring up a context menu to quickly copy the fully qualified name of the selected object to the clipboard.

- **Powerful Table Editor**
  While not a graphical modeler, the table editor in FrontBaseManager provides great control over table creation, allowing you to change almost everything about a table with a few mouse clicks. The column editor is especially helpful, since it provides ALTER SQL in real-time as you change your table definition. This helps to ensure that you know what's going to happen for a given change, and teaches you a little more about SQL 92 and FrontBase in the process.

- **Full Text Indexing**
  Using the table definition editor, you can quickly configure the powerful LookSee full text indexing engine and create full text indexes for your data. The indexing engine can handle text, HTML, and

XML, and has several options for stemming, folding case, trimming indexes, etc.

- **Powerful Table Content Editor**
  Using the content editor, you can browse table data, insert rows, update rows, and delete rows. You can even insert and download BLOB and CLOB objects by clicking on a simple hyperlink!

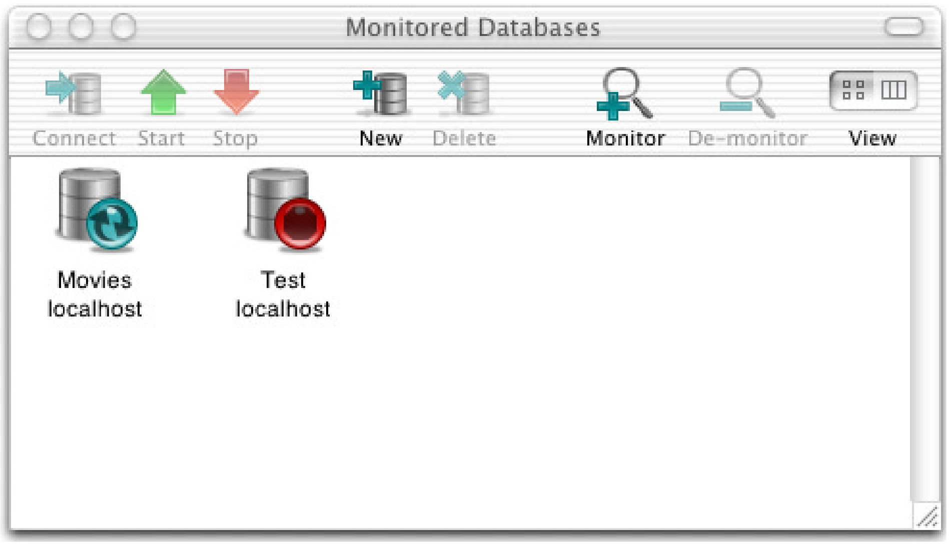
- **Drag & Drop**
  You can select rows in the content editor or in the fetch result window and drag them to the desktop, or to applications such as TextEdit, Mail, Excel and Word.

- **Easy Backup**
  The backup pane provides backup functionality with the click of a mouse. Backups are always a good idea!

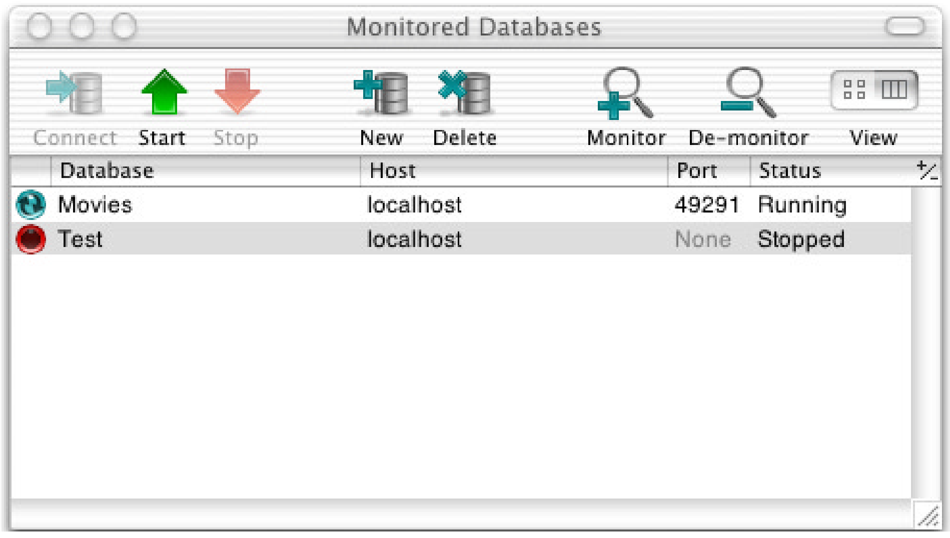
**- Flexible File Import Helper**
A powerful graphical interface to import data from a variety of different file formats. You can easily load data generated from FrontBase, OpenBase, Microsoft Access, Microsoft Excel or any program that exports to a delimited file format.

## Monitoring and Managing Databases

FrontBaseManager lets you monitor and control the state of Front-Base servers, i.e. whether they are running or stopped. When the FrontBaseManager is started, the following "Monitored Databases" window appears:



This windows shows monitored databases as icons indicating the state of the database together with the name of the database and the host computer on which the database is located. The example above shows the databases Movies and Test on host localhost. The Movies database is running and the Test database is stopped. Alternately, you can switch to List View using the view buttons. The corresponding List View window looks like this:



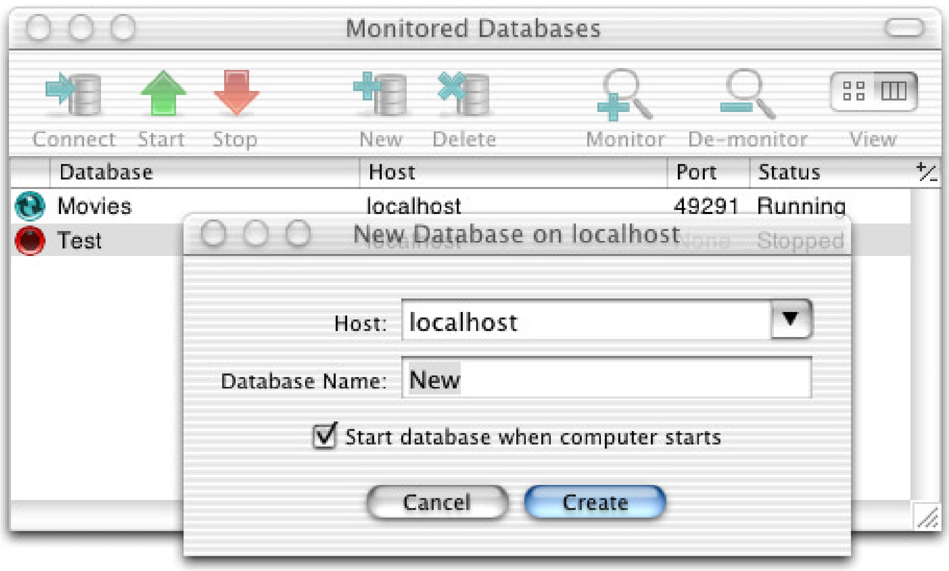
| Database | Host | Port | Status |
|---|---|---|---|
| Movies | localhost | 49291 | Running |
| Test | localhost | None | Stopped |

FrontBaseManager will remember whether you last used Icon View or List View.

## Creating Databases

The New and Delete buttons will allow you to create a new database or delete an existing, selected database. The Delete button will not be enabled unless you have selected a stopped database. When you click Delete, FrontBaseManager will ask you to confirm that you want to delete the database.

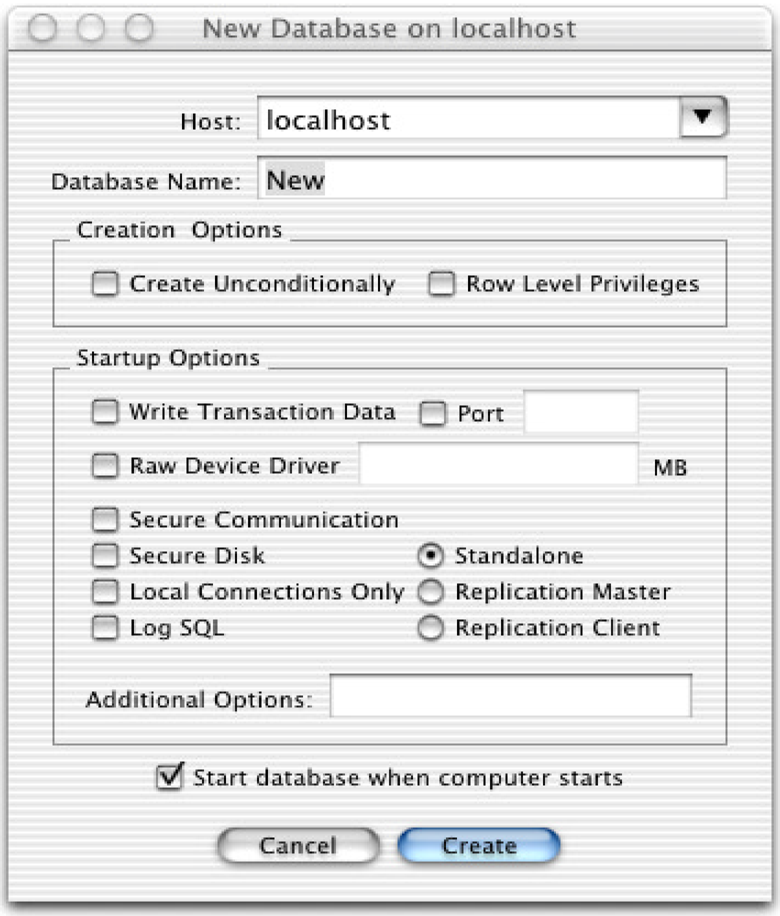
When you click New, FrontBaseManager will present you a dialog box asking which host you want to create the database on and what to name the database. That window looks like this:



| Database | Host | Port | Status |
|---|---|---|---|
| Movies | localhost | 49291 | Running |
| Test | localhost | None | Stopped |

Host: localhost
Database Name: New
☑ Start database when computer starts

Using the File->New Database Advanced option, you can create a new database with additional options. For detailed explanation of all the available options, please see "FrontBase for the Developer" on page 267.. A few options are discussed here:


New Database on localhost

Host: localhost

Database Name: New

Creation Options

☐ Create Unconditionally ☐ Row Level Privileges

Startup Options

☐ Write Transaction Data ☐ Port

☐ Raw Device Driver MB

☐ Secure Communication

☐ Secure Disk [✓ Standalone]

☐ Local Connections Only ○ Replication Master

☐ Log SQL ○ Replication Client

Additional Options:

☑ Start database when computer starts

*Create Unconditionally* - A new database is unconditionally created, overwriting an existing database with the same name (if it existed). Please use this option with caution!

*Row Level Privileges* - FrontBase offers a unique feature called Row Level Privileges which allows you to assign privileges, like on the files in a Unix filesystem, to each individual row in the tables of a database. The Row Level Privileges checkbox turns this feature on for the database when it is created. Refer to the "Row Level Privileges" on page 295. for more information on how to use this feature.

*Write Transaction Data* - After restoring from backup, you can "roll forward" any transactions performed since the backup if you've written out transaction data.

*Port* - Specifying a port directs the FrontBase database to listen on a specific port number.

*Raw Device Driver* - FrontBase offers a very advanced write-through cache mechanism that also supports use of a raw device (partition) as data storage. If you'd like to use this option (and your license permits it), click the Raw Device Driver and specify a size for this cache.

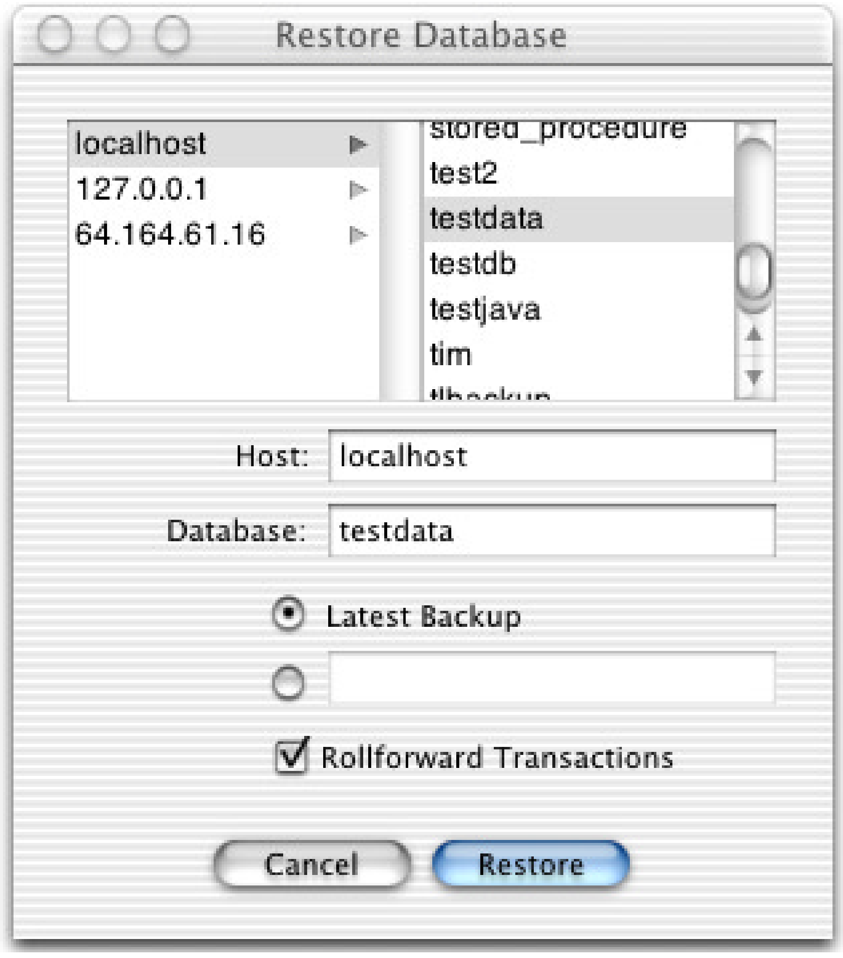
*Local Connections Only* - By specifying this option, a FrontBase database will only accept connections from clients running on the same computer as the database. The default is to accept connections from networked as well as local clients.

*Log SQL* - By specifying this option, a FrontBase database will log all the received SQL statements in a file. The file is created in the Databases directory of a FrontBase installation and is named <database name>.fb.sql. (The default is to not log the SQL statements.) For example, you can "Log SQL" on a database named Movies and (on Mac OS X) your log would be in /Library/FrontBase/Databases/Movies.fb.log.

*Replication options* - In the replication scheme offered by FrontBase, you can have one master database into which all updates must take place. Any number of read-only replication clients can be added. You can specify whether this new database should be started as a standalone server, a replication master or a replication client via the New Database Advanced window.

## Restoring from Backup

Restoring from backup is easy with FrontBaseManager. There are two cases where you might want to restore a database backup. One, you accidentally deleted your database entirely and need to restore from backup. The second case would be that you still have the database but just want to overwrite the current contents with the backup. So, if you already have a database, be sure it is stopped and then choose File->Restore Database.



Host: localhost

Database: testdata

[✓ Latest Backup]

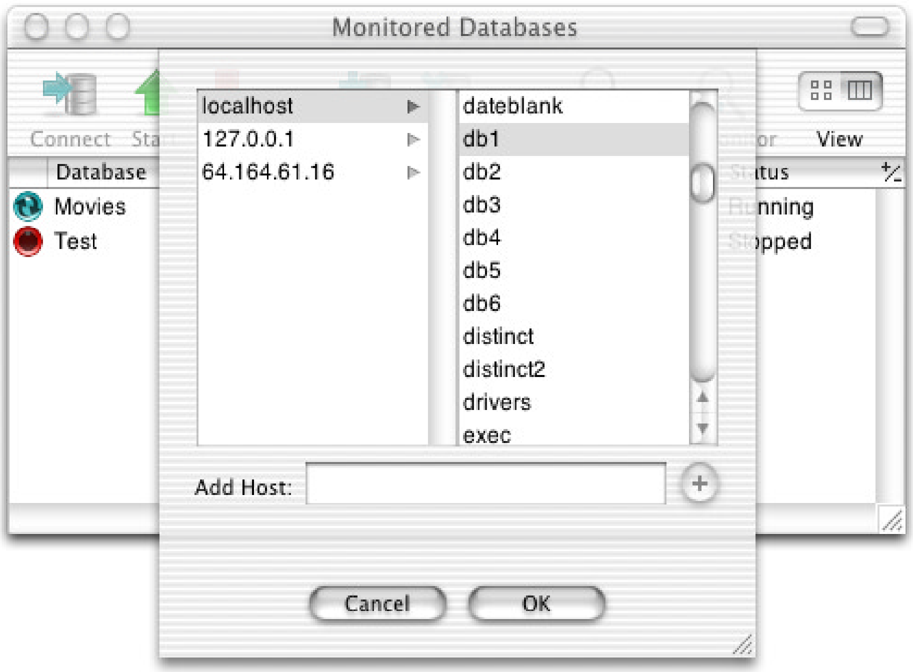
[✓] Rollforward Transactions

Select the hostname and then the database that you would like to restore. Most often you'd just want to restore the latest backup but if you'd rather restore an older one, specify the filename. If you'd like to roll forward any transactions since the backup (if you've been writing transaction data), be sure the "Rollfoward transactions" option is selected. When you click the "Restore" button, FrontBaseManager will create the database (if needed), start the database, and reload all your content.

To restore a backup if the database has been deleted, select the host where the database used to be and then type in the name of the database in the database textfield. Be sure the "Rollforward Transactions" checkbox is checked if you'd like to roll forward any transactions saved in the transaction log since the latest backup. Click the "Restore" button and your database will be restored, added to your monitored databases view, and started.

## Monitoring

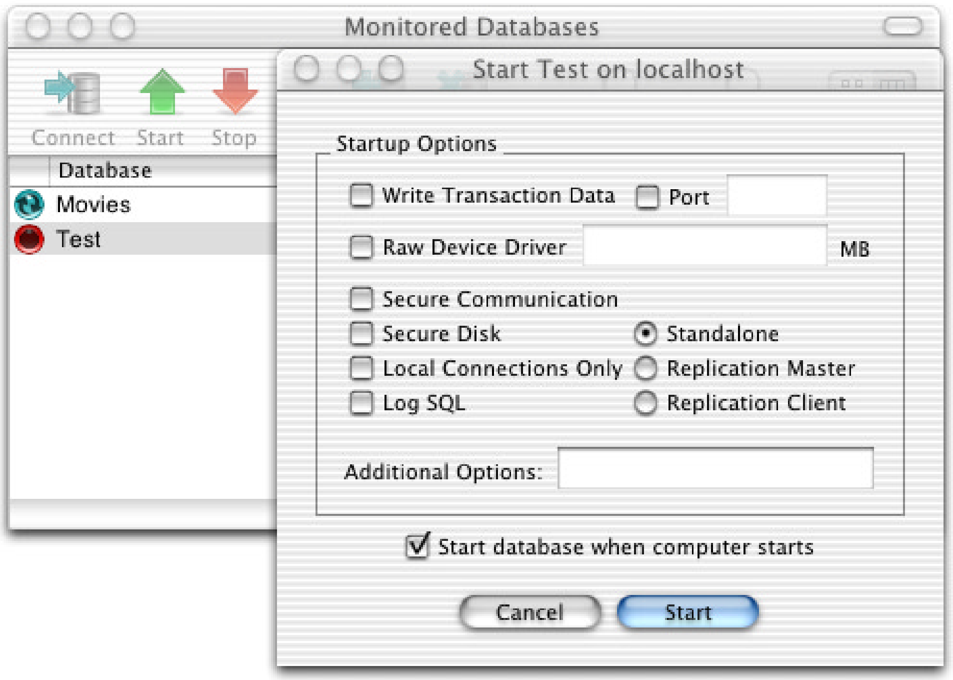
FrontBaseManager only allows you to actively manage those databases which it is currently monitoring. So, the database icons or list items which you see are only those databases it is monitoring. To add existing databases to the monitor view, click the "Add" button on the toolbar or choose Database->Monitor from the menu. The following panel appears:



The left column of the browser shows all currently monitored hosts. The right column shows all existing but not yet monitored databases on the selected host. To monitor databases on a not yet shown host, simply enter the hostname in the textfield below the browser and press <Return>. To add a database to the list of monitored databases, select the database in the browser and click the "OK" button. You can also add multiple databases at once. You can remove a database from the monitored databases list by selecting it and clicking "Remove" or by selecting it and choosing Database->"Stop Monitoring".

## Starting and Stopping Databases

To start a monitored database that is not yet running, select it in the monitor view and click Start or choose Database->Start or Database->"Start Advanced". Clicking Start or choosing Database->Start starts the database immediately. Selecting "Start Advanced" brings up a panel allowing you to specify options before actually starting the database.



Startup Options

- [ ] Write Transaction Data
- [ ] Port
- [ ] Raw Device Driver MB
- [ ] Secure Communication
- [ ] Secure Disk
- [ ] Local Connections Only
- [ ] Log SQL
- [✓ Standalone] Replication Master Replication Client

Additional Options:

- [✓] Start database when computer starts

You can select the start options on this window. The options are explained below:

*Write Transaction Data* - After restoring from backup, you can "roll forward" any transactions performed since the backup if you've written out transaction data.

*Port* - Specifying a port directs the FrontBase database to use a specific port number.

*Raw Device Driver* - FrontBase offers a very advanced write-through cache mechanism that also supports use of a raw device (partition) as data storage. If you'd like to use this option (and your license permits it), click the Raw Device Driver and specify a size for this cache.

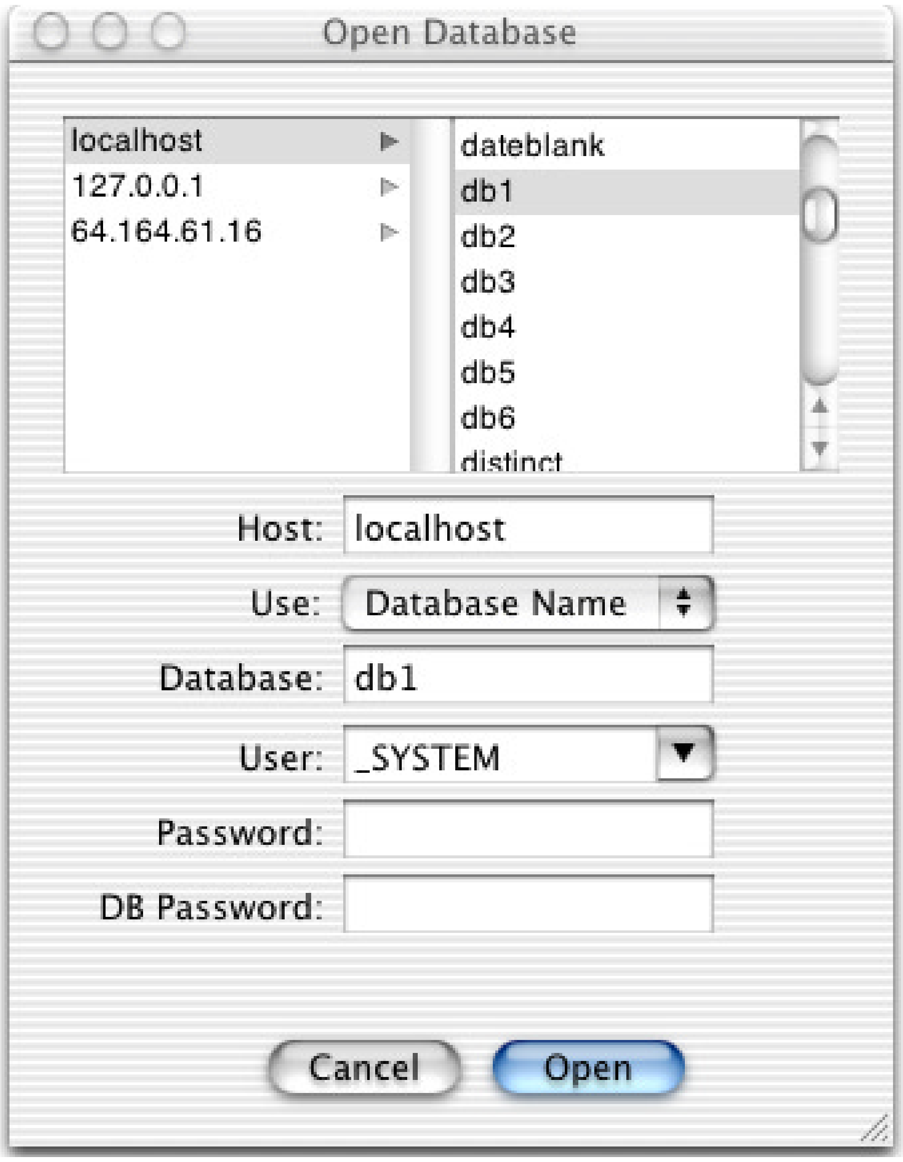
*Local Connections Only* - By specifying this option, a FrontBase database will only accept connections from clients running on the same computer as the database. (The default is to accept connections from networked as well as local clients.)

*Log SQL* - By specifying this option, a FrontBase database will log all the received SQL statements in a file. The file is created in the Databases directory of a FrontBase installation and is named <database name>.fb.sql. (The default is to not log the SQL statements.) For example, you can "Log SQL" on a database named Movies and (on Mac OS X) your log would be in /Library/FrontBase/Databases/Movies.fb.log.

*Replication options* - In the replication scheme offered by FrontBase, you can have one master database into which all updates must take place. Any number of read-only replication clients can be added. You can specify whether this new database should be started as a standalone server, a replication master or a replication client via the New Database Advanced window.

## Connecting to the database

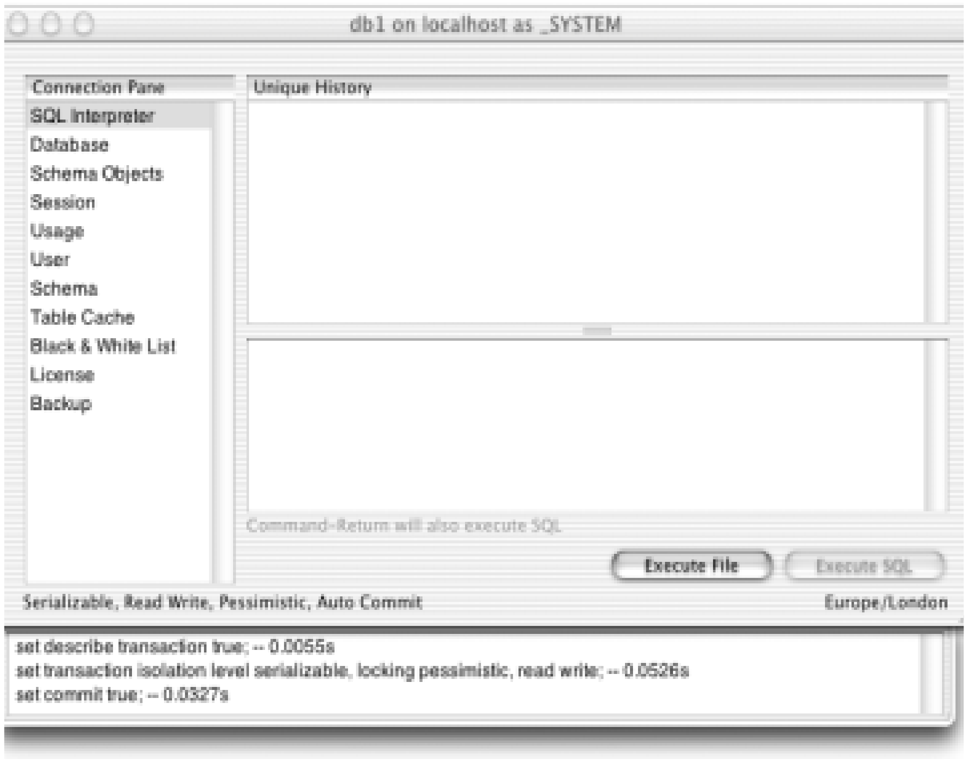
Once you have a running database, you can connect to the database by either selecting the icon and clicking the "Connect" button, selecting the icon and choosing File->Open Database, or by simply doubleclicking on the database icon. If you attempt to connect to a database that is not yet running, you will get a message saying "Unable to connect: Database is not running". When you attempt to connect to a running database, you will get the following dialog box:



Host: localhost
Use: Database Name
Database: db1
User: _SYSTEM
Password:
DB Password:

This window shows you the database you chose to connect to and also allows you to change your selection. It allows you to connect using either the database name or TCP port. Finally, it allows you to specify the user and password and the DB password, if any. You can select a user from a list of usernames that have been used at previous connects or type in a new username. When you connect to the database for the first time (directly after creating it), only superuser _SYSTEM is offered. If a connection can be established, the main Connection window appears.

## Connection Window



db1 on localhost as _SYSTEM

Serializable, Read Write, Pessimistic, Auto Commit    Europe/London

```
set describe transaction true; -- 0.0055s
set transaction isolation level serializable, locking pessimistic, read write; -- 0.0526s
set commit true; -- 0.0327s
```

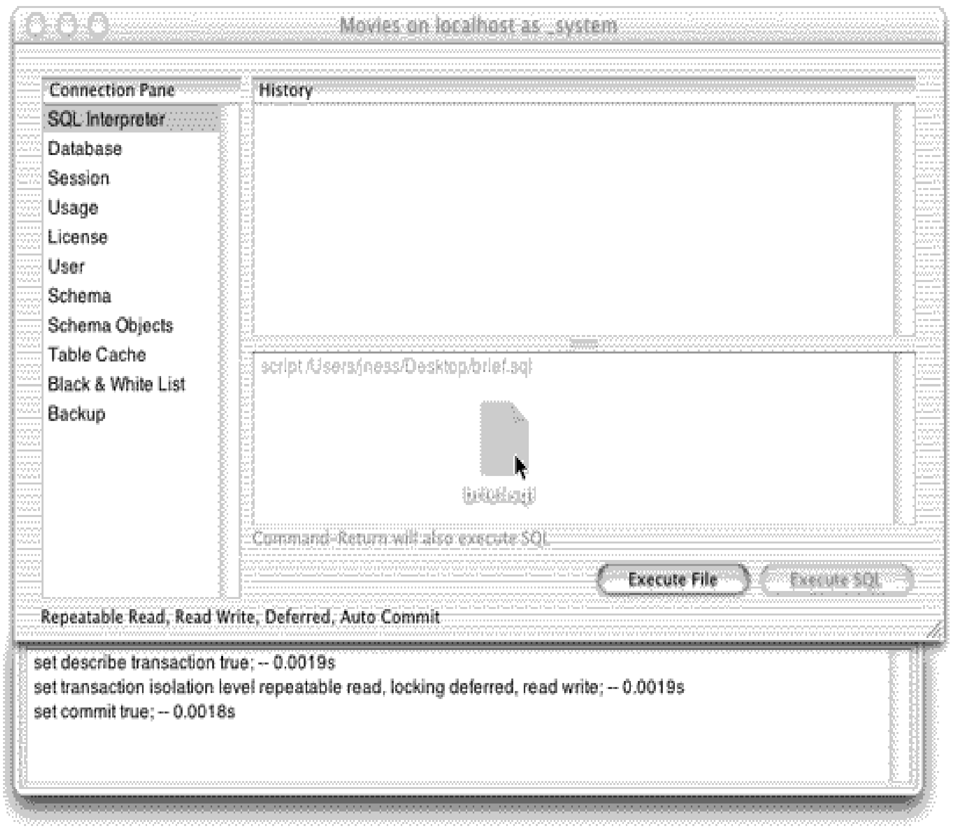
The Connection Window is FrontBaseManager's main window. This pane gives you access to most of the functionality that Front-BaseManager offers. So let's first examine the elements on the main connection window and then we'll delve into each separate switched view of the connection pane.

The Connection window title bar contains the name of the database, the host on which it is running, and the connection's user. At the bottom-left corner of the connection pane are the transaction settings, always there reminding you of your current transaction settings. These transaction settings can be changed through the Session pane. (See appendix 1 for a definition and explanation of all available transaction settings.) By default, a drawer extends below the connection window showing the log of all SQL statements sent to the database within this connection. (This drawer can be hidden by selecting View->"Hide SQL Log")

## SQL Interpreter

The Connection window starts with the SQL Interpreter pane selected as the sub-view. The SQL Interpreter allows you to send direct SQL to the database server. To do so, just click in the lower text area, type in your SQL and click "Execute SQL". As the message below the text input area states, Command-Return will also execute SQL. You will notice that as you type in SQL, the SQL Interpreter highlights SQL 92 reserved keywords in blue. This helps you debug your SQL statements. As you send SQL to the server, a history of SQL statements is kept for you in the history area. By default, the history list is kept unique. That is, if you issue "select * from foo;" then some other SQL statement, then "select * from foo;" again, you will not get a second "select * from foo;" in the history. Instead, the SQL history will move the "select * from foo;" already in the history to the bottom of the history list. You can turn this SQL uniquing feature off in the Preferences if you prefer to see every SQL statement in the history.

You can also execute files of SQL with the SQL Interpreter. There are two ways to do so. You can click the "Execute File" button to select a file to execute or you can just drag-and-drop the file onto the text entry area. As you drag the file over, a grayed out "script <filename>" will appear in the text area. Then, when you drop the file into the text area, "script <filename>" will actually be executed.



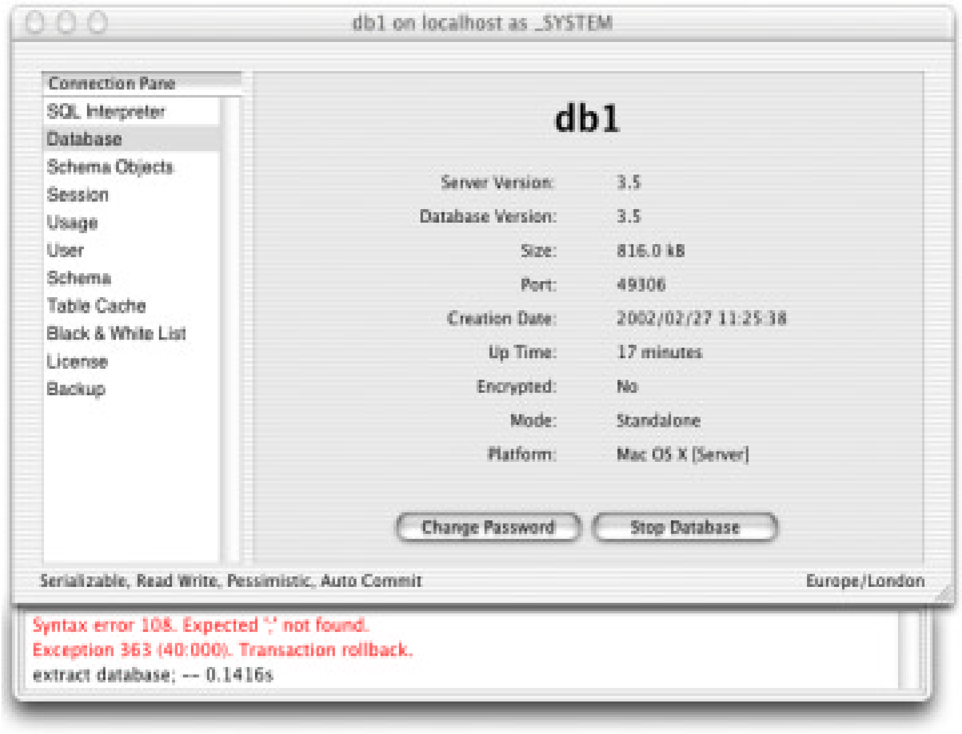
Repeatable Read, Read Write, Deferred, Auto Commit

```
set describe transaction true; -- 0.0019s
set transaction isolation level repeatable read, locking deferred, read write; -- 0.0019s
set commit true; -- 0.0018s
```

You can easily re-execute commands from your history by clicking on the history line item and then clicking "Execute SQL" or by simply double-clicking the history line.

It's easy to change your transaction settings from the SQL Interpreter as well. Just right-click in the SQL entry area and you'll be presented with a context menu giving you the option to Disable/ Enable Auto Commit and change to a number of useful transaction setting configurations.

## Database



db1

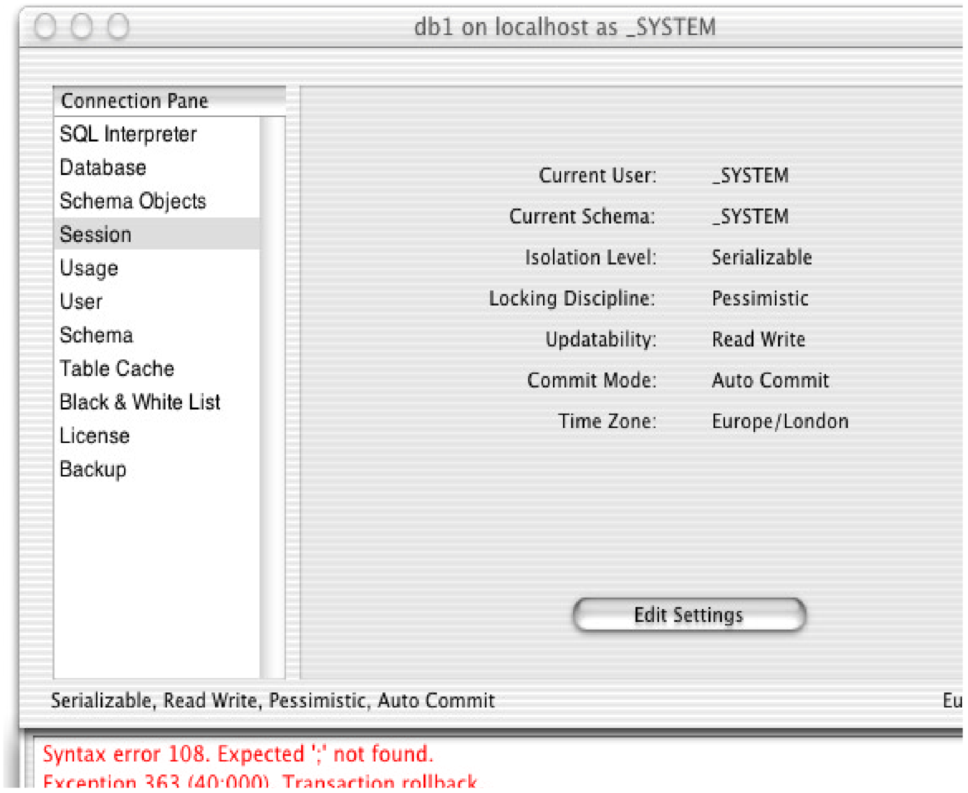
Server Version: 3.5
Database Version: 3.5
Size: 816.0 kB
Port: 49306
Creation Date: 2002/02/27 11:25:38
Up Time: 17 minutes
Encrypted: No
Mode: Standalone
Platform: Mac OS X (Server)

Serializable, Read Write, Pessimistic, Auto Commit    Europe/London

```
Syntax error 108. Expected ';' not found.
Exception 363 (40:000). Transaction rollback.
extract database; -- 0.1416s
```

The Database pane contains information about the database. Most of the items on this screen are self-explanatory. The server version is the current database *server* version number. The database version is the version of the server that created *this* database initially. You can change the database password or stop the database on this panel with the buttons at the bottom of the panel.

## Session



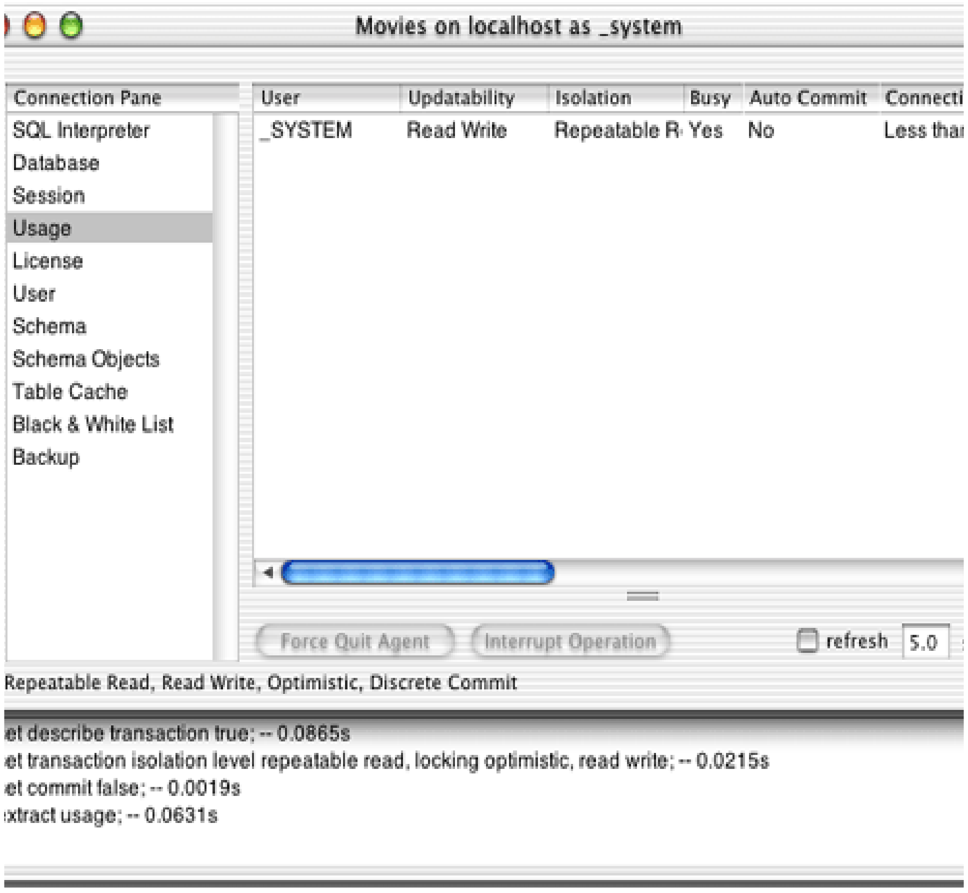
| | |
|---|---|
| Current User: | _SYSTEM |
| Current Schema: | _SYSTEM |
| Isolation Level: | Serializable |
| Locking Discipline: | Pessimistic |
| Updatability: | Read Write |
| Commit Mode: | Auto Commit |
| Time Zone: | Europe/London |

Serializable, Read Write, Pessimistic, Auto Commit Eu

```
Syntax error 108. Expected ';' not found.
Exception 363 (40:000) Transaction rollback
```

The Session pane contains information about this session's connection. It shows the current user, current schema, isolation level, locking discipline, updatability, and commit mode. There is an "Edit Settings" button which allows you to edit your current schema or any of your transaction settings.

## Usage

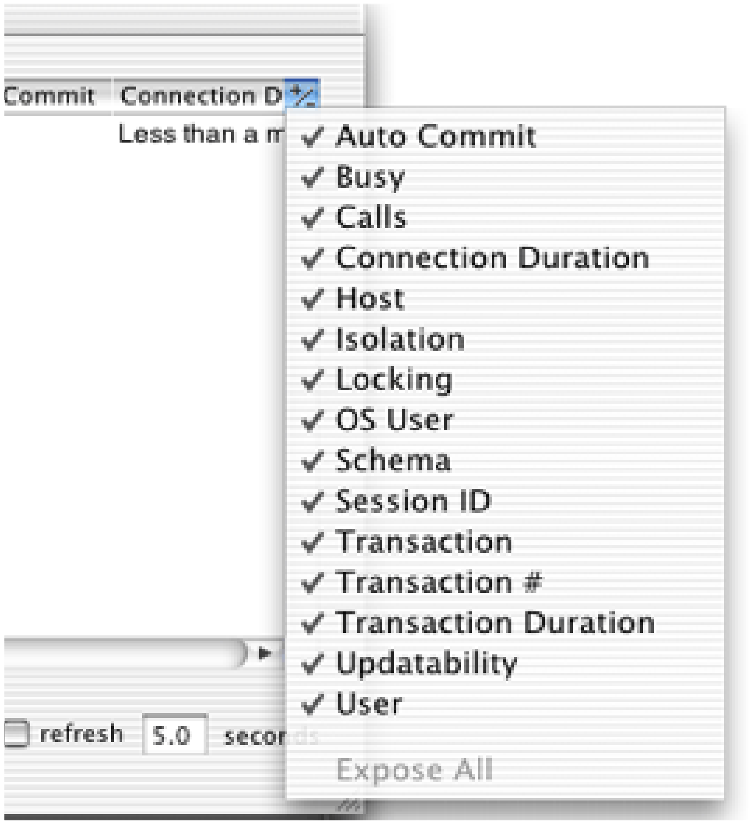


| User | Updatability | Isolation | Busy | Auto Commit | Connecti |
|---|---|---|---|---|---|
| _SYSTEM | Read Write | Repeatable R | Yes | No | Less thar |

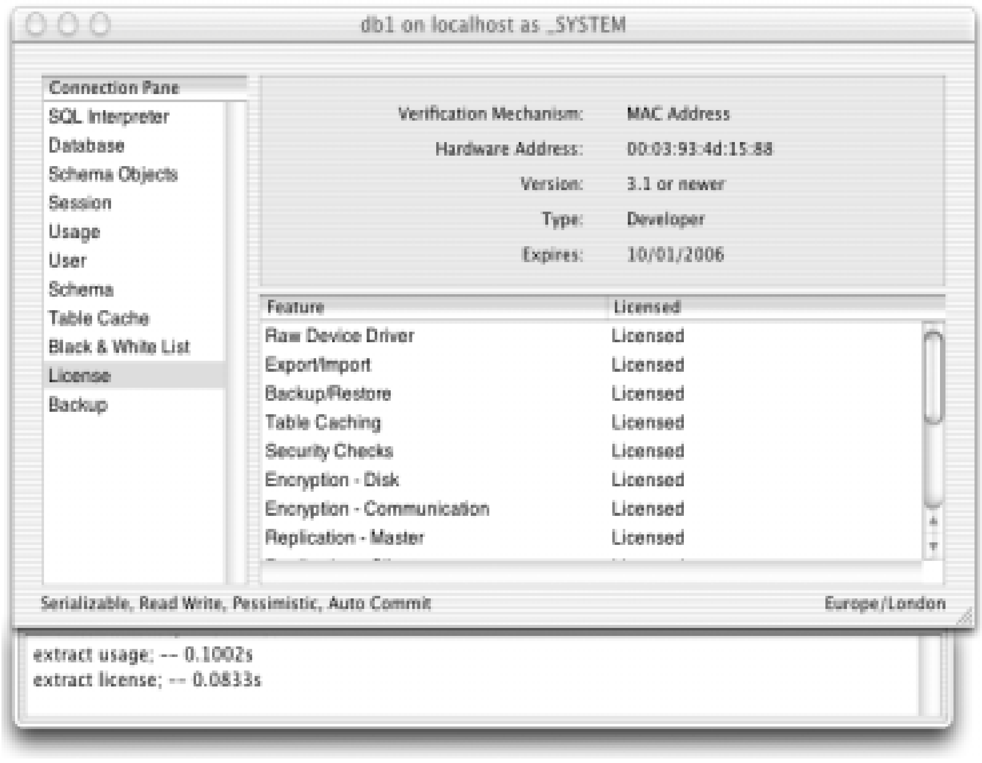
Repeatable Read, Read Write, Optimistic, Discrete Commit

```
et describe transaction true; -- 0.0865s
et transaction isolation level repeatable read, locking optimistic, read write; -- 0.0215s
et commit false; -- 0.0019s
xtract usage; -- 0.0631s
```

The Usage pane shows all the connections active to your currently connected database. It shows connection information, transaction settings, and the connection duration. The bottom pane shows the selected transaction's currently running SQL. You can use the +/- button in the upper right to filter the list of informational fields shown.



## License


db1 on localhost as _SYSTEM

| | |
|---|---|
| Verification Mechanism: | MAC Address |
| Hardware Address: | 00:03:93:4d:15:88 |
| Version: | 3.1 or newer |
| Type: | Developer |
| Expires: | 10/01/2006 |

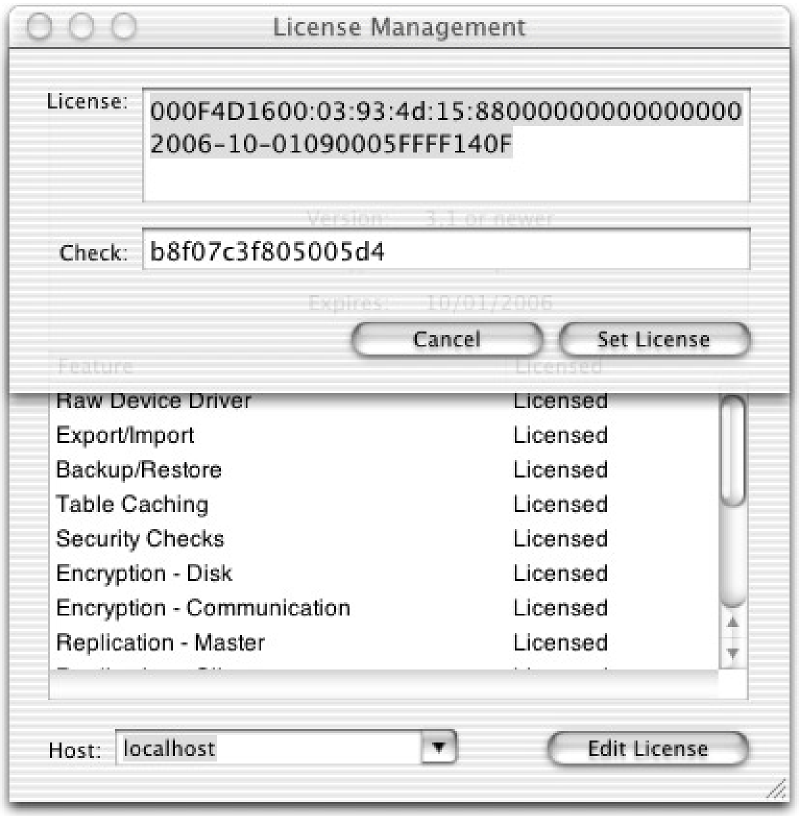
| Feature | Licensed |
|---|---|
| Raw Device Driver | Licensed |
| Export/Import | Licensed |
| Backup/Restore | Licensed |
| Table Caching | Licensed |
| Security Checks | Licensed |
| Encryption - Disk | Licensed |
| Encryption - Communication | Licensed |
| Replication - Master | Licensed |

Serializable, Read Write, Pessimistic, Auto Commit    Europe/London

```
extract usage; -- 0.1002s
extract license; -- 0.0833s
```

The License pane shows your current licensing information. It shows the verification mechanism (MAC Address or IP Address), the actual address it's verifying against, the version, the license type, and the expiration date. It then also shows all the features and that feature's licensed state (Licensed or Unlicensed). To change license information, click on Tools->"License Management", specify the host, and click the

"Edit License" button.



License: 000F4D1600:03:93:4d:15:880000000000000000 2006-10-01090005FFFF140F

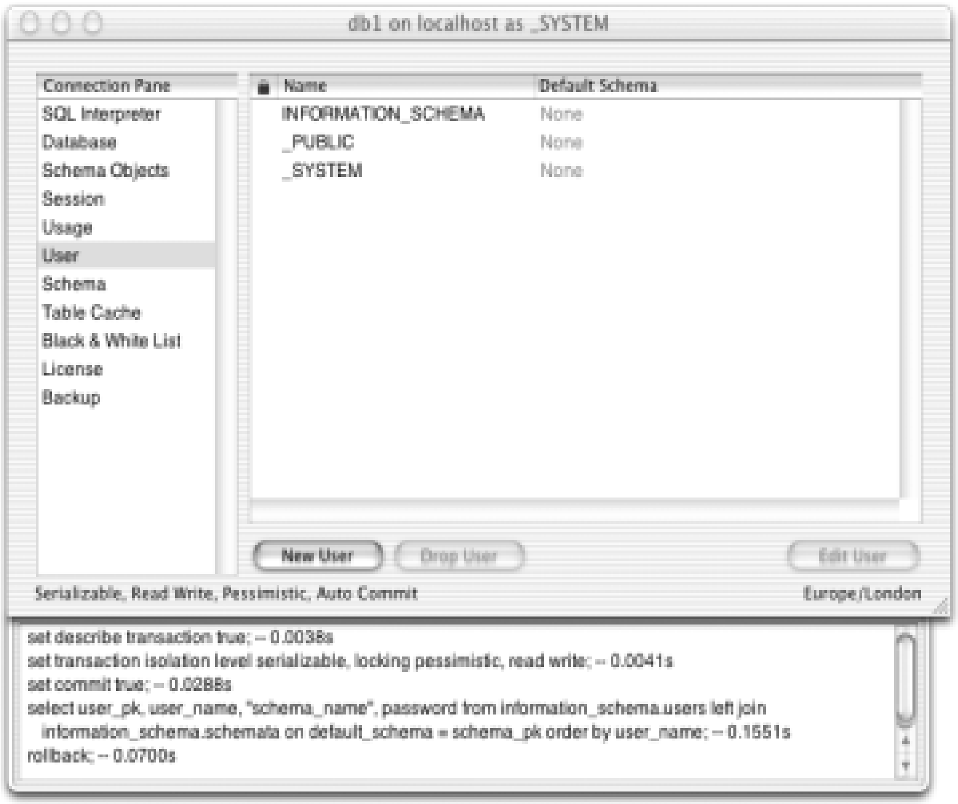
Check: b8f07c3f805005d4

| Feature | Licensed |
|---|---|
| Raw Device Driver | Licensed |
| Export/Import | Licensed |
| Backup/Restore | Licensed |
| Table Caching | Licensed |
| Security Checks | Licensed |
| Encryption - Disk | Licensed |
| Encryption - Communication | Licensed |
| Replication - Master | Licensed |

Host: localhost

You can then specify the License key and the Check key you received from FrontBase. Click "Set License" and the new license will go into effect after you restart the server. **You must restart the server for the new license to go into effect!**

## User



| Name | Default Schema |
|---|---|
| INFORMATION_SCHEMA | None |
| _PUBLIC | None |
| _SYSTEM | None |

Serializable, Read Write, Pessimistic, Auto Commit    Europe/London

```
set describe transaction true; -- 0.0038s
set transaction isolation level serializable, locking pessimistic, read write; -- 0.0041s
set commit true; -- 0.0288s
select user_pk, user_name, "schema_name", password from information_schema.users left join
  information_schema.schemata on default_schema = schema_pk order by user_name; -- 0.1551s
rollback; -- 0.0700s
```

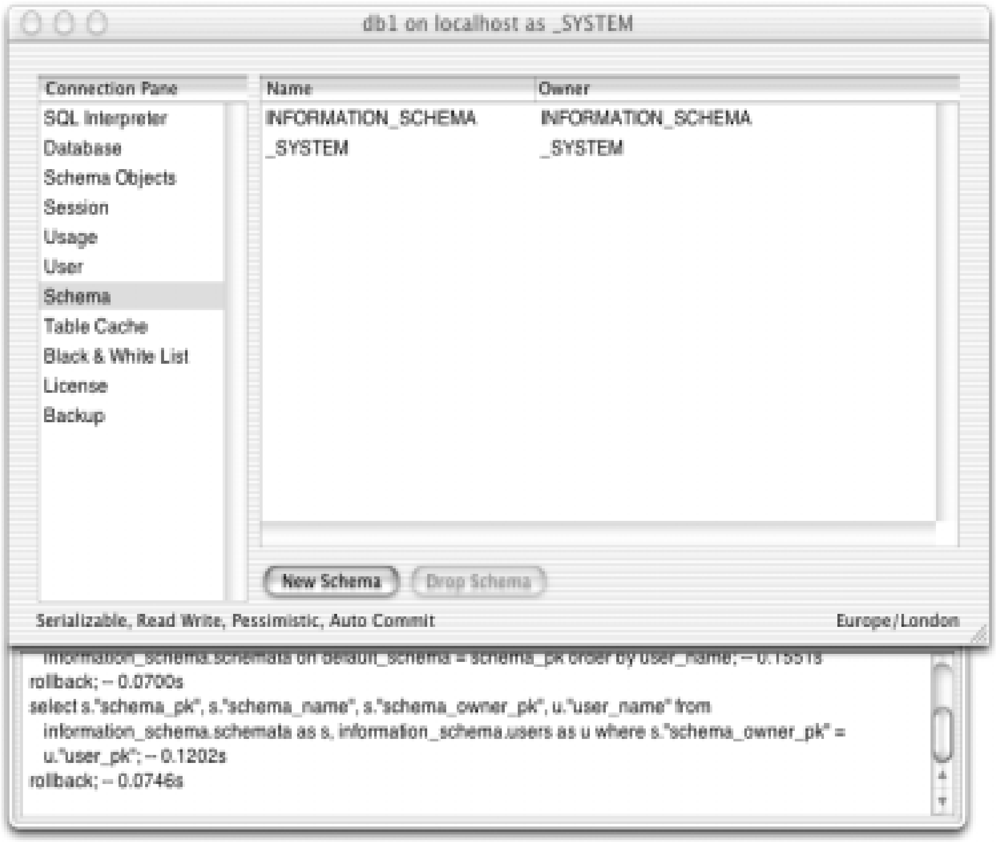
The User pane gives you the tools to add, drop, and edit users.

To create a new user, click the "New User" button and specify a username, schema option, and password. If you specify "Create Default Schema", FrontBaseManager will create a schema for that user named the same as the username. (A new user named "Bob" will have a new schema named "Bob" created and assigned as his default schema.) Otherwise, you can specify a user's default schema to be any of the other schemas defined on the database or to have no default schema.

To drop a user, select the user and click "Drop User". A sheet will come down asking whether to "Drop Cascade" or "Drop Restrict". You can only choose "Drop Restrict" if the user does not own any schemas or schema objects. Choosing "Drop Cascade" will drop the user and associated schemas and schema objects.

Editing a user allows you to change the user's default schema and password.

## Schema



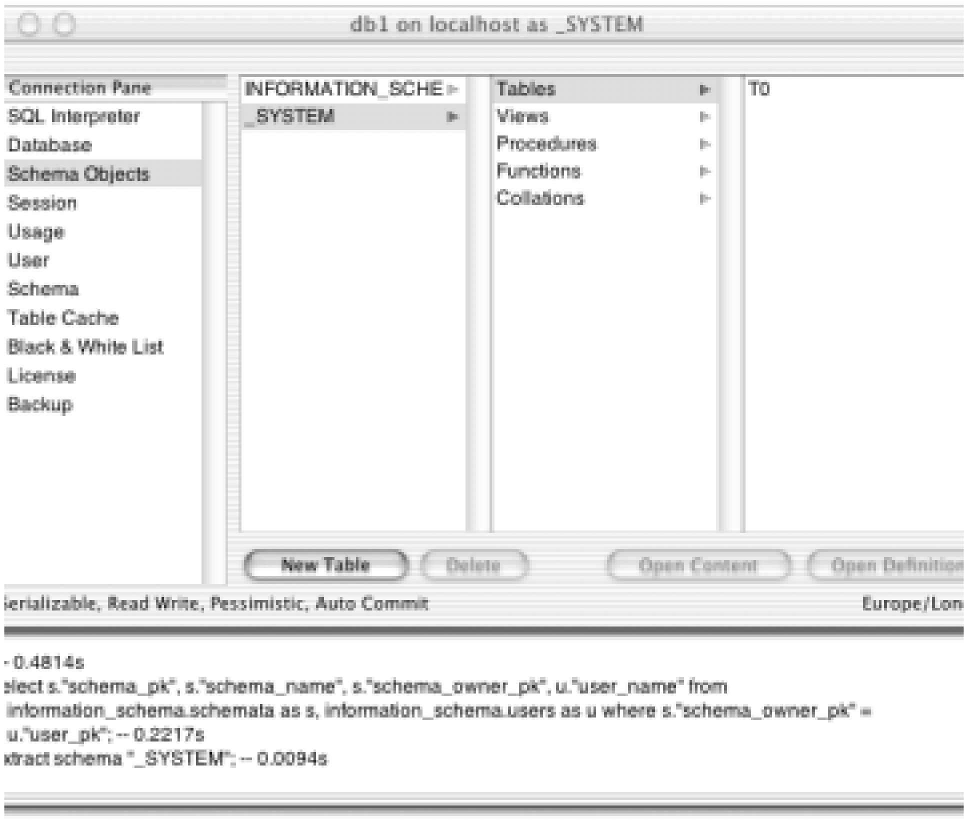
| Name | Owner |
|---|---|
| INFORMATION_SCHEMA | INFORMATION_SCHEMA |
| _SYSTEM | _SYSTEM |

Serializable, Read Write, Pessimistic, Auto Commit	Europe/London

```
  information_schema.schemata on default_schema = schema_pk order by user_name; -- 0.1551s
rollback; -- 0.0700s
select s."schema_pk", s."schema_name", s."schema_owner_pk", u."user_name" from
  information_schema.schemata as s, information_schema.users as u where s."schema_owner_pk" =
  u."user_pk"; -- 0.1202s
rollback; -- 0.0746s
```

The schema pane allows you to add or drop schema. To create a new schema, click the "New Schema" button and specify a schema name and owner. To drop a schema, select a schema that the logged-in user owns and click the "Drop Schema" button. If the schema contains schema objects, you must "Drop Cascade" to delete all the associated schema objects as well. If the schema is empty, you can just "Drop Restrict".

## Schema Objects



```
erializable, Read Write, Pessimistic, Auto Commit                Europe/Lon
```

```
- 0.4814s
elect s."schema_pk", s."schema_name", s."schema_owner_pk", u."user_name" from
information_schema.schemata as s, information_schema.users as u where s."schema_owner_pk" =
u."user_pk"; -- 0.2217s
xtract schema "_SYSTEM"; -- 0.0094s
```

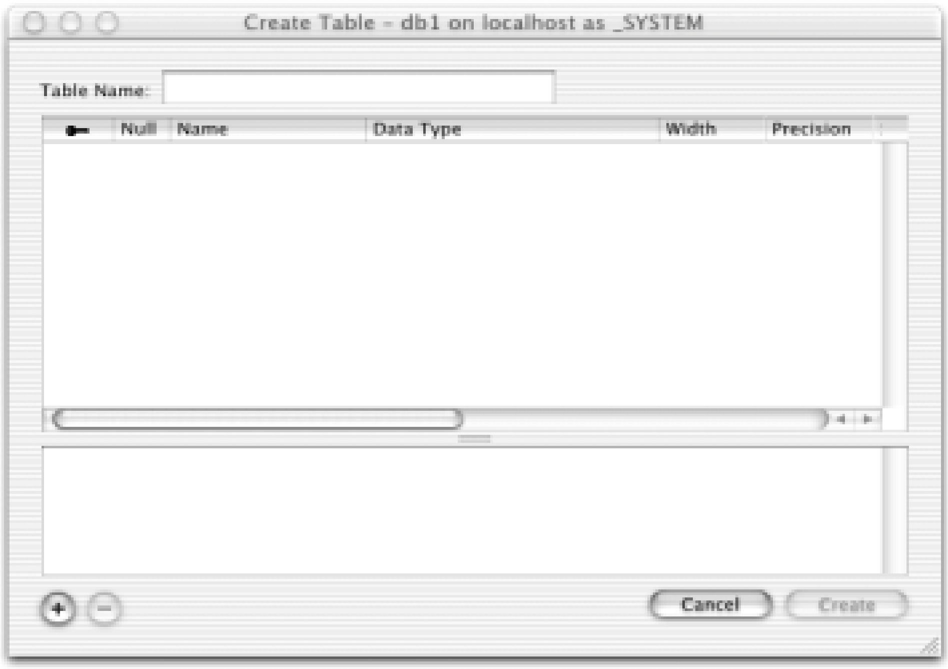
The schema objects browser contains some of the most powerful FrontBaseManager functionality. It allows you to create, delete, view, and modify the stuff you really care about -- your tables and data. This pane is organized like a browser or Finder. You first select a schema, then an object type, then you'll be presented with a list of the schema objects for you to either create, delete, open their content, or open their definition.

## New Schema Object

There are 5 different schema objects FrontBaseManager allows you to manipulate -- Tables, Views, Procedures, Functions, and Collations. FrontBaseManager provides an intuitive view to create a new object of each type. Let's first look at the New Table editor.

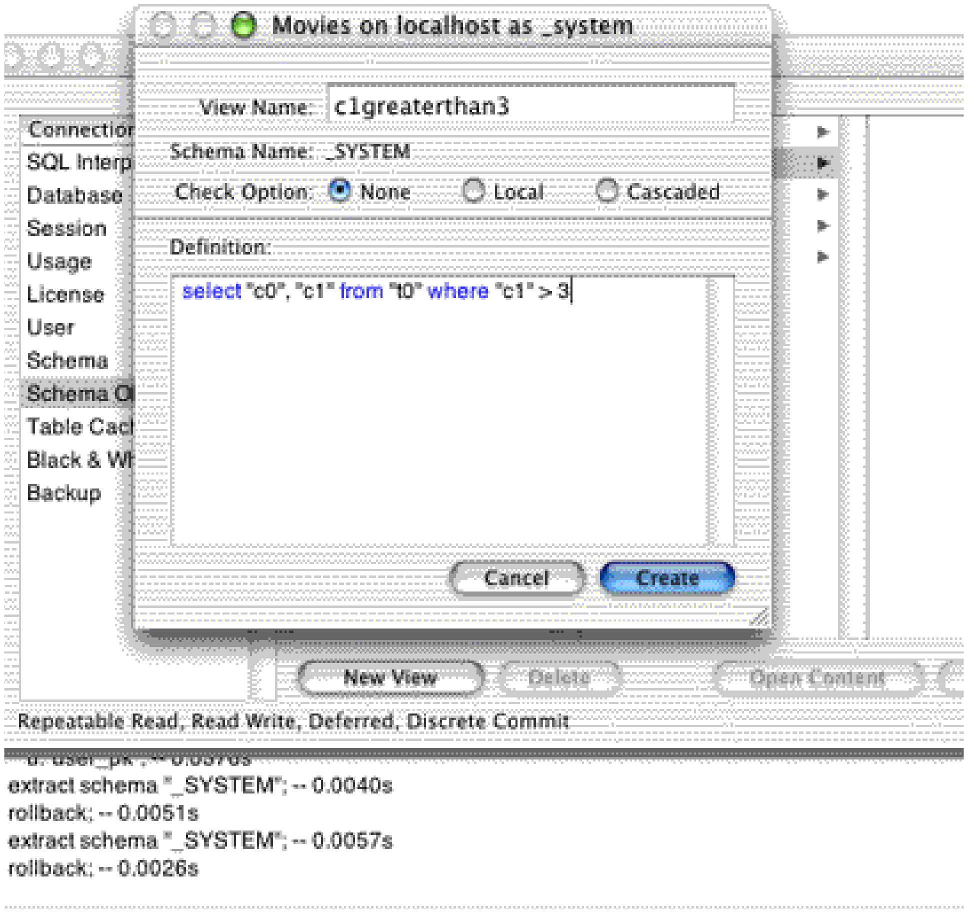
## New Table

To get there, select the schema where you'd like to create the table, then select "Tables". The "New Table" button will be enabled. Click the "New Table" button and you'll be presented with a view like this one:



You can specify a table name and then click the + button at the bottom left of the window to set up a column. You will notice that when you specify a table name and add a column, the SQL that would be executed is displayed in the lower pane. This helps ensure that you know exactly what's going to happen for a given table setup and teaches you a little more SQL 92 in the process. You'll notice that as you change the datatype or any other attributes in the upper pane, the SQL in the lower pane changes to show what SQL would be executed given the new table definition. The labels, from left to right, are primary key (If a key appears, this column is a primary key), allows null (If an indicator appears, this column allows null), column name, column data type, width, precision, scale, default value, whether normalized, and name of collation on the column. It is important to note that no actual create table SQL will get sent to the server until you press the "Create" button. This gives you a flexible canvas for defining exactly how you want to design your table before it's actually created.

## New View



View Name: c1greaterthan3

Schema Name: _SYSTEM

Check Option: [✓ None] Local Cascaded

Definition:

```
select "c0", "c1" from "t0" where "c1" > 3
```

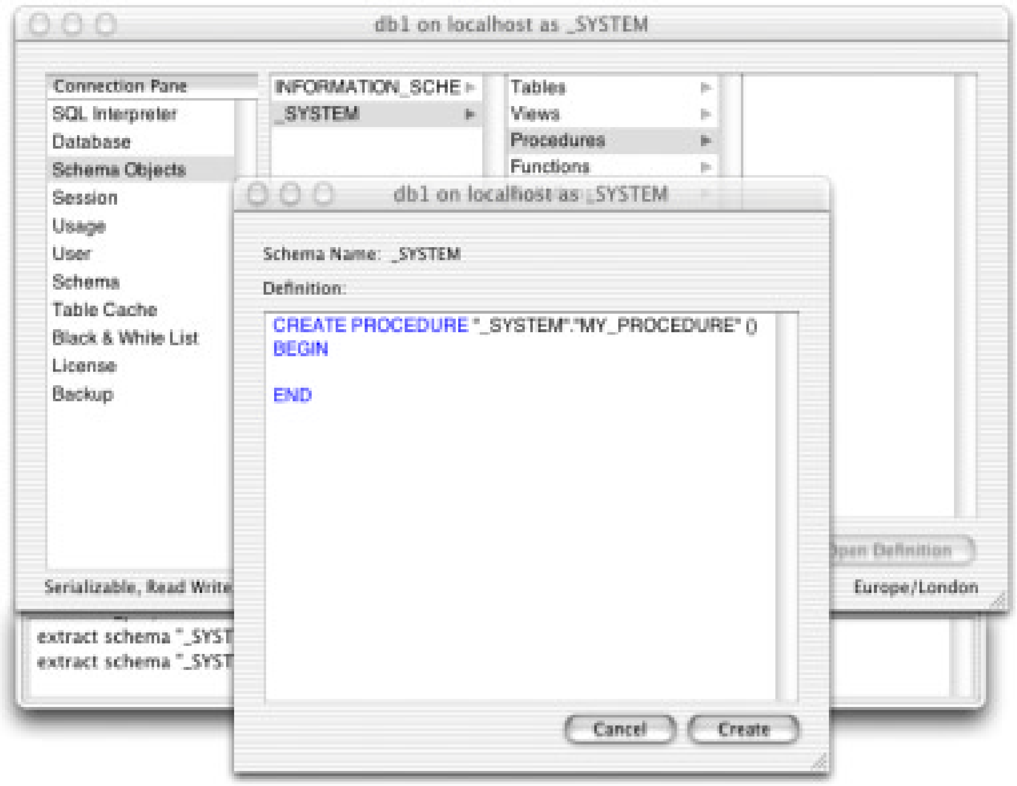
Repeatable Read, Read Write, Deferred, Discrete Commit

```
extract schema "_SYSTEM"; -- 0.0040s
rollback; -- 0.0051s
extract schema "_SYSTEM"; -- 0.0057s
rollback; -- 0.0026s
```

The New View editor is a bit more free-form than the table editor. It allows you to specify the name of the view, pre-populates the schema name, gives you a group of radio buttons to specify the check option (an advanced feature defined in the SQL92 standard), and the view definition. For example, to create a view containing columns named c0 and c1 from table t0 where c1 (an int column) is greater than 3, input the following into the definition:

select "c0", "c1" from "t0" where "c1" > 3

## New Procedures and New Functions



Schema Name: _SYSTEM
Definition:

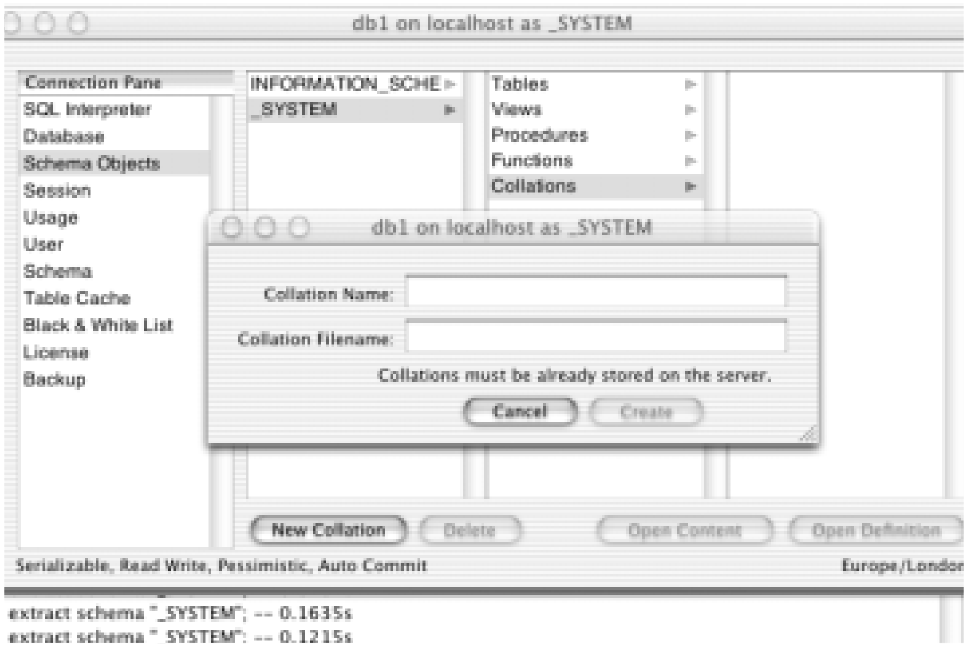
```
CREATE PROCEDURE "_SYSTEM"."MY_PROCEDURE" ()
BEGIN

END
```

extract schema "_SYST
extract schema "_SYST

Both the New Procedure and the New Function windows are quite simple. When you navigate in the schema objects browser to the Procedures or Functions item within a defined schema, the "New Function" or "New Procedure" button will become enabled. When you click the button, the resulting window will allow you to define the function or procedure.

## New Collation



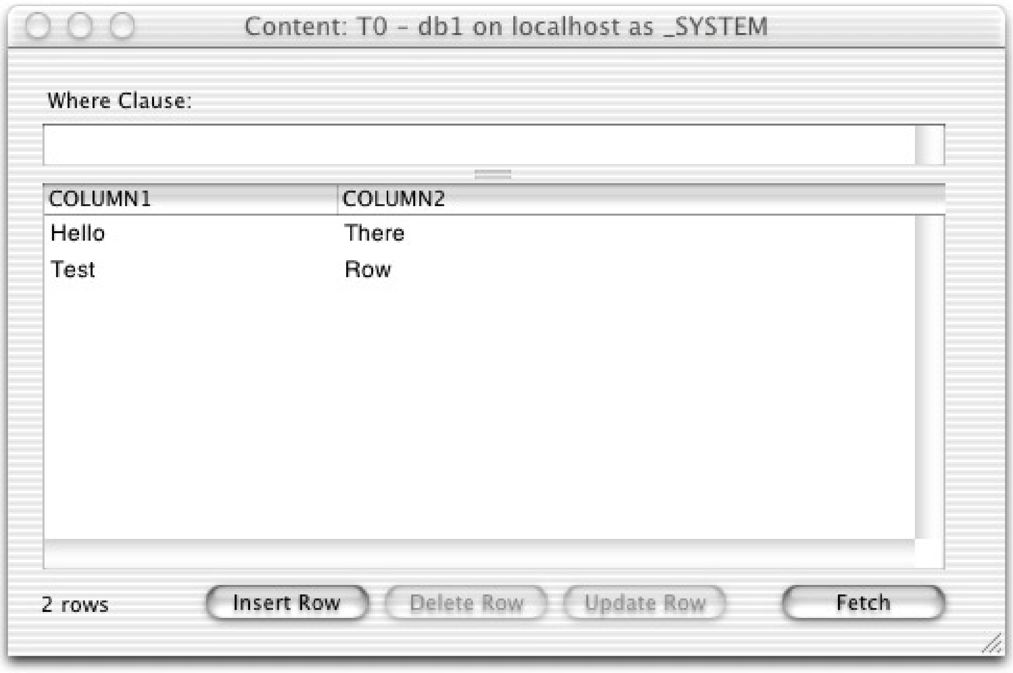
Collations must be already stored on the server.

Serializable, Read Write, Pessimistic, Auto Commit

```
extract schema "_SYSTEM"; -- 0.1635s
extract schema "_SYSTEM"; -- 0.1215s
```

Clicking the "New Collation" button in the Schema Objects browser brings up a small window which allows you to specify the collation name and the filename of the actual collation file on the server. (The Collations are all stored in /Library/FrontBase/Collations so you do not need to specify a path.)

## Open Schema Object Content



| COLUMN1 | COLUMN2 |
|---|---|
| Hello | There |
| Test | Row |

The "Open Content" button is enabled when you select a defined table or view. This window allows you to view content and insert, delete, and update rows. When the window first comes up, it shows all the rows (up to your fetch limit) in the table. You can qualify the rows fetched (and shown) by specifying a where clause in the upper pane and clicking the "Fetch" button. Specify only the part of the SQL statement you would put after a 'where'. For example, "column3 > 5" would be a valid "where clause" if you had a table with a column3 as an int column within that table. To delete rows, simply select a row or number of rows and click the delete button. If the schema object is updatable (some views are not), the rows will be deleted. Inserting and updating rows happens by pulling down a sheet from the window which allows you to specify values for any of the columns you'd like populated.

## Inserting and Updating Rows

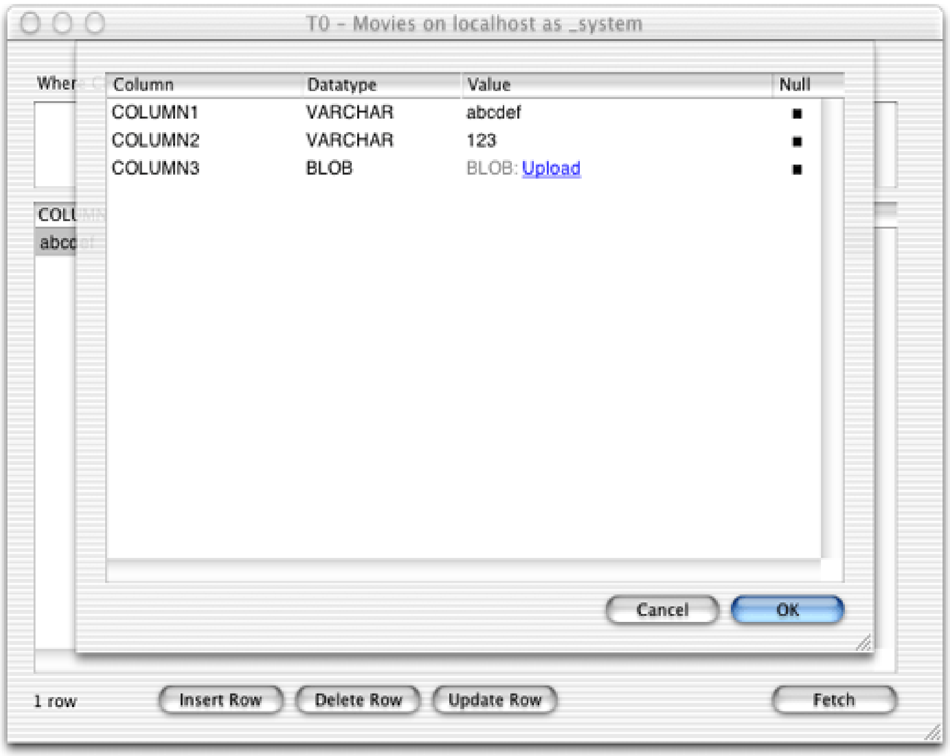
The Row Editor sheet contains a table view with columns for table column name, datatype, value, and an allows null indicator. When entering values, enter just the actual content. For example, the SQL to insert a row into a table t0 containing a date column c0 would look something like:

insert into "t0" ("c0") values (DATE '2001-08-11');

However, the Row Editor prepends the DATE for you so you only need to enter the 2001-08-11 portion of the value. When inserting a row, the "OK" button will be disabled if you have not specified a value for a column which does not allow null.

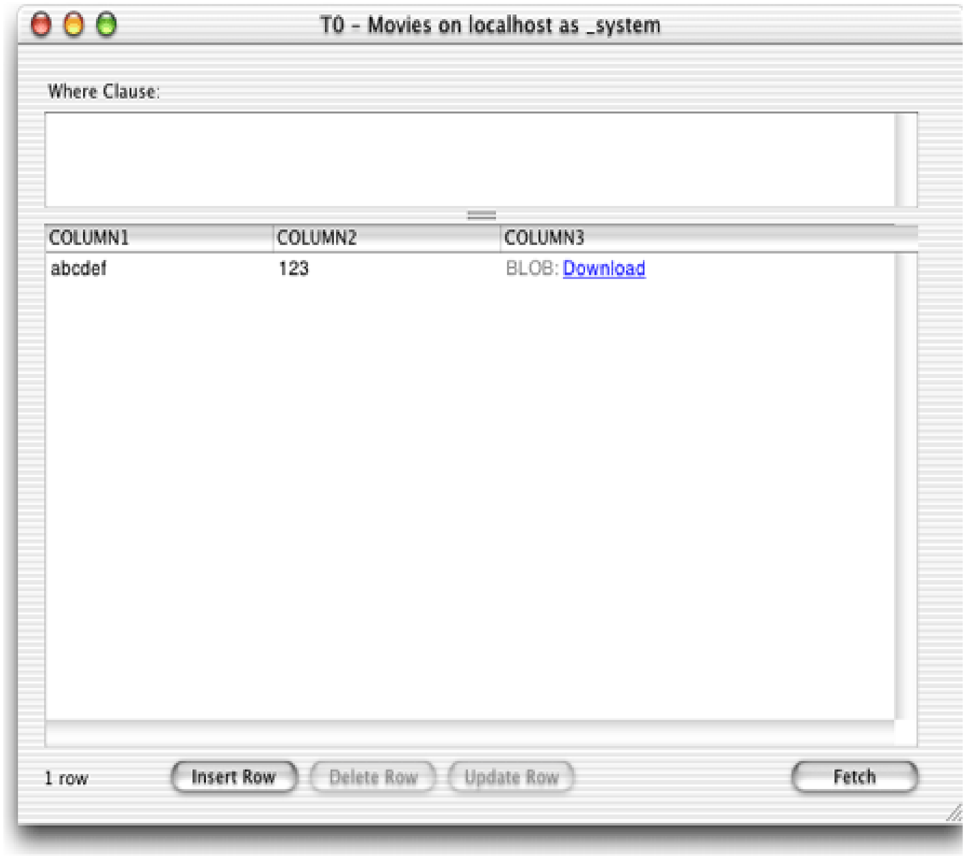
## BLOB and CLOB handling

FrontBaseManager provides an intuitive user interface to upload and download large objects. If a table or view has a column with datatype BLOB or CLOB, you cannot edit these values directly in the content editor. However, you can specify a file containing content that you would like in the CLOB/BLOB column and FrontBase-Manager will read in the file and set the column to that value.



| Column | Datatype | Value | Null |
|---|---|---|---|
| COLUMN1 | VARCHAR | abcdef | ■ |
| COLUMN2 | VARCHAR | 123 | ■ |
| COLUMN3 | BLOB | BLOB: Upload | ■ |

If a CLOB/BLOB column contains content, it will display a hyper-link in the content browser to download that content to a file on your local file system. Clicking on the Download link will bring up a "Save File" dialog, asking you for a filename in which to save the large object data.



| COLUMN1 | COLUMN2 | COLUMN3 |
|---|---|---|
| abcdef | 123 | BLOB: Download |

1 row

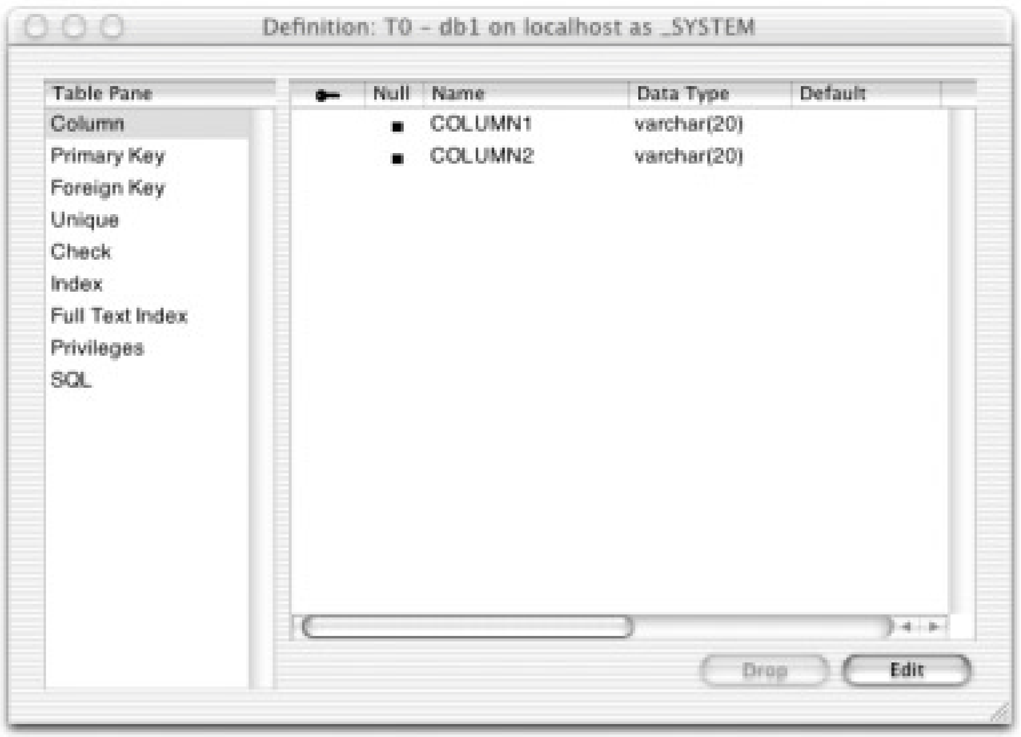
## Open Schema Object Definition

The Schema Object definition window gives you information about the schema object itself. The view, procedure, and function definition windows give you the definition information (which you specified when you created the schema object) and allows you to change the privileges on the schema object. The collation definition window gives the name of the collation, the schema name, the type, and the external name. The table definition window gives you quite a bit more information which we'll look at in detail.

## Table Definition Window

The table definition window gives you access to a great deal of information about the table. Let's go through each of the panes, one at a time. Note that to alter table columns, primary keys, foreign keys, etc., you must set your transaction settings to Serializable, Pessimistic. You can change your transaction settings through the Session pane on the Connection window or by right-clicking in the SQL Interpreter.

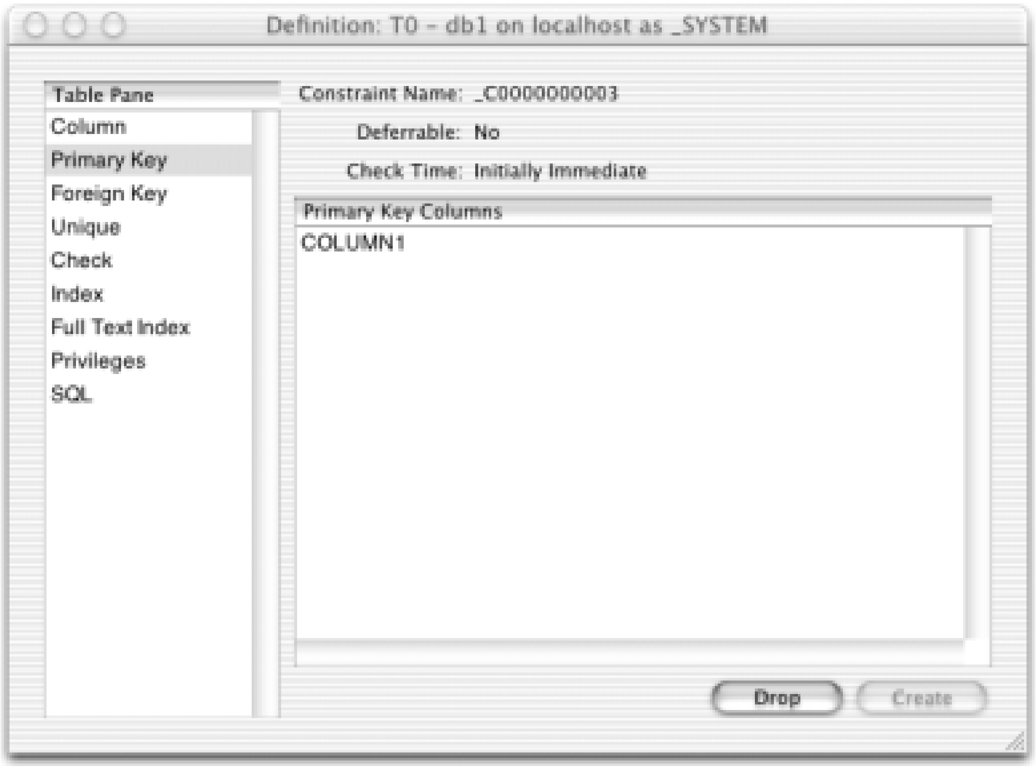
## Column Definition pane



| | Null | Name | Data Type | Default |
|---|---|---|---|---|
| | ■ | COLUMN1 | varchar(20) | |
| | ■ | COLUMN2 | varchar(20) | |

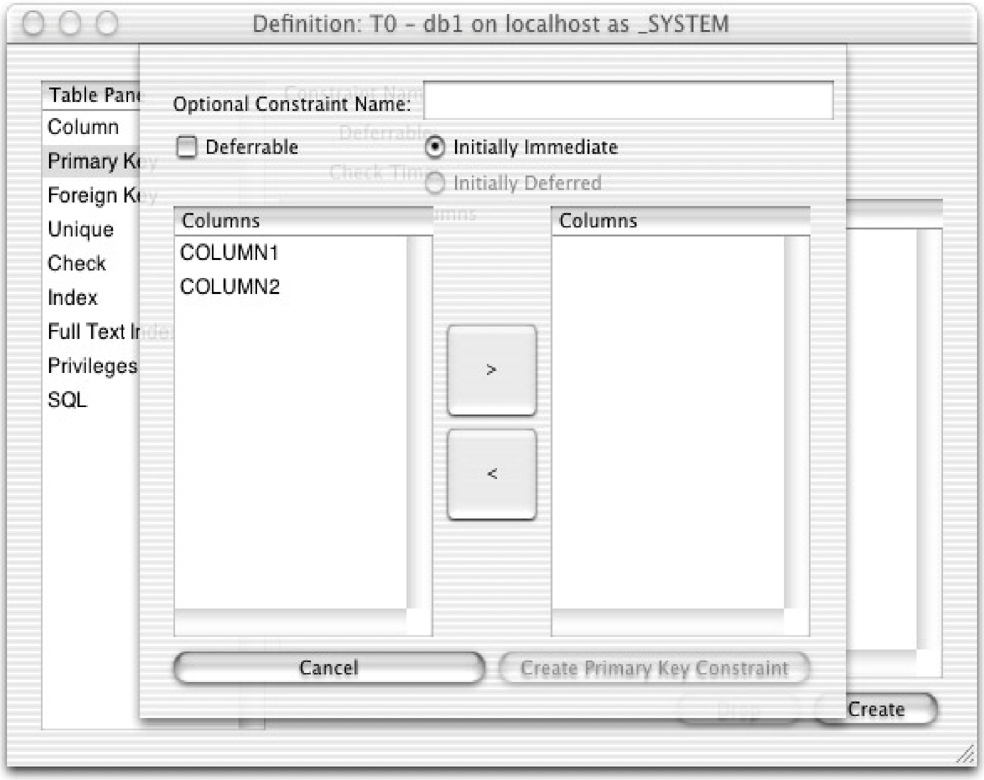
The Column definition pane of the table definition window gives you a chance to drop or edit columns in a defined table. To drop a column, simply select the column and click the "Drop" button. Note that you cannot drop all the columns in the table because a table must have at least one column. You can edit the table columns by clicking the edit button and proceed to set primary key and not null constraints, column names, datatypes, widths, precision, scale, set default values, specify a column to be normalized, and specify a collation on a column. Notice that as you're altering the table in the column definition pane, the SQL that will be executed is generated in the lower pane. Click "Update" to actually send the SQL to the server.

## Primary Key pane



Constraint Name: _C0000000003
Deferrable: No
Check Time: Initially Immediate
COLUMN1

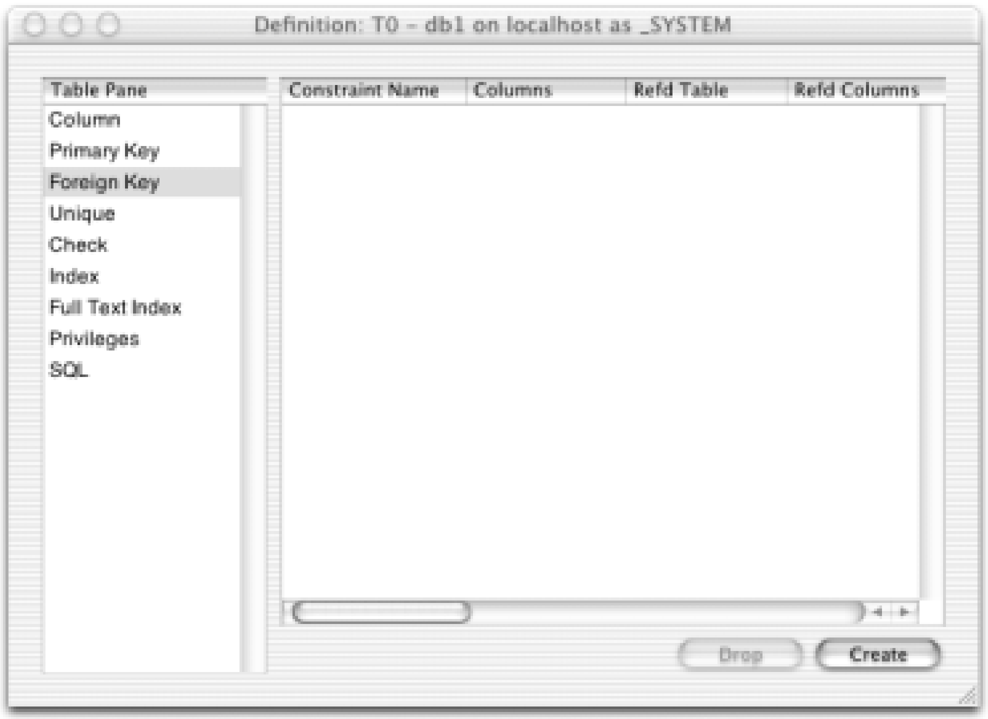
The Primary Key definition pane of the table definition window lets you drop and create primary key constraints. To create a primary key constraint, click the Create button to bring up the primary key creation sheet.



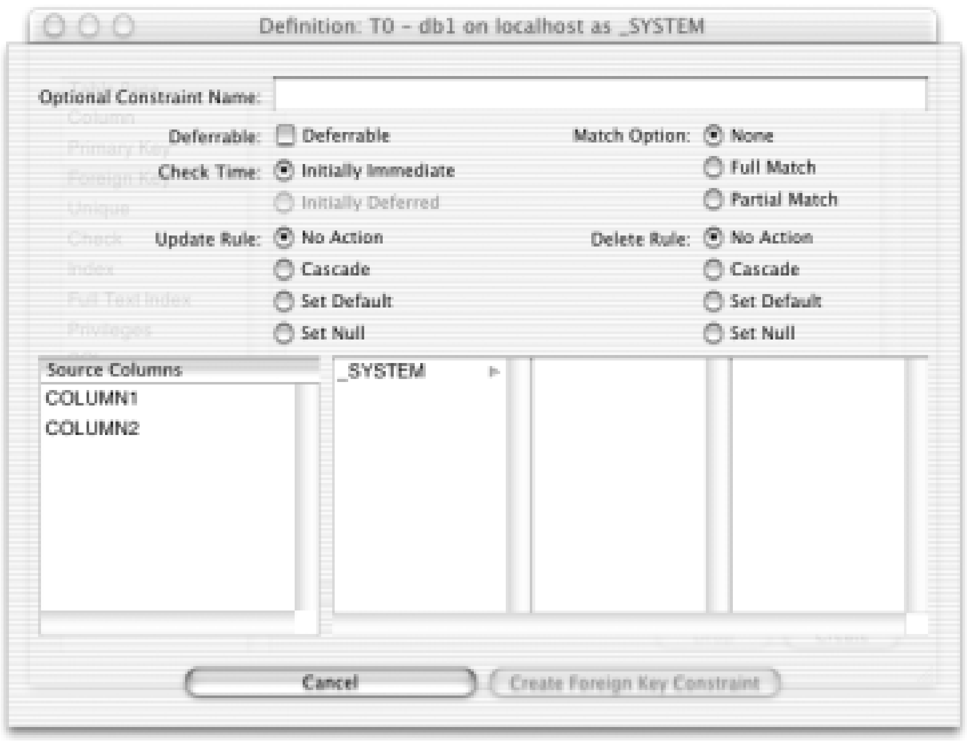
From here you can specify an (optional) constraint name, set whether the constraint is deferrable and, if it is deferrable, whether it is initially deferred. You can then specify which columns should participate in the primary key constraint and click the "Create Primary Key Constraint" button. If your transaction settings were correct, the sheet will retract and you'll have a new primary key constraint.

To drop the primary key constraint, simply click the "Drop" button.

## Foreign Key pane



The Foreign Key definition pane of the table definition window lets you drop and create foreign key constraints. To create a foreign key constraint, click the Create button to bring up the foreign key creation sheet.


Definition: T0 – db1 on localhost as _SYSTEM

Optional Constraint Name:

Deferrable: [ ] Deferrable

Check Time: [✓ Initially Immediate] Initially Deferred

Update Rule: [✓ No Action] Cascade Set Default Set Null

Match Option: [✓ None] Full Match Partial Match

Delete Rule: [✓ No Action] Cascade Set Default Set Null

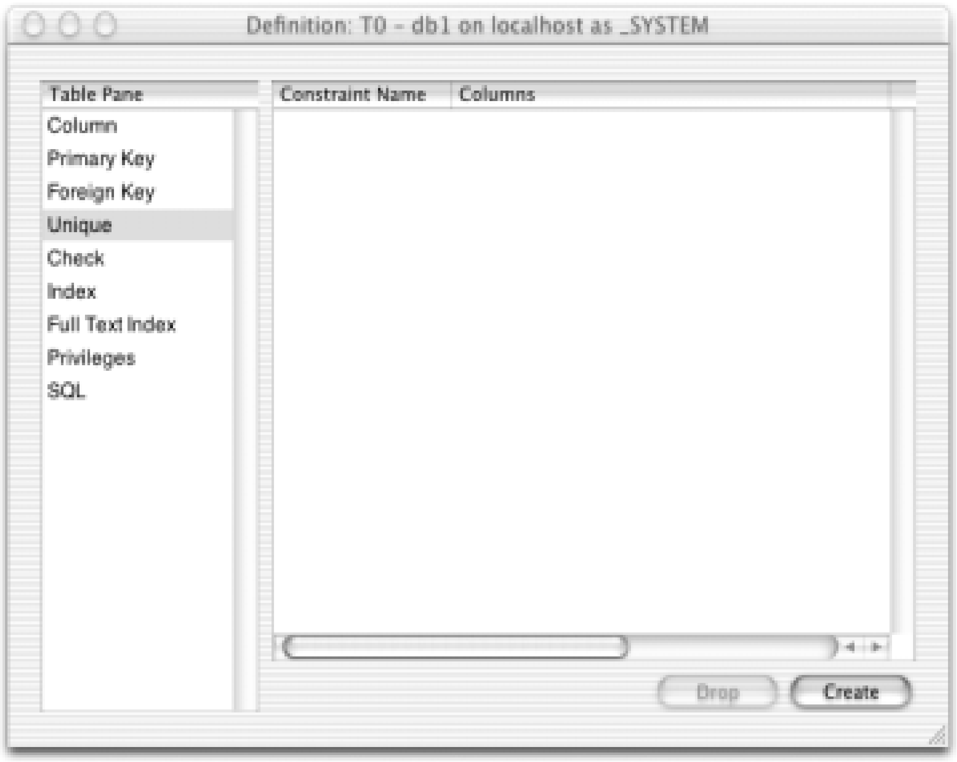
| Source Columns |
|---|
| COLUMN1 |
| COLUMN2 |

_SYSTEM

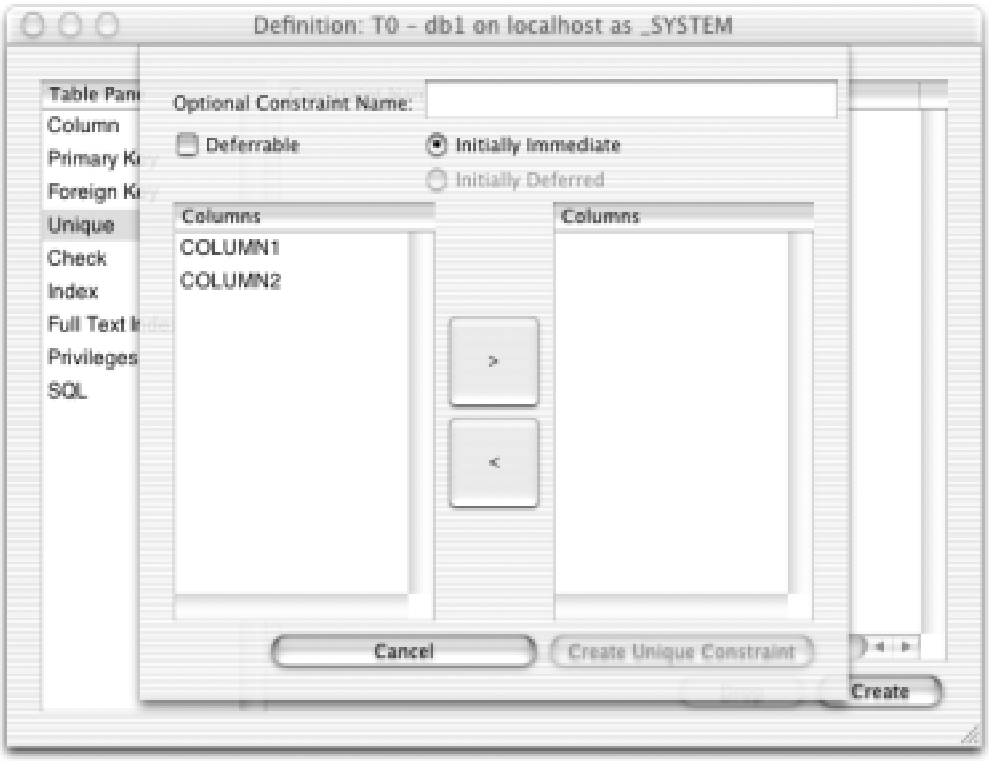
This sheet may look daunting but the only two things you really have to specify are the source column(s) (via the "Source Columns" browser) and the destination columns (via the browser to the right of the "Source Columns" browser). The number of source columns and destination columns must, of course, match. The other options (Deferrable, Check Time, Update Rule, Match Option, Delete Rule) can be changed if you want foreign key behavior other than the de-fault behavior.

To drop a foreign key constraint, simply select the constraint and click the "Drop" button.

## Unique pane



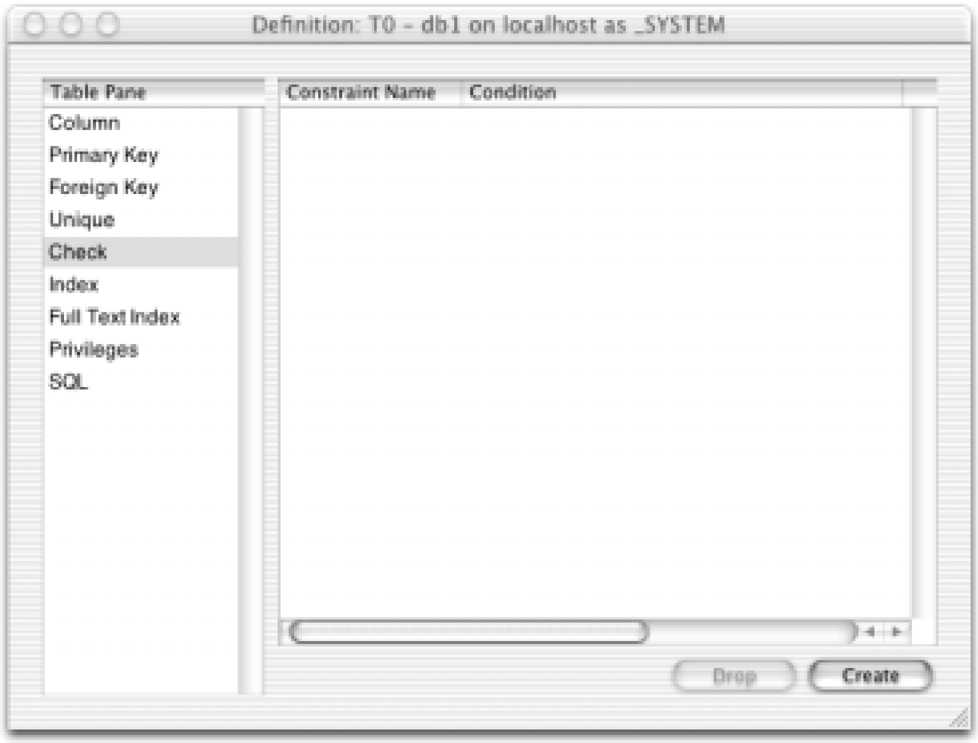
The Unique definition pane of the table definition window presents an interface to create and drop unique constraints.  To create a unique constraint, click the "Create" button to bring up the Unique creation sheet.



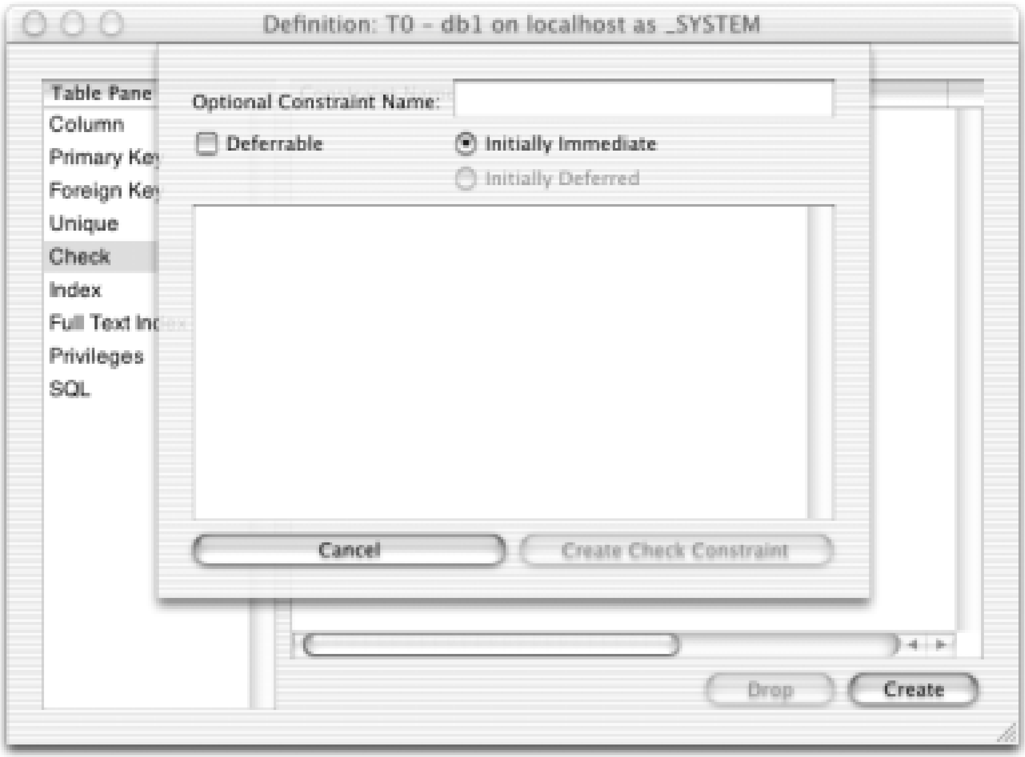
From here you can specify an (optional) constraint name, set whether the constraint is deferrable and, if it is deferrable, whether it is initially deferred. You can then specify which columns should participate in the unique constraint and click the "Create Unique Constraint" button.

To drop a unique constraint, simply select the constraint and click the "Drop" button.

## Check pane



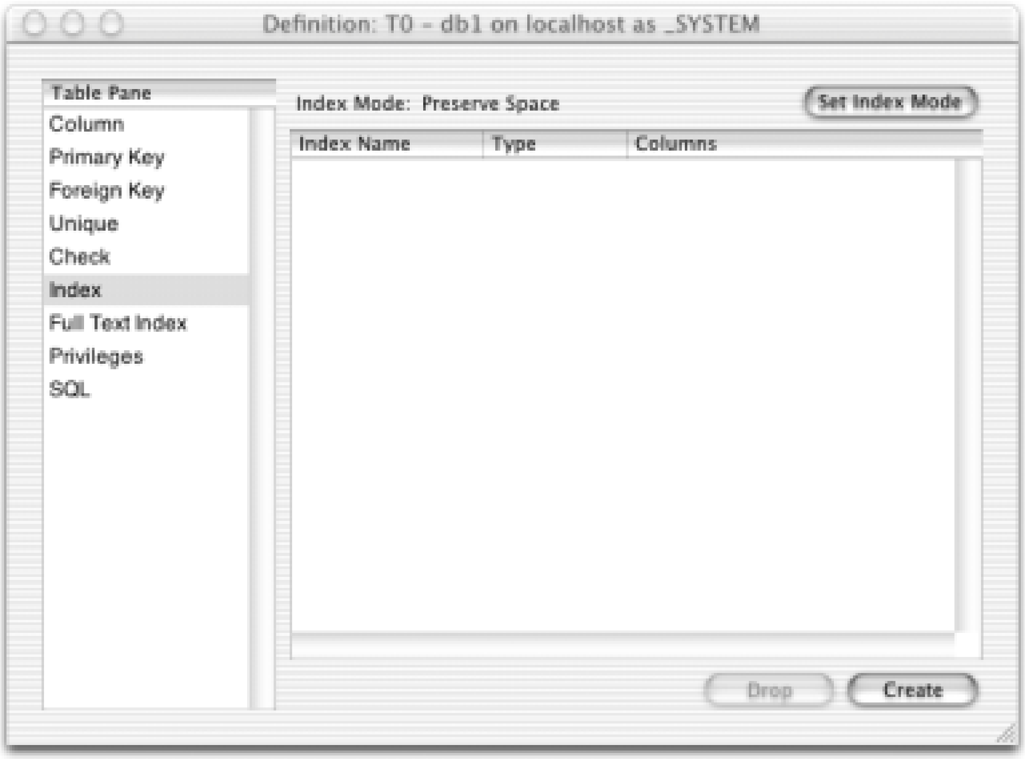
The Check definition pane of the table definition window allows to create and drop check constraints. To create a check constraints, click the "Create" button to bring up the Check creation sheet.



From here you can specify an (optional) constraint name, set whether the constraint is deferrable and, if it is deferrable, whether it is initially deferred. You can then specify a freeform check constraints in the definition area. (such as "column1 > 3" for an int column1 for which you want to enforce only values > 3). You can then click the "Create Check Constraint" to create the check constraint.

To drop a check constraint, simply select the constraint and click the "Drop" button.

## Index pane



The Index definition pane shows all the indexes you currently have in place, allows you to set the index mode, allows you to drop existing indexes, and create new indexes.

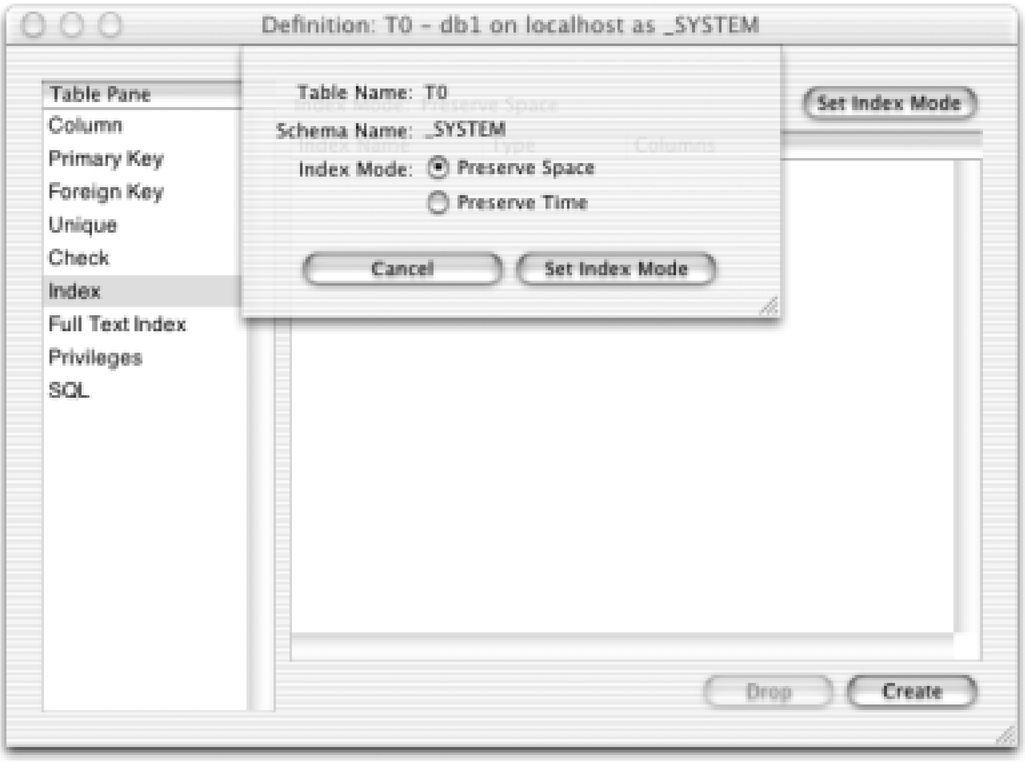
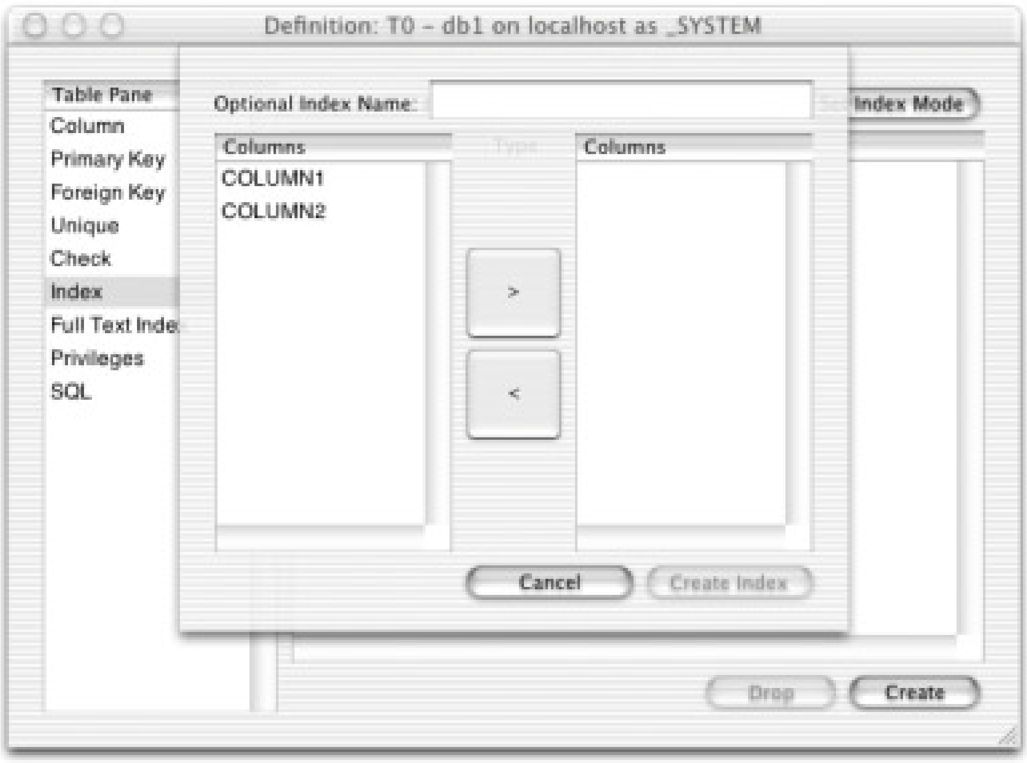


Table Name: T0
Schema Name: _SYSTEM
Index Mode: [✓ Preserve Space] Preserve Time

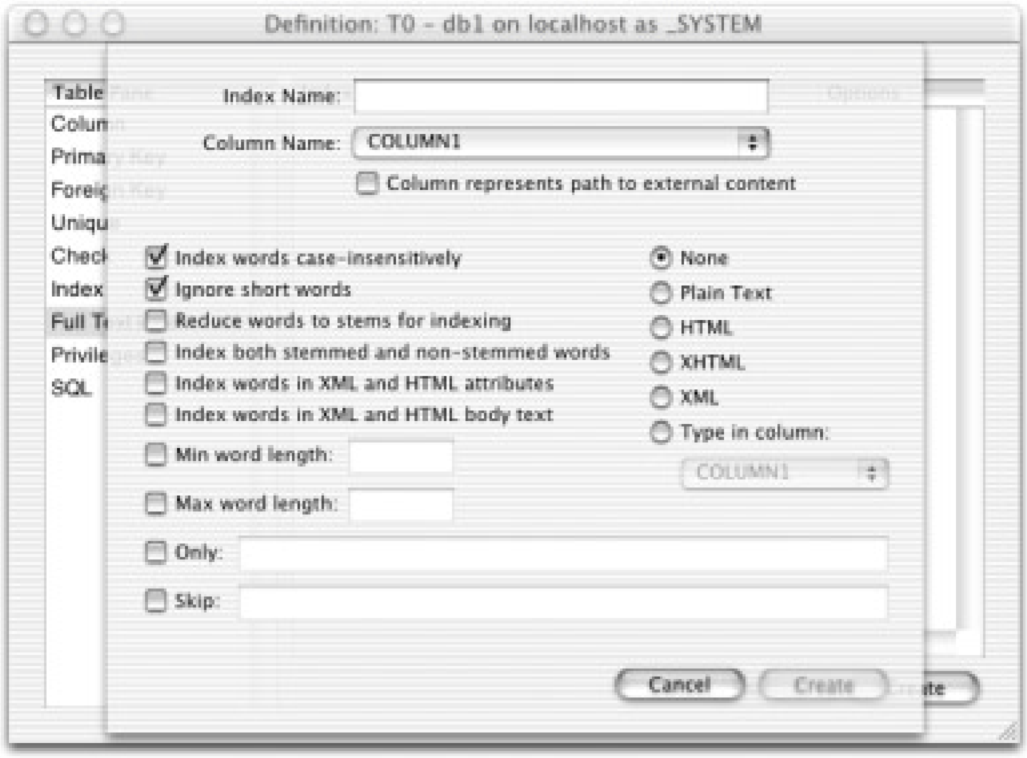
To set the index mode, click the "Set Index Mode" button and you'll be presented with the choice of either "Preserve Space" or "Preserve Time". Make your choice, click "Set Index Mode" and FrontBaseManager will change your index mode. You must be in Serializable, Pessimistic mode to change the index settings.

To drop an index, select the index and click "Drop". You must be in Serializable, Pessimistic mode to drop an index.



To create an index, click the "Create" button. When this sheet comes down, you can specify an (optional) index name and then specify which columns should participate in the index and click the "Create Index" button. You must be in Serializable, Pessimistic mode to create an index.

## Full Text Index



Index Name:
Column Name: COLUMN1
☐ Column represents path to external content

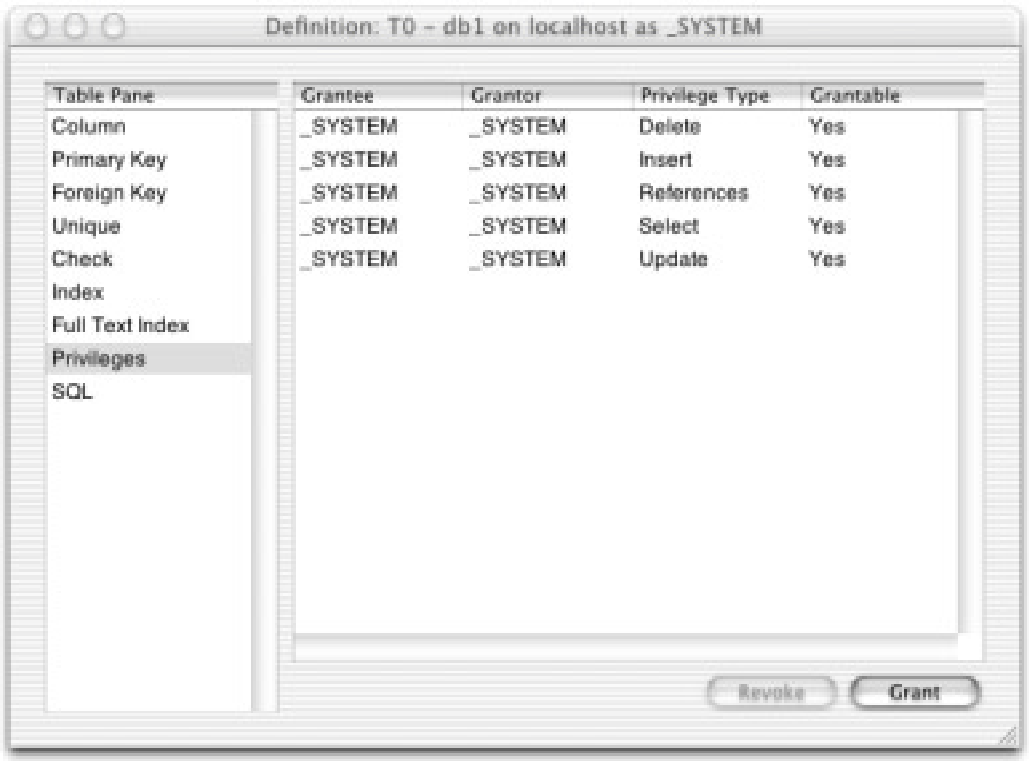
☑ Index words case-insensitively
☑ Ignore short words
☐ Reduce words to stems for indexing
☐ Index both stemmed and non-stemmed words
☐ Index words in XML and HTML attributes
☐ Index words in XML and HTML body text
☐ Min word length:
☐ Max word length:
☐ Only:
☐ Skip:

[✓ None] Plain Text HTML XHTML XML Type in column: COLUMN1

The full text index pane allows you create and drop full text indexes. To create a full text index, click the "Create" button to bring down the full text index creation sheet. For full text indeces, you must specify an index name. Next, specify a column name. For a description of all the different full text index options, see the LookSee documentation. (LookSee is the full text indexing engine.) After specifying the options, click "Create" to create the full text index.

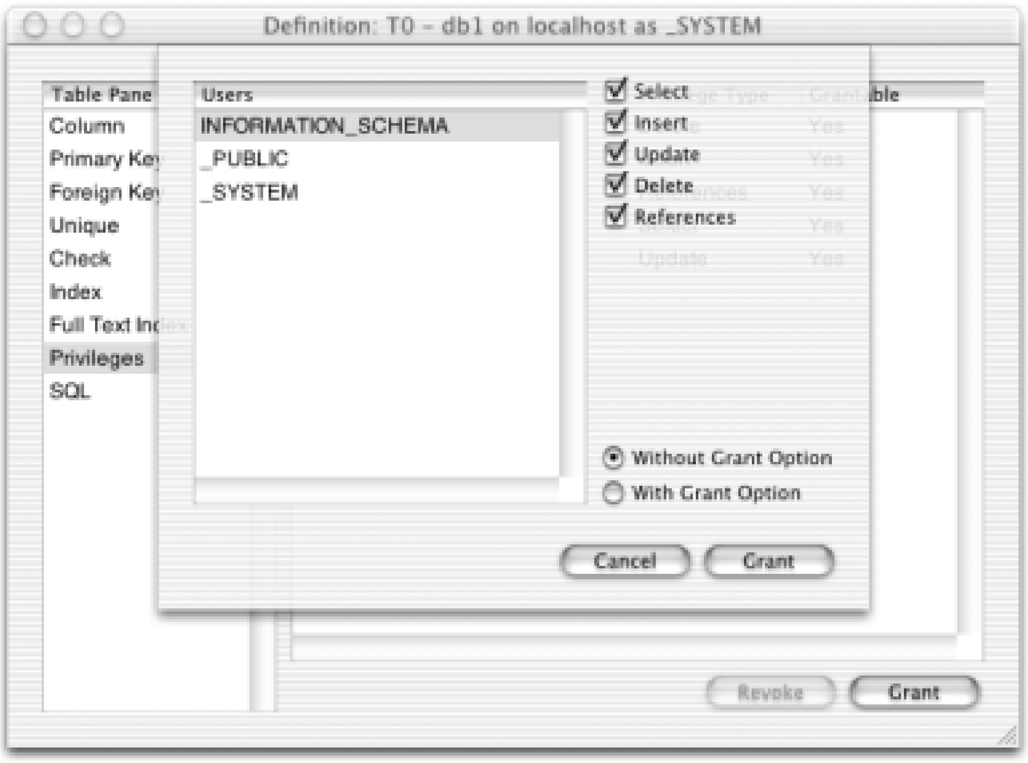
To drop a full text index, simply select the index and click the "Drop" button.

## Privileges



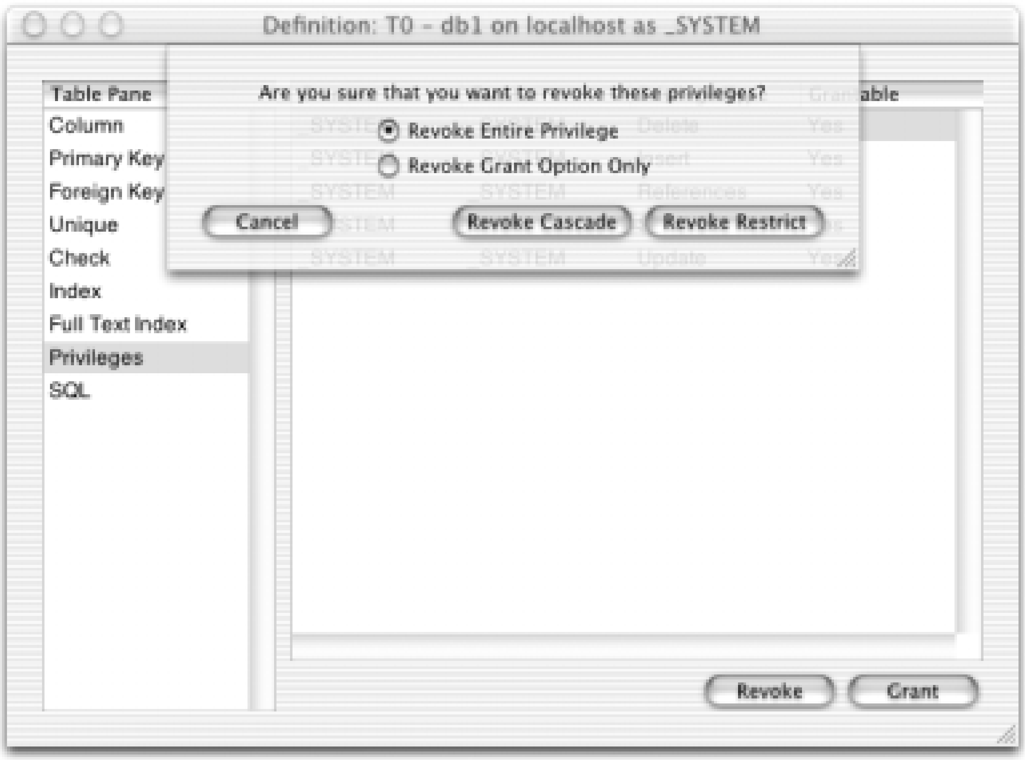
| Grantee | Grantor | Privilege Type | Grantable |
|---|---|---|---|
| _SYSTEM | _SYSTEM | Delete | Yes |
| _SYSTEM | _SYSTEM | Insert | Yes |
| _SYSTEM | _SYSTEM | References | Yes |
| _SYSTEM | _SYSTEM | Select | Yes |
| _SYSTEM | _SYSTEM | Update | Yes |

The Privileges pane shows you the currently granted privileges in the database.  The columns are Grantee, Grantor, Type, and Grantable.  Grantable means whether the Grantee can grant the privilege to other users.  To grant privileges, make sure the connection's user has grant privileges and click the "Grant" button.



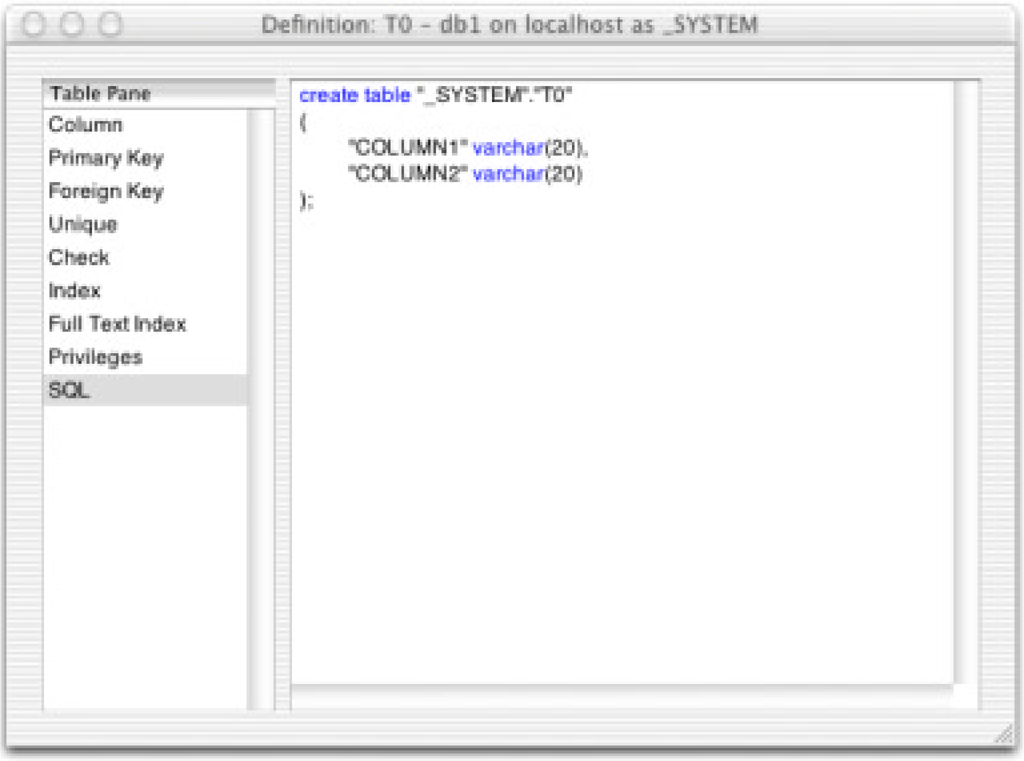
You'll see all the database users in browser on the left. Select the grantee, check all the privilege types you'd like to grant, select whether you'd like to grant without the grant option (the default) or with the grant option, and click the "Grant" button.

To revoke privileges, select the privileges you'd like to revoke and click the "Revoke" button.



You'll be given a choice to either revoke the entire privilege or revoke the grant option only (so that the user cannot grant the privilege to other users). If you revoke restrict, it will only revoke the privileges you've selected. If you revoke cascade, it will revoke the privileges you selected and all the privileges that have been granted using that privilege. (If BOB grants privileges to JIM and JIM grants privileges to SUE, revoking BOB's privilege cascade will revoke BOB's, JIM's, and SUE's privileges.)

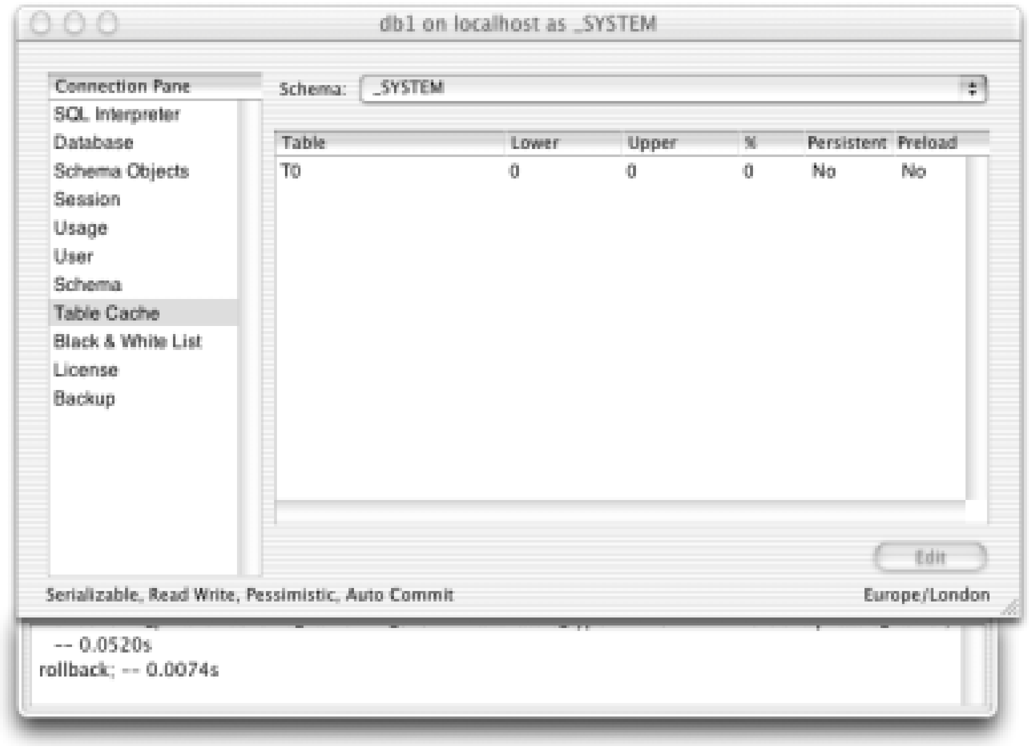
## SQL



```
create table "_SYSTEM"."T0"
(
	"COLUMN1" varchar(20),
	"COLUMN2" varchar(20)
);
```

This pane simply shows the SQL which generated the table.

## Table Cache



Schema: _SYSTEM

| Table | Lower | Upper | % | Persistent | Preload |
|---|---|---|---|---|---|
| T0 | 0 | 0 | 0 | No | No |

Serializable, Read Write, Pessimistic, Auto Commit    Europe/London

```
  -- 0.0520s
rollback; -- 0.0074s
```

The Table Cache pane contains information about the table cache. The table cache can be used to fine-tune table caching. All the tables in the selected schema are shown on this panel. The parameters are:

Lower - The minimum number of rows of the table to be kept in the cache

Upper - The maximum number of rows of the table to be kept in the cache

% The percentage of the total number of rows to be kept in the cache

Persistent - If Yes, the cache is kept across transactions. Otherwise, the cache is flushed after each COMMIT or ROLLBACK.

Preload - If Yes, all rows in the table will be loaded into the cache when the server is started. Otherwise, the rows are loaded upon the first reference to them.

To change any of these settings, select the table you want to change and click the "Edit" button.

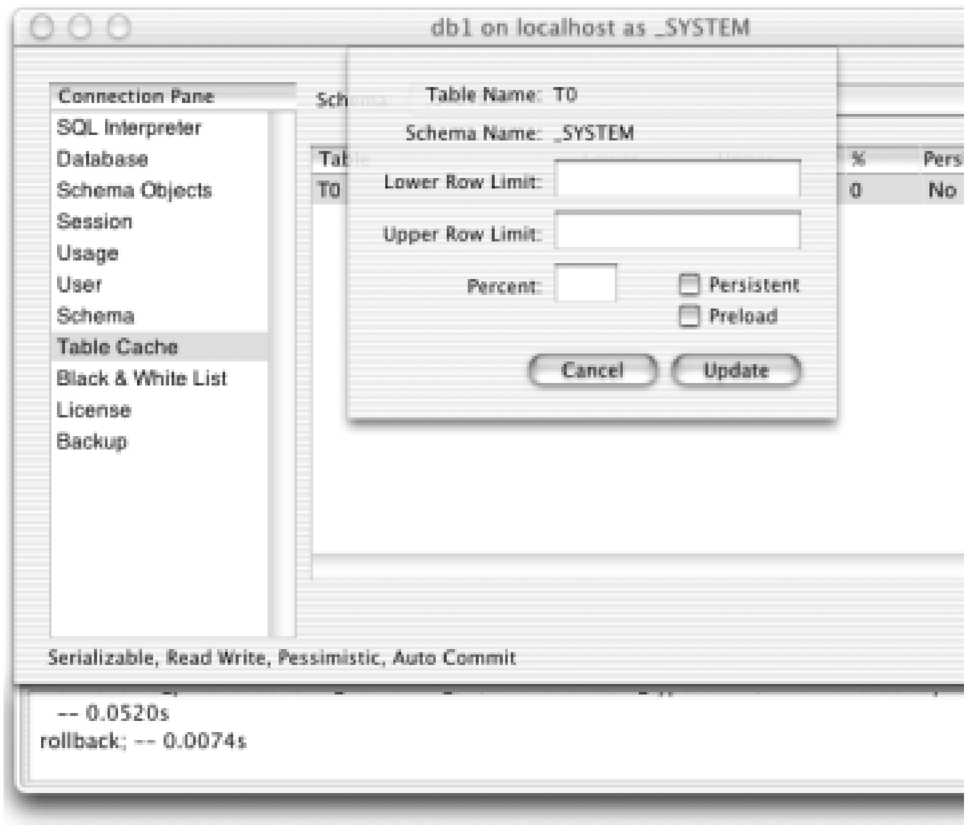


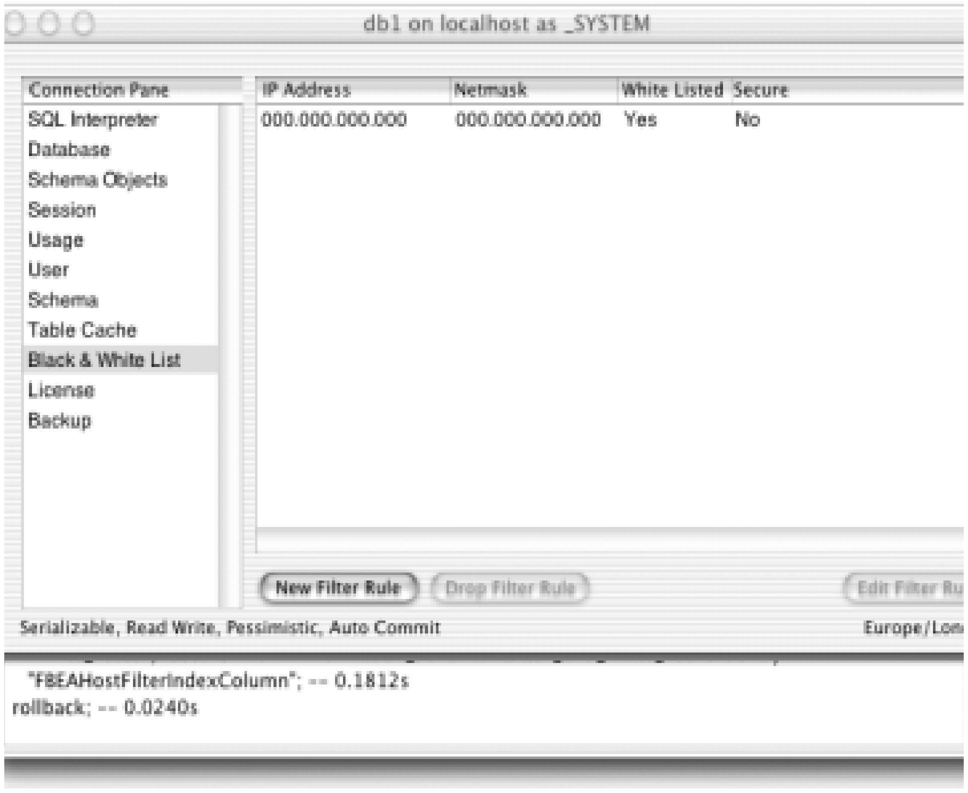
Table Name: T0
Schema Name: _SYSTEM
Lower Row Limit:
Upper Row Limit:
Percent:
Persistent
Preload

| Tab | % | Pers |
|---|---|---|
| T0 | 0 | No |

Serializable, Read Write, Pessimistic, Auto Commit

```
  -- 0.0520s
rollback; -- 0.0074s
```

## Black & White List



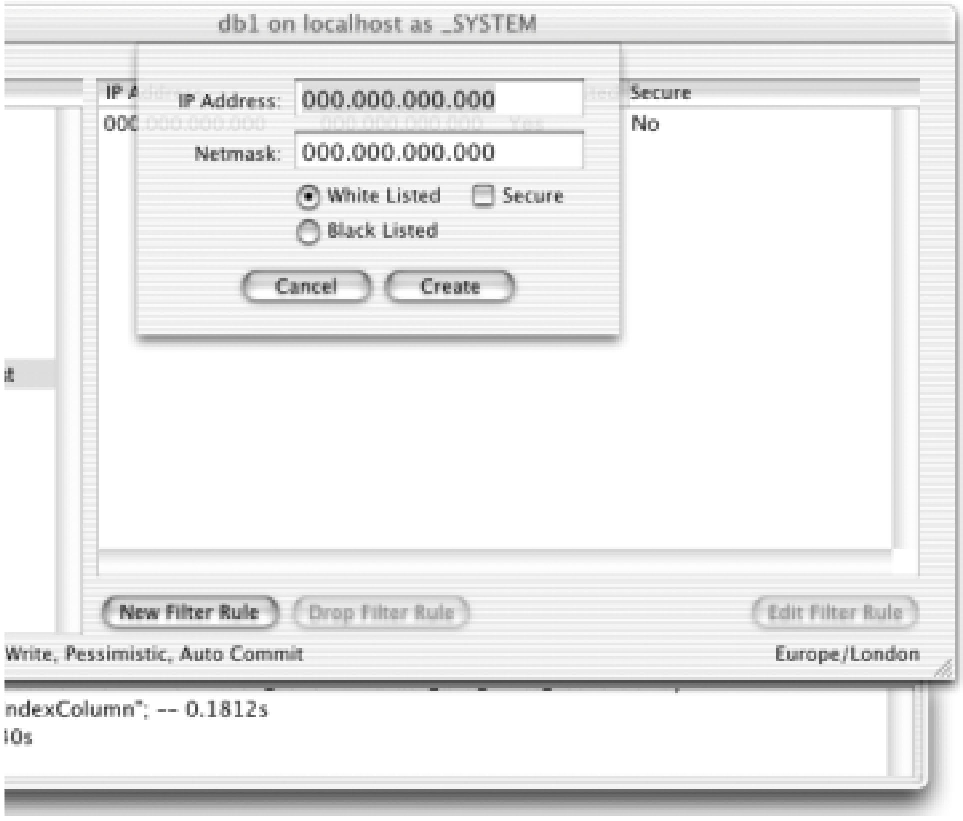
| IP Address | Netmask | White Listed | Secure |
|---|---|---|---|
| 000.000.000.000 | 000.000.000.000 | Yes | No |

Serializable, Read Write, Pessimistic, Auto Commit Europe/Lon

```
 "FBEAHostFilterIndexColumn"; -- 0.1812s
rollback; -- 0.0240s
```

The pane contains information about the black and white list. When a client connects to a FrontBase server, its IP address is checked against the black and white list to see if the client is allowed to connect. For each entry in the black and white list, the IP address of the client is AND'ed with the netmask and the result is compared with the IP address in the white and black list. If a match is found and the entry is a white listing, the client is allowed to connect. Otherwise, the connection is refused. If no match is found, the connection is refused. If a YES is entered in entered in the secure field, the client is requested to use encryption for all further communication with the server.

To add a new filter rule, click the "New Filter Rule" button.



IP Address: 000.000.000.000
Netmask: 000.000.000.000
[✓ White Listed] Black Listed
Secure [ ]

Write, Pessimistic, Auto Commit
Europe/London
ndexColumn"; -- 0.1812s

Enter the netmask and IP address that describes the host or network you'd like to white list or black list. Select White List or Black List from the radio button and click the Secure checkbox if you'd like the server to request that the client use encryption for their communication. Click the "Create" button and you'll have a new list entry.

To drop a filter rule, simply highlight the rule you'd like dropped and click the "Drop Filter Rule" button.

To edit a filter rule, highlight the rule you'd like to edit and click the "Edit Filter Rule" button. A sheet will come down allowing you to edit the filter rule. After you change the values you'd like to change, click "Update" to save your changes.

Filter rules are evaluated from the top of the black and white list to the bottom of the list. As soon as a match is found, evaluation stops and the rule is applied. So, if you have IP 000.000.000.000 / Netmask 000.000.000.000 white listed, every single IP will be able to connect, no matter how many black list entries you have below the first white list entry. Here are some examples of how to use the Black & White List

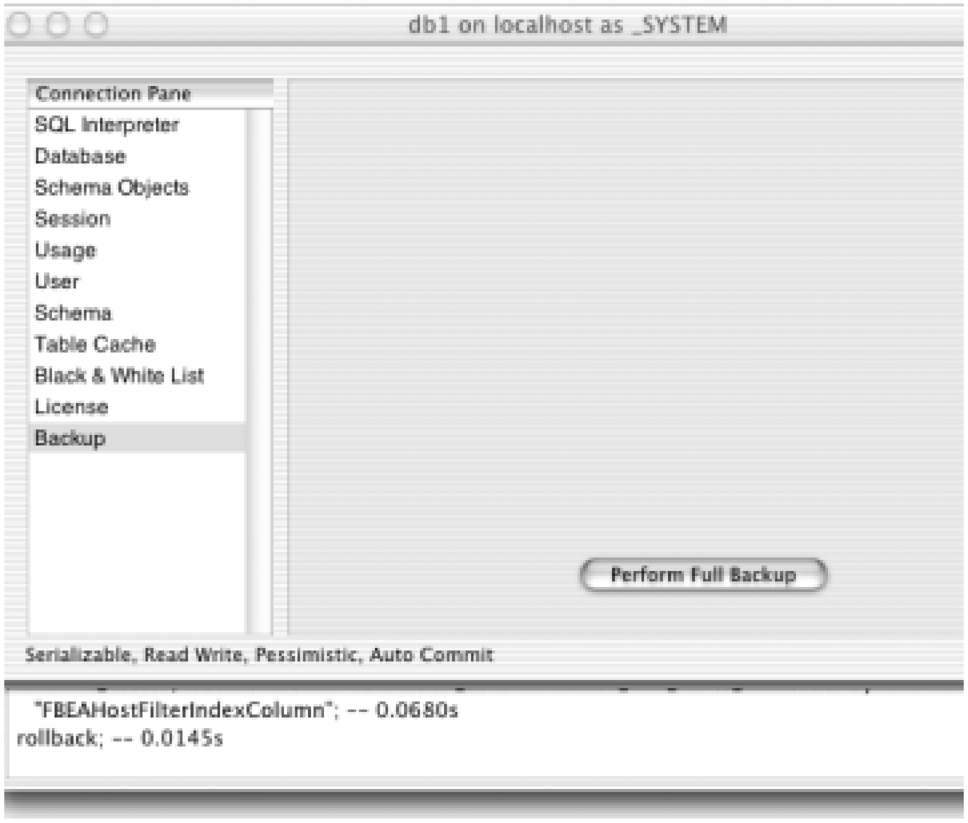
| IP | Netmask | White Listed | Secure |
|---|---|---|---|
| 24.5.126.145 | 255.255.255.255 | NO | NO |
| 000.000.000.000 | 000.000.000.000 | YES | NO |

This would lock out the single IP at 24.5.126.145 and would allow connections from every other IP.

| IP | Netmask | Whitle Listed | Secure |
|---|---|---|---|
| 24.5.126.0 | 255.255.255.0 | YES | NO |
| 24.5.127.0 | 255.255.255.0 | YES | NO |
| 128.0.64.2 | 255.255.255.255 | YES | NO |

This would allow access from two class C network (24.5.126.0-255 and 24.5.127.0-255) and one single IP address (128.0.64.2) and would refuse connection to every other IP.

## Backup



Serializable, Read Write, Pessimistic, Auto Commit

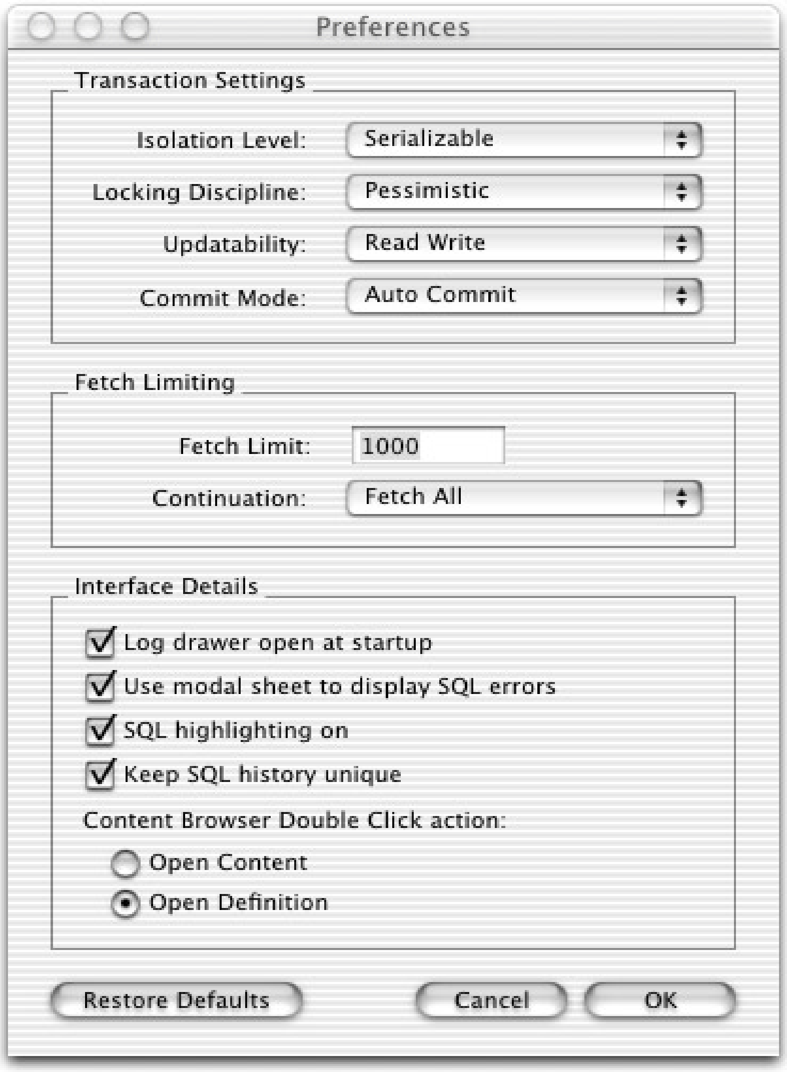
```
  "FBEAHostFilterIndexColumn"; -- 0.0680s
rollback; -- 0.0145s
```

The Backup pane simply has a "Perform Full Backup" button on it. If you click the button, a sheet will come down with a button to "Begin Backup". It also has a checkbox (defaulted to checked) which will instruct the server to keep a transaction log for incremental restore.

## Odds-n-Ends

Not all FrontBaseManager features are accessible through the main connection window panes. Some are accessed from the pull-down menus or keyboard shortcuts. This section details those features:

## Preferences



Transaction Settings

Isolation Level: Serializable

Locking Discipline: Pessimistic

Updatability: Read Write

Commit Mode: Auto Commit

Fetch Limiting

Fetch Limit: 1000

Continuation: Fetch All

Interface Details

- [✓] Log drawer open at startup
- [✓] Use modal sheet to display SQL errors
- [✓] SQL highlighting on
- [✓] Keep SQL history unique

Content Browser Double Click action:

Open Content [✓ Open Definition]

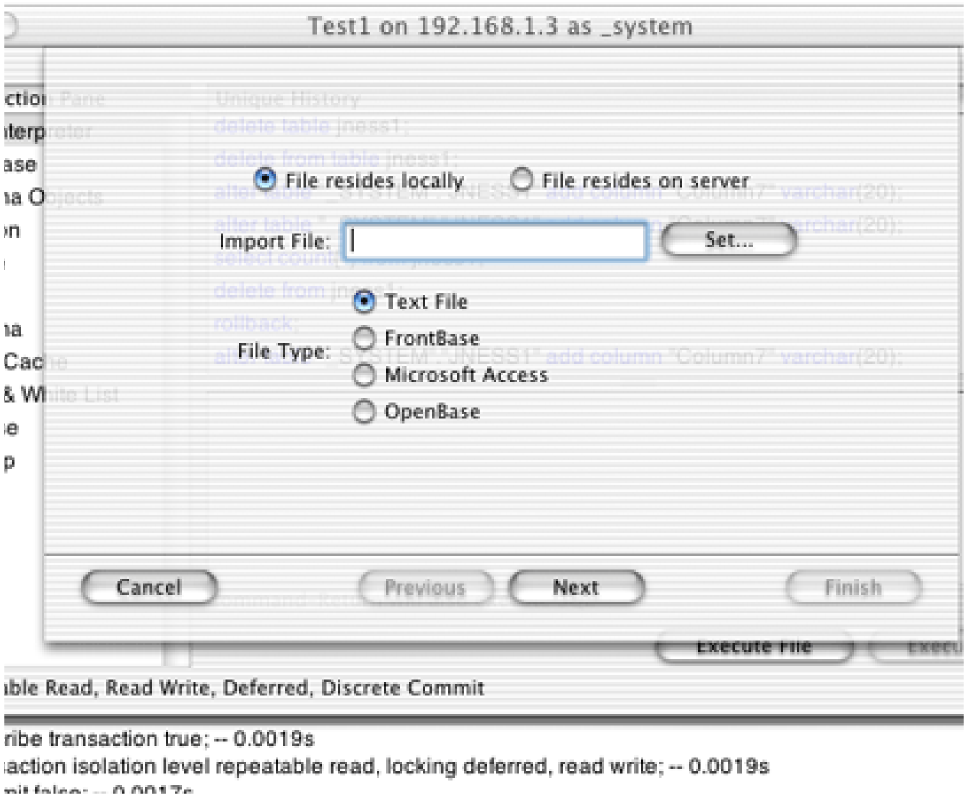
FrontBaseManager's preferences system allows you to specify your startup transaction settings, fetch limiting, and a few interface details so that you don't have to keep switching from the factory defaults to the setup that you prefer. The transaction settings specified in your preferences will be set every time you start up a new database connection.

Your fetch limit will also be set when you start a new database connection. The continuation action tells the server what to do when it hits the fetch limit. The two choices are to stop the fetch or to fetch all the available rows.

The interface details sections allows you to specify whether you want the SQL log drawer open or closed when the connection window first opens. (You can, of course, change it manually. See section 4.2 for how to do so.) SQL highlighting is a powerful feature that can greatly reduce SQL92 syntax errors by showing you which words are SQL92 reserved keywords. (One benefit of this is reminding you to wrap your column names in quotes if they are named the same as any SQL92 reserved keyword.) Finally, you can set a preference to keep your SQL history in the SQL Interpreter unique. This means that if you issue "select * from foo;" then some other SQL statement, then "select * from foo;" again, you will not get a second "select * from foo;" in the history. Instead, the SQL history will move the "select * from foo;" already in the history to the bottom of the history list.

## File Import

FrontBaseManager version <version-number> first introduced an intuitive user interface to take advantage of FrontBase's powerful import functionality. To access the Import helper interface, select File->Import.



[● File resides locally] ○ File resides on server

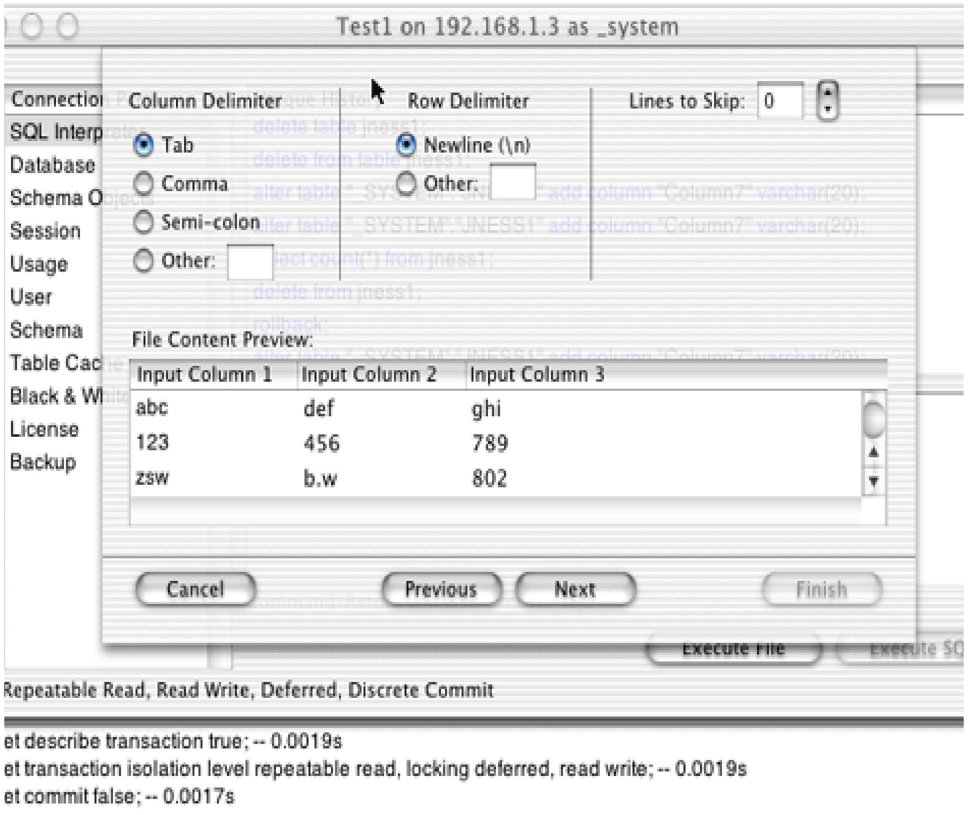
Import File:

File Type: [● Text File] ○ FrontBase ○ Microsoft Access ○ OpenBase

able Read, Read Write, Deferred, Discrete Commit

```
ribe transaction true; -- 0.0019s
action isolation level repeatable read, locking deferred, read write; -- 0.0019s
```

This first screen allows you to specify where the file is and what general format the file is in. You can choose between the file being on your local client machine or being on the database server. (If the server is your local machine, this choice will be disabled.) If the file resides locally, you can also use the "Set..." button to browse your local file system for the file. The "File Type" allows you to specify the general format of the file. If it is a FrontBase export, choose FrontBase. If Microsoft Access produced the file, choose Microsoft Access. For most flatfile imports, you'll want to choose Text File. After selecting the file location and type, click "Next" to go on to the next screen.



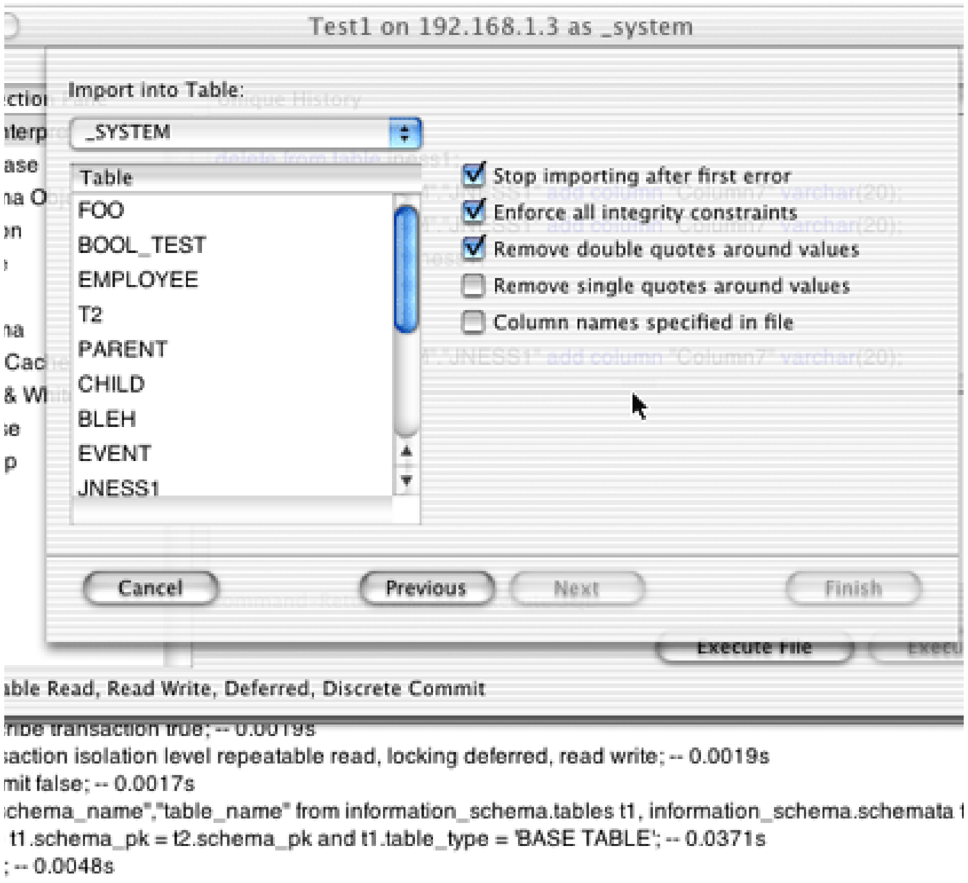
Column Delimiter: [✓ Tab] Comma Semi-colon Other:
Row Delimiter: [✓ Newline (\n)] Other:
Lines to Skip: 0

| Input Column 1 | Input Column 2 | Input Column 3 |
|---|---|---|
| abc | def | ghi |
| 123 | 456 | 789 |
| zsw | b.w | 802 |

Repeatable Read, Read Write, Deferred, Discrete Commit

```
et describe transaction true; -- 0.0019s
et transaction isolation level repeatable read, locking deferred, read write; -- 0.0019s
et commit false; -- 0.0017s
```

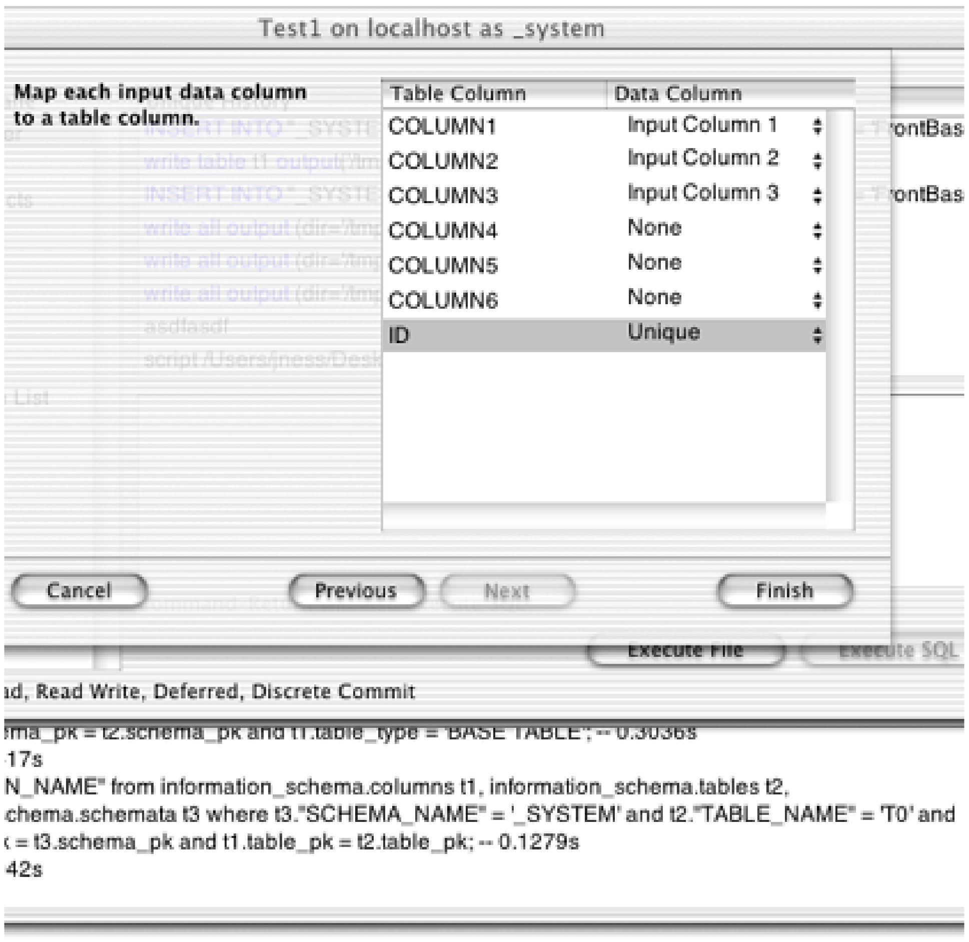
Now that you've specified where the file is located and the general format, you'll get a preview of the first few lines of the file. This is also the screen where you'll specify the column and row delimiters (the separator between each input file column and row) and the lines at the top of the file that the import process should skip. As you change the delimiters and lines to skip, your preview will change to show how the data would be imported. You'll get a chance on the next screen to choose the destination of this import.



☑ Stop importing after first error
☑ Enforce all integrity constraints
☑ Remove double quotes around values
☐ Remove single quotes around values
☐ Column names specified in file

```
able Read, Read Write, Deferred, Discrete Commit
ribe transaction true; -- 0.0019s
saction isolation level repeatable read, locking deferred, read write; -- 0.0019s
mit false; -- 0.0017s
chema_name","table_name" from information_schema.tables t1, information_schema.schemata t
 t1.schema_pk = t2.schema_pk and t1.table_type = 'BASE TABLE'; -- 0.0371s
; -- 0.0048s
```

In this screen you'll choose which table to use in this import. If you'd like, you can change schemas with the drop down above the table list to access tables from other schemas. There are also several import options on this screen. You can choose whether the process should stop if it encounters errors, whether the database server should enforce all integrity constraints (such as unique constraints and check constraints), whether single or double quotes should be removed around the values in the file, and whether the column names are specified in the file. FrontBase and OpenBase type imports have well-defined file formats with the column names specified in the file so you can just click "Finish" on this screen and you'll be done. If your file was generated from Microsoft Access, the column names are specified in the file so you can check the "Colun names specified in file" checkbox and click "Finish". However, if you are importing a text file, flat file, or Microsoft Access file without column names specified in the file, you'll need to specify the columns to use for this import. That's done on the next screen.



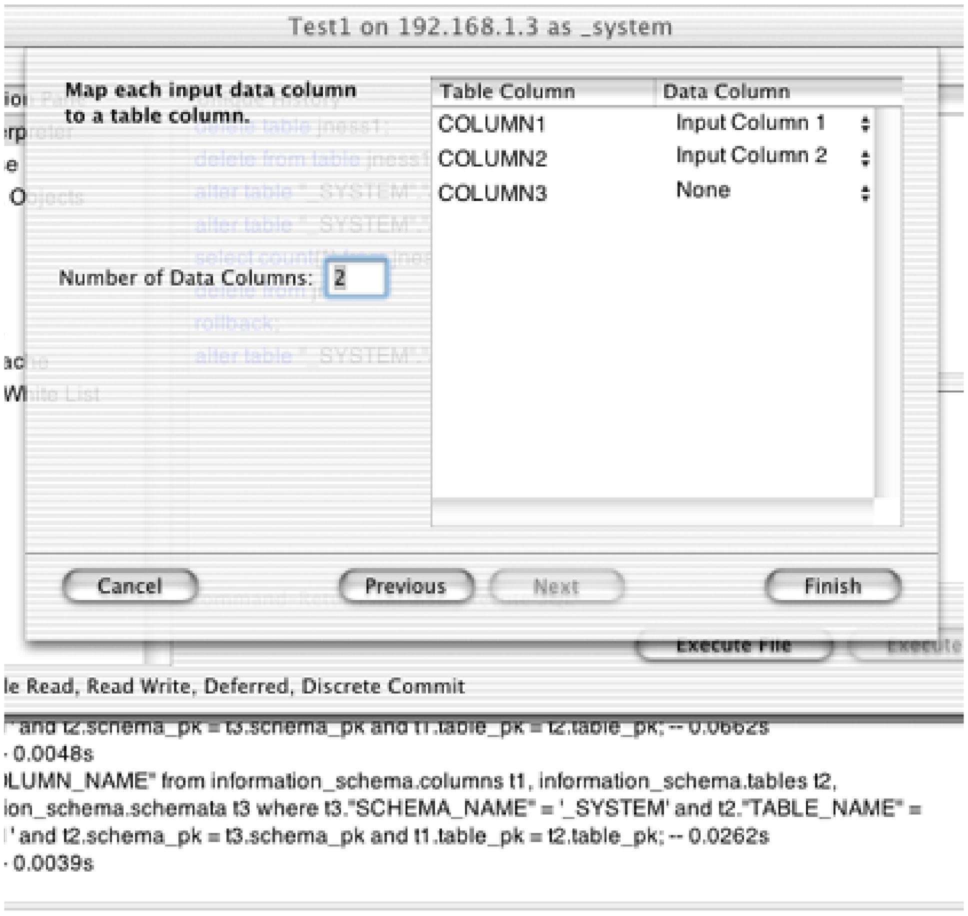
**Map each input data column to a table column.**

| Table Column | Data Column |
|---|---|
| COLUMN1 | Input Column 1 |
| COLUMN2 | Input Column 2 |
| COLUMN3 | Input Column 3 |
| COLUMN4 | None |
| COLUMN5 | None |
| COLUMN6 | None |
| ID | Unique |

ad, Read Write, Deferred, Discrete Commit

```
ma_pk = t2.schema_pk and t1.table_type = 'BASE TABLE'; -- 0.3036s
17s
N_NAME" from information_schema.columns t1, information_schema.tables t2,
chema.schemata t3 where t3."SCHEMA_NAME" = '_SYSTEM' and t2."TABLE_NAME" = 'T0' and
( = t3.schema_pk and t1.table_pk = t2.table_pk; -- 0.1279s
42s
```

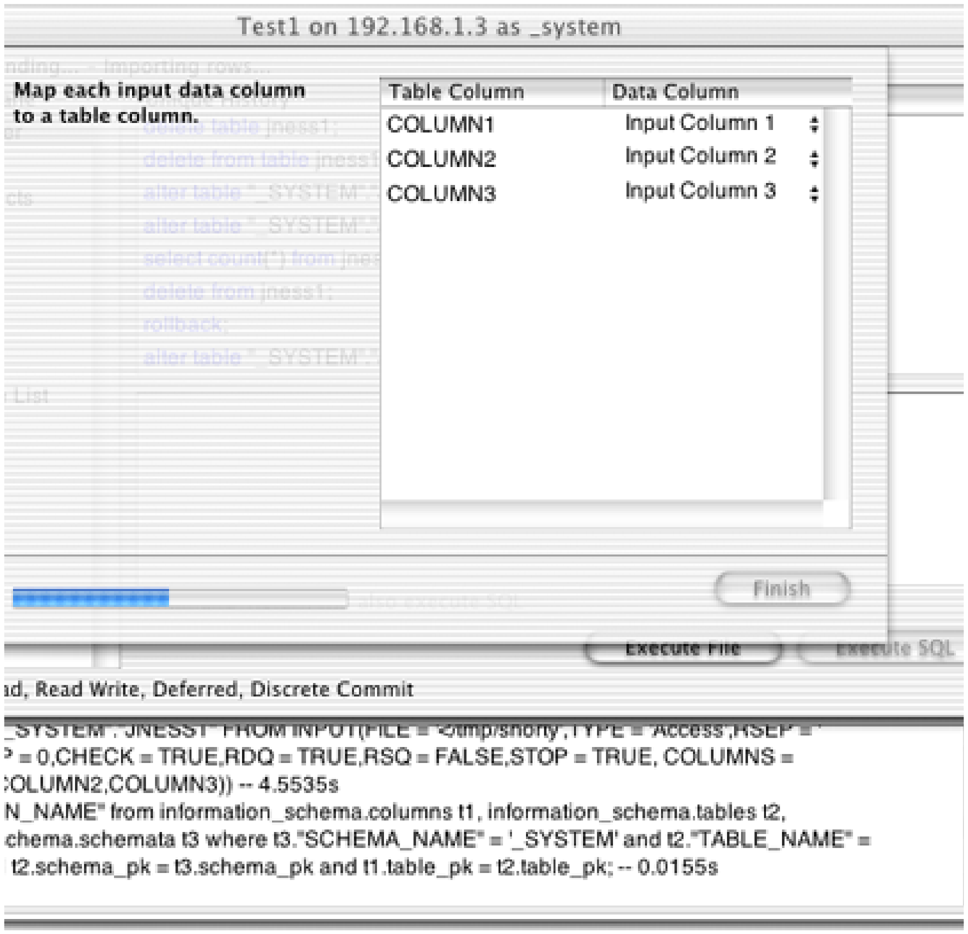
On this screen, you'll need to match up each "Input Column" with one "Table Column". Not every database table column needs to have an input column specified but *every* input column *must* have a table column specified for it. If you'd like, you can assign one of your table columns to "Unique". This table column must be a column of type INTEGER and will get a unique value for each row in the import. If the data file is local or is on the server but your server is local, FrontBaseManager will be able to open the file and count how many input columns the file has. However, if the file resides on the database server and the client cannot access the file, you'll need to specify how many input columns this file contains. If that is the case, here is the screen you'll see:


Test1 on 192.168.1.3 as _system
Map each input data column to a table column.
Number of Data Columns: 2
Cancel
Previous
Next
Finish
Execute File

| Table Column | Data Column |
|---|---|
| COLUMN1 | Input Column 1 |
| COLUMN2 | Input Column 2 |
| COLUMN3 | None |

```
le Read, Read Write, Deferred, Discrete Commit
' and t2.schema_pk = t3.schema_pk and t1.table_pk = t2.table_pk; -- 0.0662s
- 0.0048s
LUMN_NAME" from information_schema.columns t1, information_schema.tables t2,
ion_schema.schemata t3 where t3."SCHEMA_NAME" = '_SYSTEM' and t2."TABLE_NAME" =
' and t2.schema_pk = t3.schema_pk and t1.table_pk = t2.table_pk; -- 0.0262s
- 0.0039s
```

Notice how changing the number of data columns (2 in this case) changes how many input columns you'll see on the right side. Once you've matched up each input column with a table column, click "Finish" and your import will begin. If the file resides locally, you'll see a progress bar like the one in the following screen:



Map each input data column to a table column.

| Table Column | Data Column |
|---|---|
| COLUMN1 | Input Column 1 |
| COLUMN2 | Input Column 2 |
| COLUMN3 | Input Column 3 |

ad, Read Write, Deferred, Discrete Commit

```
_SYSTEM"."JNESS1" FROM INPUT(FILE = '/tmp/shorty',TYPE = 'Access',RSEP = '
P = 0,CHECK = TRUE,RDQ = TRUE,RSQ = FALSE,STOP = TRUE, COLUMNS =
COLUMN2,COLUMN3)) -- 4.5535s
N_NAME" from information_schema.columns t1, information_schema.tables t2,
chema.schemata t3 where t3."SCHEMA_NAME" = '_SYSTEM' and t2."TABLE_NAME" =
 t2.schema_pk = t3.schema_pk and t1.table_pk = t2.table_pk; -- 0.0155s
```

If the Import helper does not work for your import file, please contact tools@frontbase.com with the details of your problem.

## SQL Log

The SQL log drawer is a very handy way of knowing exactly what SQL is sent to the server. However, if you'd like, you can close the SQL Log Drawer by either dragging from the bottom edge of the SQL Log drawer up into the connection window, by selecting View->Hide SQL Log or by hitting Command-L. If the Log Drawer is hidden, you can reveal it again by selecting View->Show SQL Log or by hitting Command-L.  You can clear the SQL Log at any time by hitting Command-K.

## Known Issues

### Removing unreachable remote databases

You can wait for FrontBaseManager to time out on its attempt to connect to the unreachable database and then de-monitor the data-base in the Monitoring panel. Alternatively, you can do it manually. First, make sure that FrontBaseManager isn't running. Then, find the following file in your home directory:

Library/Preferences/com.frontbase.FrontBaseManager.plist

Open it up in a text editor and search for the "monitoredDatabases" key. You should see something like:

```
<key>monitoredDatabases</key>

<array>

    <dict>

        <key>databaseName</key>

        <string>database1</string>

        <key>hostName</key>

        <string>localhost</string>

    </dict>

    <dict>

        <key>databaseName</key>

        <string>database2</string>

        <key>hostName</key>

        <string>laptop</string>
        </dict>
    </array>
```

Delete the entry for the database(s) on your laptop. You should be left with something like:

```
    <key>monitoredDatabases</key>
    <array>
        <dict>
            <key>databaseName</key>
            <string>database1</string>
            <key>hostName</key>
            <string>localhost</string>
        </dict>
    </array>
```

8

# FBWebManager

This chapter introduces the FBWebManager web-based application, which is available for all installations of FrontBase and is accessed through any browser that supports dynamic HTML (aka JavaS-cript).

This chapter contains the following:

## Monitoring Databases

The FBWebmanager lets you monitor and control the state of Front-Base servers, i.e. whether they are running or stopped. The data-bases monitored by FBWebManager are stored in cookies on the cli-ent computer, so that the user will see the same databases next time FBWebmanager is accessed. In order to use FBWebManager you will need a Web Server such as Apache running on the server. To connect a client browser to FBWebManager you will need to pro-vide an URL. This could, for example, take the form:

http://localhost/cgi-bin/FBWebManager

where ‘localhost’ is the host name of the server machine.

### Add a Database

When FBWebmanager is started for the first time by a given user, the following "empty" window appears as in Figure 8.1.

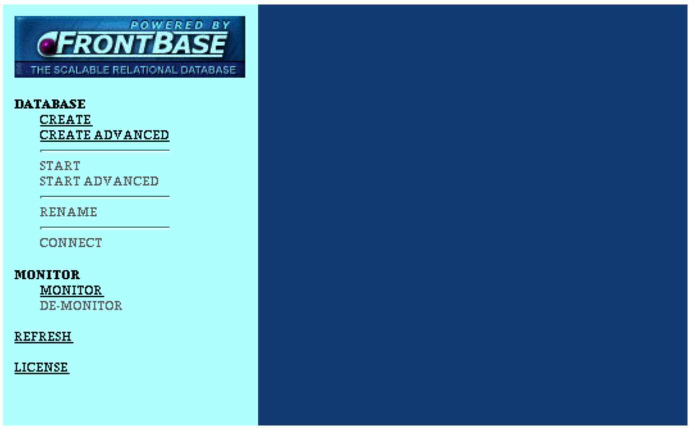


**Figure 8.1 Starting WebManager**

Monitored databases are shown as icons indicating the state of the database, together with the name of the database and the host computer on which the database is located. The example below shows the databases WhitePages and Pitsaw located on the host called macosx. See Figure 8.2.

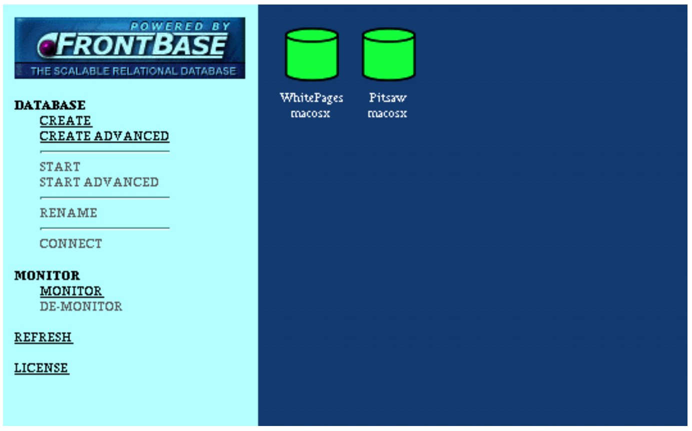


**Figure 8.2 Monitored Databases**

## Add a Database To the Monitor

A database is added to the monitor view by clicking the menu item MONITOR->MONITOR which will bring up the database selector view as seen in Figure 8.3.

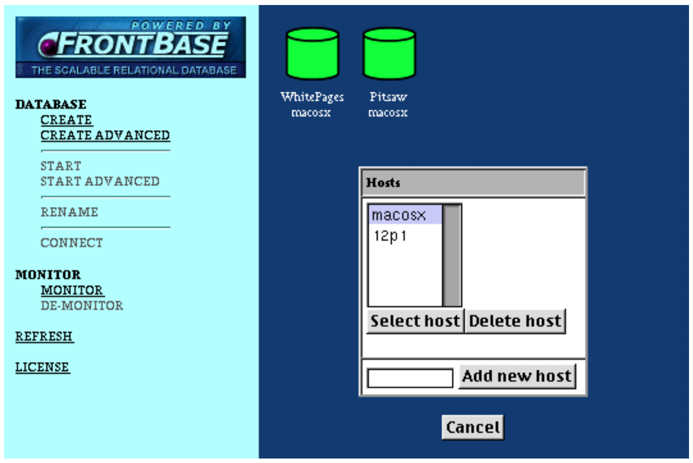


**Figure 8.3 Adding a Database**

If the wanted host is not in the list, enter the host name in the input field and click Add new host. The host is then added to the list. Now select the wanted host in the list and click Select host. This will bring up a list of databases on the specified host as seen in Figure 8.4.

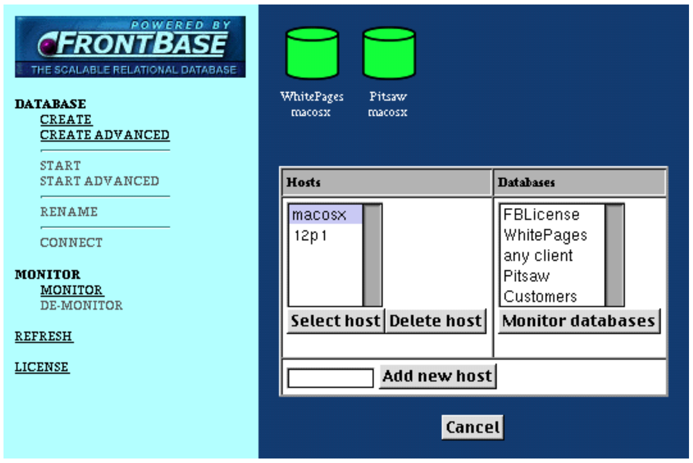


**Figure 8.4 Adding a Host**

Now select the database(s) to be monitored and click Monitor databases. This will add the database(s) to the list of monitored databases, see Figure 8.5.

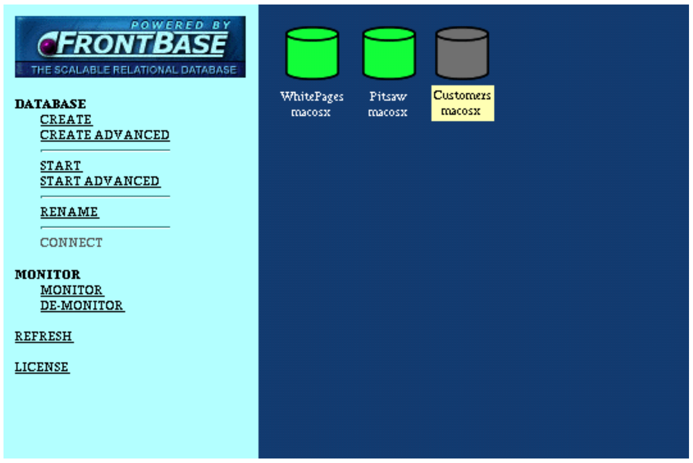


**Figure 8.5 Select a Database**

### Remove a Database

To remove a database from the monitor view, you need to select it first. The database is now presented at a white background. To de-monitor the database click on the menu item MONITOR->DE-MONITOR. This action neither stops nor deletes the database, it is only removed from the monitor view.

## Administering Databases

The web-based database manager lets you perform the necessary tasks in administrating your databases in an easy to comprehend way.

### Creating a Database

To create a new database click on the menu item DATABASE->CREATE and the following view is displayed as seen in Figure 8.6.

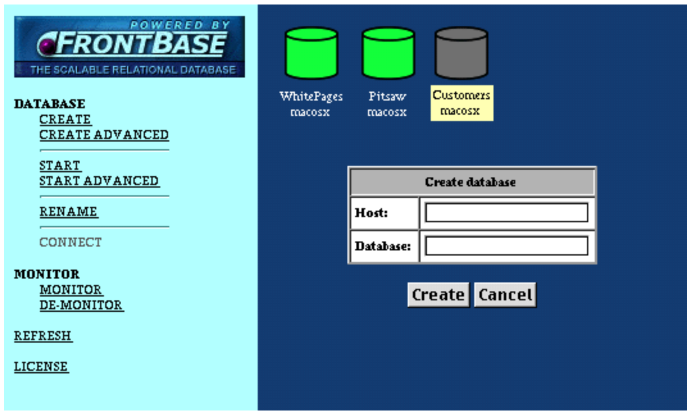


**Figure 8.6 Create a Database**

Enter the host name and the name of the new database and click on the Create button. The database is then created, started and added to the monitor view as seen in Figure 8.7.

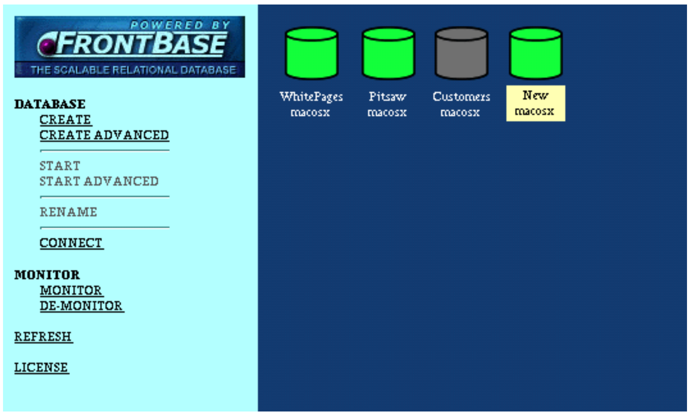
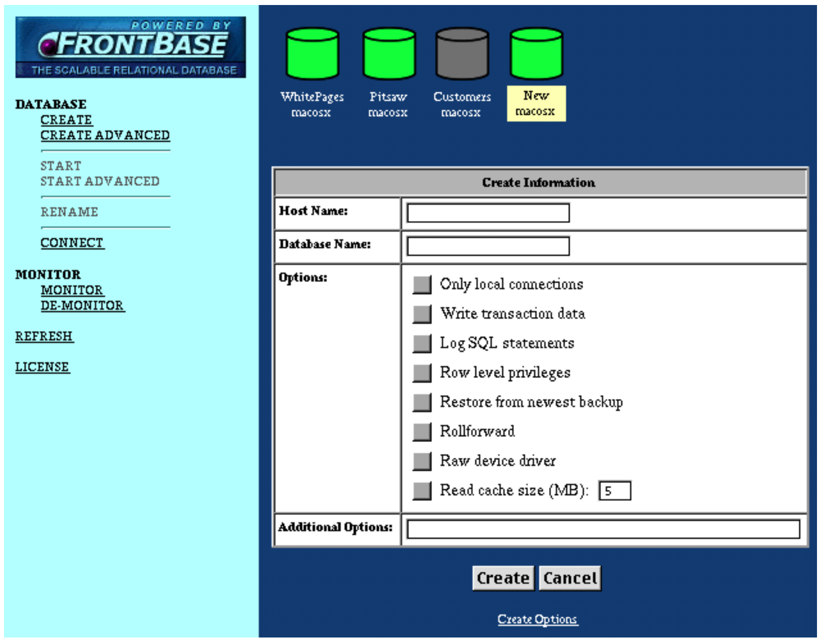


**Figure 8.7 Added to Monitor**

If you need to specify options to the database server click on the menu item DATABASE->CREATE ADVANCED. This will enable you to specify the following options as seen in Figure 8.8.



| Create Information | |
|---|---|
| Host Name: | |
| Database Name: | |
| Options: | ▢ Only local connections<br>▢ Write transaction data<br>▢ Log SQL statements<br>▢ Row level privileges<br>▢ Restore from newest backup<br>▢ Rollforward<br>▢ Raw device driver<br>▢ Read cache size (MB): 5 |
| Additional Options: | |

**Figure 8.8 Database Server Options**

1. Only local connections

   The database server will only allow local connections, i.e. connections from the same host as the one on which the database server is located.

2. Write transaction data

   The database server will log all transactions to a file.

3. Log SQL statements

   The database server will log executed SQL statements to a file. The file is located in the database directory and named

   <database name>.fb.sql.

4. Row level privileges allow the user to specify access privileges for individual rows.

Allows the user to specify access privileges for individual rows.

5. Restore from newest backup

   When the database is created the content data will be restored from the newest backup. Refer to “Transaction Logging” on page 93 for generating the backup.

6. Rollforward

   When restoring a database from a backup file, this option indicates that transactions that were not completed when the backup was done should be completed. This requires that the backup was done with the "WITH TRANSACTION DATA" option.

7. Raw device driver

   This option specifies that the host OS file system should be bypassed directly. A separate paper on this issue is under production.

8. Read cache size

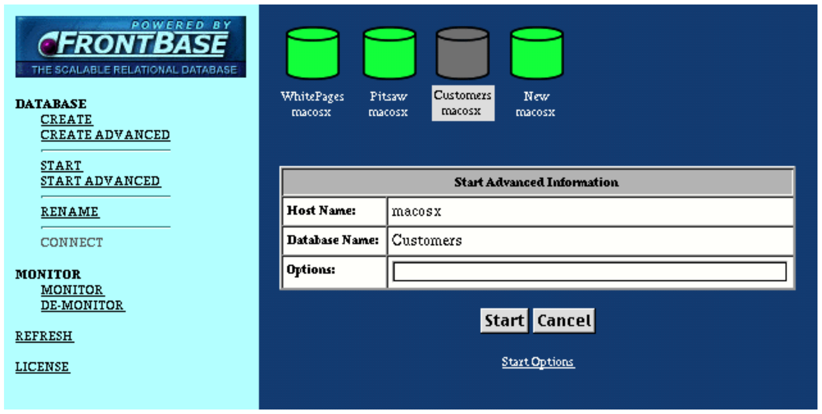
   In connection with the "Raw device driver" option this option specifies the read cache size.

9. Additional options

   Besides the above specified, a number of options can be specified when creating a database.

## Starting a Database

To start an already created, but stopped, database, select the database in the monitor view and click on the menu item DATABASE->START. If you need to specify options to the database server click on the menu item DATABASE->START ADVANCED. This will enable you to specify the following options as seen in Figure 8.9.



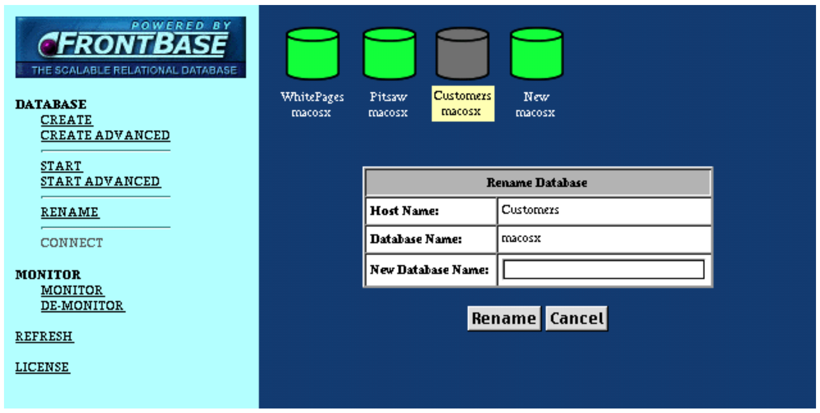
| Start Advanced Information | |
|---|---|
| Host Name: | macosx |
| Database Name: | Customers |
| Options: | |

**Figure 8.9 Start an Existing Database**

A number of options can be specified when starting a database. The options can be viewed by clicking the link 'Start Options'.

## Renaming a Database

To rename a stopped database, select the database in the monitor view and click on the menu item DATABASE->RENAME as seen in Figure 8.10.



| Rename Database | |
|---|---|
| Host Name: | Customers |
| Database Name: | macosx |
| New Database Name: | |

**Figure 8.10 Renaming a Database**

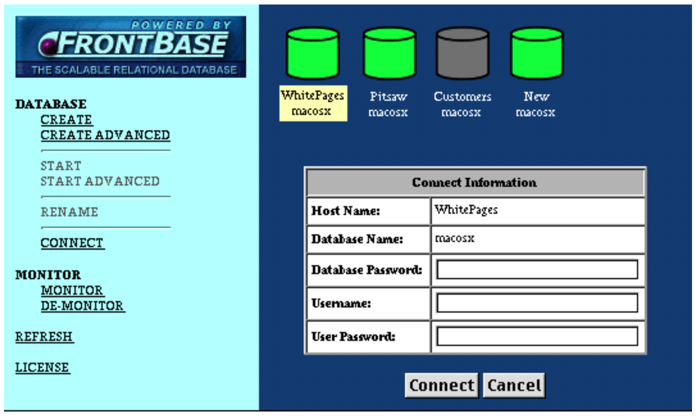
Enter the new database name and click on the Rename button.

## Refreshing the Monitor View

It may happen that a database has changed status since the monitor view was generated. To update the monitor view, click the menu item REFRESH.

## Database Management

To enter the database management module, select a running database in the monitor view and click on the menu item DATABASE->CONNECT. The monitor view will then display the connect form see Figure 8.11.



| Connect Information | |
|---|---|
| Host Name: | WhitePages |
| Database Name: | macosx |
| Database Password: | |
| Username: | |
| User Password: | |

**Figure 8.11 Entering the Database Management Module**

Enter the database password (alternatively you can create this once connected) the user name and password, if any, and click on the Connect button. If a connection can be established, the database management view is entered.

The administration tasks enabled by the management module are:

- “Stop and Delete Databases” on page 232.
- “SQL 92” on page 232.
- “User Management” on page 238.
- “Database Password” on page 242.
- “Table Cache Management” on page 243.
- “White/Black List Management” on page 244.
- “Usage Management” on page 246.
- “Keywords” on page 248.

## Stop and Delete Databases

To stop or delete a database you must be connected as user '_system'. Just click the menu items DATABASE->STOP or DATABASE->DELETE as in Figure 8.12

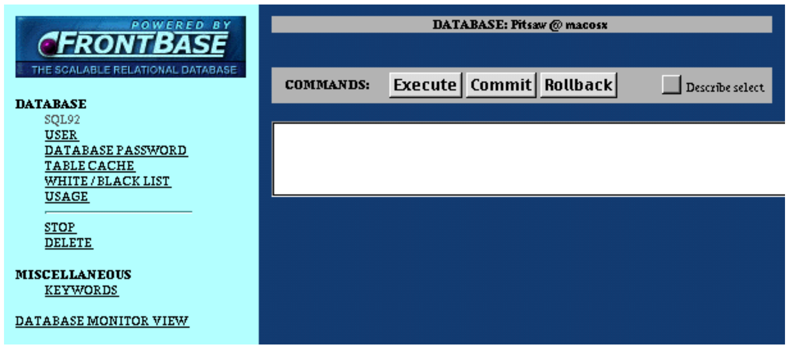


**Figure 8.12 Stop and Delete Databases**

## SQL 92

The SQL92 management view is an easy to do "command-line" SQL based interaction with the server. To enter the SQL92 manager click the menu item DATABASE->SQL92. At the first only the command entering section is visible.

Later when commands are executed also the history section and the log section will show up as seen in Figure 8.13.

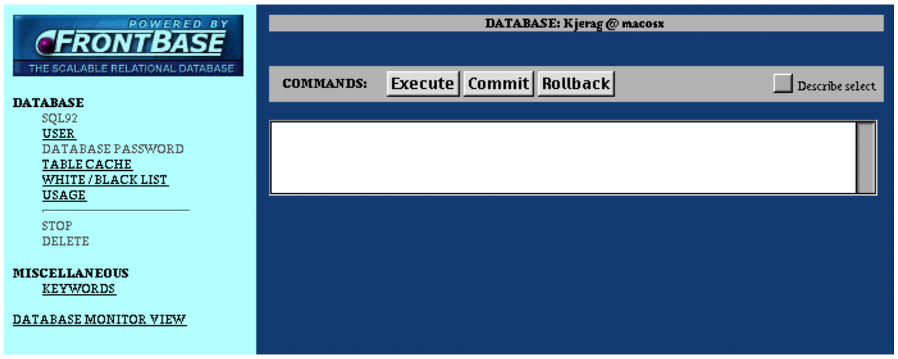


**Figure 8.13 SQL92 Management View**

### Command Entering Section

The command entering section consists of a text field to enter commands and a header with 3 buttons: one for executing the entered command and two shortcut buttons for the commands COMMIT and ROLLBACK. To the right is a check box ('Describe select'). When enabled and a select is executed, a link ('Query Plan') will be visible in the command header. When you click on this link a new window will pop up showing the query plan. Also when a select is executed a link 'Result Set') is visible in the command header. When you click on this link, a window pops up with the result of the select.

### History Section

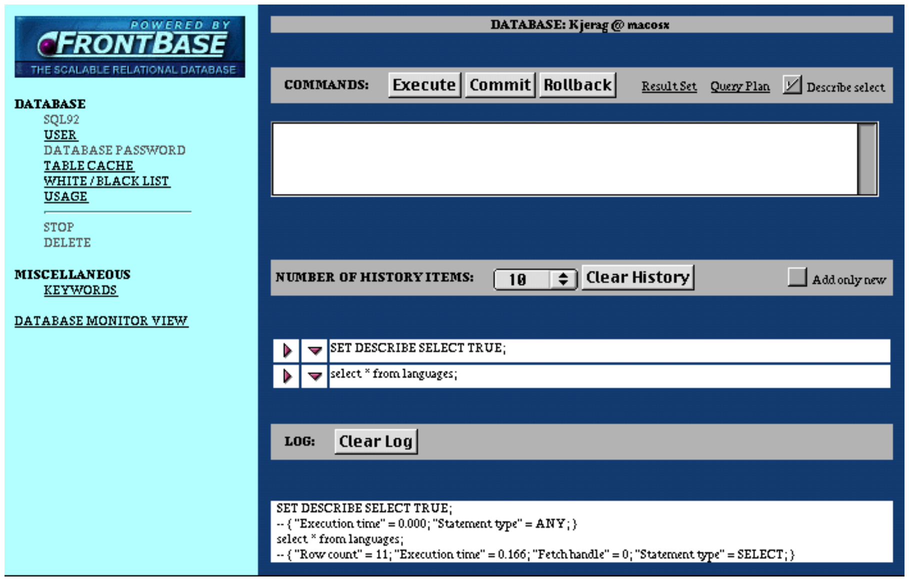
The history section consists of a header and a list of executed statements. In the history header you can change the number of displayed commands. Also you can clear the history list. To the right is a check box ('Add only new'). When this check box is enabled only commands not already in the list are added to the list.

Each entry in the history list has two small images a ‘right triangle’ and a ‘down triangle’. The ‘right triangle’ icon the statement is executed. Clicking the ‘down triangle’ icon copies the statement to the command field where it can be modified before execution.

**Figure 8.14 The Right and Down Triangle**

### Log Section

The log section consists of a header and a log view. In the log header you can clear the log. See Figure 8.15.



DATABASE: Kjerag @ macosx

[✓ Describe select]

NUMBER OF HISTORY ITEMS: 10

[ ] Add only new

```
SET DESCRIBE SELECT TRUE;
select * from languages;
```

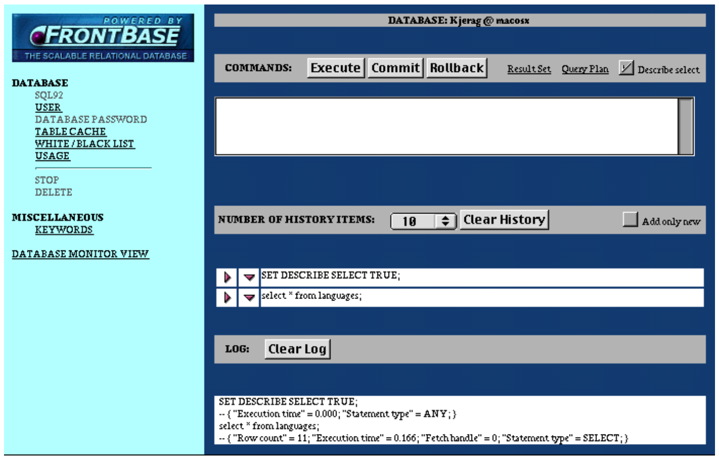
LOG:

```
SET DESCRIBE SELECT TRUE;
-- { "Execution time" = 0.000; "Statement type" = ANY; }
select * from languages;
-- { "Row count" = 11; "Execution time" = 0.166; "Fetch handle" = 0; "Statement type" = SELECT; }
```

**Figure 8.15 The Right and Down Triangle**

### Example

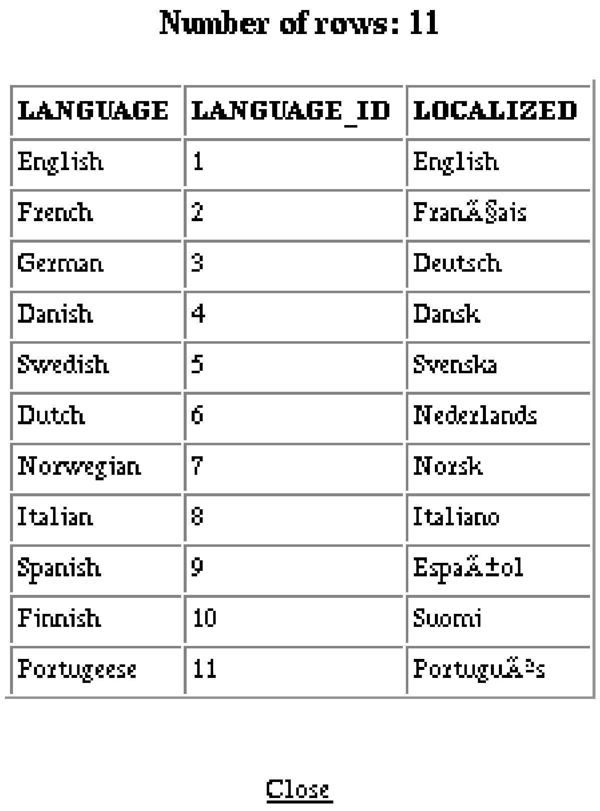
The following shows the SQL92 management view after execution of the command 'select * from languages;' with the 'Describe select' enabled. First is the SQL92 management view, then comes the result set and last comes to query plan as seen in Figure 8.16.



```
SET DESCRIBE SELECT TRUE;
select * from languages;
```

```
SET DESCRIBE SELECT TRUE;
-- { "Execution time" = 0.000; "Statement type" = ANY; }
select * from languages;
-- { "Row count" = 11; "Execution time" = 0.166; "Fetch handle" = 0; "Statement type" = SELECT; }
```

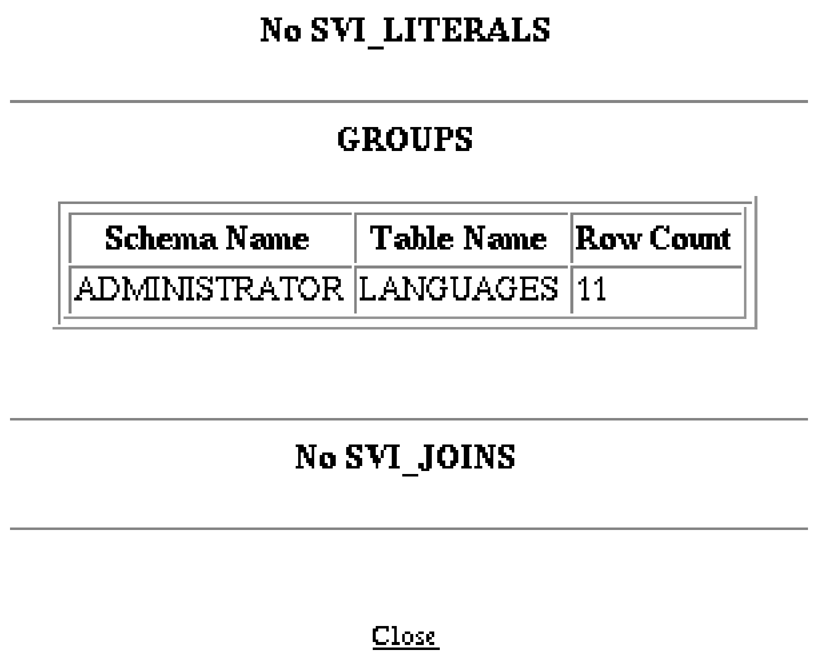
**Figure 8.16 SQL Management View**



**Number of rows: 11**

| LANGUAGE | LANGUAGE_ID | LOCALIZED |
|---|---|---|
| English | 1 | English |
| French | 2 | FranÃ§ais |
| German | 3 | Deutsch |
| Danish | 4 | Dansk |
| Swedish | 5 | Svenska |
| Dutch | 6 | Nederlands |
| Norwegian | 7 | Norsk |
| Italian | 8 | Italiano |
| Spanish | 9 | EspaÃ±ol |
| Finnish | 10 | Suomi |
| Portugeese | 11 | PortuguÃªs |

Close



**No SVI_LITERALS**

**GROUPS**

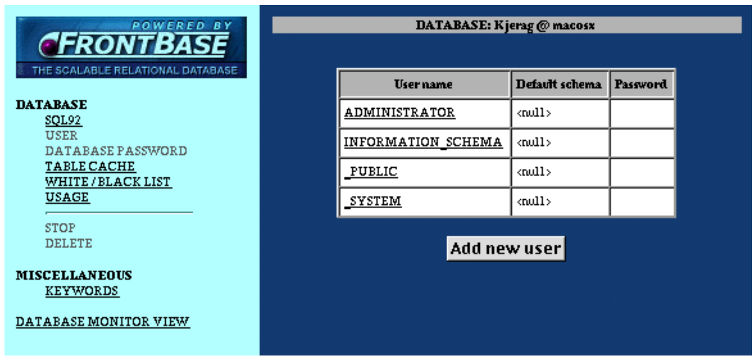
| Schema Name | Table Name | Row Count |
|---|---|---|
| ADMINISTRATOR | LANGUAGES | 11 |

**No SVI_JOINS**

Close

## User Management

To enter the user manager click the menu item DATABASE->USER. When entering the user management all the existing users are shown in Figure 8.17.



DATABASE: Kjerag @ macosx

| User name | Default schema | Password |
|---|---|---|
| ADMINISTRATOR | <null> | |
| INFORMATION_SCHEMA | <null> | |
| _PUBLIC | <null> | |
| _SYSTEM | <null> | |

**Figure 8.17 User Management View**

User management includes:

– "Adding a New User" on page 238.
– "Deleting a User" on page 239.
– "Setting a User Password" on page 240.
– "Setting a Default Schema" on page 241.

### Adding a New User

To add a new user you must be connected as user '_system'. Click on the Add new user button and the user view will look like Figure 8.18.

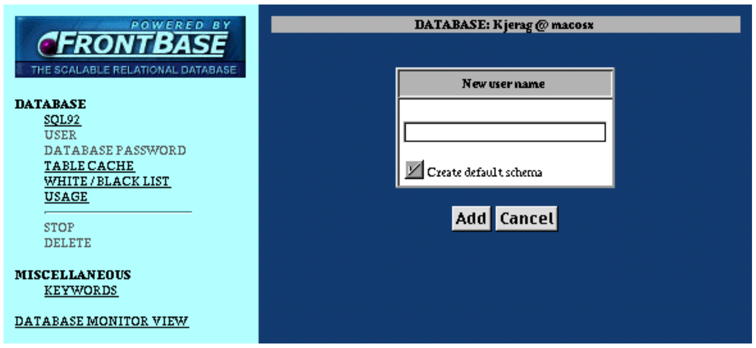
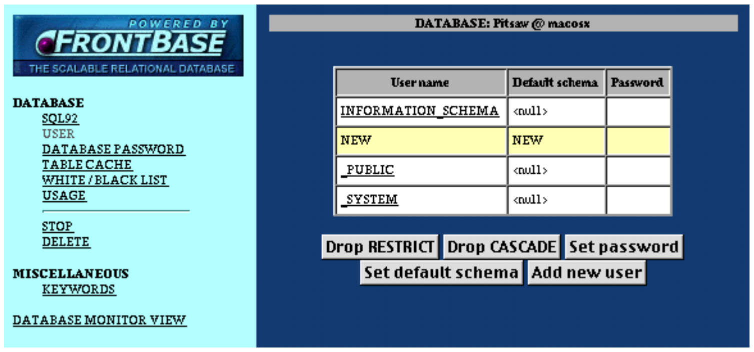


**Figure 8.18** **Add a New User**

If the "Create default schema" switch is set, a schema with the same name as the user is created. Please note that the new user becomes the current user, i.e. as if a connection was established with the new user name. The corresponding SQL statements are shown in the log window.

## Deleting a User

To delete a user, you first have to select the user in the table and then click on the Drop RESTRICT or Drop CASCADE buttons. Dropping restricted will only succeed if the user doesn't own any schema's. Dropping cascaded will unconditionally drop all schema's owned by the user. See Figure 8.19.

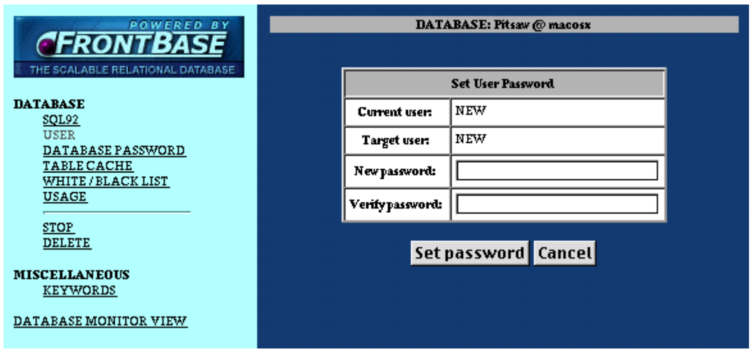


| User name | Default schema | Password |
|---|---|---|
| INFORMATION_SCHEMA | <null> | |
| NEW | NEW | |
| _PUBLIC | <null> | |
| _SYSTEM | <null> | |

**Figure 8.19 Delete a User**

### Setting a User Password

To set or change a user password, select the user in the table, click on the Set password button and the following view is shown in Figure 8.20.



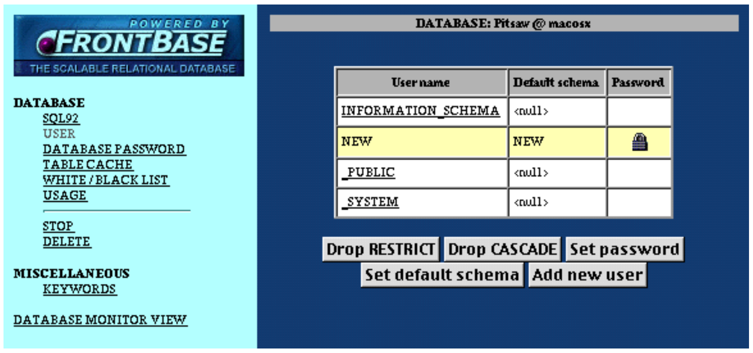
DATABASE: Pitsaw @ macosx

Set User Password

| | |
|---|---|
| Current user: | NEW |
| Target user: | NEW |
| New password: | |
| Verify password: | |

**Figure 8.20 Set or Change a User Password**

Enter the new password, verify the password, i.e. type it one more time, and click on the Set password button. The user entry in the table is updated and shows that a password has been set for the user as in Figure 8.21.



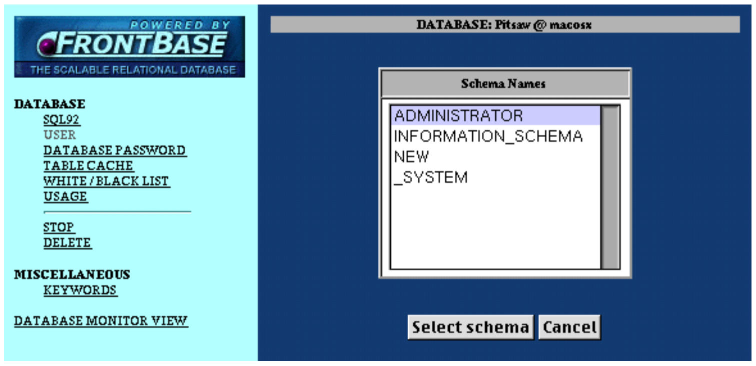
DATABASE: Pitsaw @ macosx

| User name | Default schema | Password |
|---|---|---|
| INFORMATION_SCHEMA | <null> | |
| NEW | NEW | 🔒 |
| _PUBLIC | <null> | |
| _SYSTEM | <null> | |

**Figure 8.21 Verify Password Setting**

### Setting a Default Schema

The default schema is the schema that becomes the current schema for the user whenever a connection to the database is made. To set or change the default schema, click on the Set default schema button and the possible schemas are listed in Figure 8.22.

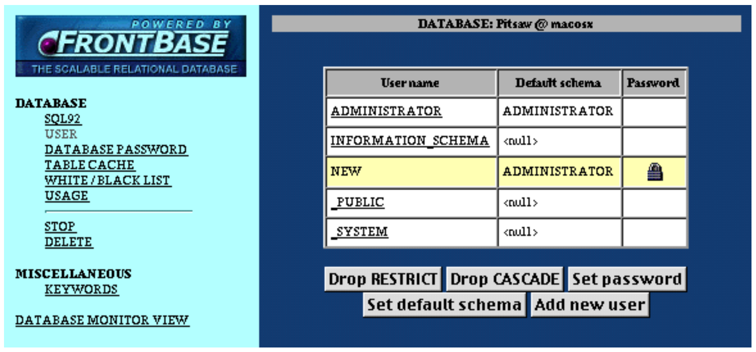


DATABASE: Pitsaw @ macosx

| Schema Names |
|---|
| ADMINISTRATOR |
| INFORMATION_SCHEMA |
| NEW |
| _SYSTEM |

**Figure 8.22 Set the Default Schema**

Select a schema and click on the Select schema button, The user entry is updated and shows the new default schema for the user as seen in Figure 8.23.



DATABASE: Pitsaw @ macosx

| User name | Default schema | Password |
|---|---|---|
| ADMINISTRATOR | ADMINISTRATOR | |
| INFORMATION_SCHEMA | <null> | |
| NEW | ADMINISTRATOR | 🔒 |
| _PUBLIC | <null> | |
| _SYSTEM | <null> | |

**Figure 8.23 The New Default Schema**

## Database Password

To set the database password, click the menu item DATABASE->DATABASE PASSWORD. You have to be connected as the user '_system' to set the database password. See Figure 8.24.

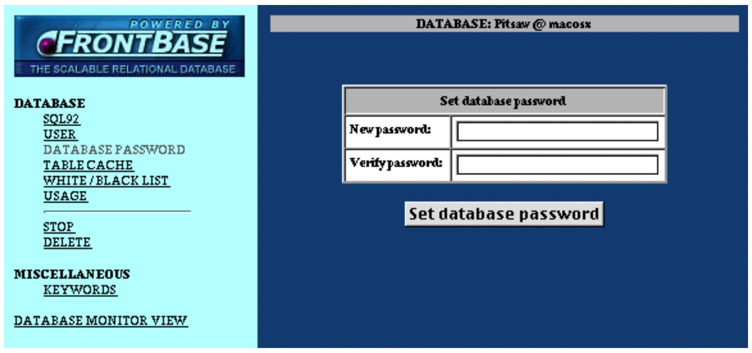
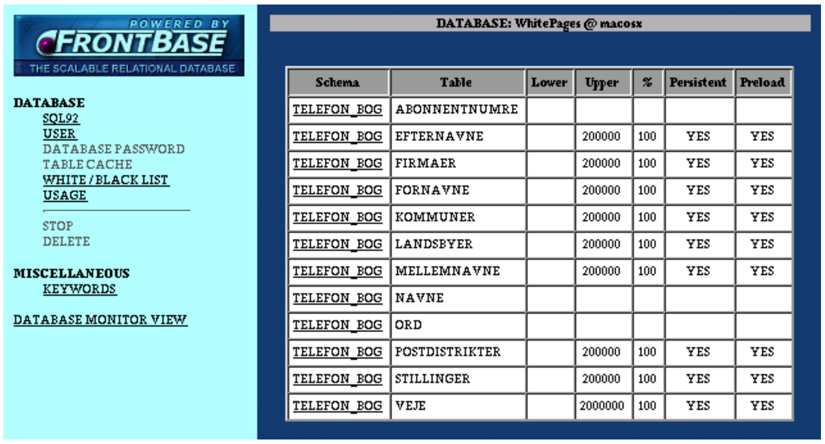


**Figure 8.24 Setting the Database Password**

Enter the new password, verify the password, i.e. type it once more, and click on the Set Database Password button.

## Table Cache Management

To enter the table cache manager, click the menu item DATABASE->TABLE CACHE as seen in Figure 8.25.



DATABASE: WhitePages @ macosx

| Schema | Table | Lower | Upper | % | Persistent | Preload |
|---|---|---|---|---|---|---|
| TELEFON_BOG | ABONNENTNUMRE | | | | | |
| TELEFON_BOG | EFTERNAVNE | | 200000 | 100 | YES | YES |
| TELEFON_BOG | FIRMAER | | 200000 | 100 | YES | YES |
| TELEFON_BOG | FORNAVNE | | 200000 | 100 | YES | YES |
| TELEFON_BOG | KOMMUNER | | 200000 | 100 | YES | YES |
| TELEFON_BOG | LANDSBYER | | 200000 | 100 | YES | YES |
| TELEFON_BOG | MELLEMNAVNE | | 200000 | 100 | YES | YES |
| TELEFON_BOG | NAVNE | | | | | |
| TELEFON_BOG | ORD | | | | | |
| TELEFON_BOG | POSTDISTRIKTER | | 200000 | 100 | YES | YES |
| TELEFON_BOG | STILLINGER | | 200000 | 100 | YES | YES |
| TELEFON_BOG | VEJE | | 2000000 | 100 | YES | YES |

**Figure 8.25 The Table Cache Manager**

All tables owned by the connected user are shown in the table and the cache parameters for each table can be set. The parameters are:

– Lower

The min. number of rows of the table to be kept in the cache.

– Upper

The max. number of rows of the table to be kept in the cache.

– %

The percentage of the total number of rows in the table to be kept in the cache.

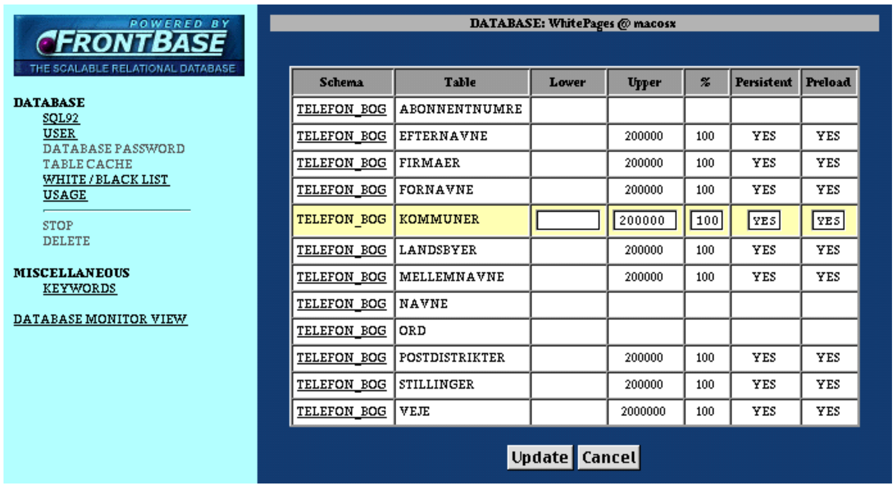
– Persistent

If set to YES, the cache is kept across transactions, otherwise the cache is flushed after each COMMIT or ROLLBACK.

– Preload

If set to YES, all rows in the table will be loaded into the cache when the server is started, otherwise the rows are loaded upon the first reference to them.

To modify the cache parameters, select a row in the table by clicking at the schema name. This will enable editing the parameters as seen in Figure 8.26.



DATABASE: WhitePages @ macosx

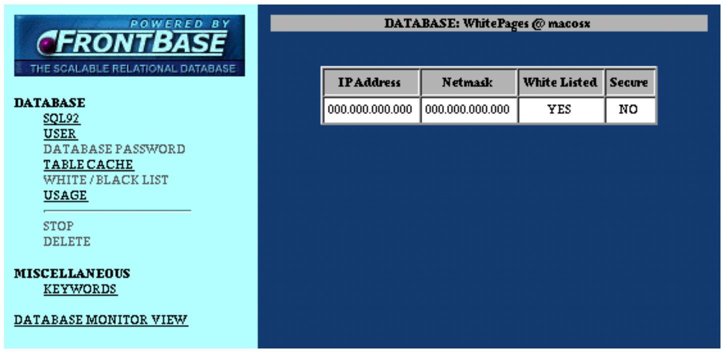
| Schema | Table | Lower | Upper | % | Persistent | Preload |
|---|---|---|---|---|---|---|
| TELEFON_BOG | ABONNENTNUMRE | | | | | |
| TELEFON_BOG | EFTERNAVNE | | 200000 | 100 | YES | YES |
| TELEFON_BOG | FIRMAER | | 200000 | 100 | YES | YES |
| TELEFON_BOG | FORNAVNE | | 200000 | 100 | YES | YES |
| TELEFON_BOG | KOMMUNER | | 200000 | 100 | YES | YES |
| TELEFON_BOG | LANDSBYER | | 200000 | 100 | YES | YES |
| TELEFON_BOG | MELLEMNAVNE | | 200000 | 100 | YES | YES |
| TELEFON_BOG | NAVNE | | | | | |
| TELEFON_BOG | ORD | | | | | |
| TELEFON_BOG | POSTDISTRIKTER | | 200000 | 100 | YES | YES |
| TELEFON_BOG | STILLINGER | | 200000 | 100 | YES | YES |
| TELEFON_BOG | VEJE | | 2000000 | 100 | YES | YES |

**Figure 8.26 Modifying the Cache Parameters**

After entering the new parameters, click the Update button.

## White/Black List Management

To enter the white and black list manager, click the menu item DATABASE->WHITE BLACK LIST as seen in Figure 8.27.



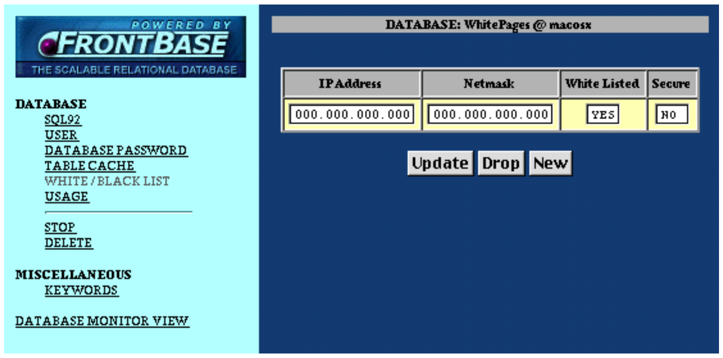
DATABASE: WhitePages @ macosx

| IP Address | Netmask | White Listed | Secure |
|---|---|---|---|
| 000.000.000.000 | 000.000.000.000 | YES | NO |

**Figure 8.27 The White/Black List Manager**

When a client connects to a FrontBase server, its IP address is checked against the white and black list to see if the client is allowed to connect. For each entry in the white and black list, the IP address of the client is AND'ed with the netmask and the result is compared with the IP address in the white and black list. If a match is found and the White values is YES, the client is allowed to connect otherwise the connection is refused. If no match is found, the connection is refused. If a YES is entered in the secure field, the client is requested to uses encryption for all further communication with the server.

To update the values you have to be connected as the user '_system'. To modify an entry in the list, select a row in the table by clicking at the IP Address. This will enable editing the values as seen in Figure 8.28.



DATABASE: WhitePages @ macosx

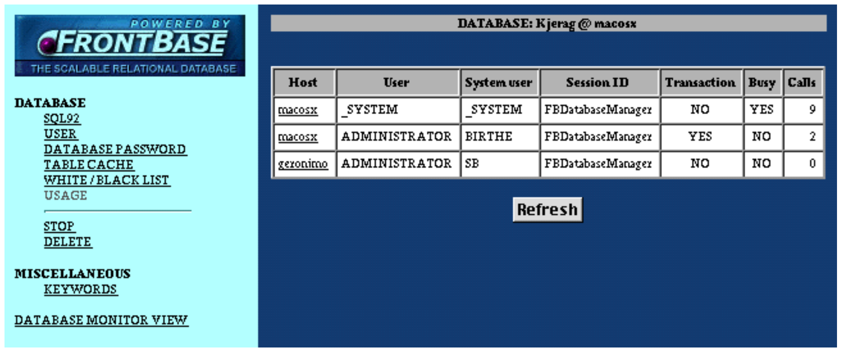
| IP Address | Netmask | White Listed | Secure |
|---|---|---|---|
| 000.000.000.000 | 000.000.000.000 | YES | NO |

**Figure 8.28 Update the Values**

The values can now be modified or the selected entry can be deleted by clicking the Drop button. A new entry is created with the New button.

## Usage Management

The usage manager allows you to monitor all connections to your database. To see the connections, click the menu item DATABASE->USAGE as seen in Figure 8.29.



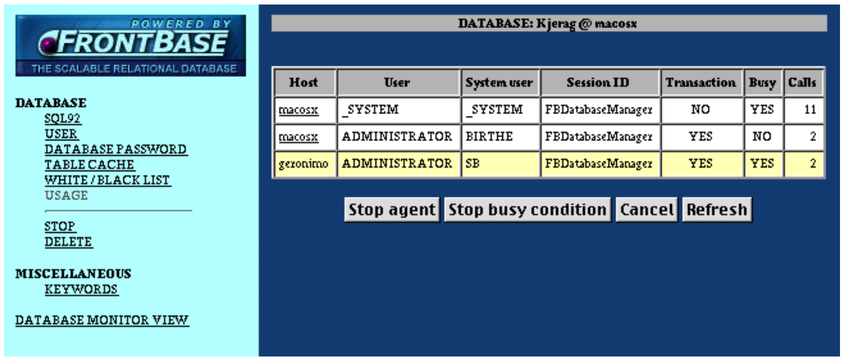
DATABASE: Kjerag @ macosx

| Host | User | System user | Session ID | Transaction | Busy | Calls |
|---|---|---|---|---|---|---|
| macosx | _SYSTEM | _SYSTEM | FBDatabaseManager | NO | YES | 9 |
| macosx | ADMINISTRATOR | BIRTHE | FBDatabaseManager | YES | NO | 2 |
| gezonimo | ADMINISTRATOR | SB | FBDatabaseManager | NO | NO | 0 |

**Figure 8.29 The Usage Manager**

The usage table displays the clients currently connected to the database and the host from where they connected. More important are the transaction and busy columns. The transaction shows if the connection has an ongoing transaction. The busy column indicates whether the client is busy.

If for some reason you need to stop a client, either because you just want to or because it is busy with e.g. a very complicated select you must be logged in as user '_system'. Select the client you want to stop by clicking on its host name as seen in Figure 8.30.



DATABASE: Kjerag @ macosx

| Host | User | System user | Session ID | Transaction | Busy | Calls |
|---|---|---|---|---|---|---|
| macosx | _SYSTEM | _SYSTEM | FBDatabaseManager | NO | YES | 11 |
| macosx | ADMINISTRATOR | BIRTHE | FBDatabaseManager | YES | NO | 2 |
| geronimo | ADMINISTRATOR | SB | FBDatabaseManager | YES | YES | 2 |

**Figure 8.30 Stopping a Client**

If the client is busy then click the Stop busy condition, otherwise click the Stop agent button.

## Keywords

The keyword manager allows you to lookup keywords used by the database server. To start the keyword manager, click the menu item MISCELLANEOUS->KEYWORDS as seen in Figure 8.31.

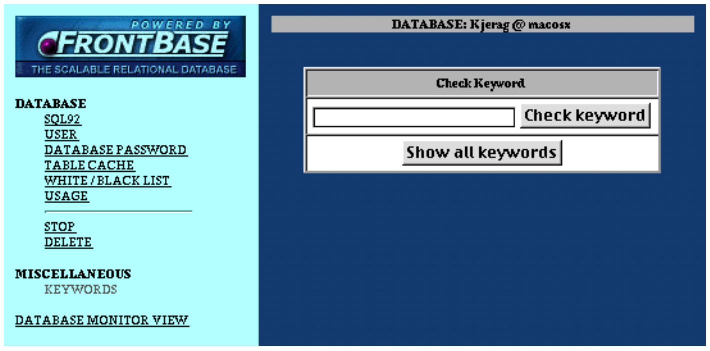
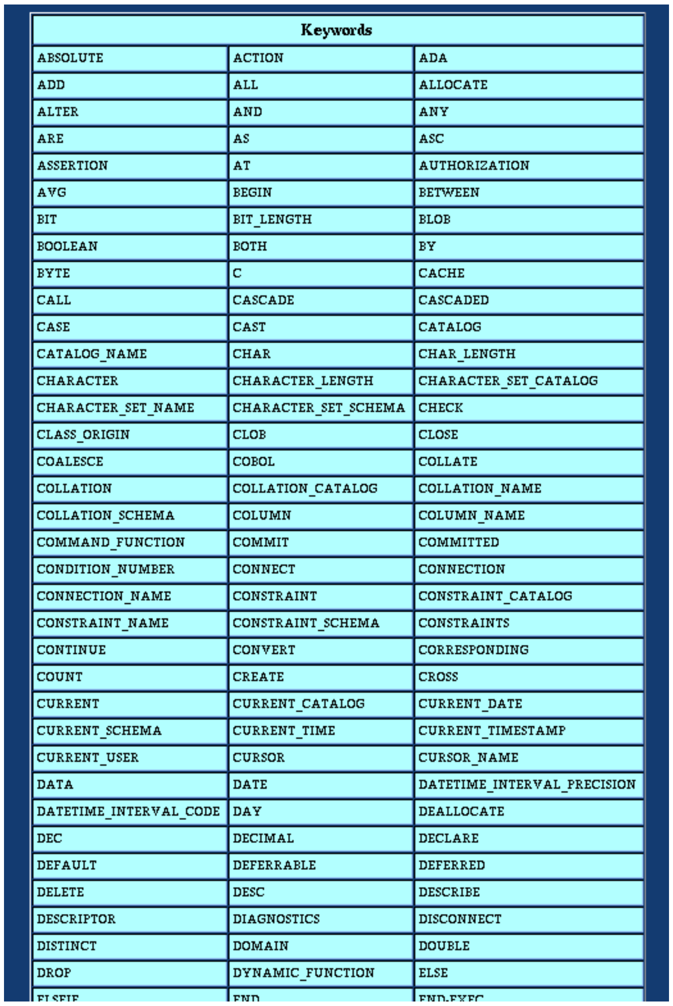


**Figure 8.31 Look up Keywords**

You can now check if a word is a keyword by entering the word in the text field and click Check keyword.

To see all keywords used by the database server click the Show all keywords button. The keywords are shown in a separate window as seen in Figure 8.32.

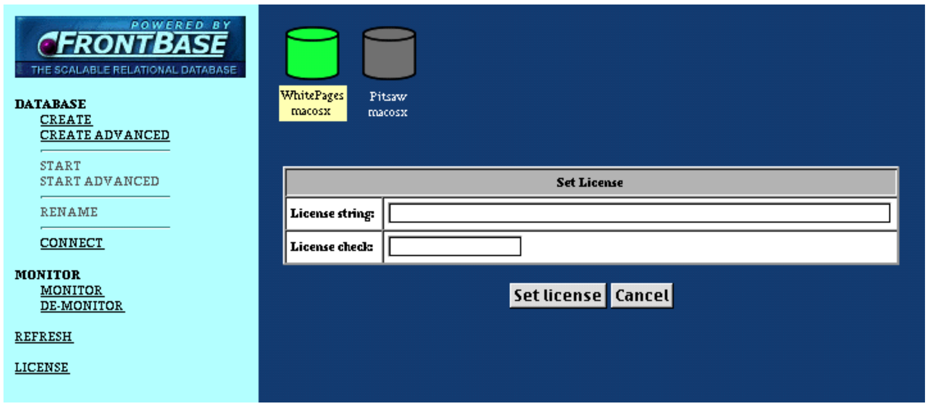


| Keywords | | |
|---|---|---|
| ABSOLUTE | ACTION | ADA |
| ADD | ALL | ALLOCATE |
| ALTER | AND | ANY |
| ARE | AS | ASC |
| ASSERTION | AT | AUTHORIZATION |
| AVG | BEGIN | BETWEEN |
| BIT | BIT_LENGTH | BLOB |
| BOOLEAN | BOTH | BY |
| BYTE | C | CACHE |
| CALL | CASCADE | CASCADED |
| CASE | CAST | CATALOG |
| CATALOG_NAME | CHAR | CHAR_LENGTH |
| CHARACTER | CHARACTER_LENGTH | CHARACTER_SET_CATALOG |
| CHARACTER_SET_NAME | CHARACTER_SET_SCHEMA | CHECK |
| CLASS_ORIGIN | CLOB | CLOSE |
| COALESCE | COBOL | COLLATE |
| COLLATION | COLLATION_CATALOG | COLLATION_NAME |
| COLLATION_SCHEMA | COLUMN | COLUMN_NAME |
| COMMAND_FUNCTION | COMMIT | COMMITTED |
| CONDITION_NUMBER | CONNECT | CONNECTION |
| CONNECTION_NAME | CONSTRAINT | CONSTRAINT_CATALOG |
| CONSTRAINT_NAME | CONSTRAINT_SCHEMA | CONSTRAINTS |
| CONTINUE | CONVERT | CORRESPONDING |
| COUNT | CREATE | CROSS |
| CURRENT | CURRENT_CATALOG | CURRENT_DATE |
| CURRENT_SCHEMA | CURRENT_TIME | CURRENT_TIMESTAMP |
| CURRENT_USER | CURSOR | CURSOR_NAME |
| DATA | DATE | DATETIME_INTERVAL_PRECISION |
| DATETIME_INTERVAL_CODE | DAY | DEALLOCATE |
| DEC | DECIMAL | DECLARE |
| DEFAULT | DEFERRABLE | DEFERRED |
| DELETE | DESC | DESCRIBE |
| DESCRIPTOR | DIAGNOSTICS | DISCONNECT |
| DISTINCT | DOMAIN | DOUBLE |
| DROP | DYNAMIC_FUNCTION | ELSE |
| ELSEIF | END | END-EXEC |

**Figure 8.32 Show All Keywords**

## Add License

To add a license to the server where FBWebManager is running, click the menu item LICENSE as seen in Figure 8.33.



| Set License | |
|---|---|
| License string: | |
| License check: | |

**Figure 8.33 Add a License**

Enter the license string and license check keys that you have received from FrontBase.com. Then click the Set license button.

9

# sql92

This chapter introduces the sql92 command-line tool, which is available for all installations of FrontBase. sql92 is a command line tool for execution of SQL 92 statements.

This chapter contains the following:

Please note that only basic commands are included in this section.

## Getting Started

The sql92 command-line tool is located in the /Library/Frontbase/bin directory. In order to use it step through the following:

– Open a terminal application/command-line.

– Navigate to the FrontBase/bin directory

– Start sql92

The following example demonstrates starting up sql92, creating a database and connecting to it on MacOS X.

```
Welcome to Darwin!
[localhost:~] root# cd /library/frontbase/bin
[localhost:/library/frontbase/bin] root# ./sql92
sql92#1> create database db0 host localhost;
sql92#2> connect to db0 user _system;
```

```
> Auto committing is on: SET COMMIT TRUE;
db0@localhost#3>
```

At this point it is possible to start entering your SQL statements.

## Command syntax

```
sql92 <options> [<filename>]
```

If a filename is specified, sql92 reads and executes the SQL 92 statements stored in the named file. If the input file is omitted, SQL 92 statements are read from standard input. All output is written to standard output.

## Options

- - a

  Do not commit transactions automatically.

- - c
  Compare. Perform a test, compare the output with the input, write a passed comment if they are equal, and a failed when not.
- - e
  Exit on error.
- - i
  Ignore. Normally sql92 will exit when an error is encountered unless the ignore option is specified.
- - l <number>
  The most number of rows that is fetched from any result set.
- - m
  Print metadata as returned from the server when it has executed sql.
- - n
  Print a note that tells operator that he must inspect the result.

- - p
  Prompt. By default sql92 will prompt for statements when the input source is a terminal device. When the prompt option is specified, sql92 will always prompt.
- - q
  Allow control characters inside quotes.
- - s
  Silent. Do not print commands as they are executed. By default, statements are printed when the input source is not a terminal device.
- - t
  Print timing information.
- - u <encoding>
  Input encoding. The <encoding> argument specifies the encoding of the input files. The default is UTF8 which will also accept us-ascii.
- - v
  Verbose. Print statements as they are executed.
- - w
  Print warnings.

## General

A sql92 command is always terminated by a ';' character. When you type input to sql92 it will not interpret the statement until a ';' character is input. This allows an SQL 92 statement to span several lines. Most SQL 92 statements are interpreted by the FrontBase SQL server, but a number of commands are interpreted by sql92. These commands typically deal with connections to the server and administration of the input files.

Comments start with a '#' character or with the '--' string and are terminated by the end of line. The '#' as a comment delimiter allows you to write interpreters. Comments are considered part of the input and are thus part of the verbose output. Lines beginning with the character '>' are also regarded as comments, but are not regarded as part of the verbose output. Result from executing SQL 92 statements are written with lines beginning with '>'. This allows you to write test scripts where the output of the script is the same as the input, simplifying regression testing.

When sql92 is invoked it attempts to execute an initialization file, with the name .slq92rc.sql. sql92 searches the current working directory and the user's home directory.

## Commands interpreted by sql92

### Connect

**Syntax:**

```
CONNECT TO <database-name>
    [DATABASE_PASSWORD <database-password>]
    [ON <host-name>]
    [AS <connection-name>]
    [USER <user-name>]
    [PASSWORD <password>];

CONNECT TO DEFAULT;
```

The first form establishes a connection to the database named <database-name> with the optional database password <database-password> on the host named <host-name>. If the <host-name>is not specified, the local host is assumed. The connection is named with <connection-name>, which is also used as prompt value. If the connection name is not specified, the connection is named with the name of the database name and the name of the host. The <user-name> specifies the authorization identifier used for the SQL 92 session established. If the <user-name> is not specified, the login name of the host operating system is used. The <password> specifies the user password. The connect command is an SQL 92 statement but is interpreted by sql92.

The second form establishes the default connection.

## Create Database

**Syntax:**

```
CREATE DATABASE <database> [(ON | @ | HOST) <hostname>]
    [PASSWORD <password>]
    [OPTIONS <option>...];
```

Create a new database with the name <database-name> on the host with the name <host-name>. If the host name is not specified the name of the local host is used. If the FBExec on the target host requires a password it can be specified, if it is not and sql92 is running interactively, the user will be propted for the password, otherwise the statement will fail. The options will be used when creating the database.

## Define Blob and Define Clob

**Syntax:**

```
DEFINE BLOB <blob-name> LENGTH <length> VALUE {<hex-bytes>};
DEFINE CLOB <clob-name> LENGTH <length> VALUE {<hex-bytes>};
```

Define a blob (or clob) object with the name <blob-name> (or <clob-name>), length <length> in bytes, and the value which is a list of <hex-bytes>. A <hex-byte> is two hexadecimal digits. The list of <hex-bytes> may contain white space (newlines, tabs, spaces, etc.). The blob (or clob) object is defined and can be referenced in subsequent SQL 92 statements. A reference has the form @'<blob-name>' (or @'<clob-name>') and will create the blob (or clob) object for that particular column. The blob (or clob) object will disappear when the next commit or rollback is executed. An incoming BLOB is stored as a file in the file system that is (almost) the bottom layer of a FrontBase database, if such a file isn't linked to a column in a row (which happens during the INSERT), the file will naturally have to be deleted when executing a COMMIT or ROLLBACK.

## Delete Database

**Syntax:**

```
 DELETE DATABASE <database> [(ON | @ | HOST) <hostname>]
        [PASSWORD <password>];
```

The database <database> on the host <hostname> is deleted. If the host name is not specified the local host is assumes. If the FBExec of th host requires a password is can be specified as <password>, if the password is not specified, sql92 is running interactively and the FBExec requires a password the user user is prompted.

## Disconnect

**Syntax:**

```
DISCONNECT <connection-name>;
DISCONNECT CURRENT;
DISCONNECT ALL;
```

The first form disconnects the connection with the name <connection-name>. The second form disconnects the current connection, and the third form disconnects all connections.

## Script

**Syntax:**

```
SCRIPT <path-to-file>/schema.sql;
```

Reads and executes the sql92 commands from the specified file (schema.sql)

## Exit and Quit

**Syntax:**

```
EXIT
QUIT
```

Exits sql92. The EXIT and QUIT commands are the only ones that do not need to be terminated with a ';'.

## Set Connection

**Syntax:**

```
SET CONNECTION <connection-name>;
SET CONNECTION DEFAULT;
```

The first form makes the connection with the name <connection-name> the current connection. The second form makes the default connection the current connection.

## Set Database Password

**Syntax:**

```
SET DATABASE_PASSWORD [<password>];
```

Sets the database password to <password>. If <password> is omitted, future connections will not require a password.

## Set Password

**Syntax:**

```
SET PASSWORD
    [USER <user-name>]
    [OLD <old-password>]
    [NEW <new-password>];
```

Sets the password for the user with the name <user-name>. If the user already has a password it must be specified and the new password <new-password> is established for the user. If <new-password> is omitted, the user may login without a password.

## Start Database

**Syntax:**

```
START  DATABASE <database> [(ON | @ | HOST) <hostname>]
    [PASSWORD <password>]
    [OPTIONS <option>...];
```

Starts the database with the name <database-name> on the host with the name <host-name>. If the host name is not specified the name of the local host is used. If the FBExec on the target host requires a password it can be specified, if it is not and sql92 is running interactively the user will be propted for the password otherwise the statement will fail.  The options will be used when starting the database.

## Stop Database

**Syntax:**

```
STOP DATABASE;
STOP DATABASE <database-name> [@|ON|HOST <host-name>];
```

The first form stops the currently connected database.  The second form stops the database on the host specified.  In order to stop a database you must be connected as user _SYSTEM. The command will attempt to create a system connection and if needed prompt for the database password and the _SYSTEM password.

## Autostart

**Syntax:**

```
AUTOSTART [ON|HOST <host-name>];
```

With the autostart commands the user can control which databases are started on a given host with the <host-name> when it is booted. If the host name is omitted the local host is assumed. The autostart command has the following sub-commands:

```
SHOW;
```

Show the list of autostarted databases. The list is numbered and the number can be used to reference to a specific database.

```
ADD  <database-name> [ <options>...];
```

Add a database to the set of autostarted databases. The server is started with the specified options.

```
OPTIONS <number> <options>...;
```

Set the options for the selected database.

```
DELETE  <number>;
```

Remove the specified database from the set of autostarted databases.

```
SAVE   [PASSWORD <password>];
```

Commit the changes. If the FBExec on the host requires a password it must be specified, if not and sql92 is running interactively the user is prompted, otherwise the command will fail.

```
EXIT;
QUIT;
```

Exit the autostart command loop.

## Show Connections

**Syntax:**

```
SHOW CONNECTIONS;
```

Shows all the connections that have been established.

## Show Usage

**Syntax:**

```
SHOW USAGE;
```

Prints a summary of the sessions for the database currently connected.

## Show Database Log

**Syntax:**

```
SHOW DATABASE LOG <datbase-name> [@|ON|HOST <host-name>];
```

Show the tail of the database log file for the database on the specified host. If the host name is not specified the localhost is assumed.

## Switch To New Database Log

**NOTE**: Available in FrontBase 3.5b and later

**Syntax:**

```
SWITCH TO NEW DATABASE LOG;
```

Will rename the existing database log file from <database name>.fb.log to <database name>.fb.log.<timestamp> and create a new <database name>.fb.log file. Once the statement has been executed, the *.<timestamp> log file can be moved or deleted.

## Show Database

**Syntax:**

```
SHOW DATABASE [FULL];
```

Show information about the currently connected database. If FULL is specified all available information is shown.

## Show License

**Syntax:**

```
SHOW LICENSE;
```

Show license details.

## Show Logs

**Syntax:**

```
SHOW LOGS [ALL|<number>];
```

Display the status for the transaction log files for the currently connected database. If all is specified all transaction logs are shown, if number is specified the latest number is printed.  The default is 1.

## Show Memory

**Syntax:**

```
SHOW MEMORY [ALL];
```

Show the status of the memory allocated by the database srever currently connected. Please refer to the memory use section in the administration chapter.

## Show Caches

**Syntax:**

```
SHOW CACHES;
SHOW CACHES <table-name>;
SHOW CACHES <schema-name>.<table-name>;
```

Shows statistics for the caches. The first form show the caches in the current schema, the second form shows the caches for the specified tables, and the last one shows the tables in the specified schemas. The schema name and the table name may contain the SQL 92 wild card characters % and _.

## Show Transaction

**Syntax:**

```
SHOW TRANSACTION;
```

Show the current transaction parameters.

## Show Schema

**Syntax:**

```
SHOW SCHEMA <schema name>;
```

Show the tables and views defined in the specified schema.

## Show Table

**Syntax:**

```
SHOW TABLE <table name> [FULL];
```

Print the definition of the table. If FULL is specified all available information is printed.

### Show View

**Syntax:**

```
SHOW VIEW <view name> [FULL];
```

Print the view definition. If FULL is specified all available information is printed.

### Show Size

**Syntax:**

```
SHOW SIZE;
SHOW SIZE <table-name>;
SHOW SIZE <schema-name>.<table-name>;
```

Show the amount of disk space used for implementing the tables specified. The first form shows all tables in the current schema, the second form shows the specified tables in the current schema, and the last form shows the the specified tables in the specified schemas. The schema name and the table name may contain the SQL92 wild card character % and _.

### Show History

**Syntax:**

```
SHOW HISTORY;
```

Show the command history.

## Start Replicator

**Syntax:**

```
START  REPLICATOR;
```

Starts a replicator. If the replicator is already running an error is reported.

## Create Client

**Syntax:**

```
   CREATE CLIENT <database-name>
    [@|ON|HOST <host-name>]
    [DATABASE PASSWORD <password>]
    [PASSWORD <system-password>];
```

Creates a new client with the name on the host . If the host name is omitted localhost is used. The create command creates and initializes the new client such that it can later can be added to client list.

## Add Client

**Syntax:**

```
   ADD CLIENT <datbase-name>
    [@|ON|HOST <host-name>]
    [DATABASE PASSWORDS <password>]
    [PASSWORD <password>];
```

Add the client with the name on the host to the replicators client list. The replicator will keep the client up to date. The passwords are the database password and the password for the _SYSTEM user which is required by the replicator to be able to connect to the client.

## Remove Client

**Syntax:**

```
REMOVE CLIENT <database-name> [@|ON|HOST <host-name>];
```

Remove the client from the list, the client is no longer updated, but may later be added to the client list again.

## Show Clients

**Syntax:**

```
SHOW CLIENTS;
```

Show the client list and the status for each client.

## Stop Replicator

**Syntax:**

```
STOP REPLICATOR;
```

Stop the replicator.

10

# FrontBase for the Developer

This chapter addresses issues that will be of interest for the developer of applications that integrate FrontBase for their data source. The chapter is divided into the following sections:

## Invocation Options Available

FrontBase sports an extensive list of so called options that can be specified when a database is started. The option can be given on either the command line or when using e.g. the FBWebManager.

- autocommit - Automatically COMMIT of transactions Auto commit all transactions after each statement
- autocreate - Automatically create new users Auto create user names as the users connect
- create - Create a new database unconditionally
- fbexec - Run with or without the FBExec Run with or without the FBExec process
- index - Set default index mode Specify the default mode for new indexes
- localonly - Accept only local connections
- logSQL - Log SQL statements to file Log all SQL statements as they are received
- port - Port number Specify a port number to use
- prvchk - Privilege checks Disable privilege checks
- rlpriv - Row Level Privileges
- restore, rollforward, wtd - Restore from a backup
- RDD - Raw Device Driver Enable the Raw Device Driver
- keys, scomm, sdisk - Encryption Specify the keys to be used for encryption
- rmaster, rclient - Replication Replication master
- rcluster - Cluster Become a member of a cluster

### autocommit - Automatically COMMIT of transactions

```
Syntax: -autocommit[=no|yes]      yes is the default
```

FrontBase 2.0 offers an auto commit feature whereby a transaction is automatically committed after the SQL statement that initiated the transaction, has been successfully executed.

The auto commit feature can also be turned off or on by executing the following SQL statement:

```
   SET COMMIT FALSE;    -- Turn the auto commit feature off
   SET COMMIT TRUE;     -- Turn the auto commit feature on
```

If the option is omitted when a FrontBase 2.0 database is started, the auto commit feature is turned off.

## autocreate - Automatically create new users

```
Syntax: -autocreate[=no|yes]      yes is the default
```

By specifyfing -autocreate=yes, all unknown users connecting to the database will automatically get created. Please note that by turning the feature on, you are violating the security of the database.

If the option is omitted when a FrontBase 2.0 database is started, unknown users will not be allowed to connect.

## create - Create a new database unconditionally

```
Syntax: -create
```

By specifying -create, a new database is unconditionally created overwriting an existing database with the same name (if it existed).

**WARNING!** Please use this option with caution.

## fbexec - Run with or without the FBExec

```
Syntax: -fbexec[=no|yes]      yes is the default
```

By specifying -fbexec=no, the starting FrontBase database will not try to establish communication with the FBExec process. This enables FrontBase to run in special environments where the FBExec for specialized reasons isn't needed. Please note that this option requires use of the -port option and requires custom client side software.

Normally a FrontBase database will communicate with the FBExec, which functions as an information gateway on a host where FrontBase is installed.

## index - Set default index mode

```
Syntax: -index[=time|space]      space is the default
```

All indexes created in a FrontBase database are created either with the PRESERVE SPACE or the PRESERVE TIME mode set. PRESERVE SPACE is a very memory efficient mode, which works well for tables with up to 1.000.000 rows. PRESERVE TIME uses more memory, but bodes for excellent performance for tables with millions of rows.

**NOTE:** The default is PRESERVE SPACE.

The mode of the indexes created for a given table can be changed by executing the following SQL statement:

```
   ALTER TABLE <table> SET INDEX PRESERVE SPACE;
```

or

```
   ALTER TABLE <table> SET INDEX PRESERVE TIME;
```

## localonly - Accept only local connections

```
Syntax: -localonly
```

By specifying this option, a Frontbase database will only accept connections from clients running on the same computer as the database.

The default is to accept connections from networked as well as local clients.

## logSQL - Log SQL statements to file

```
Syntax: -logSQL
```

By specifying this option, a Frontbase database will log all the received SQL statements in a file. The file is created in the Databases directory of a FrontBase installation and is named:

```
   Databases/<database name>.fbsql   Windows NT
   Databases/<database name>.fb.sql  All other platforms
```

The default is not to log the SQL statements.

## port - Port number

```
Syntax: -port=<port number>
```

A FrontBase database communicates with clients using BSD style sockets, requiring the clients to know the so called port number before a connection can be made. Clients will communicate with the FBExec process on port 20020 to get the port number of a given FrontBase database.

A FrontBase database will normally acquire a port number from the host operating system, but in e.g. setups with a firewall, static and known port numbers may have to used. By using the -port option, a FrontBase database can be directed to use a specific port number.

## prvchk - Privilege checks

```
Syntax: -prvchk[=no|yes]      Default is yes
```

In FrontBase 2.0 a number of privilege checks have been enforced, in particular the check for proper SELECT privileges on referenced columns. This may “break”existing apps, so by specifying the -prvchk=no option, no privilege checks will occur, i.e. existing apps should work as with FrontBase 1.2.

## rlpriv - Row Level Privileges

```
Syntax: -rlpriv
```

FrontBase offers a unique feature called Row Level Privileges, which allows you to assign privileges, like on the files in a Unix file-system, to each individual row in the tables of a databases. The -rl-priv option need only be given when a new database is created.

Refer to “SELECTing the access privileges for a row” on page 297, for more information.

## restore, rollforward, wtd - Restore from a backup

```
Syntax: -restore[=<file name>]
        -rollforward
        -wtd
```

All three options are related to the backup mechanism offered by FrontBase.

## RDD - Raw Device Driver

```
Syntax: -RDD[=<size of cache>]
```

FrontBase offers a very advanced write-through cache mechanism, that also supports use of a raw device (partition) as data storage, hence the name.

The size of the cache need only be given the first time the option is specified for a given databases or when the size needs to be decreased or increased.

## keys, scomm, sdisk - Encryption

```
Syntax: -keys=<file name>
        -scomm
        -sdisk
```

FrontBase offers a high degree of protection of your data by deploying a unique encryption scheme. Not only can the communication between clients and the FrontBase database be encrypted (streaming), but also the actual datastore on the hard disk can be encrypted (block mode).

## rmaster, rclient - Replication

```
Syntax: -rmaster
        -rclient
```

In the replication scheme offered by FrontBase, you can have one master database into which all updates must take place. Any number of read-only replicated client databases can be added.

## rcluster - Cluster

```
Syntax: -rcluster=<cluster members>
```

FrontBase offers also a cluster feature whereby N databases in a cluster can all be updated into, while they all synchronize changes with each other.

# FrontBase and Security

With FrontBase you have several options for protecting your data, as well as protecting communication against eavesdropping and tampering. The protection is obtained by applying encryption of communication and data storage, passwords for authorization, and black/white listing of computers. It is also possible to actually protect usage of the FrontBase server using a special built-in authentication feature.

## Password protection

Two layers of password protection are provided, database password and users passwords.

## Database password

If the database password is set, a client must send the database password to the server as part of the connection protocol, if the server cannot verify the password, the client connection is closed down immediately.

## User password

Each database user can have a password, the password is verified by the server when a session is created for that user, if the verification fails the session is not created. When a session is successfully created, the SQL92 defined protection takes over.

## Server Authentication

FrontBase has an authentication feature which is not enabled by default, since it would be an annoyance to most developers who are not working in secure environments.

To enable this feature as root, or a user with write permission to /Library/FrontBase, run:

- /Library/FrontBase/bin/FBExec -newpasswd=secret (you can see the usage via FBExec -h).

- Restart the running FBExec service.

At this point, creating databases, starting them, stopping them, etc., will require authentication. Individual databases can be further secured through the use of database passwords and user passwords.

### Password handling in general

Passwords may be of any length and passwords are never exposed outside the client software, and they are not even in the database. A soon as an application passes password to the FrontBase client software from the application, a oneway function is applied to generate a password digest. The function will throw away parts of the password so it is impossible to deduce the password from the digest. The user name is part of the digest, so two users with the same password will not have the same digest. The password digest is used for verification instead of the password.

## Encryption

Encryption is used to protect communication channels and data storage. When you create a FrontBase you may optionally specify that data stored on the disk should be encrypted, and optionally specify that communication channels between the server and its clients must be secure. You must provide an encryption key for each option specified.

### Encryption of data

Data stored on the disk is encrypted using a triple DES in cipher block chaining mode on 512 byte blocks, the data store is block oriented with 512 bytes / block. The initialization vector depends on the logical position of the block within the system, thus blocks with the same contents will never generate different cipher text blocks.

The key used for encryption of data is a 64 bits initialization vector, and 3x56 bits for the DES encryption.

## Secure channels

A client and the server is able to establish a secure channel. When a client connects to the server, it receives a public RSA key from the server, the client generates a set of random session keys, one for outgoing data and one for incoming data, encrypts those with the public RSA key, and sends the result to the server. The server decrypts the result from the client with use of its private key, and thus the client and the server.have established a common set of secret keys.

The algorithm used for encryption of communication data is a triple DES in byte stream mode with cipher text and clear text feed back. The clear text feedback ensures that an error will propagate to all bytes following the error, which makes detection of errors simple, introduces a small amount of redundancy and uses that for verification in the receiving end.

# Tools and Options

A few tools are provided to support key management, and FrontBase has a number of options related to security.

## FBKeyGenerator

The key generator is used to generate a set of keys that can be used by FrontBase:

```
FBKeyGenerator <RSA-key-bit-size> [<key-filename>]
```

The key-bit-size must be between 1024 and 4096 bits. If the key-filename is specified a the keys are written to that file, otherwise the keys are written to standard output. Each key set is assigned an identification

Example key text:

```
FrontBase Key Generator Tue Dec  1 12:22:45 1998
Key ident: fc0c719d
DES disk keys:
  iv(008):9f68d3d1 e25fb047
  k1(008):a2dc2f97 7a1a9b5b
  k2(008):f29ba4e0 346bea31
  k3(008):16bf92f4 311cc834
RSA private key:
    n(128):c7763cbb 9627ff8e 756173f8 2507257c
         3b4c349f 0401cc49 8473dff6 9d61752c
         020ba963 bd726d91 af249fd0 3b099e3a
         543b701c aa26be06 48f7b277 3071db45
         9607b9d5 69d2012b 6cc93b62 0536ef69
         41b815ce 265e8933 df5ab416 916ee017
```

```
          a23fe996 99212754 d1168f98 8ba78144
          d92bb918 4e5c5c2b d140d79d 602e86ed
   e(003):010001
   d(128):a61baf84 410e5a63 1719eb6c 31d9fbbb
          b3de48cc 4c97fabb 16d53124 bdf8158e
          6abdaf79 62a1d2e1 ca4be3d3 93f6f490
          7ae96cfa 5231257b 32752568 e12d507e
          eed424dd dd3e377a a41f8aeb f7158cf7
          e5388bf3 adfd2b1a 62bf2377 a1a49809
          1d9c79f8 c4d1a25f 26b464da ddbd8e7a
          cbb01ca5 847746c2 008821a0 84d93d79
   p(064):f8a87b44 77c89ba2 9a70a505 a2618d8c
          8814e4d1 e14a920d 868947b0 c10a0542
          9477cf9f 7c46e6ed 7c094b8c 85a7f866
          81b021c7 06eaa4d5 80c31473 27476b6f
   q(064):cd59e60b dfdd7d65 5825ba48 86bcb5b6
          7a469109 81bdcb91 4381aa21 83bfe3e5
          4db26077 b2001209 e916547a 084dcf2b
          299ae74d 5f0fe010 535a1376 bdce3563
dmp1(064):f45a0c98 61e59f1b 910cf4fd ae6e92fc
          f09527af 1fe779d3 14cc3ec8 f149f995
          4df4d6f0 f0dd73fd 1810c24f 1ae0cf77
          5c264b6b 1bdee590 717242dc 4e531ebd
dmq1(064):2e77bf54 c48dac48 788fe0fd 746fd376
          bd68a67d 4e3c928a 068a9ae3 069b2f88
          bdf2355e 208b5f89 9a9310fe a44e6728
          cf3b5c47 e7d101d4 efe79c2b ec7b731d
```

```
iqmp(064):8fe8b77c 53069c4d 020c2b98 c8fedf59
          5381e3a6 8d437b03 f2950da7 c7d6927b
          ef7b2ea1 e2f17add c5717232 f83bbdcd
          fa373d94 5fc066f3 4453df6d c1250d7a
```

## FBKeyGenerator

FrontBase has three security options:

- scomm

  If present the communication between the server and its clients is secure.
- sdisk

  If present data stored on the disk is encrypted
- keys=<file-name>

  The keys are read from the named file.

If one or both of the -sdisk or -scomm is specified the server requires a set of keys in order to operate. If -keys= is specified the keys are read from that file, otherwise they are read from standard input. When the server is started it checks that the correct and required keys that have been specified.

## FBChangeKey

The FBChangeKey tool can be used to encrypt a database that was not previously encrypted, to change the keys used for encryptiion, or decrypt an encrypted database.

```
FBChangeKey <database-file> [<keys-filename>]
```

If the is not provided the keys are read from standard input. If you provide one key, the database is encrypted if it was not, and decrypted if it was encrypted. If you provide two keys the database is decrypted by the old key, and encrypted with the new. Two keys is simply created by appending the text for the two keys. The tool checks the identity of the keys, so the order in which you append key texts is not significant.

## IP Address checks

When a client connects to the FrontBase server, the IP address of the client is checked against a black and white list. If the IP is blacklisted the connection is refused, if the IP is white listed the connection is accepted. The list is arranged such that it will work both as a white list and as a black list. If an IP is white listed you can specify if you require a secure communication channel to that IP. In most cases it will be ok to allow local connections to run without encryption.

A related option is the -localonly option, which makes sure that the Frontbase only allows connections from clients running on the same host as the FrontBase server.

# Datatypes

SQL 92 offers an extensive list of datatypes all of which are supported by FrontBase. Additionally, FrontBase also supports a number of datatypes from SQL3. Although the list of datatypes seem long and maybe even confusing, don't worry, many of the names denote the same datatype (such is the work that comes from a committee).

## TINYINT

Implemented as a 8-bit integer.

Example:

```
CREATE TABLE T0(C0 TINYINT, ...);
```

## SMALLINT

Implemented as a 16-bit integer.

Example:

```
CREATE TABLE T0(C0 SMALLINT, ...);
```

## INTEGER, INT

Implemented as a 32-bit integer. Apart from the obvious use, this datatype is often used for single column PRIMARY KEYs. If you are using EOF, you may want to look into using EOF's auto-generated primary keys and the BYTE type as EOF can then generate keys without a database access. The trade off is that the 12-byte primary keys thus generated are unintelligible, while a 32-bit integer is pretty simple.

Example:

```
CREATE TABLE T0(C0 INTEGER PRIMARY KEY, ...);
```

## LONGINT

Implemented as a 64-bit integer.

Example:

```
CREATE TABLE T0(C0 LONGINT, ...);
```

## DECIMAL[ ( <precision> [ , <scale> ] ) ]

Implemented as a 128-bit integer + 32 bits to hold sign and exponent. Default value for <precision> is 38 (the max.) and 0 for <scale>. This representation is identical to that of NSDecimalNumber. If you want fixed point numbers this is the datatype for it. A popular use of DECIMAL is to hold currency values.

**NOTE:** FrontBase, by using a base 10 representation, does not lose precision. If you INSERT e.g. 1.23, this is the value that gets stored and returned, not 1.229994599 or whatever. This also applies to the NUMERIC, FLOAT, REAL, and DOUBLE PRECISION (see below) datatypes.

Example:

```
CREATE TABLE T0(C0 DECIMAL, PROFITS DECIMAL(20,2), ...);
```

## NUMERIC[ ( <precision> [ , <scale> ] ) ]

Implemented as a 64-bit integer + 32 bits to hold sign and exponent. Default value for <precision> is 19 (the max.) and 0 for <scale>. NUMERIC can be used instead of DECIMAL if you don't need the 38 digit precision (and thus reduce the storage requirement).

Example:

```
CREATE TABLE T0(C0 NUMERIC, SALARY NUMERIC(10,2), ...);
```

## FLOAT[ ( <precision> ) ]

Implemented as a 64-bit integer + 32 bits to hold sign and exponent. Default value for <precision> is 19 (the max.).

Example:

```
CREATE TABLE T0(C0 FLOAT, C1 FLOAT(10), ...);
```

## REAL

Implemented as a 64-bit integer + 32 bits to hold sign and exponent. Default value for <precision> is 19 (the max.). REAL and FLOAT are implemented identically, except that you can specify the max. precision when using FLOAT.

Example:

```
CREATE TABLE T0(C0 REAL, ...);
```

## DOUBLE PRECISION

Implemented as a 128-bit integer + 32 bits to hold sign and exponent. Default value for <precision> is 38 (the max.). For many purposes this is the best choice for mapping an NSDecimalNumber / java.math.BigDecimal. See the “Mapping of Foundation / Java objects into FrontBase” document for details.

Example:

```
CREATE TABLE T0(C0 DOUBLE PRECISION, ...);
```

## CHARACTER, CHAR

Implemented as the traditional fixed length character string. Please note that FrontBase supports Unicode exclusively and stores all character strings in the UTF8 encoding. This means that character strings with values other than ASCII will occupy more bytes than the number of characters. Most non-ASCII characters, e.g. æøåÆØÅ, when encoded into the UTF8 format, occupies two bytes.

**NOTE:** The max. length of a CHARACTER value is 2GB.

Example:

```
CREATE TABLE T0(C0 CHAR(1), C1 CHARACTER(100000), ...);
```

## NATIONAL CHARACTER, NATIONAL CHAR, NCHAR

As FrontBase supports Unicode exclusively, the NATIONAL CHARACTER datatype is mapped into CHARACTER.

Example:

```
CREATE TABLE T0(C0 NATIONAL CHAR(1), C1 NCHAR(100000), ...);
```

## CHARACTER VARYING, CHAR VARYING, VARCHAR

Implemented as the traditional variable length character string. The implementation of variable length strings is very efficient, and there is no extra overhead associated with very long strings. Strings up to 16 bytes in length are stored directly in the row record (as if it was a fixed length string). A so called spelling table is associated with each table and all identical variable length strings inserted in the rows of a table are only stored once.

**NOTE:** Since FrontBase encodes varchars very efficiently, use of variable length strings is in general recommend over fixed length strings.

Example:

```
   CREATE TABLE T0(C0 VARCHAR(128), C1 CHARACTER VARYING(200000), ...);
```

## NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING, NCHAR VARYING

As FrontBase supports Unicode exclusively, the NATIONAL CHARACTER VARYING datatypes are all mapped into CHARACTER VARYING.

Example:

```
   CREATE TABLE T0(C0 NATIONAL CHAR VARYING(10), C1 NCHAR VARYING(10000),
...);
```

## BIT

The bit datatype is conceptually a string of 1's and 0's, but is implemented as an opaque binary datatype, i.e. BIT(8) occupies one byte. See below as concerns EOF and BYTE.

Example:

```
   CREATE TABLE T0(C0 BIT(32), C1 BIT(256)...);
```

## BIT VARYING

As BIT, but with the obvious exception that the bit strings are variable in length.

Example:

```
CREATE TABLE T0(C0 BIT VARYING(32), C1 BIT VARYING(256)...);
```

## BYTE

A simple wrapper for BIT, i.e. BYTE(n) is identical to BIT(n*8). This datatype is not part of the SQL92 standard, but has been introduced to better support EOF's automatic primary key generation. If you use 12-byte binary key's, EOF can automatically generate a primary key without doing a roundtrip to the database server (and thus cause a transaction to be initiated).

Example:

```
CREATE TABLE T0(C0 BYTE(12), ...);
```

## DATE

The traditional date datatype. Please note that DATE does not include any time components. DATE values are internally represented as seconds (2001-01-01 is zero) and are stored as NUMERIC(0) values.

Example:

```
CREATE TABLE T0(C0 DATE, ...);
```

## TIME

Holds only the time component of a complete timestamp. TIME values ('12:34:23') are internally represented as seconds and are stored as NUMERIC values. Please note that TIME values, which can be negative, are assumed to be expressed in the servers time zone, i.e. the servers time zone is applied to the time value when it is inserted.

Example:

```
CREATE TABLE T0(C0 TIME, ...);
```

## TIME WITH TIME ZONE

As TIME, except that the time zone offset is included and stored with the time values ('12:34:23-08:00'). The explicit time zone is returned to clients.

Example:

```
CREATE TABLE T0(C0 TIME WITH TIME ZONE, ...);
```

## TIMESTAMP

Holds a complete timestamp value which includes both the date and time components. TIMESTAMP values ('2001-01-24 12:34:23') are internally represented as seconds (2001-01-01 is zero) and are stored as NUMERIC values. Please note that TIMESTAMP values will be expressed in the servers time zone, i.e. the servers time zone is applied to the time value when it is inserted. This means that TIMESTAMP values can end up having a time zone that is different from the client!

Example:

```
CREATE TABLE T0(C0 TIMESTAMP, ...);
```

## TIMESTAMP WITH TIME ZONE

As TIMESTAMP except that the time zone offset is included and stored with the time values ('2001-01-24 12:34:23-08:00'). The explicit time zone is returned to clients. This datatype is needed if you want to be in complete control over how time zone information is stored and displayed.

Example:

```
CREATE TABLE T0(C0 TIMESTAMP WITH TIME ZONE, ...);
```

## INTERVAL

INTERVAL is actually two separate datatypes: a datatype called year-month interval and a datatype called day-time interval.

A year-month interval is internally represented as months and is stored as a 32-bit integer.

A day-time interval is internally represented as seconds and is stored as a NUMERIC value.

One way to use intervals is when manipulating dates and timestamps, e.g. when adding a day or month:

```
DATE '2000-01-25' + INTERVAL '02' MONTH (result: DATE '2000-03-25' )
```

or

```
DATE '2000-02-28' + INTERVAL '02' DAY (result DATE '2000-03-01' )
```

Example:

```
CREATE TABLE T0(C0 INTERVAL YEAR TO MONTH, C1 INTERVAL MONTH, ...);
CREATE TABLE T1(D0 INTERVAL DAY TO SECOND, C1 INTERVAL HOUR, ...);
```

## BLOB

A Binary Large OBject is an opaque binary datatype, i.e. the bytes you store are not interpreted in any way and are returned in the same form as when inserted. FrontBase implements BLOBs very efficiently which includes streaming on the server side, i.e. no unnecessary copying. Client side streaming is planned enabled in FrontBase 2.0. A BLOB value can be up to 2 GB in size.

Example:

```
CREATE TABLE T0(C0 BLOB, ...);
```

### CLOB

A Character Large OBject is a datatype for very large character strings, i.e. strings that you don't want to search on and where you would like the increased efficiency compared to normal CHARACTER/VARCHAR values (which gets copied into e.g. INSERT or UPDATE SQL statements). CLOBs are implemented as efficiently as BLOBs. CLOB values are encoded in the UTF8 format with encoding and decoding taking place on the client side.

Example:

```
   CREATE TABLE T0(C0 CLOB, ...);
```

### BOOLEAN

Implemented as an unsigned byte. Please note that SQL 92 uses three-valued logic, i.e. the possible values are FALSE (0), TRUE (1), and UNKNOWN (255).

Example:

```
   CREATE TABLE T0(C0 BOOLEAN, ...);
```

## Mapping of Foundation/Java objects into FrontBase datatypes

The following is a brief description of the recommend mapping of the most common Foundation/Java objects into FrontBase datatypes.

## String

| | | |
|---|---|---|
| NSString<br>java.lang.String<br>java.io.Reader<br>(for CLOB) | Suggested datatypes: | CHARACTER, VARCHAR, CLOB |
| | Recommended: | VARCHAR |

**NOTE:** If you are working with large strings and if you most likely are not going to perform searches on the character strings, the CLOB datatype can be a good choice. The main advantage of the CLOB datatype is that the string is sent to the database separately. That is, rather then the database having to parse and copy a huge SQL statement, it can instead efficiently transfer the data on a binary channel. VARCHAR is in general recommended over CHARACTER, as the implementation of VARCHAR is very efficient and allows reuse of identical strings. You can use “open ended” VARCHAR definitions (e.g. VARCHAR(1000000)) without worry for any performance penalty. The internal representation of character strings is UTF8 with the encoding/decoding taking place on the client side, i.e. handled by the native methods of NSString/ java.lang.String.

## Integer Numbers

| | | |
|---|---|---|
| NSNumber java.lang.Number | Suggested datatypes: | BOOLEAN, SMALL-INT, INTEGER, FLOAT, REAL, DOUBLE PRECISION |
| | Recommended: | INTEGER or DOUBLE PRECISION depending on actual use. |

**NOTE:** As NSNumber/Number is a cover for the C/Java number datatypes, the FrontBase type to choose depends on the actual use.

## Decimal Numbers

| | | |
|---|---|---|
| NSDecimalNumber java.math.BigDecimal | Suggested datatypes: | NUMERIC, DECIMAL, REAL, FLOAT, DOUBLE PRECISION |
| | Recommended: | DOUBLE PRECISION, DECIMAL |

**NOTE:** If you are using NSDecimalNumber for accurate calculations, that maps to DOUBLE PRECISION (or DECIMAL if integer) directly. If you wish to reduce storage requirements, you can use FLOAT/NUMERIC instead (the reduction is 8 bytes per value per row).

## Dates

| | | |
|---|---|---|
| NSCalendarDate<br>java.sql.Date<br>java.sql.Timestamp) | Suggested datatypes: | DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE |
| | Recommended: | TIMESTAMP |

**NOTE:** TIMESTAMP WITH TIME ZONE is directly equivalent to the data provided by NSCalendarDate. You might want to consider just TIMESTAMP if you don't have to deal with data in different time zones or if you want data always displayed using the clients time zone.

## Time

| | | |
|---|---|---|
| java.sql.Time | Suggested datatypes: | TIME, TIME WITH TIME ZONE |
| | Recommended: | Either. |

**NOTE:** java.sql.Time is a simple wrapper (or just a plain gruesome hack) around java.sql.Date.

## Stream Data

| | | |
|---|---|---|
| NSData<br>java.io.InputStream | Suggested datatypes: | BLOB |
| | Recommended: | BLOB |

### Primary Key

| | |
|---|---|
| Suggested datatypes: | INTEGER, BYTE(12) |
| Recommended: | Either. |

**NOTE:** The BYTE(12), which is a cover for BIT(96), datatype should be used if the generation of primary key is done by EOF (client side calculation facility). Any FrontBase datatype can be used for a primary key and since all numeric datatypes are represented as exact numeric values, it is safe to use e.g. DOUBLE PRECISION or TIMESTAMP as a datatype for a primary key column. However, EOF does not allow certain datatypes (DOUBLE PRECISION, BLOB and CLOB) to be specified for primary key columns, so you will need to consider this if you are using EOF. FrontBase supports multi-column or compound primary keys, while you should apply some caution if you are using compound keys with EOF.

## Primary Keys and Auto Generation

FrontBase supports, as required by the SQL92 standard, multiple-column primary keys, but in the case of single-column integer primary keys, FrontBase can help you in generating such keys.

### Generation of Keys

In FrontBase, each table has an associated counter. The counter is accessed and incremented by the following SQL statement:

```
SELECT UNIQUE FROM <table>;
```

which returns a single row with a single integer column. The SELECT UNIQUE construct can also be used as a scalar subquery:

```
INSERT INTO <table> VALUES(SELECT UNIQUE FROM <table>, ...);
```

When a table is created, the associated counter is set to an initial value of 1000000.

The counter can be set in two ways:

```
     SET UNIQUE=<value> FOR <table>;
```

or

```
     SET UNIQUE FOR <table>(<column>);
```

with the latter form being a short-hand for:

```
SET UNIQUE=(SELECT MAX(<column>)+1 FROM <table>) FOR <table>;
```

A primary key column in a new row can be automatically set by specifying a special default value:

```
ALTER TABLE <table> ALTER <column> SET DEFAULT UNIQUE;
```

If an explicit value for the column isn't given in a INSERT statement, the default will be used.

## Row Level Privileges

### Defining

FrontBase offers a unique feature called Row Level Privileges, which allows you to specify access privileges for individual rows. Each is row is said to be owned by a specific user and belonging to a specific group. Access privileges (SELECT, UPDATE and DELETE) for a row can be specified for the owner, the group and the world.

**NOTE:** Row Level Privileges is licensed as a separate option and that the feature is not available in the free version.

## Deploying

To use the Row Level Privileges feature, a given database has to be initialized with the feature given as an option:

```
/Local/Library/FrontBase/bin/FrontBase -rlpriv <database name>
```

You can also specify the -rlpriv option when creating a database via the FBDatabaseManager.

Once created, the option is recorded in the database, i.e. you don't need to specify the option when the database server is subsequently stopped and started.

### Managing the meta data

```
      CREATE GROUP <group name>;
            -- CURRENT_USER must be _SYSTEM
      DROP GROUP <group name> RESTRICT|CASCADE;
            -- CURRENT_USER must be _SYSTEM
```

```
      ALTER GROUP <group name> ADD USER <user name>;
         -- CURRENT_USER must be _SYSTEM
      ALTER GROUP <group name> DROP USER <user name>;
         -- CURRENT_USER must be _SYSTEM
```

```
      ALTER USER <user name> SET DEFAULT GROUP <group name>;
         -- CURRENT_USER must be _SYSTEM or <user name>
      ALTER TABLE <table name> SET DEFAULT PRIVILEGES(<row
privileges>)
         [USER <user name>];
         -- CURRENT_USER must be _SYSTEM or <user name>, if no
user name
         -- is given, the current user is used
      <row privileges> ::= <row privs> | <row privileges> , <row privs>
      <row privs>      ::= <owner privs> | <group privs> | <world privs>
      <user privs>     ::= USER = * | <priv mask>
      <group privs>    ::= GROUP = * | <priv mask>
      <world privs>    ::= * = * | <priv mask>
      <priv mask>       ::= <priv> | <priv mask> + <priv>
      <priv>            ::= SELECT | UPDATE | DELETE
```

Example:

```
ALTER TABLE T0 SET DEFAULT PRIVILEGES(USER=*, GROUP=SELECT+UPDATE, *=SELECT);
```

### Managing the content data

```
UPDATE <table name> SET PRIVILEGES(<row privileges>) [WHERE <cond expr>];
UPDATE <table name> SET GROUP <group name> [WHERE <cond expr>];
UPDATE <table name> SET USER <user name> [WHERE <cond expr>];
         -- CURRENT_USER has to either own the row or be _SYSTEM
```

### SELECTing the access privileges for a row

The owner, group and privileges for a given set of rows can be fetched as follows:

```
SELECT USER, GROUP, PRIVILEGES FROM <table> WHERE <cond expr>;
```

By wrapping the SELECT in a VIEW, the values can be used in queries:

```
CREATE VIEW(ROW_OWNER, ROW_GROUP, ROW_PRIVS) T0_PRIVS
SELECT USER, GROUP, PRIVILEGES FROM T0;
SELECT * FROM T0_PRIVS WHERE ROW_OWNER = '<user name>';
```

# What Collations can do for you

Collations are basically a way for you to control how two characters should be compared or rather whether two given characters compare equal, less than or greater than.

Why bother with this?

There are two main reasons for having to bother with collations:

1. International characters
2. Case insensitive compare operations

## International Characters

FrontBase implements Unicode and support thus use of all the so-called international characters including Kanji, Hangul etc. The positional values of the international characters in the Unicode universe can not be used for ordering two characters, at least if the ordering is to turn out as most people expect it.

An example: The French character ç (Latin Small Letter C With Cedilla) has the ordinal value of 231 (decimal) while a lower case C has 99 as ordinal value. If e.g. ç and d are compared, d would then compare to be smaller than ç, which may not be what you want.

## Case Insensitive Compare Operations

Normally character strings are stored in the database using the same case as they were entered by a user. Some users prefer to enter just lower case characters, other prefer upper case characters, and a few uses capitalization as in FrontBase. When searching, users generally don't know in which case characters were entered, i.e. a search has to take care of this.

This can be dealt with by doing something like:

```
SELECT * FROM T0 WHERE UPPER(CITY) = 'COPENHAGEN';
```

The problem with above SELECT is that an index defined on T0.CITY cannot be used, i.e. the SELECT will execute slower than if an index could be used.

By defining a so called COLLATION, you can effectively decide how characters are to be ordered, i.e. mapping the ordinal values into ordering values. This means for example that if 'a' is mapped into the same ordering value as 'A', 'a' is considered to be equal to 'A'.

Included with any FrontBase distribution is a collation table called CaseInsensitive.coll1 (located in the <FB home>/Collations directory) and as implied by its name, this collation can be used to do case insensitive compares with.

First you need to define the collation:

```
CREATE COLLATION CASE_INSENSITIVE
   FOR INFORMATION_SCHEMA.SQL_TEXT
   FROM EXTERNAL('CaseInsensitive.coll1');
COMMIT;
```

The collation is then used when creating a table:

```
CREATE TABLE T0(
   ...
   DB VARCHAR(128) COLLATE CASE_INSENSITIVE,
   ...
);
CREATE INDEX ON T0(DB);
COMMIT
```

The specified collation will now automatically be used whenever a DB column value is compared with another string, this includes compares done when building an index.

An example:

```
INSERT INTO T0(DB) VALUES 'frontbase', 'FrontBase', 'FRONTBASE';
COMMIT;
SELECT DB FROM T0 WHERE DB = 'FrOnTbAsE';
```

-- Will return all 3 rows

```
SELECT DB FROM T0 WHERE DB LIKE 'f%';
```

-- Will return all 3 rows

If for some reason you want to compare case sensitive, you need to define an identity collation (using the FBUnicodeManager application) and save the collation as e.g. CaseSensitive.coll1 in the <FB home> / Collations directory.

```
CREATE COLLATION CASE_SENSITIVE
   FOR INFORMATION_SCHEMA.SQL_TEXT
   FROM EXTERNAL('CaseSensitive.coll1');
COMMIT;
SELECT DB FROM T0 WHERE DB = 'FrontBase' COLLATE CASE_SENSITIVE;
```

-- Will return 1 row

```
SELECT DB FROM T0 WHERE DB LIKE 'F%' COLLATE CASE_SENSITIVE;
```

-- Will return 2 rows

Please note that the above two SELECTs will not use the index created on column DB, i.e. for large tables these two SELECTs will execute slower than if the index could be used.

Now what if you want to search case insensitive and then limit the result set further by requiring that an exact case match should also apply? Easy done:

```
SELECT DB FROM T0 WHERE
   DB = 'FrontBase'
      AND
   DB = 'FrontBase' COLLATE CASE_SENSITIVE;
```

The first WHERE component will return 'frontbase', 'FrontBase', 'FRONTBASE', while the second WHERE component will reduce the result to 'FrontBase';

# Embedding FrontBase into your own application or solution

When an end-user downloads and installs FrontBase for a given platform, the installation will typically go into a default location specific for the platform. FrontBase is, after installation, then accessible to all applications etc. for which access is granted.

When FrontBase is embedded into another application or solution, it is normally desirable to make sure that FrontBase will operate only with the given application and independent from a normal end-user version of FrontBase that may already be installed.

Technically this means that:

1) FrontBase is  installed ("embedded") inside the normal directory structure of the parent application or solution, i.e. the normal FrontBase installer package isn't used.

2) An application or solution-specific license string is to be used. This license string is generic, i.e. it isn't tied to a specific IP or MAC address.

3) The FBExec, which is like a DNS service for FrontBase databases on a given host, isn't used, meaning that the parent application or solution will have to connect to the database using a port number. The port number will be embedded (hard coded) into the license string.

4) An embedded license string allows for the application or solution to work with one (1) FrontBase database.

## Directory Structure

Although FrontBase can be fully embedded into the directory structure of the parent application, there still has to be the notion of a FrontBase directory structure as well. The FrontBase server simply uses a "relative to where I am" scheme to locate the few pieces it needs, including the actual database file.

The FrontBase directory structure and files for a normal end-user installation is usually:

FrontBase / Collations / CaseInsensitive.coll1
FrontBase / Databases
FrontBase / Java / frontbasejdbc.jar
FrontBase / Library / DefinitionSchema.sql
FrontBase / Library / InformationSchema.sql
FrontBase / Library / OpenBaseImport.sql
FrontBase / Library / KeyWords.txt
FrontBase / Library / FBSQLErrors.array
FrontBase / Library / *.ucm
FrontBase / LicenseString
FrontBase / Templates / *
FrontBase / TransactionLogs
FrontBase / Translations / ToLower.trans
FrontBase / Translations / ToUpper.trans
FrontBase / bin /
FrontBase / include / *
FrontBase / lib / libFBCAccess.a

The function of each subdirectory in the FrontBase directory is:

**Collations**

Placeholder for all collation definitions to be used by the given database schema. If your schema doesn't make use of collations, this directory can be empty or taken out.

**Databases**

Will hold the actual database file. Although the actual database file can be located anyplace in the host file system, it is recommended to use this directory as it makes support easier.

**Java**

Holds the JDBC driver for FrontBase. This directory can be empty or taken out.

**Library**

Holds various house-keeping files incl. files used during bootstrapping of a new database. If a parent application includes a pre-boostrapped database, the InformationSchema.sql and DefinitionSchema.sql files are not needed. If the FrontBase is required to, as part of an installation, to bootstrap a new database, these two files MUST be available in the Library directory.

The **FBSQLErrors.array** file is a list of paradigm error messages, used by client applications to map error messages returned by the server into textual error messages. A parent application will typically embed (or simply ignore) this file into it self. FBSQLErrors.array can be edited for providing localized error messages. This file is NOT used by the server.

The **.ucm** files are used by client applications to map the UTF8 encoded strings, returned by the server into the chosen character set. Client applications can decide to use other means of mapping UTF8 into a given character set. The .ucm files are NOT used by the server. Other files in this directory can be taken out.

**Templates**

HTML files used by the FBWebManager.  This directory can be empty or taken out.

**TransactionLogs**

Created and maintained exclusively by the server. This directory is NOT to be deleted.

**Translations**

Holds two translation files for support of UPPER and LOWER (SQL functions). If the parent application doesn't use LOWER and UPPER, this directory can be empty or taken out.

**bin**

Holds the executables (the binaries) of a Frontbase distribution. Only the FrontBase server executable is needed, but the sql92 (command line tool) executable could be advantageous to include as well.

#### include

Holds various files used by a developer. This directory can be empty or taken out.

#### lib

Holds various files used by a developer. This directory can be empty or taken out.

### Embedded Deployment

The minimal FrontBase directory structure that can be deployed in an embedded situation is:

FrontBase / Databases / <pre-bootstrapped database>
FrontBase / LicenseString
FrontBase / TransactionLogs
FrontBase / bin / FrontBase

The recommended FrontBase directory structure that can be deployed in an embedded situation is:

FrontBase / Collations / CaseInsensitive.coll1
FrontBase / Databases
FrontBase / Library / DefinitionSchema.sql
FrontBase / Library / InformationSchema.sql
FrontBase / LicenseString
FrontBase / TransactionLogs
FrontBase / Translations / ToLower.trans
FrontBase / Translations / ToUpper.trans
FrontBase / bin / FrontBase
FrontBase / bin / sql92

## Starting the FrontBase server - Windows NT/ 2000/XP

The Windows platforms deviates enough from all other supported platforms to warrant its own description.

During the installation process, the FrontBase server and database must be installed as a normal service application:

<drive>:<install path>\FrontBase\bin\FrontBase -install <database name>

The FrontBase server can then be started and stopped like any other service application, e.g. automatically via the Service Control Manager and/or programmatically by the parent application.

Currently, the server will, per default, try to create the database file in C:/usr/FrontBase/Databases. By defining a system wide environment variable called FB_HOME_DRIVE, the Databases directory can be located where appropriate. Typically the FB_HOME_DRIVE variable is, in an embedded situation, defined to be

<drive>:<install path>\FrontBase\

## Starting the FrontBase server - Other platforms than Windows NT/2000/XP

The server is simply started as any other background application:

<install path>/FrontBase/bin/FrontBase [<options>] <database name> &

If there is no path information prefixing the database name, the server will assume that the database resides in the Databases directory.

### How to tell the server what port number to use

The portnumber is hardcoded in the LicenseString file, any use of the -port<number> option is ignored.