FrontBase® 4.x
Users Guide

Updated 2007-05-31 BB

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. 26741 Portola Pkwy. Suite 1E #414 Foothill Ranch, CA 92610
	Frontline Software Aps Blokken 15 DK-3460 Birkerød Denmark
Ordering	Voice: +1 949 636 8026 Fax: +1 949 330 6371
	Voice: +45 4582 6262 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
Document Feedback	doc-feedback@frontbase.com

Table of Contents

1 Foreword

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

1.1 Architecture

1.1.1 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.

1.1.2 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.

1.1.3 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.

1.1.4 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.

1.1.5 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 plug-in and an Omnis Studio DAM.

2 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 on page 13
- Standards Compliance on page 14
- Key Features on page 16
- Drivers, Adaptors and Plug-ins on page 19
- Migration on page 23
- FrontBase-Related Processes on page 23

2.1 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 on page 13
- Designed with Forethought on page 13
- Solid Foundation on page 14

2.1.1 Supported Platforms

- Mac OS X
- Windows
- Linux
- Unix

2.1.2 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 – no particular precautions are needed.

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

2.2 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 discusses the following:

- Full SQL 92 Compliance on page 15

- Unicode Character Representation on page 15
- TCP/IP Client/Server Interaction on page 15
- ANSI C Codebase on page 16

2.2.1 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 SQL Standard (4th Edition)”.

http://www.amazon.com/exec/obidos/ASIN/0201964260/002-2828760-6502439

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.

2.2.2 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.

2.2.3 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.

2.2.4 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.

2.3 Key Features

This section introduces the key features of FrontBase that 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 discusses the following:

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

2.3.1 Petabyte Databases, Gigabyte Column Values

FrontBase supports petabyte-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 partitions of up to 2^40 512-byte blocks, yielding one half petabyte of usable space per partition. FrontBase 4.x uses a block-size of 512 bytes for low-level disk access. Partitions may reside in distinct physical devices, enabling FrontBase to use 64 bit addressing capabilities resulting in low level disk access to handle up 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 (the few and rare)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.

2.3.2 FrontBase Security

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

2.3.2.1 Passwords

FrontBase provides two layers of passwords for protecting access to databases: database passwords and user 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 immediately.
User password	Each database user can have a password. The server verifies the password 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.
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.

FrontBase furthermore offers password authorization of overall database administration: The so-called Server Authentication feature may be enabled, which means that password authorization is needed in order to perform operations such as database creation and deletion as well as starting and stopping database servers.

2.3.2.2 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 specify that communication channels between the server and its clients must be secure.

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 per 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.
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.

2.3.2.3 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.

2.3.3 Transactions

Transaction Logging on page 71 describes FrontBase's transaction support.

2.3.4 Row Level Privileges

FrontBase offers a unique feature called Row Level Privileges (see page 224), 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.

2.3.5 Live Backup

Backup and Restore on page 85 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).

2.3.6 Caching

Tuning FrontBase on page 96 describes FrontBase's caching schemes.

2.3.7 Multi-Server Deployment

Replication on page 76 describes FrontBase's replication features and Clustering on page 80 the features for multi-server deployment for both redundancy and enhanced performance.

2.4 Drivers, Adaptors and Plug-ins

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 Download section at http://www.frontbase.com/ along with associated documentation.

2.4.1 ODBC

The ODBC driver is for use on Windows (NT/2000/XP/ME/98) and Mac (PowerPC/Intel) with any ODBC compliant application. Details about establishing a connection with each driver can be found in the Readme files supplied with the drivers.

For the driver to correctly query the columns in a database schema using the SQLColumns ODBC API call it must create a VIEW called VIEW_NEEDED_BY_FBODBC. This is automatically attempted when a connection on that schema first issues the SQLColumns call.

However, in some circumstances when using an ODBC application an error may be returned specifying that the view could not be found. Typically, this occurs because the user of the connection is not the owner of the schema and will not have permission to create the view. It will also fail to be created if the connection has been placed in a read-only transaction mode.

If this error occurs then first make sure the view exists. To do this through the driver, connect as the owner of the schema and query the columns in a table, e.g. (using the MSQuery wizard). Make sure that the connection's transaction mode allows writes to the database.

Alternatively, you can manually create the VIEW (from FBManager/FBJManager) by issuing the following,

CREATE VIEW VIEW_NEEDED_BY_FBODBC AS SELECT COLUMN_PK, TABLE_PK, "COLUMN_NAME", IS_NULLABLE, ORDINAL_POSITION, COLUMN_DEFAULT FROM DEFINITION_SCHEMA.COLUMNS;

GRANT SELECT ON VIEW_NEEDED_BY_FBODBC TO "_PUBLIC";

This only needs to be done once for each schema and then any user can connect and query the columns.

2.4.2 JDBC

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

The URL syntax is as follows:

```
jdbc:FrontBase://<host info> [<arg list>]

<host info>            ::= <regular> | <cluster list
<arg list>             ::= <user> | <user password> |
                           <database password> | <session> |
                           <system> | <isolation level> |
                           <locking discipline> | <access mode>

<regular>              ::= host_name <database info>
<cluster list>         ::= <cluster member> {/<cluster member>}

<database info>        ::= /database_name | :port

<cluster member>       ::= database_name@host_name

<user>                 ::= /user=user_name
<user password>        ::= /upasswd=user_password
<database password>    ::= /dbpasswd=database_password
<session>              ::= /session=session_id
<system>               ::= /system=system_user
<isolation level>      ::= /isolation=<isolation value>
```

```
<locking discipline>  ::= /locking=<locking value>
<access mode>         ::= /readOnly=<access value>

<isolation value>     ::= read_uncommitted | read_committed |
                          repeatable_read | serializable | versioned
<locking value>       ::= pessimistic | deferred | optimistic
<access value>        ::= true | false
```

Establishing a connection requires that the driver has been installed and that classpaths have been set up correctly.

Example:

```
import java.sql.*;

public class Test {

   public static void main(String[] args) {

      // Register driver.
      try {
         Class.forName("com.frontbase.jdbc.FBJDriver");
      }
      catch ( Exception e ) {
         System.err.println("Unable to load driver");
         e.printStackTrace();
      }

      // The URL syntax is described above
      try {
         String url      = "jdbc:FrontBase://hostName/databaseName";
         String user     = "myusername";
         String password = "mypassword";

         Connection con = DriverManager.getConnection(url, user,
                                                      password);

         Statement stmt  = con.createStatement();

         someCode...

         stmt.close();
         con.close();
      }
      catch( SQLException ex ) {
         System.err.println("SQLException: " + ex.getMessage());
         ex.printStackTrace();
      }
   }
}
```

2.4.3 WebObjects 5 Plug-in

A WebObjects plug-in enables connection to FrontBase on any supported operating system. There are separate plug-ins for Win32 and Mac OS X.

2.4.4 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).

2.4.5 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.

2.4.6 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) Mac OS X and Win32. Documentation is included in the download package.

2.4.7 REALBasic

These plug-ins are for REALBasic 3.0 PPC running on MacOS Classic (8/9) and REALBasic 3.0 Carbon running on Mac OS X.

2.4.8 Tcl Driver

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

2.4.9 EOF Adaptor

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

2.5 Migration

FrontBase has tools for importing databases from FileMaker and MySQL.

2.5.1 FileMaker

The FileMaker migration (see page 116) is a two-step process. The tables of a FileMaker database are exported from FileMaker. The exported files are moved to Mac OS X, where an application imports them into FrontBase.

2.5.2 MySQL

The MySQL migration tool (see page 117) 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 88.

2.6 FrontBase-Related Processes

There are two main processes that run in a FrontBase installation: FBExec and FrontBase. In case of replicated configurations, the FBReplicator is the third major active FrontBase process. This section briefly identifies these processes.

2.6.1 FBExec

FBExec acts as a broker between FrontBase databases running on your computer and client software running on your computer or over the network. FBExec knows about database names and how to establish connections to their FrontBase Servers.

When you install FrontBase, your computer will be set up so that FBExec is launched at start-up. A detailed description of the FBExec process and how to start it is found in FBExec Invocation om page 197.

2.6.2 FrontBase

One instance of FrontBase will be running for each database that is hosted on your computer. Each FrontBase instance is started directly from the command-line (UNIX installations), by the Service Manager (Windows NT), or by using the FrontBaseManager or the sql92 tool.

A detailed description of a FrontBase Server and how to start it is found FrontBase Invocation on page 199.

2.6.3 FBReplicator

When FrontBase is configured for a replicated setup, the FBReplicator serves as the connection between the replication master and the replication clients. The FBReplicator simply reads the Transaction Log produced by the replication master and distributes the SQL statements written here to (all of) the replication clients.

A detailed description of the FBReplicator process and how to start it is found in FBReplicator Invocation on page 208.

3 Installation

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

3.1 Downloading FrontBase

To download FrontBase, please visit the Download section of http://www.frontbase.com. The size of the archive is less than 10 MB and should require only a few minutes to download.

The following platforms are supported:

- Mac OS X
- Windows
- Linux
- Unix

3.2 FrontBase Directory Structure

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

3.2.1 Directory Location

The default location of the FrontBase installation depends on the platform on which you install FrontBase.

Platform	**Path**
Mac OS X	/Library/FrontBase
Windows	<drive>:/usr/FrontBase
RedHat Linux (x86)	/usr/local/FrontBase
SuSE Linux (x86)	/opt/FrontBase
YellowDog Linux (PPC)	/opt/FrontBase
Debian Linux (x86)	/usr/lib/FrontBase
Mandrake Linux (x86)	/usr/lib/FrontBase
Solaris	/opt/FrontBase
FreeBSD (x86)	/usr/local/FrontBase

The FrontBase installers by default installs FrontBase in the platform-specific default locations described above, but you may actually install FrontBase anywhere in your file system. Currently, the FrontBase directory needs to be owned by "root" or "administrator", but that mostly artificial requirement should also disappear in a future version.

3.2.2 Directory Contents

The structure of the FrontBase home directory is independent of the platform on which it is installed, and it normally contains the following subdirectories and files:

Backups	The Backups directory contains backup directories, one for each database

Collations	The Collations directory holds all defined collations (orderings of Unicode characters which can be instantiated as COLLATIONs in your SCHEMAs).
Databases	The Databases directory holds all databases served from the host. For each database there are three or more files: <database-name>.fb, which contains (some of) the actual database data, <database-name>.fb.log containing miscellaneous status and error messages and <database-name>.fb.pid containing the Process ID of the current (or latest) database server. If the database is a member of a cluster, the cluster composition is described in <database-name>.cluster. If SQL logging is enabled for a database, it is written to the file <database-name>.fb.sql. 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 one of the database managers to delete your databases. On Windows NT, you should make sure that you've removed the database as a service before deleting it.
Documentation	The Documentation directory (if present) contains an SQL example, the FrontBase license agreement, the platform specific Readme file and the Release Note for the present FrontBase installation.
FBExec.autostart	FBExec.autostart specifies which databases the FBExec should start on its own start-up, if any
FBExec.log	FBExec.log is a file to which the FBExec process appends miscellaneous status and error messages, when enabled
FBExec.passwd	FBExec.passwd contains a digested form of the password used by the Server Authentication feature of FrontBase, when enabled.
Java	The Java directory (if present) contains miscellaneous FrontBase related Java code, for example the JDBC driver for FrontBase
Library	The Library directory contains miscellaneous files required to create a new database.
LicenseString	The LicenseString file contains the license string for FrontBase on this machine. FrontBase licenses are free and may be obtained from the Buy section of http://www.frontbase.com.
TransactionLogs	The TransactionLogs directory contains transaction logs in separate sub-directories, one for each database.
Translations	The Translations directory holds all defined translations.

bin	The bin directory contains the FrontBase executables. You may want to add the <FB home>/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.
include	The include directory contains header files used for development.
lib	The lib directory contains FrontBase libraries used for development.

3.3 FrontBase Components

A FrontBase installation consists of a number of executables (servers, applications and tools), and may be supplemented by a number of Client Libraries.

3.3.1 FrontBase Servers

These are part of any FrontBase installation and are found in the bin directory of the FrontBase home directory:

FBExec	The FBExec is a service background process that is typically started automatically when the computer is restarted. The FBExec provides status, management and access information to all client applications concerning all databases on its host machine.
FrontBase	FrontBase is the actual database server. There is a FrontBase process running for each running database. A FrontBase process is typically started by various management applications, but can also be started from the command line.
FBReplicator	The replication daemon distributing transactions from a replication master to its replication clients.

3.3.2 FrontBase Applications

The FrontBaseManager is relevant only for the Mac OS X platform and the FrontBaseJManager is relevant for any platform on which Java is installed. The FrontBaseJManager is found in the Java directory of the FrontBase home directory (or may be downloaded separately from the FrontBase website):

FrontBaseManager	A Cocoa application for managing databases, database schemas and content. The FrontBaseManager also contains an "sql92" functionality; i.e. SQL statements can be sent to a FrontBase database for execution. Please note that the FrontBaseManager can manage databases on all hosts in a network with a FrontBase installation.
FrontBaseJManager	The Java equivalent of the FrontBaseManager. Relevant for all platforms on which Java is installed.

3.3.3 FrontBase Tools

These are command line tools which are part of any FrontBase installation. They are found in the bin directory of the FrontBase home directory:

sql92	A command line tool for managing and accessing databases. sql92 will let you create, start and stop databases, connect to databases, and execute SQL statements.
FBInfoCenter	A command line tool which prints out details about your FrontBase installation and your machine.
FBRAccess	A command line tool for inspecting and managing a replication setup.
FBTLogs	A command line tool for summarizing the transaction log of a database.
FBTLog	A command line tool for displaying (portions of) the transaction log of a database in an ASCII form.
FBKeyGenerator	A command line tool for generating encryption keys.
FBChangeKey	A command line tool for changing the encryption of a database.

3.3.4 Client Libraries

Of the Client Libraries mentioned below, only FBCAccess is relevant for all platforms and is included in the default FrontBase installation. The other libraries may be downloaded separately when and if they are needed.

FBAccess	A Cocoa framework providing the API that allows Cocoa applications to administrate and connect to FrontBase databases.
FBCAccess	A C library providing the API that allows client applications to administrate and connect to FrontBase databases.
PHP3/4	The FrontBase driver for a PHP connection
Perl	The FrontBase driver for a Perl connection
ODBC	The FrontBase driver for an ODBC connection
JDBC 3	The FrontBase driver for a JDBC connection
EOF Adaptor	A Cocoa framework providing the functionality required by the Enterprise Objects Framework which is a part of WebObjects 4.5.1 from Apple Computer, Inc.

3.4 Platforms

3.4.1 Mac OS X and Mac OS X Server 10.x

The following Administrative Tools are provided:

- FrontBaseManager (see page 123)
- sql92 (see page 177)

The following Client Libraries are available:

- FBAccess
- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 3
- EOF Adaptor

3.4.1.1 Installation

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 in to your Mac OS X computer. FrontBase can be installed and run by any user.

2. If you have a previous version of FrontBase running on your server, the install script will stop all FrontBase related processes automatically. Refer to “Administration Tools” on page 70 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.

3. 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.

4. Select the .dmg file and double click. To install the .pkg, 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.

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) is running, the system should reply with something like:

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

   You can also check this process on Mac OS X by running the Activity Monitor.

   If it is not running, try to launch it from the command-line:

   ```
   /Library/FrontBase/bin/FBExec &
   ```

   Please note that the FBExec is installed so that it will 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 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 have 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 70. Then restart your client software.

7. Adjusting the search path (PATH) to include the /Library/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.

3.4.1.2 Removing FrontBase

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

1. Log in 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/FBScriptAgent.app
rm -r /Applications/FBUnicodeManager.app

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

rm /Library/StartupItems/FrontBase*

rm -r /Library/FrontBase
```

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

3.4.2 Windows 2000 / NT / XP

The following Administrative Tools are provided:

- FBJManager (a Java variant of the FrontBaseManager described on page 123)
- sql92 (see page 177)

The following Client Libraries are available:

- FBAccess
- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 3
- EOF Adaptor

3.4.2.1 Installation

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 in to 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. Use your favorite zip file utility (such as PKZip) to unzip the downloaded archive. Unzipping will yield two executable (.exe) files. Run each of them and follow the installation instructions provided.

   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.

3. 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:).

4. The installation process will automatically define FBExec as an NT service and start it. 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).

5. 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 that accompanies your FrontBase server.

3.4.2.2 Removing FrontBase

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

1. Log in as administrator

2. Stop the FBExec service using the ServiceControlManager application.

3. Remove the FBExec as a service:

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

4. Remove all FrontBase databases as services:

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

5. Repeat step 4 for each database that has been installed as a service (find the list of installed databases using the ServiceControlManager application).

6. Remove the following folders (using e.g. the Windows NT Explorer):

```
<drive>:\Apple\Local\Library\Frameworks\FBAccess.framework
<drive>:\Apple\Local\Library\Frameworks\FrontbaseEOAdaptor.frame
work
<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 contain entries to remove installed FrontBase components, Server, FrameWorks and Tools. By removing the last folder in step 7, you irreversible remove all your databases.

3.4.3 RedHat Linux (x86)

The following Administrative Tools are provided:

- FBJManager (a Java variant of the FrontBaseManager described on page 123)
- sql92 (see page 177)

The following Client Libraries are available:

- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 3

3.4.3.1 Installation

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 in to 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. 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.

   If you are also running client software (such as PHP), 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.

3. 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.

4. 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) is running, the system should reply with something like:

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

If it is not running, try to launch it from the command-line:

```
/usr/local/FrontBase/bin/FBExec &
```

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

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

   First, start your databases as described in “Administration Tools” on page 70. Then restart your client software.

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

7. 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 that accompanies your FrontBase server.

3.4.3.2 Removing FrontBase

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

1. Login as root

2. Remove the FrontBase installation:

```
rpm -e FrontBase
```

3. Remove the remains of the installation directory:

```
rm -r <FB home>/FrontBase
```

3.4.4 SuSE Linux (x86)

The following Administrative Tools are provided:

- FBJManager (a Java variant of the FrontBaseManager described on page 123)
- sql92 (see page 177)

The following Client Libraries are available:

- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 3

3.4.4.1 Installation

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 in to 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. 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 PHP), 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.

3. 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.

4. 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) is running, the system should reply with something like:

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

    If it is not running, try to launch it from the command-line:

    ```
    /opt/FrontBase/bin/FBExec &
    ```

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

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

    First, start your databases as described in “Administration Tools” on page 70. Then restart your client software.

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

7. 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 that accompanies your FrontBase server.

3.4.4.2 Removing FrontBase

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

1. Login as root

2. Remove the FrontBase installation

```
rpm -e FrontBase
```

3. Remove the remains of the installation directory:

```
rm -r <FB home>/FrontBase
```

3.4.5 YellowDog Linux (PPC)

The following Administrative Tools are provided:

- FBJManager (a Java variant of the FrontBaseManager described on page 123)
- sql92 (see page 177)

The following Client Libraries are available:

- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 3

3.4.5.1 Installation

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. 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 70 if you wish to stop things manually.

   If you are also running client software (such as PHP), 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.

3. 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.

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

```
ps axc | grep FB
```

f the FBExec process (a key FrontBase component) is running, the system should reply with something like:

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

If it is not running, try to launch it from the command-line:

```
/opt/FrontBase/bin/FBExec &
```

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

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

   First, start your databases as described in “Administration Tools” on page 70. Then restart your client software.

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

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

3.4.5.2 Removing FrontBase

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

1. Login as root

2. Remove the FrontBase installation

```
rpm -e FrontBase
```

3. Remove the remains of the installation directory:

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

NOTE: By removing the FrontBase directory, you irreversibly remove all your databases.

3.4.6 Debian Linux (x86)

The following Administrative Tools are provided:

- FBJManager (a Java variant of the FrontBaseManager described on page 123)
- sql92 (see page 177)

The following Client Libraries are available:

- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 3

3.4.6.1 Installation

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. 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 70 if you wish to stop things manually.

   If you are also running client software (such as PHP) 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.

3. Once you have downloaded the .deb file, install it as follows (terminal window, logged in as root):

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

   By default, the script will install FrontBase into the /usr/lib/FrontBase directory.

4. 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) is running, the system should reply with something like:

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

   If it is not running, try to launch it from the command-line:

   ```
   /usr/lib/FrontBase/bin/FBExec &
   ```

   Please note that the FBExec is installed so that it will be started automatically 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 results in an error, please send e-mail to support@frontbase.com. We will be happy to help you.

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

   First, start your databases as described in “Administration Tools” on page 70. Then restart your client software.

6. Adjusting the search path (PATH) to include the /usr/lib/FrontBase/bin directory will make your life on the command-line simpler.

7. 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 that accompanies your FrontBase server.

3.4.6.2 Removing FrontBase

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

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 FrontBase directory, you irreversibly remove all your databases.

3.4.7 Mandrake Linux (x86)

The following Administrative Tools are provided:

- FBJManager (a Java variant of the FrontBaseManager described on page 123)
- sql92 (see page 177)

The following Client Libraries are available:

- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 3

3.4.7.1 Installation

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. 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 PHP), 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.

3. 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.

4. 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) is running, the system should reply with something like:

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

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

```
/usr/local/FrontBase/bin/FBExec &
```

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

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

   First, start your databases as described in “Administration Tools” on page 70. Then restart your client software.

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

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

3.4.7.2 Removing FrontBase

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

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
```

NOTE: By removing the FrontBase directory, you irreversibly remove all your databases.

3.4.8 Solaris

The following Administrative Tools are provided:

- FBJManager (a Java variant of the FrontBaseManager described on page 123)
- sql92 (see page 177)

The following Client Libraries are available:

- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 3

3.4.8.1 Installation

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: FrontBaseManager 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 in to 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. 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 or you. Refer to “Administration Tools” on page 70 if you wish to stop things manually.

   If you are also running client software (such as PHP), 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.

3. 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
```

4. By default, the script will install FrontBase into the /opt/FrontBase directory.

5. 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) is running, the system should reply with something like:

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

   If it is not running, try to launch it from the command-line:

   ```
   /opt/FrontBase/bin/FBExec &
   ```

   If launching FBExec 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. PHP) running prior to starting your upgrade, you should restart those now.

   First, start your databases as described in “Administration Tools” on page 70. Then restart your client software.

7. Adjusting the search path (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 that accompanies your FrontBase server.

3.4.8.2 Removing FrontBase

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

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 parent directory and remove the remains of the installation directory:

   ```
   cd ..
   rm -r FrontBase
   ```

   NOTE: By removing the FrontBase directory, you irreversibly remove all your databases.

3.4.9 FreeBSD (x86)

The following Administrative Tools are provided:

- FBJManager (a Java variant of the FrontBaseManager described on page 123)
- sql92 (see page 177)

The following Client Libraries are available:

- FBCAccess
- PHP3/4
- Perl
- ODBC
- JDBC 3

3.4.9.1 Installation

Following the instructions below, you will 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: FreeBSD installs with shell script and requires some manual configuration.

1. Log in to 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. 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 70 if you wish to stop things manually.

   If you are also running client software (such as PHP), 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.

3. 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
```

4. By default, the script will install FrontBase into the /usr/local/FrontBase directory.

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
```

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

```
cd /usr/local/FrontBase (default FrontBase directory)
./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. PHP) running prior to starting your upgrade, you should restart those now.

First, start your databases as described in “Administration Tools” on page 70. Then restart your client software.

8. Adjusting the search path to include the /usr/local/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. You should now spend some time reading the documentation that accompanies your FrontBase server.

3.4.9.2 Removing FrontBase

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

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 parent directory.

```
cd ..
```

5. Remove the remains of the installation directory:

```
rm -r FrontBase
```

NOTE: By removing the FrontBase directory, you irreversibly remove all your databases.

3.5 FrontBase Licenses

3.5.1 Obtaining a License

FrontBase requires a valid license to run with full functionality enabled. FrontBase licenses are free of charge and are obtained from the Buy section of the FrontBase Web site at: http://www.frontbase.com.

You obtain a license string from the website by registering the platform on which you wish FrontBase to run and by supplying the IP or MAC (Ethernet) address of the server. A license string for your server will be sent to you by e-mail.

There is only one license kind available, called E-Enterprise with LookSee, which enables the full, non-expiring functionality of FrontBase.

3rd party companies (ISVs/OEMs) can incorporate FrontBase into their solution using an Embedded license. Please contact FrontBase for further information. Technical details can be found in the section "Embedding FrontBase into your own application or solution" on page 229.

3.5.2 Installing a License

There are two ways to install the license string:

1. If you are running FrontBase on a platform that includes the FrontBaseManager (Mac OS X) or the FrontBaseJManager (all platforms with Java installed), you can install it from there. Choose License management from the Tools menu, select the server, and paste the license string and license check that were e-mailed to you.

2. You can create the license file yourself using a text editor or, for example, the Shell command 'cat > LicenseString' from the command line. The file is named LicenseString and resides in the FrontBase installation directory. Its format is as follows:

```
<64 character license string>:<16 character license checksum>
```

3.6 Keeping Current

FrontBase is continually being improved. However you have obtained FrontBase, if it is some time ago, you most likely are not running the latest version (nor reading the latest documentation!).

3.6.1 Determining the Latest Version

The latest version of FrontBase available for your platform of choice is identified at the Download section of the FrontBase Web site: http://www.frontbase.com

3.6.2 Determining your Current Version

3.6.2.1 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.

3.6.2.2 FrontBaseManager (Mac OS X) and FrontBaseJManager (Java)

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.

3.6.2.3 Command Line / Terminal

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

```
FrontBase -v
```

3.6.3 Upgrading your FrontBase Server

When you need to upgrade, go to the Download section of the FrontBase Web site: http://www.frontbase.com, choose your platform and download the FrontBase software. Then proceed to section “Installation” on page 25 and follow instructions there.

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

4 Basic Concepts

This chapter is divided into the following sections:

4.1 SQL 92 Concepts

SQL 92 is the latest official standard for SQL from the ANSI/ISO bodies and as such represents 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:

4.1.1 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.

4.1.2 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.

4.1.3 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;
```

4.1.4 DATE, TIME and TIMESTAMP

SQL 92 has an elaborate time concept that includes the following data types:

- 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 time in January).

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

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

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

4.1.5 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":

1. There is no keyword in SQL 92 ending with an underscore, e.g. it will be perfectly legal to use SELECT_ as an identifier.

2. By enclosing the identifier in double quotes, essentially any spelling can be used as an identifier, e.g. "SELECT" is a legal identifier.

3. 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.

4.1.6 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.

4.2 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.

4.2.1 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.

4.2.2 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.

4.2.3 Updateability

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

4.2.4 Isolation level

SQL 92 defines 4 isolation levels:

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

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 did not really exist.
Non-repeatable Read	A transaction reads 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.

4.2.5 Locking Discipline

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

PESSIMISTIC	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 resources and when none of them can continue before they have acquired all the requested recourses. The connection that causes the deadlock to become a problem is the transaction that will get rolled back.
OPTIMISTIC	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	DEFERRED (also called upgradeable) 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.

4.2.6 Locking and Enterprise Objects Framework

Apple’s Enterprise Objects Framework (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. This 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.

5 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 straightforward, 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 <FB home>/FrontBase/bin directory is in your $PATH.

This chapter contains the following sections:

5.1 Administration Tools

The sql92 command-line tool and the FrontBaseManager application (for Mac OS X and Mac OS X Server 10.x) are each described in separate chapters as indicated below:

5.1.1 FrontBaseManager

FrontBaseManager on page 123, introduces the application that lets you monitor and administer local and remote database servers. It is available for Mac OS X installations of FrontBase. Windows installations use the original FBManager.

5.1.2 FrontBaseJManager

The FrontBaseJManager is a Java application with much the same functionality as the FrontBaseManager on page 123.

5.1.3 sql92

sql92 on page 177 introduces the FrontBase command-line tool, which is available for all installations of FrontBase.

5.1.4 FBScriptAgent

FBScriptAgent is a “scriptable” application that 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 be 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/

5.2 Transaction Logging

By default, FrontBase 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 by executing and committing those transactions.

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. Replication and clustering are two distinct methods for obtaining both redundancy (protect from downtime when a server becomes unavailable) and load distribution (better response time in heavy load situations).

The following sections explain the various elements of transaction logging in FrontBase:

- Implementation on page 71
- Administration on page 72
- SQL syntax on page 73
- Transaction Log Commands in sql92 on page 73
- Options on page 74
- FBTLogs on page 74
- FBTLog on page 75

5.2.1 Implementation

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

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. The directory also contains a file named .lock, which is used to prevent more than one FrontBase server at a time writing to this transaction log.

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 indivisibly take place. The names of the backup file and the new transaction log directory will be identical, except for the first character (B and L, respectively).

5.2.2 Administration

The use of transaction logging is optional, but it is the FrontBase default and strongly 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 (it may be hundreds of transactions, or more). Therefore it is safest to state that transaction log files are relevant back to the latest 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.

5.2.3 SQL syntax

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

```
CREATE TRANSACTION LOG;
```

Enables transaction logging (default and strongly recommended). If transaction logging is disabled, a new transaction log directory will 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>.

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

Creates a new transaction log directory.

```
WRITE BACKUP [ TO <path-expr> ] [ COMPRESSED ];
```

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.

5.2.4 sql92 Command

```
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).

5.2.5 FrontBase Options

Options may be specified when a FrontBase server is started – see FrontBase Invocation on page 198 for a complete description. The following are relevant for transaction logging:

-clients=no | yes	If no is specified, FrontBase will exit after its initialization phase, just prior to otherwise accepting client connections. This simply brings the database up-to-date with its transaction log.
-keeptlog	Prevents the server from clearing the transaction log, if one exists.
-rollforward=no | <number>	Changes the default behavior of the server with respect to the transaction log, if present. If <number> is specified, its value determines the last transaction number that will be committed to the database; if no is specified, no transactions from the transaction log are committed to the database.
-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.
-transaction=<number>	If <number> is specified, its value becomes the transaction number of the database, just prior to transaction log handling. This specifies the starting point in the transaction log for bringing the database up-to-date.

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

5.2.6 FBTLogs

This utility is part of your FrontBase installation:

```
FBTLogs [-c] [-d <transactionlog-directory>] [-s <file-count>] [-v]
[<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.

The -d option identifies the specified transaction log directory as the object of the summaries, and only if this is specified may <database-name> be omitted.

The -c option forces FBTLogs to interpret all transaction log files in order to verify that they may be read correctly.

The -v option specifies that slightly more information is output.

5.2.7 FBTLog

This utility is part of your FrontBase installation:

```
FBTLog [-c] [-d <transactionlog-directory>] [-k <key-filename>] [-n
<number>] <database-name> [<first-number> [<last-number>]]
```

Lists transactions in a transaction log, by default starting from the oldest transaction in the latest coherent transaction log sequence. If specified, <first-number> and <last-number> identify the range of transaction numbers that should be listed.

The -d option identifies the specified transaction log directory as the starting point.

The -k option identifies an encryption key file; this is needed if a decryption of an encrypted transaction log is wanted.

The -c option forces FBTLog to interpret all transaction log files in order to verify that they may be read correctly.

The -n option makes FBTLog list transactions in a form that may directly serve as SQL input to a database; <number> specifies how many transactions should be included in every commit, ie. when <number> equals 1, transactions are committed in the same way as when the transaction log was created.

5.3 Replication

A replication setup is a number of database servers that logically implement one and the same database, even though the database exists in several copies, one for the master and one for each replication client. FrontBase supports replication of a master database to several read-only clients that 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.

The transaction log of the master database represents an externally accessible serialization of the transactions committed to the master database. The replication mechanism (the Replicator) described below thus only needs to be able to read the transaction log of the master database, and the server for the master database does not need to be aware of the presence or not of the replication mechanism. This implies a very loose coupling between the master database and the replication clients, as well as absolutely no overhead on the part of the master database (we strongly recommend that transaction logging is enabled, whether replication or clustering are used or not!). Therefore, replication of a master database can be used to off-load time consuming backup and report writing tasks from the master, and a client may serve as a backup in itself. Apart from that, replication may be used for the more traditional distribution of read-only copies of a database.

As the clients are all read-only, a transaction can be committed on the master without any synchronization with the clients, and 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 replication setup consists of:

- The Replication Master
- The Replicator
- The Replication Clients

5.3.1 Replication Master

Any database for which transaction logging is activated may serve as a replication master. Technically, the server for the replication master database should be started with the -rmaster option.

5.3.2 Replicator Daemon

The replication of a master database is performed by a separate program (or process), the Replicator, which reads the transaction log produced by the server for the master database. 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 SQL that actually modifies the database is written to the log.

The Replicator 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 Replicator on all the accessible clients. The monitoring of the transaction log is basically a polling with a configurable interval that controls the reaction time of the Replicator. 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 Replicator may be managed by e.g. the FrontBaseManager or by the sql92 command line tool, as illustrated below. It may also be started from the command line, as follows:

```
FBReplicator [-bsv] [-i <interval>] [-d <transactionlog-directory>]
[-p <portno>] [-k <keyfile] <database>
```

The Replicator accepts the following options:

-b	Run as a background program (i.e., without controlling terminal).
-s	Silent. The Replicator will not log events to stdout (default).
-v	Verbose. The Replicator will log events to stdout.
-i <interval>	Seconds. The polling interval determines how often the Replicator will examine the transaction log of the master. The Replicator will only sleep the specified amount of time when no transactions are pending replication. The default value is 10 seconds.
-d <transactionlog-directory>	Specifies the transaction log directory to be used for replication. To be used when the transaction log is not in the default position.
-p <portno>	Specifies the port number at which the Replicator will talk to the controlling program (e.g., FBRAccess)
-k <keyfile>	Specifies the file containing the encryption key for an encrypted database. Must be specified when database is encrypted.

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

5.3.3 Replication Clients

The replication clients are ordinary database servers that must be started with the -rclient option (otherwise the Replicator will not talk to them). This disallows the users of the replication clients from updating the database. The client databases must share a common history with the master database which means that they must have been created in a replication setup with the master, or have been started from some sort of copy of the master database (see section "Backup and Restore" on page 85).

5.3.4 Replication and Passwords

The Replicator 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, and is thus required to know the database and _SYSTEM passwords in order to establish a connection. The Replicator 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.

5.3.5 Replication and Encryption

When the master database is encrypted, the Replicator must be informed of the encryption key, so that it can decrypt transactions written to the master transaction log. The replication clients do not have to be encrypted, or may be encrypted with an encryption key different from the master. In all cases, the communication of transactions by the Replicator to the replication clients is in clear SQL.

5.3.6 Database Identification Checks

Each FrontBase database has a 48-bit unique identification. The master database and all the client databases must share the same unique database identifier, signifying that they share a common history. This property is checked by the Replicator whenever a new client is added to a replication setup.

5.3.7 Replicating a Database

The operations available for the administration of a replication setup are, for the purposes of this description, best illustrated in terms of the commands implemented by the sql92 command line tool, which is part of the FrontBase installation. The full set of sql92 commands for dealing with replication is described in section "Replication Commands" on page 187.

All replication commands require that the so-called "default database" is established. The default database identifies the master database for a replication setup.

Any existing database (whether it has a server running or not) may serve as a master for replication:

```
SET DEFAULT DATABASE <master>@<masterhost>;
SHOW DEFAULT DATABASE;
```

The first step in setting up a replication is to start the Replicator for the master identified by the default database:

```
START REPLICATOR;
```

Once the Replicator is running, the state of the replication setup may be inspected:

```
SHOW CLIENTS;
```

There are two ways of adding a new client to an existing replication setup:

```
ADD CLIENT <client>@<clienthost>;
CREATE CLIENT <client@<clienthost>;
```

The first form expects that the specified client exists and that it has a server running for it (a server started with the -rclient option). It checks that the specified client database may enter a replication setup with the master (database identification and transaction number checks) and, if that is the case, adds it to the client list of the Replicator. The second form additionally makes a copy of the master database (using the symbolic WRITE ALL ...) if the specified client database does not exist, and starts a server for the client if no server is running.

5.4 Clustering

A cluster is a number of database servers that logically implement one and the same database, even though the database exists in several copies, one for each cluster member. A client (a user) can connect to any server in the cluster and access and update the database. It is a crucial property of a cluster that when a server updates the database - i.e., commits a transaction - this transaction should (ultimately) be committed to all the other databases in the cluster, and with the same transaction number, too. This requires:

1. **Two-Phase commit**
   A mechanism that ensures that if a transaction is committed in one database server, it is committed in all (available) database servers.

2. **Provisions for absent servers**
   A mechanism that ensures that, under suitable circumstances, a cluster may continue operation (possibly including committing of transactions) without compromising data consistency within the cluster, even if not all cluster members are currently available.

The implementation of these mechanisms is described below.

5.4.1 Background

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

Enhanced data security	Each database server in the cluster holds a complete copy of the database, implying that so long as one machine remains intact no data will be lost.
Load distribution	A typical client application will execute read only transactions in a rate, orders of magnitude larger than the rate of read/write transactions, and a read only transaction will only access the cluster member it is connected to, thus the load can be distributed.
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 are those of the transaction outstanding at the point of connection loss.

Loosely speaking, the databases involved in a cluster should be identical at all times, but in a real world this may not be entirely possible.

5.4.2 Two-Phase Commit

Whenever a database server running as part of a cluster wishes to commit a transaction, it must make sure that the transaction is committed to all of the other active members of the cluster, or to none. This is done in the following way:

1. The committing server distributes the (SQL of the) transaction to each of the other active cluster members; each of these verifies that the transaction may be locally committed.

2. The committing server requests a "lock" for a specific transaction number in each the other active cluster members, in a particular order.

3. The committing server requests a "commit" for the specific transaction number in each the other active cluster members, in the same order as above.

If any of these points fail, the entire commit has failed.

5.4.3 Provisions for Absent Servers

At any one point in time, a cluster has a *static composition*, a fixed set of members that is called the current set of members. The *current set of members* is known to all the members in the cluster. A member of the cluster will only accept a cluster connection from another database server, if that other server can identify itself as one in the current set of members.

The static composition of a cluster may be changed. This implies changing the current set of members for all members of the new cluster. Clearly, the connection check just mentioned prevents a new database server from joining a cluster until it has been made a member of the current set of members.

The current set of members is recorded in the *ClusterDescr* file next to the database file. Ideally, at all times all members of a cluster should have identical ClusterDescr files, but since the members of a cluster may execute on different host machines, connected by inherently unstable physical connections, inconsistencies between the ClusterDescr files of a cluster may occur. Such inconsistencies must be resolved by user intervention, i.e. there is no automatic mechanism to maintain consistency between ClusterDescr files.

At any one point in time, a cluster has a *dynamic composition*, a set of members - running database servers - which is called the *active set of members*. The active set of members is necessarily a subset of the current set of members. Within the active set of members, all servers have a connection to all other servers, which makes it possible to carry out the two-phase commit algorithm described above.

Ideally speaking, a cluster member may only commit a transaction if it may commit this transaction in all the other members of the cluster; this way, identity between the databases involved in the cluster is inherently maintained. However, in a real world, the dynamic composition of a cluster may change. Database servers may be stopped (voluntarily as well as involuntarily), physical connections between machines may break, disks may crash, etc. It would not be acceptable if the disappearance of one server in a cluster would make it impossible to update the remaining members of the cluster. However, care must be taken so that inconsistencies do not arise.

Consider, for instance, a cluster with exactly two members where the physical connection between the members break down, but the servers are still running and have client connections. If both are able to commit transactions, an inconsistency is immediately the consequence.

In order to handle situations where not all members of the cluster are running and connected to each other (i.e., the set of active members is smaller than the set of current members), we introduce the notion of *majority*. This notion provides assistance in permitting an incomplete cluster to continue execution. Majority is a logical property (true or false) that may be assigned true to exactly one member of a cluster; this member is said to have majority. On top of this notion of individual server majority, we talk about an active set of members having majority; the servers in a set of active members with majority have the right to commit transactions.

Whenever the dynamic composition of a cluster changes, the set of connections from one server to other servers in the cluster changes. This change forces the servers in the cluster to (individually) consider whether they are a part of an active set of members with majority, and thus have the right to commit transactions for the cluster. A server without the right to commit transactions is said to be readonly. A server is a member of an active set of members with majority in the following situations:

1. The number of databases in the current set of members is odd, and our member is a member of an active set of members that numbers more than half the number of current members. For instance, if the cluster has a current set of three members, two of these members must be active together to have majority; if the cluster has five current members, any three active members have majority.

2. The number of databases in the current set of members is even, and our member is a member of an active set of members that numbers more than half the number of current members, or the active members numbers exactly half the number of current members, and one of them has majority. For instance, if the cluster has a current set of four members, any three of these four will have majority, but any two of them will only have majority if one of them has been assigned the majority property. If the cluster has a current set of two members, only the member with assigned majority, if any, has majority.

3. The number of databases in the current set of members is two, and our member is running, and it can reach the host of the other member and thus make sure that the other member is stopped rather than out of communication reach.

Incidentally, it is possible, from the outside, to assign a readonly server the right to commit transactions. Since this opens for the possibility to create inconsistencies within the databases of a cluster, it is an inherently unsafe operation that should only be performed by someone knowledgeable.

5.4.4 Clustering and Passwords

Cluster members establish connections to each other as the _SYSTEM user and are thus required to know the database and _SYSTEM passwords of the other members. Each cluster member assumes that these passwords are the same for the other members as for itself. This is almost always true. However, if not all the members in a cluster are up-to-date, it may not be true, and the necessary connections may not be established; in this case, it is necessary to provide copies of the most advanced cluster member for the other cluster members. This situation occurs because passwords were changed for one cluster member while not all cluster members were active; it may thus be concluded that it is unfortunate to change passwords if not all of the cluster members are active.

5.4.5 Clustering and Encryption

Cluster members do not have to be encrypted in the same way. This means that some members of a cluster may be encrypted and others are not, or the cluster members may be encrypted with different encryption keys. In all cases, the communication of transactions between cluster members are in clear SQL.

5.4.6 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.

5.4.7 Clustering a Database

Most of the operations for cluster handling/administration are non-trivial, and for the purposes of this description are best illustrated in terms of the commands implemented by the sql92 command line tool, which is part of the FrontBase installation. The full set of sql92 commands for dealing with clustering is described in section "Clustering Commands" on page 187.

All (well, most, anyway) clustering commands require that the so-called "default database" is established. The default database is an arbitrary member of a cluster (before cluster creation, it is an arbitrary database, soon to become the first member of a cluster). It serves merely as an access point to a ClusterDescr file and plays otherwise no special role in the cluster:

```
SET DEFAULT DATABASE <database>@<host>;
SHOW DEFAULT DATABASE;
```

The creation of a cluster should not be confused with the creation of a database, these are two distinct operations. Essentially, the creation of a cluster consists of adding a ClusterDescr file, containing only the name of the database itself, to an existing database. The fact that the command below may actually create the default database, and certainly attempts to also start a server for the default database, is for convenience only.

```
CREATE CLUSTER;
```

Once a cluster has come into existence, new members may be added. The essential part of the operation here is the addition of the relevant database name to the ClusterDescr files of the resulting cluster members. The database identification checks must succeed, and furthermore, the transaction number of the new cluster member must be less than or equal to the transaction number of any existing member of the cluster; in case of less than, it must be possible to bring the new member up-to-date from the transaction log of one of the existing members. One way of ensuring that all of these conditions are satisfied is to "initialize" the new member with a backup of sorts from an existing member. See the section "Backup and Restore" on page 85.

```
ADD MEMBER <database>@<host>;
```

It is often relevant to inspect the current composition (static as well as dynamic) of a cluster:

```
SHOW CLUSTER;
SHOW CLUSTER DESCRIPTOR;
SHOW CLUSTER ALL;
```

It is possible to start and stop an entire cluster in one go. Notice in particular, that the start command will examine the transaction numbers of the cluster members, and start them in the proper order (highest number first):

```
START CLUSTER;
STOP CLUSTER;
```

Likewise it is possible to start and stop individual members of a cluster. In this case, it is your responsibility to start members in the proper order.

```
START MEMBER <database>@<hostname>;
STOP MEMBER <database>@<hostname>;
```

It may be that the default majority handling is not completely satisfactory for all cluster setups; it is thus possible explicitly to assign majority to a particular cluster member:

```
SET MEMBER <database>@<hostname> MAJORITY TRUE;
```

Finally, in emergencies, it is possible to remove the readonly property from a cluster member that has (by accident!?) acquired it:

```
SET MEMBER <database>@<hostname> READONLY FALSE;
```

5.5 Backup and Restore

FrontBase allows you to export 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.

5.5.1 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 on page 85
- Backup of a Live Database on page 86
- Restoring a Database on page 86
- Export Into Flat-Files on page 87
- Replication on page 87

5.5.2 Copy the Database Files

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

FrontBase stores its database files in <FB home>/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> <FB home>/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.

5.5.3 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 BACKUP [TO <path-name-expr>] [ COMPRESSED ];
```

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. If COMPRESSED is specified, the backup is compressed to reduce its size.

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

5.5.4 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 <FB home>/FrontBase/Databases first.

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

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

If the backup has been written to a path other than the default location, the full path to the backup file must be specified:

```
FrontBase -create –restore=<path-name-expr> <database-name>
```

5.5.5 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. Export and Import 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 88.

5.5.6 Replication

The replication approach is fully described in the section “Replication” on page 76. 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 that handles the usual database traffic and one that 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.

5.6 Enhanced Flat-File Import and Export Functions

FrontBase provides the simple symbolic import/export functionality described in Export into Flat-Files on page 87. This section seeks to explain and demonstrate an enhanced implementation of Flat-File Export/Import which is more complete and flexible.

5.6.1 Enhanced Flat-File Export Function

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

```
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 that 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).

5.6.2 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:

```
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>
```

5.6.2.1 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.

5.6.2.2 RSEP - Row SEParator

<value> must be a string expression that 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'
```

5.6.2.3 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'
```

5.6.2.4 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'
```

5.6.2.5 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)
```

5.6.2.6 SKIP

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

```
Example:      SKIP = 2
```

5.6.2.7 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
```

5.6.2.8 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 well formed if CHECK is set to FALSE, e.g. to avoid duplicate PRIMARY KEYs.

```
Example:      CHECK = TRUE
```

5.6.2.9 RDQ - Remove Double Quotes

<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
```

5.6.2.10 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
```

5.6.2.11 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
```

5.6.2.12 POSITIONS

<value> is one of two types of a list that 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))

5.6.2.13 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.

5.6.2.14 <column-name>

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

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

5.6.3 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 separator 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")
        );
```

5.6.4 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 data file before attempting an import

5.7 Index Management

5.7.1 Indexing

```
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;
```

5.7.2 Strategies

FrontBase offers two strategies for maintaining indexes:

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.

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.

Consider a typical indexing setup with a word table, a document table and a relation table:

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

CREATE INDEX ON word(word);

CREATE TABLE document(
   document_pk INT PRIMARY KEY,        -- Implies an index
   document CLOB);

CREATE TABLE relation(
   word_fk INT,
   document_fk INT,
   PRIMARY KEY(word_fk, document_fk)); -- Implies an index

CREATE INDEX ON relation(document_fk, word_fk);

COMMIT;
```

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

```
SELECT document_fk
   FROM relation, word
  WHERE relation.wordf_k = word.word_pk AND
         word.word = '<some-word>';
```

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

```
SELECT word_fk
   FROM relation
   WHERE document_fk = <some-document-pk>;
```

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

```
SELECT document_fk
   FROM relation, word
   WHERE relation.word_fk = word.word_pk AND word.word = '<word_1>'
   INTERSECT
   SELECT document_fk
   FROM relation, word
   WHERE relation.word_fk = word.word_pk 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 indexes were set to PRESERVE TIME. The indexes 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.

5.7.3 Index Tuning

Indexes are subject to considerations about Storage Management and Database Optimization as described on page 97 and on page 101, respectively. Indexes are otherwise auto-managed, ie there is no need to drop and rebuild indexes for performance reasons.

5.8 SQL 99 Triggers

The concept of Triggers was introduced with the publication of SQL:1999 and are now (with FrontBase Release 4.2.7) supported by FrontBase. An excellent description of the concept may be found in “SQL:1999 Understanding Relational Language Components” by Jim Melton and Alan R. Simon (ISBN 1-55860-456-1).

The syntax for Triggers are as follows:

```
CREATE TRIGGER <trigger name>
    <trigger action time> <trigger event>
    ON <table name> [ ORDER <int expr> ]
    [ REFERENCING <old or new values alias list> ]
    <triggered action>
<trigger action time> ::= BEFORE | AFTER
<trigger event> ::=
    INSERT | DELETE | UPDATE [ OF <trigger column list> ]
<triggered action> ::=
    [ FOR EACH {ROW | STATEMENT} ]
    WHEN <cond expr> ]
    BEGIN <list of statements> END
<old or new values alias list> ::=
    {<old or new values alias>}+
<old or new values alias> ::=
    OLD [ROW] [AS] <old values correlation name> |
    NEW [ROW] [AS] <new values correlation name>
```

Please note that the syntax for OLD TABLE and NEW TABLE aliases is not supported.

In order for FrontBase to handle the full semantics of Triggers, the server must be started with the option '-triggers' (see FrontBase Invocation on page 198).

5.9 Storage Management

FrontBase 4 lets you precisely control where the various components of a table are stored. This feature allows you to optimize access to your data by taking into account the characteristics of the given deployment configuration. It is possible, for example, to make large amounts of data reside on several distinct disks, or you can exploit the availability of several disk drives with independent IO controllers to optimize the speed of data access. Oracle has a similar, although less modern, feature called "Table Space".

Storage management is done by means of FrontBase Advanced Device Management (FADM), the components of which are disk zones, partitions and devices. Disk zones are basic "storage entities" onto which particular storage components of a table may be mapped. A disk zone consists of one or more partitions, and each such partition reside on a given device. Note that a disk zone can comprise partitions residing on several distinct devices.

A device is a mechanism for storing data in "lumps" of a particular size. Such a lump is called an IO block, and within a device, IO blocks are numbered from 0 and consecutively up to the size of the device. In FrontBase, the size of an IO block is 512 bytes.

By default, a Frontbase database defines one device, named SYSTEM and with an access path equal to that of the database file, one partition, named SYSTEM and mapped onto the entire SYSTEM device, and one disk zone, named SYSTEM and comprising (only) the SYSTEM partition. Initially, all storage components of all tables are mapped onto the SYSTEM disk zone.

The storage components of a table are:

1) DATA
2) VARYING
3) INDEX
4) BLOB or CLOB (these are synonyms)

The DATA component holds all the fixed size structures associated with each row in a table.

The VARYING component holds all the varying length character strings stored in a table (that is all strings longer than 16 bytes).

The INDEX component holds all the storage associated with all the indexes defined on a table. This can be further refined so the storage associated with each individual index can be delegated to a particular disk zone.

The BLOB/CLOB component holds all the storage associated with all the BLOBs and CLOBs stored in the rows of a table.

The set of SQL commands that are available for FrontBase Advanced Device Management are described in the following sections.

5.9.1 Device Management

A FrontBase device corresponds to a storage device accessible through the operating system (OS) of the machine hosting a FrontBase database. A FrontBase device has a *name* and a *size* which is the number of IO blocks that the device contains; the size indicates the maximum number of IO blocks that may be allocated to partitions. A device is furthermore identified by an *access path*, which (in the underlying OS) is interpreted like any other file access path. Device names reside in their own name space, ie cannot collide with eg the names of partitions or disk zones.

New devices may be added to a database by specifying their name and an access path and possibly a maximum size. If the size is not given, FrontBase derives the maximum size from the file system of the underlying OS:

```
ADD DEVICE <device name> PATH ‘<device access path>’ [SIZE
<number>];
```

<device access path> is any file access path and <number> specifies the number of IO blocks, eg

```
ADD DEVICE Blobs PATH '/Library/FrontBase/Databases/Blobs' SIZE
4000000;
```

This implies that the size of Blobs is about 2GB.

Existing devices may be changed which means that their size may be changed (unless creating a conflict with partitions mapped onto the device):

```
ALTER DEVICE <device name> SIZE <number>;
```

An existing device may be dropped (if it is not in use):

```
DROP DEVICE <device name>;
```

Devices defined by a database may be inspected:

```
SELECT * FROM INFORMATION_SCHEMA.DEVICES;
```

5.9.2 Partition Management

A partition is a region of a device, characterized by having a base address (an IO block offset from the beginning of the device), and a size (measured in IO blocks). The entire partition must live inside the total size of the device. Partitions have names (with their own name space). A device may hold several partitions, but such partitions may not overlap.

New partitions may be created:

```
CREATE PARTITION <partition name> ON DEVICE <device name> BASE
<number> SIZE <number>;
```

Existing partitions may have their base address or size changed (if this does not create inconsistencies):

```
ALTER PARTITION <partition name> [BASE <number>] [SIZE <number>];
```

Partitions may be dropped (if they are not in use):

```
DROP PARTITION <partition name>;
```

Finally, existing partitions may be inspected:

```
SELECT * FROM INFORMATION_SCHEMA.PARTITIONS;
```

5.9.3 Disk Zone Management

A disk zone has a list of one or more partitions in which data may be stored. Disk zones have names (with their own name space).

New disk zones may be created:

```
CREATE DISK ZONE <disk zone name> [WITH PARTITION <list of partition
names>];
```

No data may be allocated in a disk zone before it is associated with at least one partition. If a disk zone is associated with more than one partition, data are attempted allocated in partitions in the order in which the partitions are specified.

Disk zones may be changed by adding to or removing partitions from them:

```
ALTER DISK ZONE <disk zone name> ADD PARTITION <list of partition
names>;
ALTER DISK ZONE <disk zone name> DROP PARTITION <list of partition
names>;
```

Any data specified to reside in a disk zone and actually stored in a partition which is dropped from that disk zone, will be moved to a partition remaining as a member of that disk zone. That data move will only happen when the disk zone change is committed.

Disk zones may be dropped (if they are not in use):

```
DROP DISK ZONE <disk zone name>;
```

Finally, existing disk zones may be inspected:

```
SELECT * FROM INFORMATION_SCHEMA.DISK_ZONES;
SELECT * FROM INFORMATION_SCHEMA.DISK_ZONE_PARTITIONS;
```

5.9.4 Table Storage Component Management

The interesting part of disk zones, partitions and devices is to actually use them for storing data. As mentioned above, FrontBase identifies four types of storage component associated with a table (BLOB and CLOB are synonyms). It is now possible to specify onto which disk zone(s) the storage components of a table should be mapped.

```
ALTER TABLE <table name> SET DISK ZONE <disk zone name> | DEFAULT
[FOR <list of storage components>];
<storage component> ::= DATA | VARYING | INDEX | BLOB | CLOB
```

The DEFAULT disk zone is a synonym for the predefined SYSTEM disk zone. If no <storage component> is specified, the new disk zone applies for all storage components associated with the identified table.

It is also possible for individual indexes of a table to be mapped onto a disk zone.

```
ALTER INDEX <index name> SET DISK ZONE <disk zone name> | DEFAULT;
```

The syntax for a LookSee index is slightly different:

```
ALTER TABLE <table name> UPDATE INDEX <LookSee index name> DISK ZONE
<disk zone name> | DEFAULT;
```

Notice that in all cases, if a specified storage component contains any data that should be moved to a new disk zone, actual data movement will only take place when these SQL statements are committed.

5.10 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).

5.10.1 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.

5.10.2 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.

5.10.2.1 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 that 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 indexes for a table are cached, either in a cache shared by all indexes 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.

5.10.3 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 103
- Table Caching on page 104
- Raw Device Driver (RDD) on page 111

5.10.4 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 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.

5.10.5 Table Caching

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

There are a number of factors which can be detrimental to the performance of a database server, but broadly speaking, 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 that 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 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 number of rows to be cached.
- Persistent, keep the cache across transactions.
- Preload, cache the table on server start.

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.

5.10.5.1 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:

Row Cache	the static part of rows.

String Cache	variable length strings that aren't stored in the static part of a row.
Index Caches	most indexes can be cached individually.

5.10.5.2 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;
```

The cache will get pre-loaded to its maximum size during server start-up.

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.

5.10.5.3 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 given as absolute row counts.

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> = 100 and they will apply if PRESERVE TRUE is specified, but no explicit setting of the row cache is given.

5.10.5.4 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 data type:

CHARACTER	If the maximum size given in the data type specification is less than or equal to 64, inserted string values will be stored in the static part of the row. If the maximum size is larger than 64, inserted string values are stored indirectly in the string table.
VARCHAR	If the maximum size given in the data type specification is less than or equal to 32, inserted string values will be stored in the static part of the row. If the maximum size is larger than 32, 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.

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 is given.

5.10.5.5 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	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	efficient for larger tables. The overhead depends on the data types 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 512 bytes);

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.

5.10.5.6 Frequently Asked Questions

Which cache will store BLOB and CLOB values?
Such values are typically large and are as such not cached in a table cache, but are either streamed directly from the disk subsystem or the so-called RDD cache (described in Raw Device Driver (RDD) on page 111).

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

5.10.5.7 Performance Indicators

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

```
SHOW CACHES [<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:

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.
Type	Row caches are marked with an R, dope caches with a D, space optimized index caches with an S and time optimized index caches with a T.
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 currently used for the cache.
Max Items	The maximum number of items that the cache will hold. This number is calculated from the total number of items in the structure, under the specified cache settings.
Byte Size	The estimated number of bytes that a completely filled cache will use.
Lookups	The number of times an item was accessed.
Hits	The number of lookups where the item was found in the cache.
Hit Rate	The relative number of lookups that was found in the cache and thus did not cause one or more read operations. A lookup in the cache that loaded an item into a cache that 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 times will be stored only once. Refs gives the 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. The larger the number, the more advantage is gained from string normalization. If the ratio is close to 1, it may be worth considering turning off the string normalization mechanism.

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.

5.10.6 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 file system. 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 data store. Additionally, RDD allows you to specify the size of its combined write-through and read cache.

5.10.6.1 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. FrontBaseManager lets you specify RDD settings with its "Start Advanced" command. 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:

Pages Used	The number of pages currently in use
Pages Free	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 from the cache
Read Waits	The number of reads that had to wait for a block to have been read off the disk
Read Stalls	The number of reads that had to wait for an RDD 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 an RDD 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.

5.10.7 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 that 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 lookups 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.

5.10.8 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 enhances the performance and reliability of FrontBase as the overhead of the file system is avoided and finer control over the disk is achieved.

5.10.8.1 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 a partition on the disk, use 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.

5.10.8.2 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 that 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.

5.10.9 Memory Usage

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

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

The following headings are listed:

Name	There are a number of distinct memory zones: A fixed set of system related zones (currently up to 7), a zone for each persistent table, and a zone for each client connection (Agent)
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 (in bytes) 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.

5.10.10 Database Optimization

In FrontBase, the (smallest) unit of storing data is an IO block (see Storage Management on page 97). All storage components of tables, etc, are stored in IO blocks. Optimally, the data of a storage component is stored in consecutive IO blocks with as low addresses as possible. This is optimal both in the sense of the time taken to read the data off the disk and in the sense of using no more disk space than necessary.

Over time, updates of tables may induce less optimal layout of the data associated with the table (fragmentation). This occurs in two senses:

1) The data of a particular storage component is scattered over the disk zone in which it resides. This leads to slower access to non-cached table data (including re-loading the caches when the server starts).

2) A disk zone contains many free IO blocks, ie. the data in the disk zone is not densely allocated in a partition. This leads to consumption of more disk space than strictly necessary..

Attempts to repair fragmentation in the first sense may actually lead to more fragmentation in the second sense.

FrontBase offers two SQL statements for improving the situation when fragmentation in the first sense has become a problem.

```
OPTIMIZE DATABASE;
OPTIMIZE DISK ZONE <disk zone name> | DEFAULT;
```

The first optimizes all disk zones defined by the database, and the second optimizes only the identified disk zone.

It should be noted that the very optimal layout of an entire database is to be found immediately after a restore of an up-to-date backup of the database.

5.11 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 116
- MySQL on page 117
- Other import mechanisms are available via the enhanced import facility Enhanced Flat-File Import and Export Functions on page 88

5.11.1 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 Mac OS X platform. The tool is available from the Download section of the FrontBase website.

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

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.
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 Mac OS and Windows versions of FileMaker. In Mac OS, you may not be able just to double-click the script file. Instead, start FileMaker and open it using the "Open..." menu item.
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.

5.11.2 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: <database-name>@<host-name>.
Destination	The destination field is the desired location and name of the FrontBase database. There has to be a running version of FrontBase 2.0 on the specified host. Input must have the following format: <database-name>@<host-name>.
Username	The username needed to log on to the MySQL database.
Password	The password for the MySQL user name.

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

5.12 Troubleshooting

5.12.1 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.

SQL logging may be turned on when starting the server:

```
<FB home>/FrontBase/bin/FrontBase -logSQL [<other-options>]
<database-name> &
```

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 has first been turned on 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.

5.12.2 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.

These instrumentation comments are found in a FrontBase SQL log:

--0 <connection id><timestamp><encryption key id>
Connection created

--1 <connection id><session id><timestamp> "<user name>" "<session name>"
Session created

--2 <connection id><session id><timestamp><no of bytes in the following SQL statement>
Execute SQL

--3 <connection id><session id><timestamp>
Session terminated

--4 <connection id><session id><timestamp><no of bytes in the following SQL statement>
Execute SQL, one shot auto COMMIT

--5 <connection id> <timestamp>
Connection terminated

--6 <connection id><session id><timestamp> "<result OK> <error count>"
End of SQL execution

--7 <connection id><session id><timestamp>
Start of result set fetch

--8 <connection id><session id><timestamp> "<number of rows returned>"
End of result set fetch

--9 <connection id><session id><timestamp><exception>

Exception

--A <connection id><session id><timestamp>
Auto COMMIT OK

--B <connection id><session id><timestamp>
Auto COMMIT failed

--C <connection id><session id><timestamp><statement handle>
PREPARE statement

--D <connection id><session id><timestamp><length>
Execute SQL, return batch

--E <connection id><session id><timestamp><length>
Execute SQL, one shot auto COMMMIT, return batch

5.12.3 Location of the SQL Log File

The log file will per default get created as

```
<FB home>/FrontBase/Databases/<database-name>.fb.sql
```

This location can be changed by creating a directory named LogFiles in the FrontBase installation directory:

```
mkdir <FB home>/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.

5.12.4 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;
```

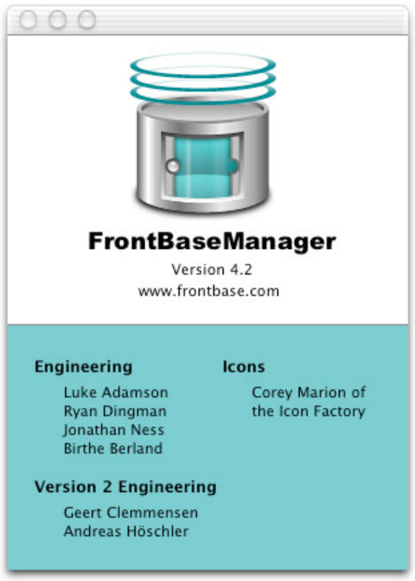
The name of the renamed log file will stay the same except that a timestamp is added as a suffix:

```
<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.

6 FrontBaseManager

FrontBaseManager is a Mac OS X application that lets you start, stop, monitor, create, remove, and generally manage every aspect of your FrontBase database on the Mac OS. 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
Version 4.2
www.frontbase.com

Engineering
Luke Adamson
Ryan Dingman
Jonathan Ness
Birthe Berland

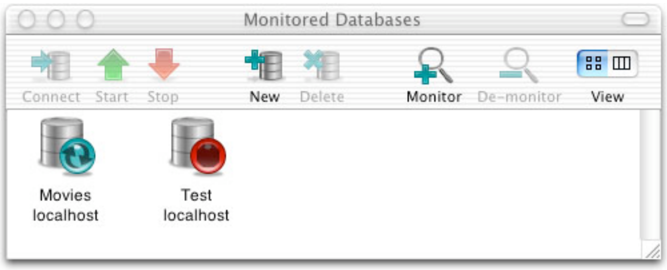
Icons
Corey Marion of the Icon Factory

Version 2 Engineering
Geert Clemmensen
Andreas Höschler

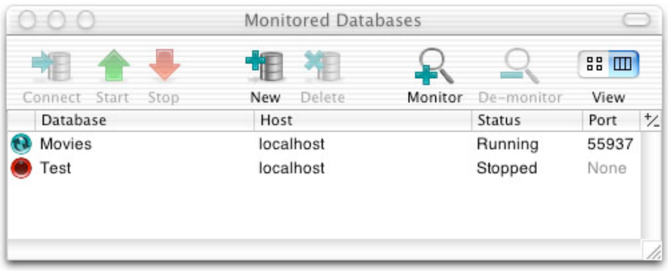
This chapter has the following sections:

6.1 Monitoring and Managing Databases

FrontBaseManager lets you monitor and control the state of FrontBase servers, i.e. whether they are running or stopped. When the FrontBaseManager is started, the following Monitored Databases window appears:



This window 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:



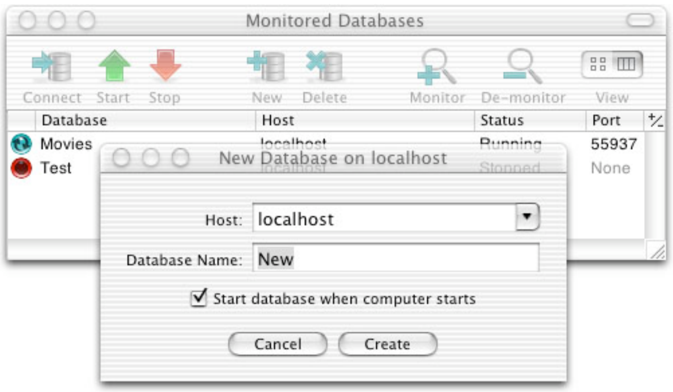
Database	Host	Status	Port
Movies	localhost	Running	55937
Test	localhost	Stopped	None

FrontBaseManager will remember whether you last used Icon View or List View.

6.1.1 Creating Databases

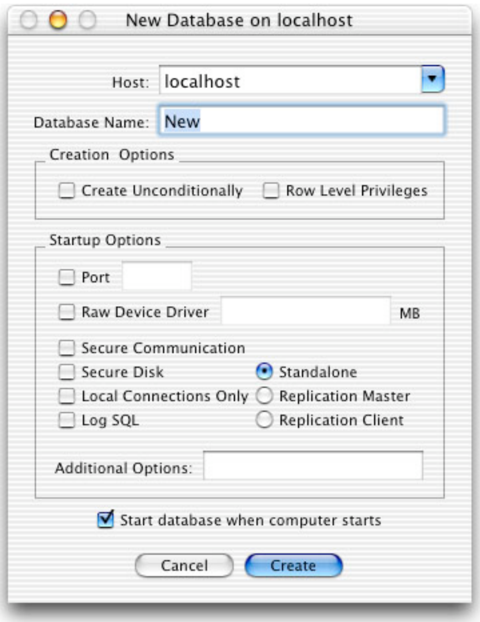
The New and Delete buttons will allow you to create a new database or delete an existing, selected database. The Delete button will not be enabled unless you have selected a stopped database. When you click Delete, FrontBaseManager will ask you to confirm that you want to delete the database.

When you click New, FrontBaseManager will present you with a dialog box asking which host you want to create the database on and what to name the database. That window looks like this:



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 197.



Host: localhost
Database Name: New
Creation Options: [ ] Create Unconditionally [ ] Row Level Privileges
Startup Options:
[ ] 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

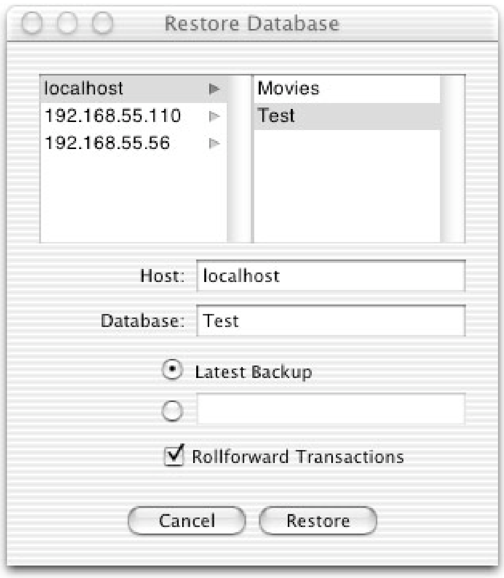
A few options are discussed here:

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 that allows you to assign privileges, like on the files in a Unix file system, 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 224 for more information on how to use this feature.
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.

6.1.2 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 it 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*.

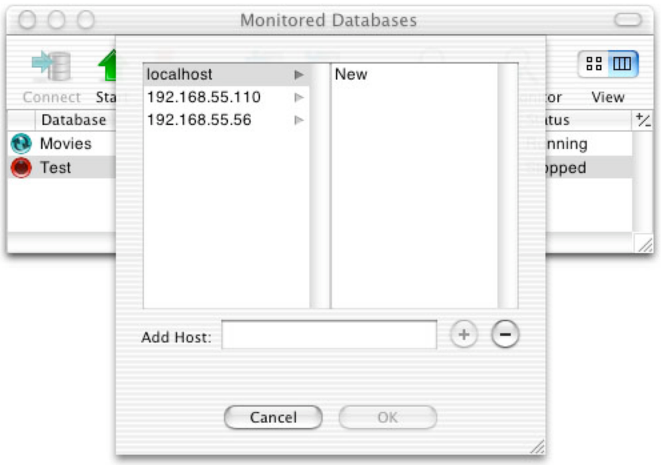


Host: localhost
Database: Test
[✓ Latest Backup]
[✓ Rollforward Transactions]

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 text field. Be sure the Rollforward Transactions checkbox is checked if you would 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.

6.1.3 Monitoring

The FrontBaseManager only allows you to actively manage those databases that it is currently monitoring. So, the database icons or list items that you see are only those databases it is monitoring. To add existing databases to the monitor view, click the Monitor button on the toolbar or choose *Database→Monitor* from the menu. The following panel appears:

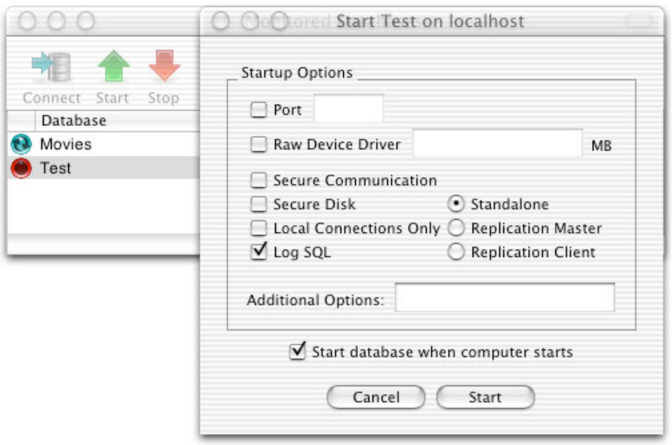


The left column of the browser shows all currently monitored hosts. The right column shows all existing but not yet monitored databases on the selected host. To monitor databases on a not yet shown host, simply enter the host name in the text field below the browser and press Return. To add a database 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 De-monitor or by selecting it and choosing *Database*➔*Stop Monitoring*.

This panel also allows you to remove a monitored host from the host list. When you have selected a host, just click the Minus button.

6.1.4 Starting 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 *Database*➔*Start Advanced* brings up a panel allowing you to specify options before actually starting the database.



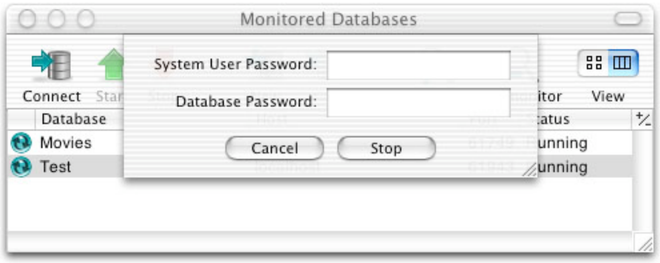
[ ] Port
[ ] Raw Device Driver MB
[ ] Secure Communication
[ ] Secure Disk
[ ] Local Connections Only
[✓] Log SQL
[✓ Standalone] Replication Master Replication Client
Additional Options:
[✓] Start database when computer starts

You can select the start options on this window. The options are explained below:

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.

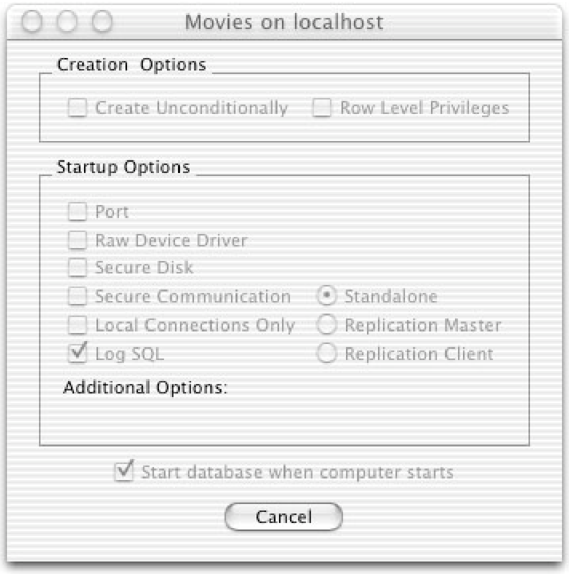
6.1.5 Stopping Databases

To stop a monitored database that is running, select it in the monitor view and click Stop or choose *Database→Stop*. Stopping a database needs to be done by the user _system. If this user has a password and/or there is a database password you will be asked to provide the password(s).



6.1.6 Showing Database Start Options

You can see the options that were used last time a database was started by choosing *Database→Show Options*:


Movies on localhost

Creation Options
- [ ] Create Unconditionally
- [ ] Row Level Privileges

Startup Options
- [ ] Port
- [ ] Raw Device Driver
- [ ] Secure Disk
- [ ] Secure Communication
- [ ] Local Connections Only
- [✓] Log SQL

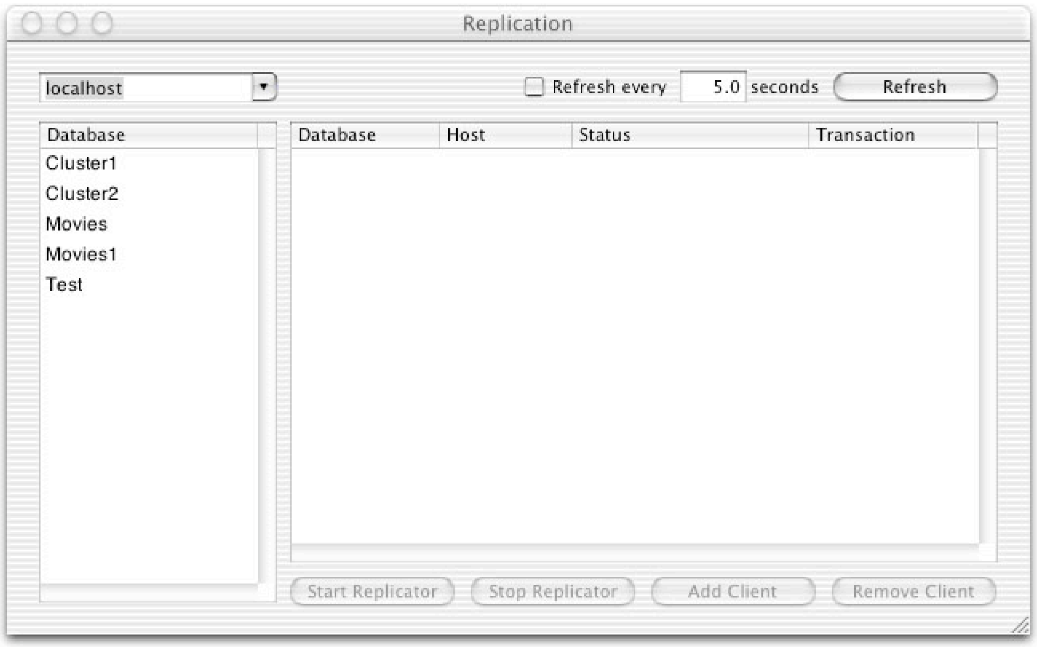
[✓ Standalone] Replication Master Replication Client

Additional Options:

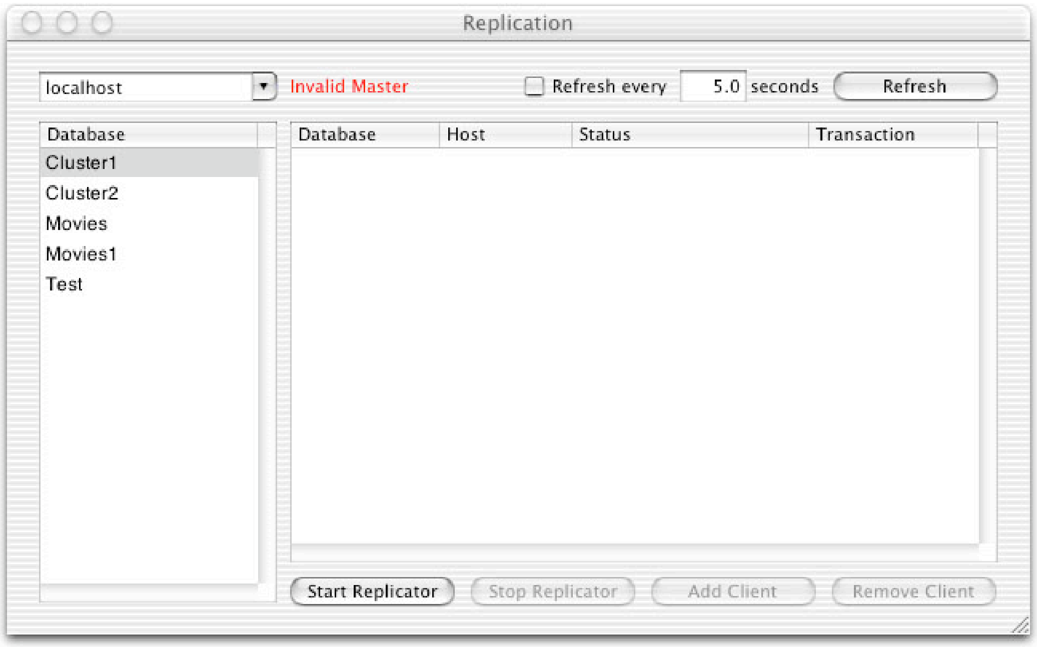
- [✓] Start database when computer starts

6.1.7 Replication Management

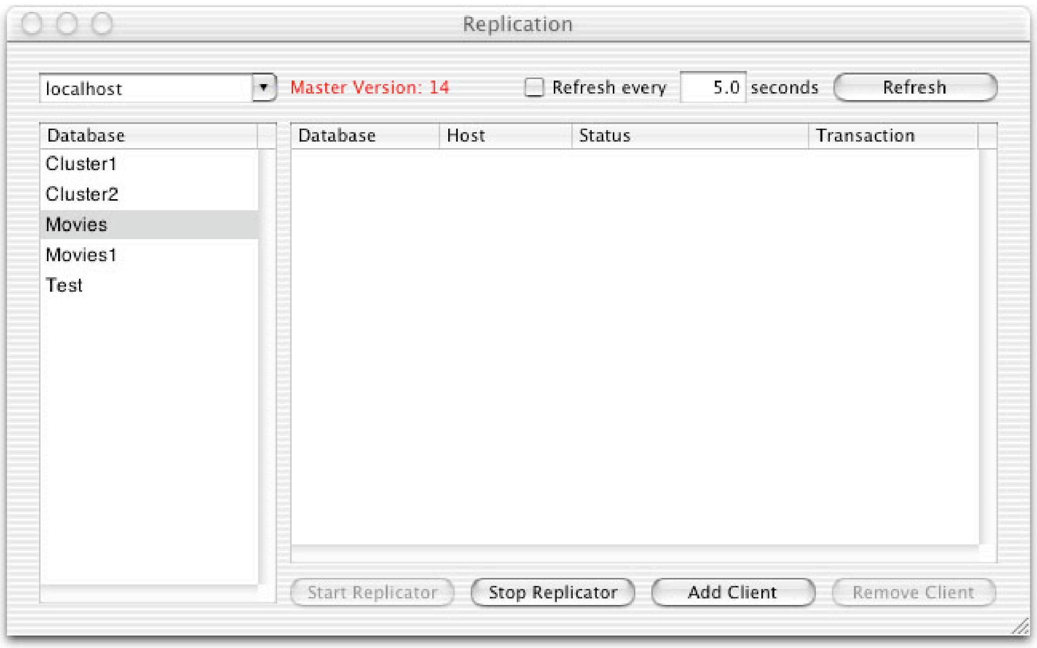
FrontBaseManager allows you to set up and manage a replication setup of databases. To open the Replication Management pane, choose *Tools→Replication Management*. Just as always when you select a host you will see the list of known databases on that host, in this case localhost has been chosen.



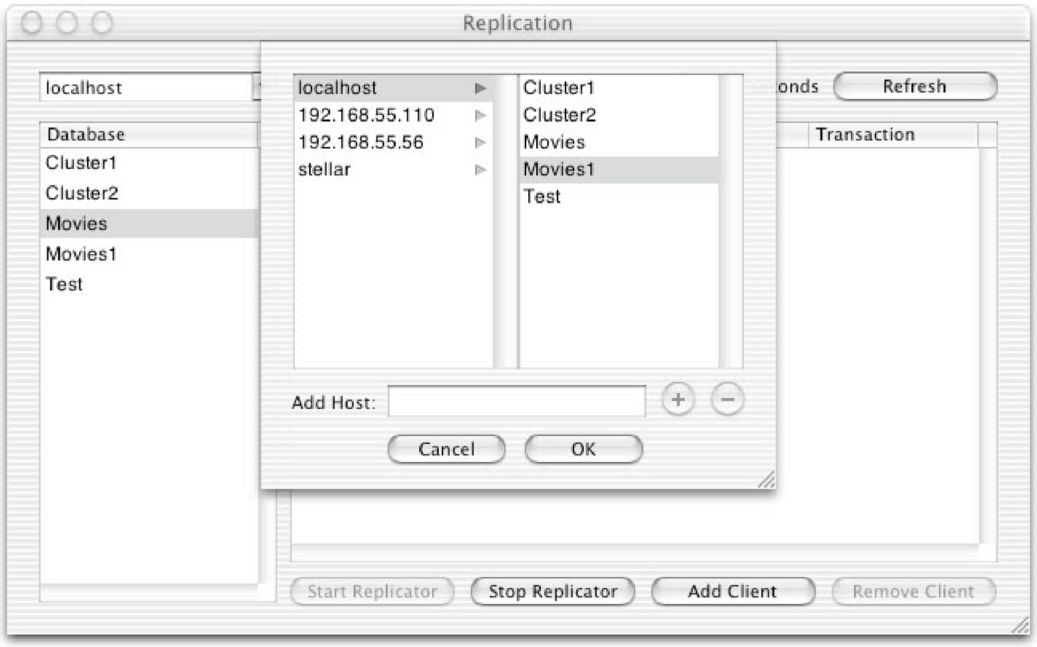
If you select a database that is not a replication master a message is displayed.

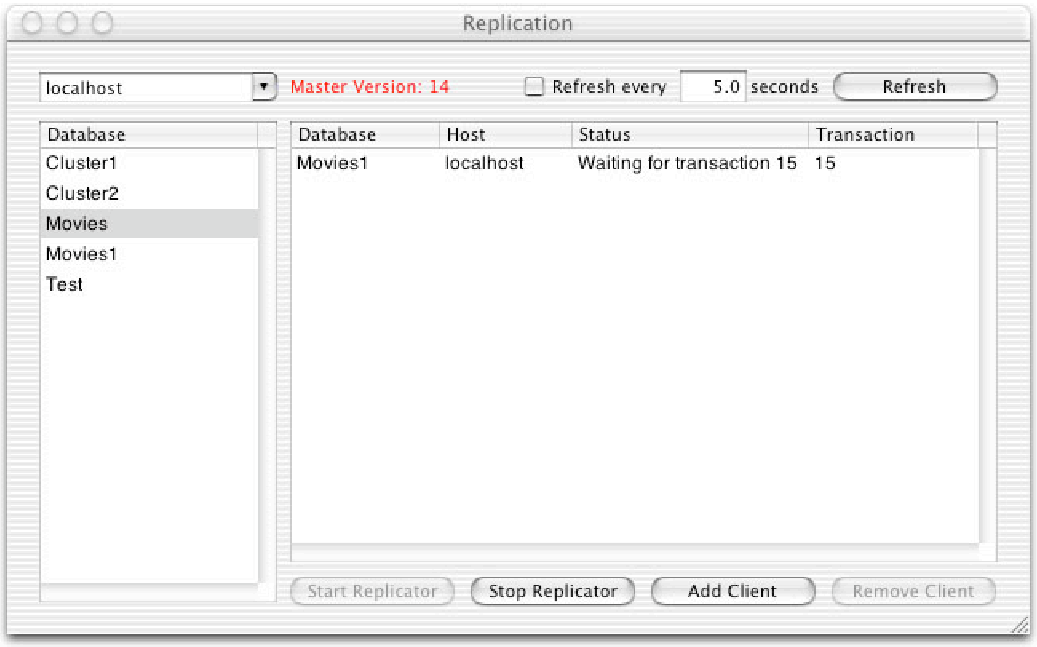


To start the replicator, you just click the Start Replicator button. This will set up the database as a replication master. You may need to make a refresh to see the result.



You can now stop the replicator or add clients. To add a client, just click the Add Client button.





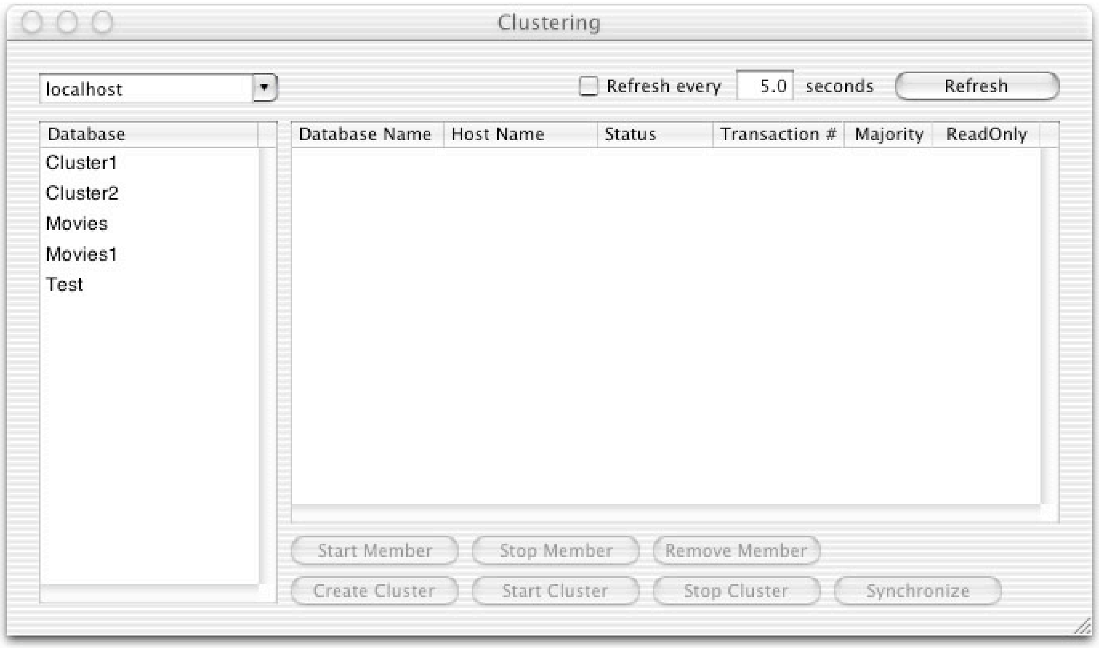
Database
Cluster1
Cluster2
Movies
Movies1
Test

Database	Host	Status	Transaction
Movies1	localhost	Waiting for transaction 15	15

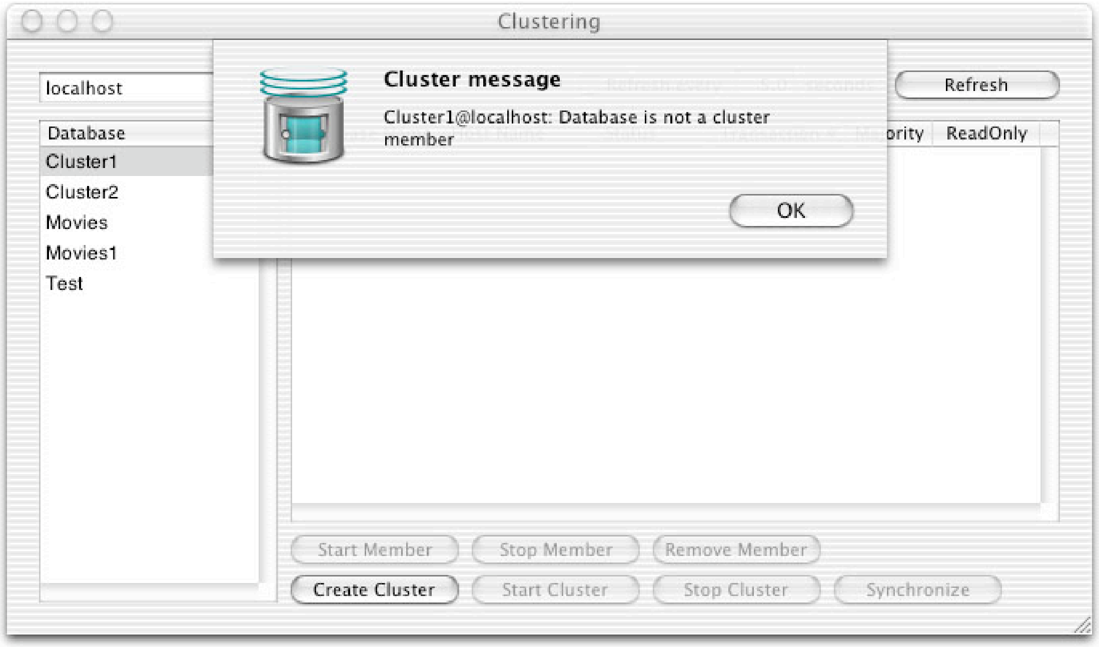
By selecting a client you can remove it.

6.1.8 Cluster Management

FrontBaseManager allows you to set up and manage a cluster of databases. To open the Cluster Management pane, choose *Tools*➔*Cluster Management*. Just as always when you select a host you will see the list of known databases on that host, in this case localhost has been chosen.



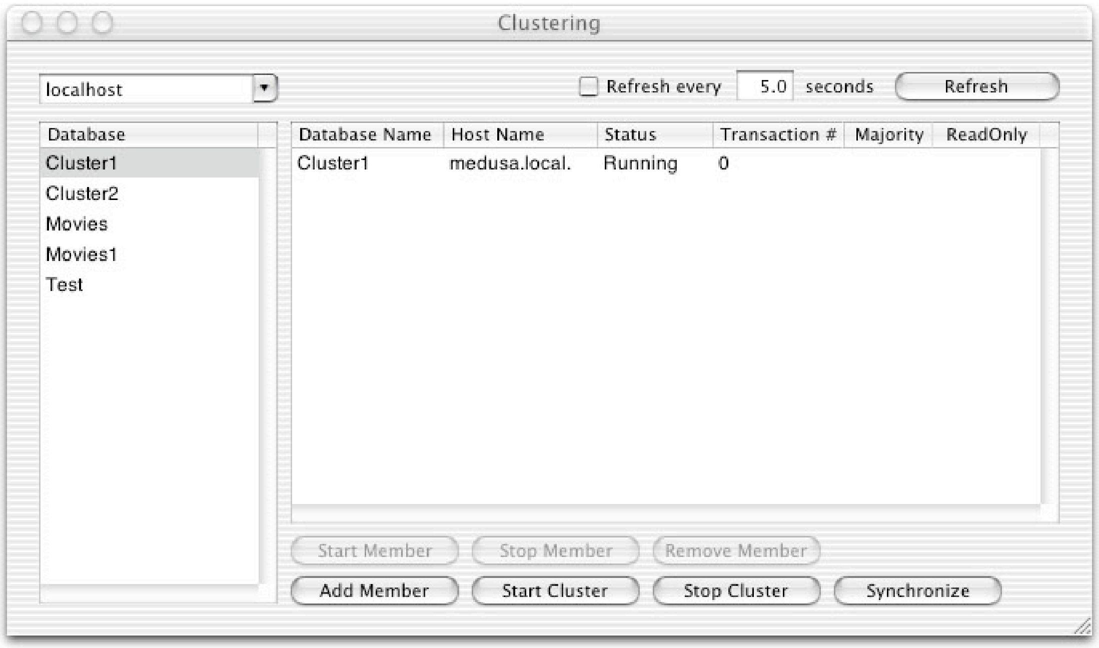
If you select a database that is not a member of a cluster a message is displayed.



Cluster message

Cluster1@localhost: Database is not a cluster member

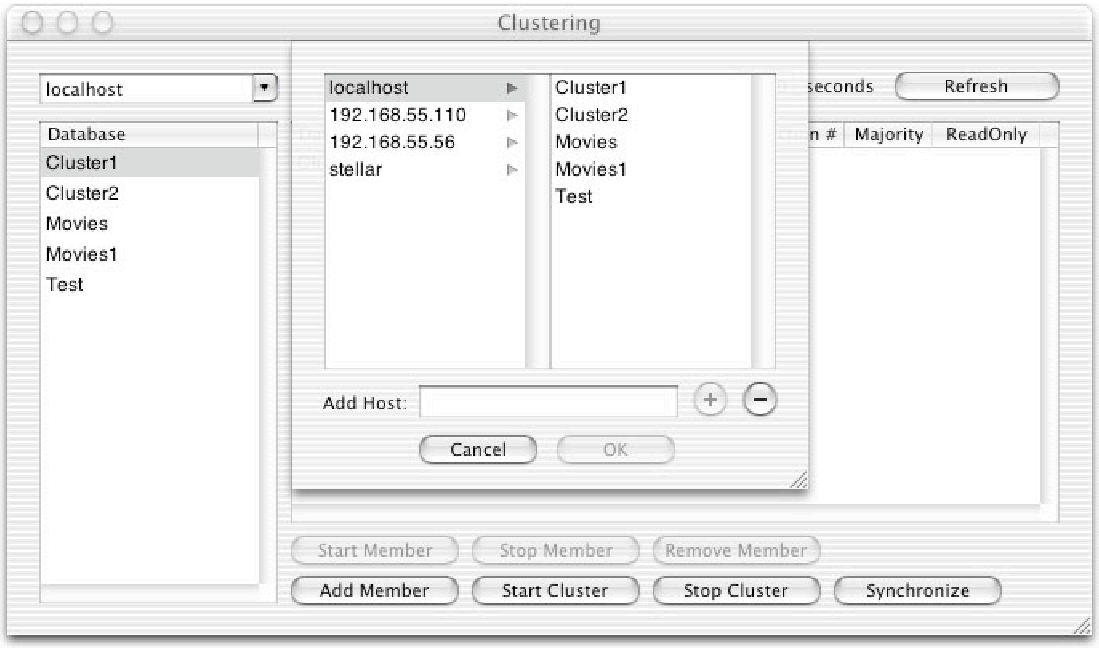
To create a cluster, you just click the Create Cluster button. This will set up the cluster and start the database if it is not running.

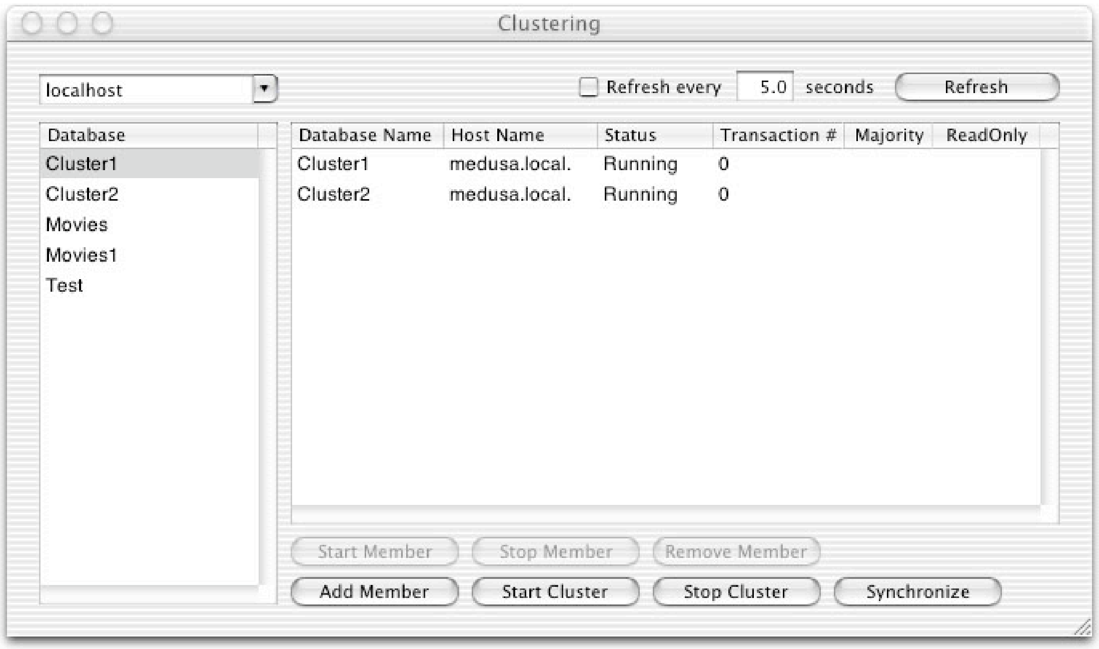


Database
Cluster1
Cluster2
Movies
Movies1
Test

Database Name	Host Name	Status	Transaction #	Majority	ReadOnly
Cluster1	medusa.local.	Running	0		

You can now add another member to the cluster by clicking the Add Member button.





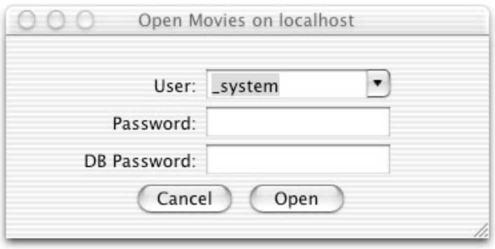
Database
Cluster1
Cluster2
Movies
Movies1
Test

Database Name	Host Name	Status	Transaction #	Majority	ReadOnly
Cluster1	medusa.local.	Running	0		
Cluster2	medusa.local.	Running	0		

By selecting one of the members you are able to start, stop, or remove the member.

6.1.9 Connecting to the Database

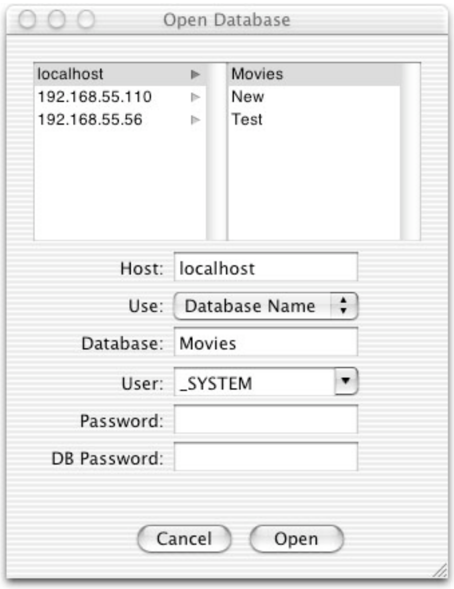
Once you have a running database, you can connect to the database either by selecting the database and clicking the Connect button, or by simply double-clicking on the database icon. This will bring up the following dialog box:



User: _system
Password:
DB Password:

This dialog box allows you to 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. It also allows you to specify the user password and the database password, if any.

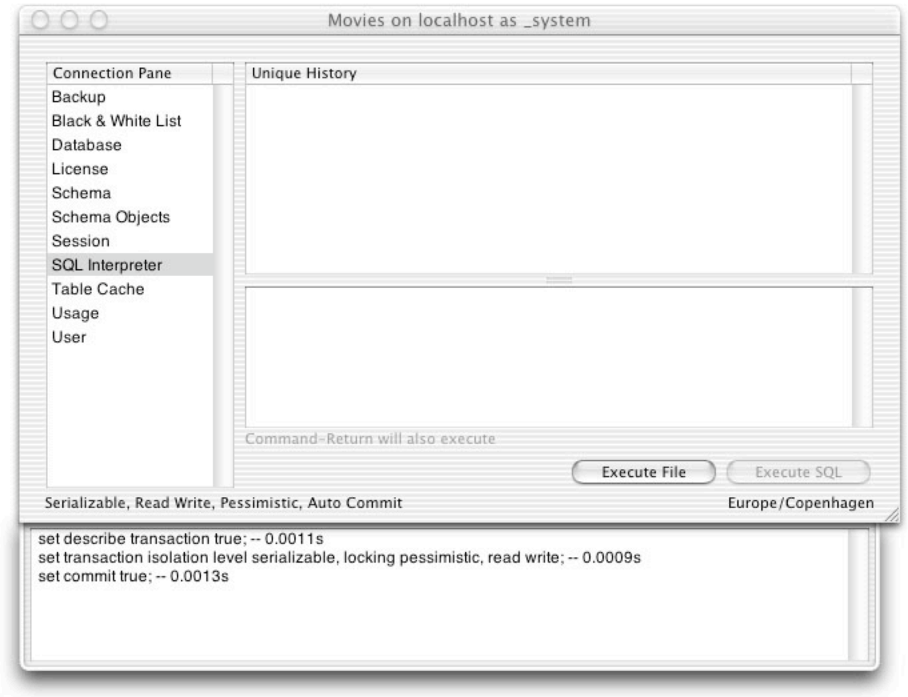
If you want to connect to a database that is not in the monitor window you can do so by choosing *File→Open Database*. You will then get the following dialog box:



Host: localhost
Use: Database Name
Database: Movies
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.

6.2 Connection Window



```
set describe transaction true; -- 0.0011s
set transaction isolation level serializable, locking pessimistic, read write; -- 0.0009s
set commit true; -- 0.0013s
```

The Connection Window is FrontBaseManager's main window. This pane gives you access to most of the functionality that FrontBaseManager 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.

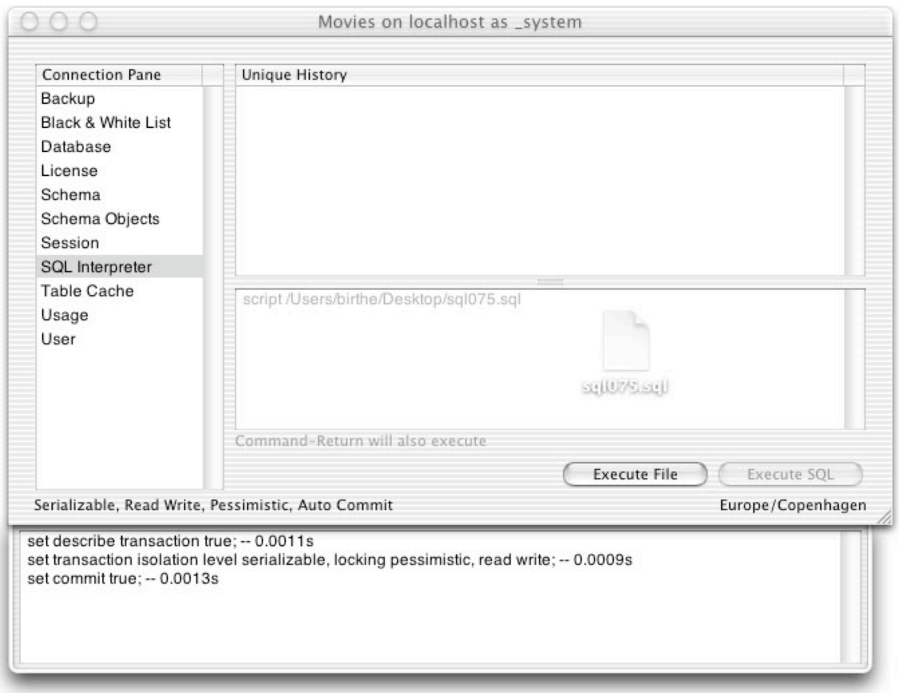
The Connection window title bar contains the name of the database, the host on which it is running, and the connection's user. At the bottom-left corner of the connection pane you can see the transaction settings, always present to remind you of your current transaction settings. These transaction settings can be changed through the *Session* pane. 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*.)

6.2.1 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 the 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 <file-name>" will appear in the text area. Then, when you drop the file into the text area, "script <filename>" will actually be executed.



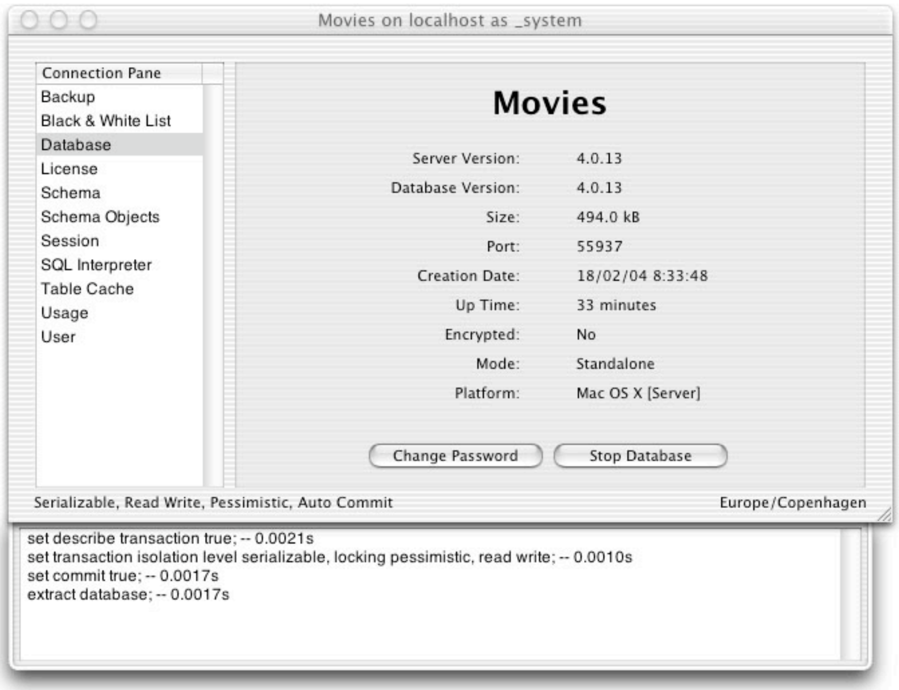
```
script /Users/birthe/Desktop/sql075.sql
```

```
set describe transaction true; -- 0.0011s
set transaction isolation level serializable, locking pessimistic, read write; -- 0.0009s
set commit true; -- 0.0013s
```

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 will 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.

6.2.2 Database



Movies

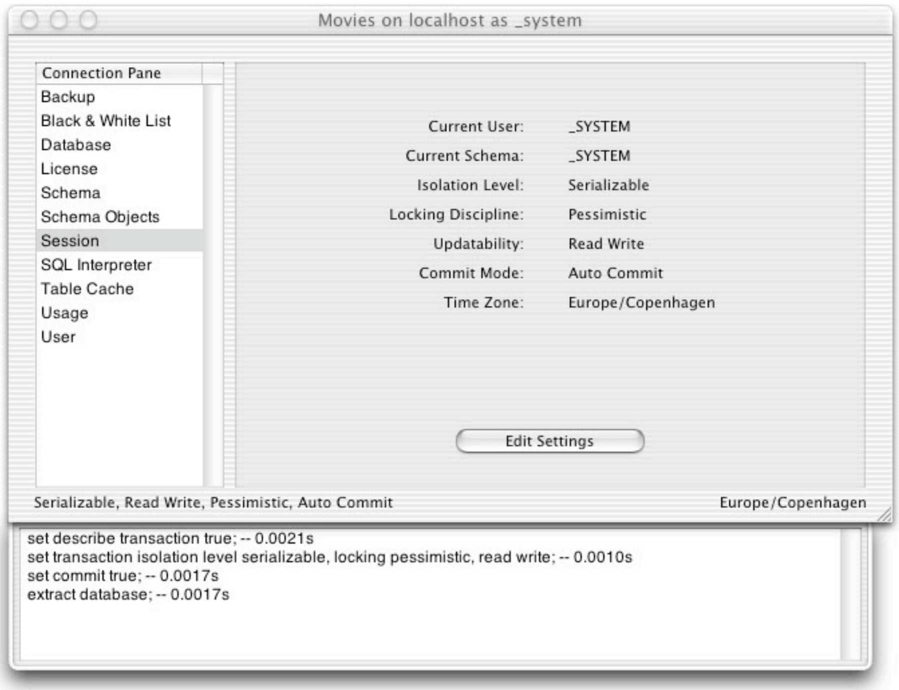
Server Version:	4.0.13
Database Version:	4.0.13
Size:	494.0 kB
Port:	55937
Creation Date:	18/02/04 8:33:48
Up Time:	33 minutes
Encrypted:	No
Mode:	Standalone
Platform:	Mac OS X [Server]

Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

```
set describe transaction true; -- 0.0021s
set transaction isolation level serializable, locking pessimistic, read write; -- 0.0010s
set commit true; -- 0.0017s
extract database; -- 0.0017s
```

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.

6.2.3 Session



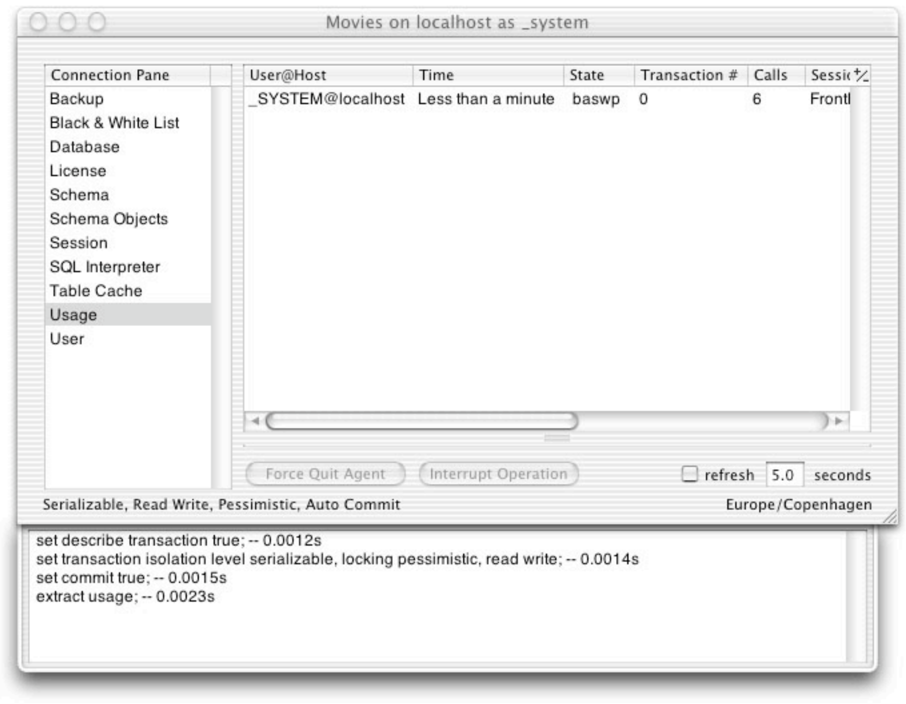
Current User:	_SYSTEM
Current Schema:	_SYSTEM
Isolation Level:	Serializable
Locking Discipline:	Pessimistic
Updatability:	Read Write
Commit Mode:	Auto Commit
Time Zone:	Europe/Copenhagen

Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

```
set describe transaction true; -- 0.0021s
set transaction isolation level serializable, locking pessimistic, read write; -- 0.0010s
set commit true; -- 0.0017s
extract database; -- 0.0017s
```

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 that allows you to edit your current schema or any of your transaction settings.

6.2.4 Usage

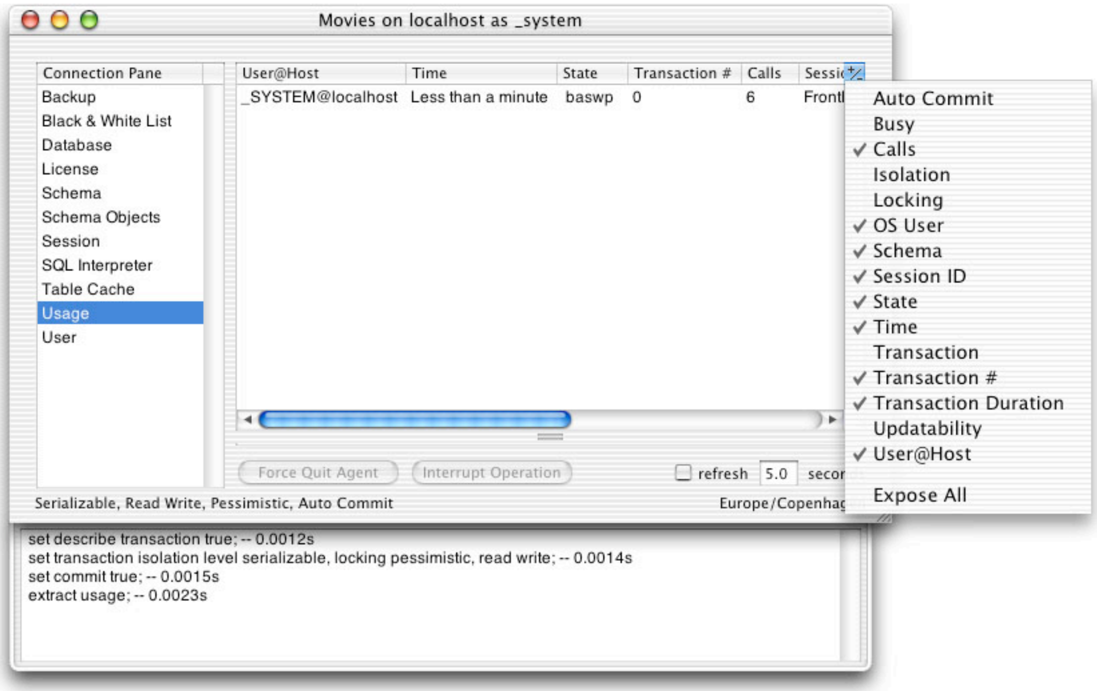


User@Host	Time	State	Transaction #	Calls	Sessi
_SYSTEM@localhost	Less than a minute	baswp	0	6	Frontl

Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

```
set describe transaction true; -- 0.0012s
set transaction isolation level serializable, locking pessimistic, read write; -- 0.0014s
set commit true; -- 0.0015s
extract usage; -- 0.0023s
```

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. The settings are saved so that you will see the same information the next time you show the usage panel, and in the same order.

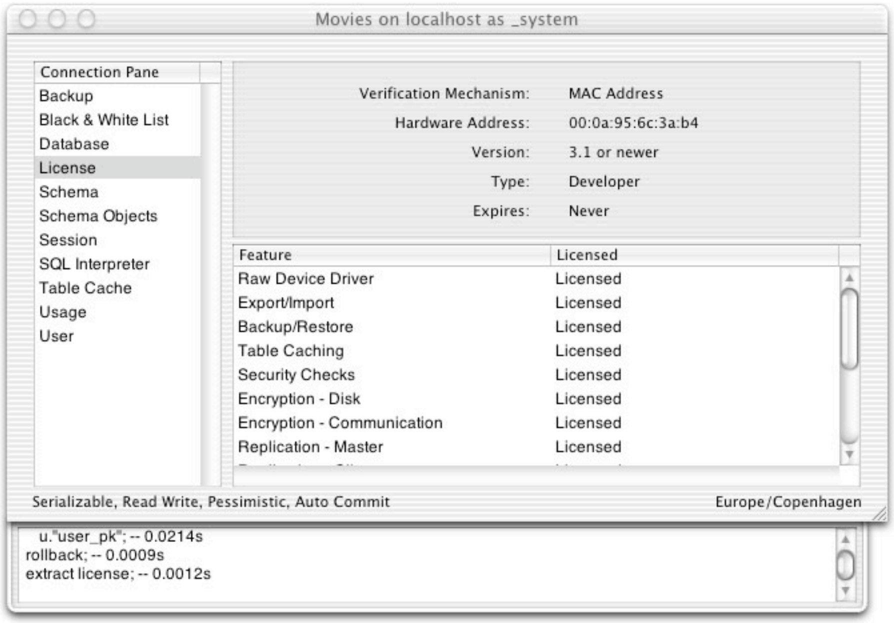


User@Host	Time	State	Transaction #	Calls	Sessic
_SYSTEM@localhost	Less than a minute	baswp	0	6	Frontl

Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhag

```
set describe transaction true; -- 0.0012s
set transaction isolation level serializable, locking pessimistic, read write; -- 0.0014s
set commit true; -- 0.0015s
extract usage; -- 0.0023s
```

6.2.5 License



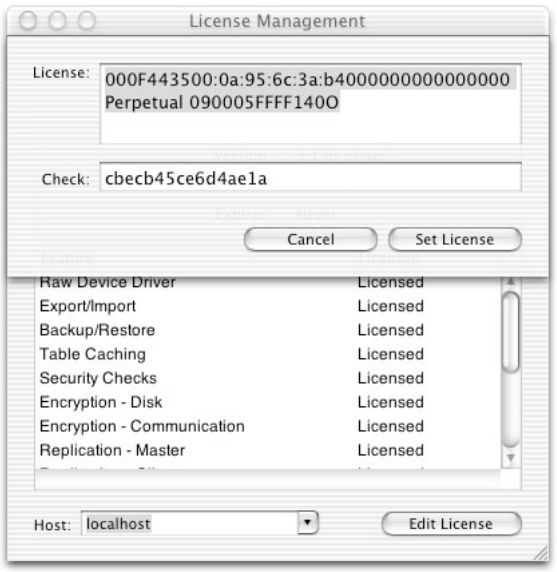
Verification Mechanism:	MAC Address
Hardware Address:	00:0a:95:6c:3a:b4
Version:	3.1 or newer
Type:	Developer
Expires:	Never

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/Copenhagen

```
  u."user_pk"; -- 0.0214s
rollback; -- 0.0009s
extract license; -- 0.0012s
```

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: 000F443500:0a:95:6c:3a:b4000000000000000 Perpetual 090005FFFF140O

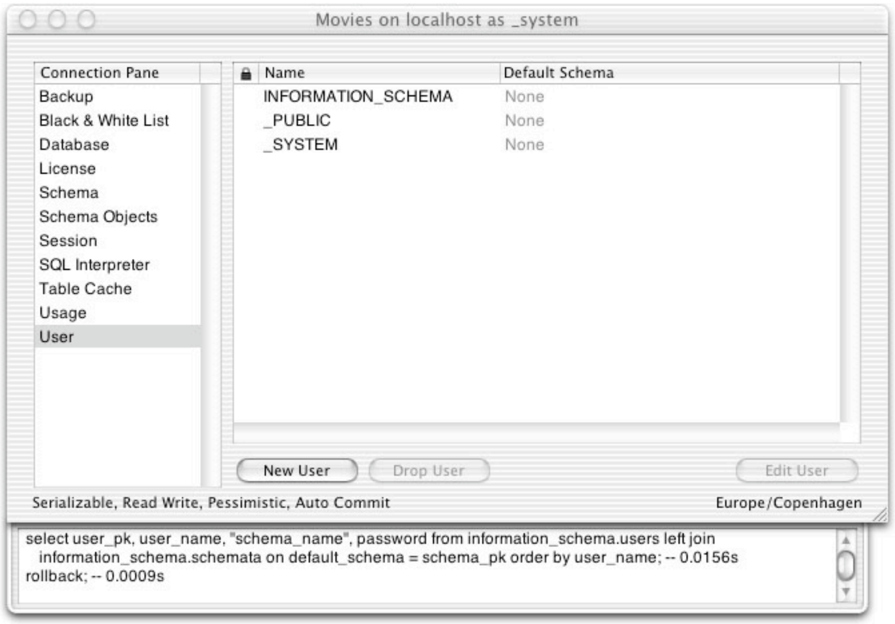
Check: cbecb45ce6d4ae1a

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!

6.2.6 User



Name	Default Schema
INFORMATION_SCHEMA	None
_PUBLIC	None
_SYSTEM	None

Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

```
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; -- 0.0156s
rollback; -- 0.0009s
```

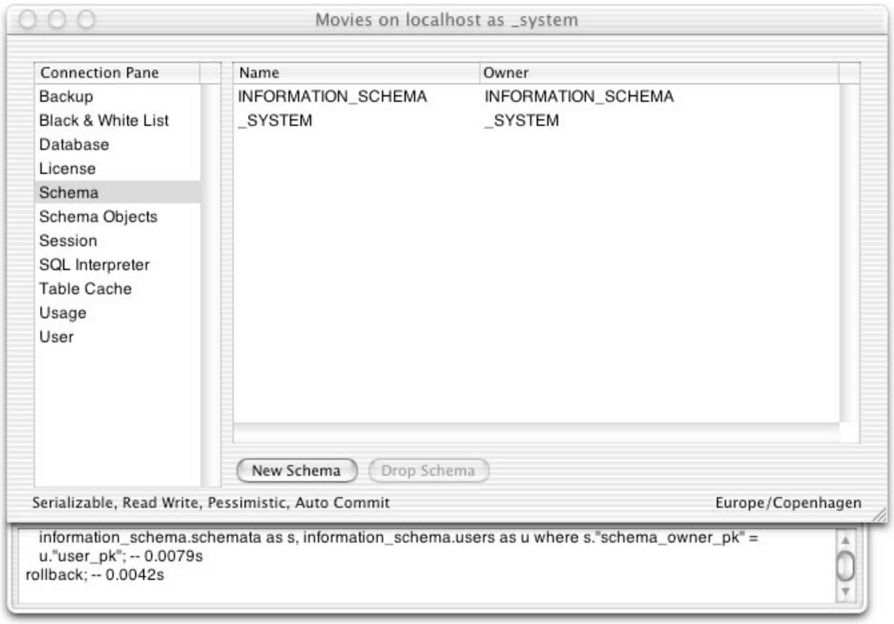
The User pane gives you the tools to add, drop, and edit users.

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.

6.2.7 Schema



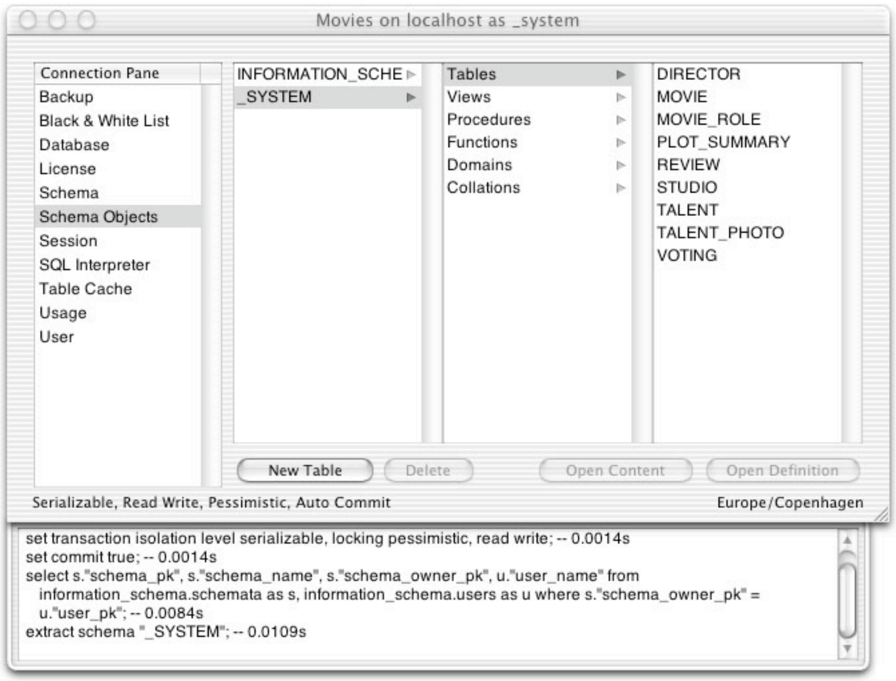
Name	Owner
INFORMATION_SCHEMA	INFORMATION_SCHEMA
_SYSTEM	_SYSTEM

Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

```
  information_schema.schemata as s, information_schema.users as u where s."schema_owner_pk" =
  u."user_pk"; -- 0.0079s
rollback; -- 0.0042s
```

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.

6.2.8 Schema Objects



Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

```
set transaction isolation level serializable, locking pessimistic, read write; -- 0.0014s
set commit true; -- 0.0014s
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"; -- 0.0084s
extract schema "_SYSTEM"; -- 0.0109s
```

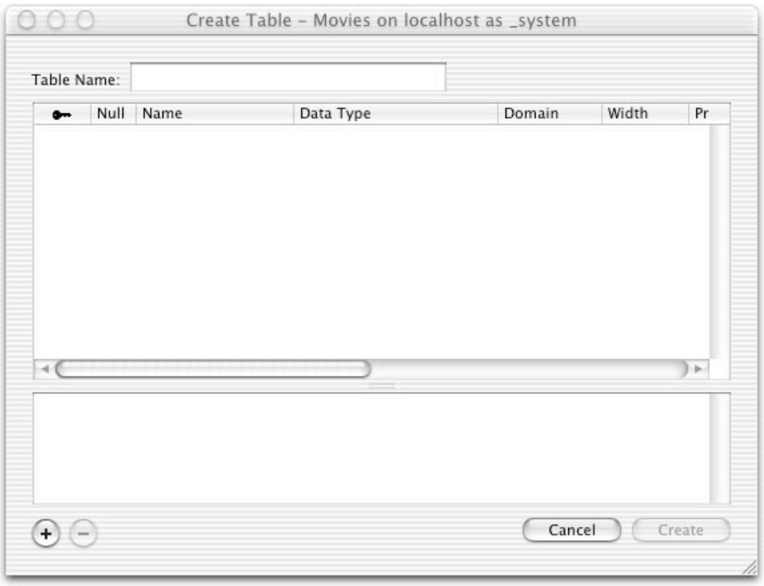
The schema objects browser contains some of the most powerful FrontBaseManager functionality. It allows you to create, delete, view, and modify the stuff you really care about -- your tables and data. This pane is organized like a browser or Finder. You first select a schema, then an object type, then you'll be presented with a list of the schema objects for you to create, delete, open their content, or open their definition.

6.2.9 New Schema Object

There are 6 different schema objects FrontBaseManager allows you to manipulate -- Tables, Views, Procedures, Functions, Domains, and Collations. FrontBaseManager provides an intuitive view to create a new object of each type. Let's first look at the New Table editor.

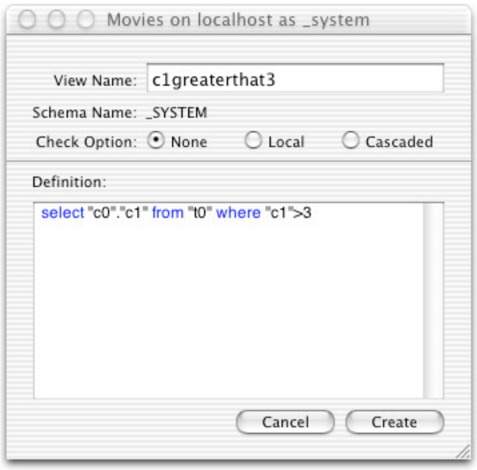
6.2.9.1 New Table

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 data type 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, domain, 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.

6.2.9.2 New View



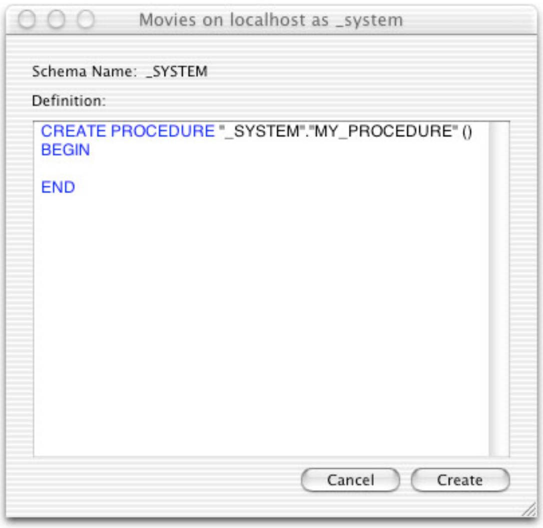
View Name: c1greaterthat3
Schema Name: _SYSTEM
Check Option: [✓ None] Local Cascaded
Definition:

```
select "c0","c1" from "t0" where "c1">3
```

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 SQL 92 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
```

6.2.9.3 New Procedures and New Functions



Schema Name: _SYSTEM
Definition:

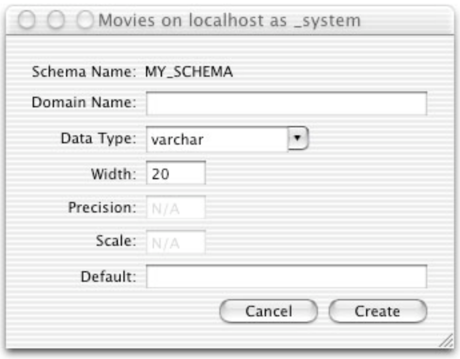
```
CREATE PROCEDURE "_SYSTEM"."MY_PROCEDURE" ()
BEGIN

END
```

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.

6.2.9.4 New Domain

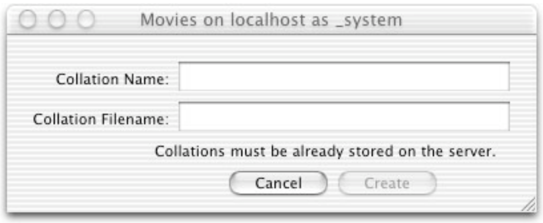
To get there, select the schema where you'd like to create the domain, then select "Domains". The New Domain button will be enabled. Click the New Domain button and you'll be presented with a view like this one:



Schema Name: MY_SCHEMA
Domain Name:
Data Type: varchar
Width: 20
Precision: N/A
Scale: N/A
Default:

You must specify the name of the domain and the data type including possible width, precision and scale. If you need to, you can also specify a default value.

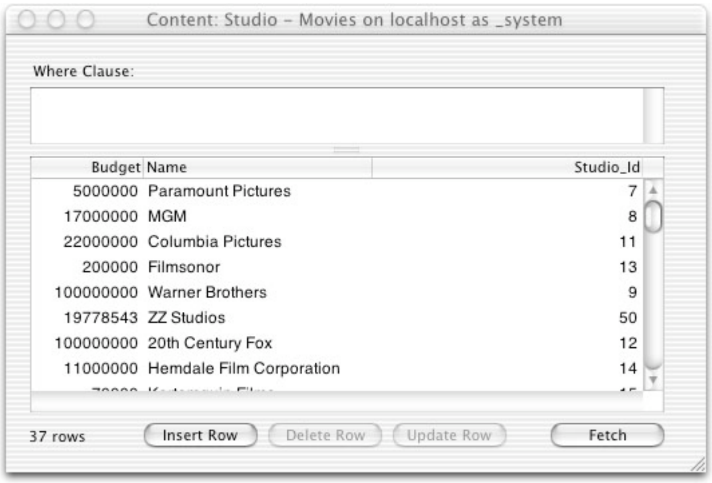
6.2.9.5 New Collation



Collation Name:
Collation Filename:
Collations must be already stored on the server.

Clicking the New Collation button in the Schema Objects browser brings up a small window that 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.)

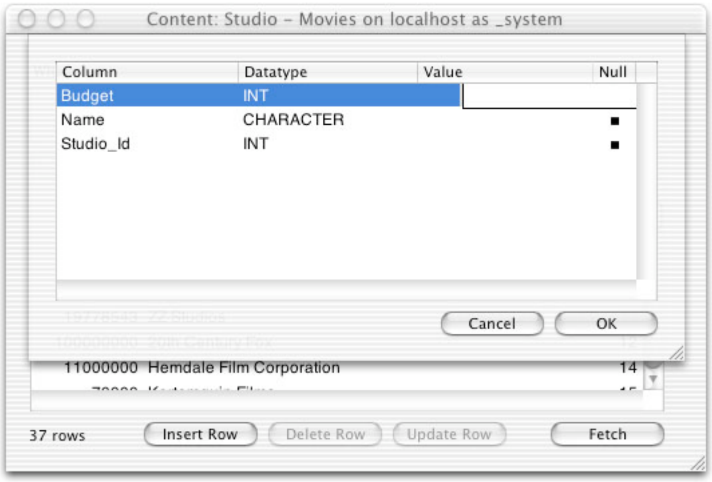
6.2.10 Open Schema Object Content



Budget	Name	Studio_Id
5000000	Paramount Pictures	7
17000000	MGM	8
22000000	Columbia Pictures	11
200000	Filmsonor	13
100000000	Warner Brothers	9
19778543	ZZ Studios	50
100000000	20th Century Fox	12
11000000	Hemdale Film Corporation	14

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, "Budget > 1000000" would be a valid "where clause". 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 that allows you to specify values for any of the columns you'd like populated.

6.2.11 Inserting and Updating Rows


Content: Studio – Movies on localhost as _system
Cancel
OK
11000000 Hemdale Film Corporation 14
37 rows
Insert Row
Delete Row
Update Row
Fetch

Column	Datatype	Value	Null
Budget	INT		
Name	CHARACTER		■
Studio_Id	INT		■

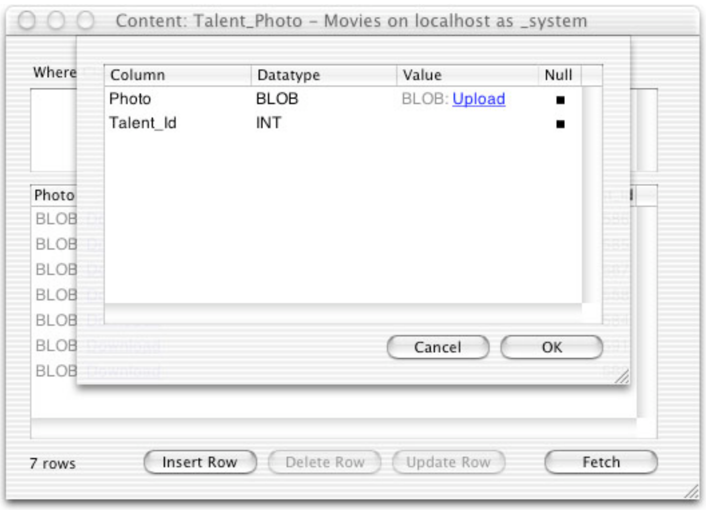
The Row Editor sheet contains a table view with columns for table column name, data type, value, and an allows null indicator. When entering values, enter just the actual content. For example, the SQL to insert a row into the table Studio containing the columns Budget, Name and Studio_Id would look something like:

```
INSERT INTO "studio" ("Budget", “Name”, “Studio_ID”) VALUES
(1000000, ‘Nordisk Film’, 100);
```

When inserting a row, the OK button will be disabled if you have not specified a value for a column that does not allow null.

6.2.12 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 data type 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 FrontBaseManager will read in the file and set the column to that value.



Column	Datatype	Value	Null
Photo	BLOB	BLOB: Upload	■
Talent_Id	INT		■

If a CLOB/BLOB column contains content, it will display a hyperlink 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.

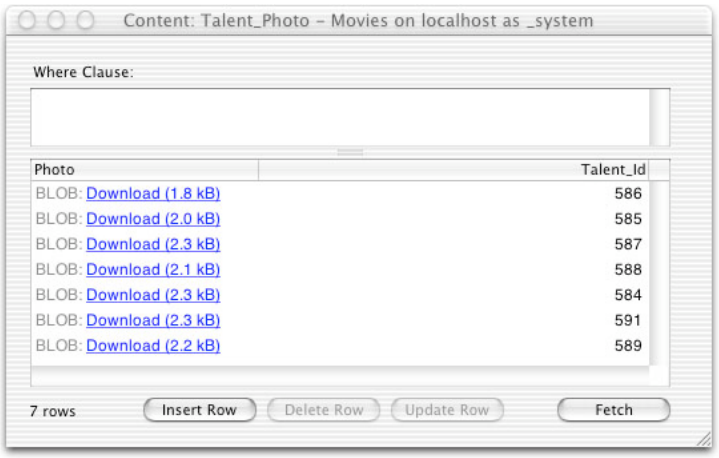


Photo	Talent_Id
BLOB: Download (1.8 kB)	586
BLOB: Download (2.0 kB)	585
BLOB: Download (2.3 kB)	587
BLOB: Download (2.1 kB)	588
BLOB: Download (2.3 kB)	584
BLOB: Download (2.3 kB)	591
BLOB: Download (2.2 kB)	589

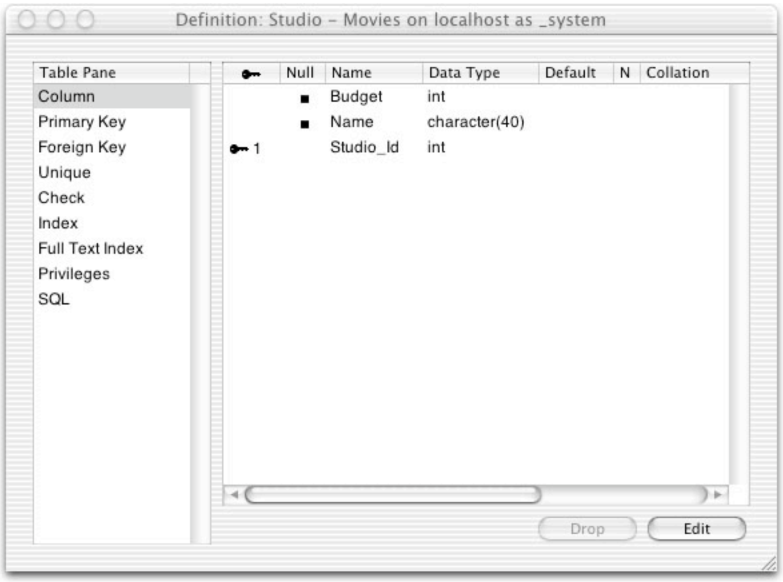
6.2.13 Open Schema Object Definition

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 allow 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 that we'll look at in detail.

6.2.13.1 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.

6.2.13.2 Column Definition Pane

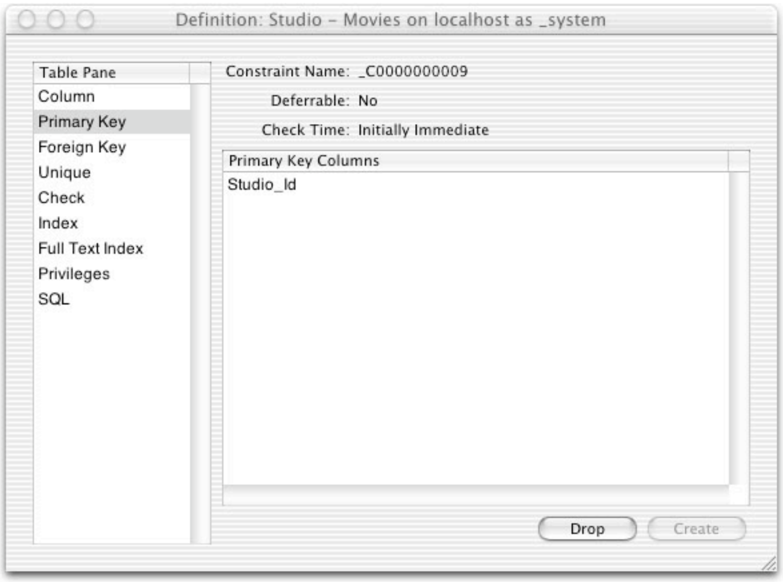


🔑	Null	Name	Data Type	Default	N	Collation
	■	Budget	int			
	■	Name	character(40)			
🔑 1		Studio_Id	int			

Initially columns are ordered as retrieved from the database. You can change the order by clicking at the header of the Name column.

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, data types, 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.

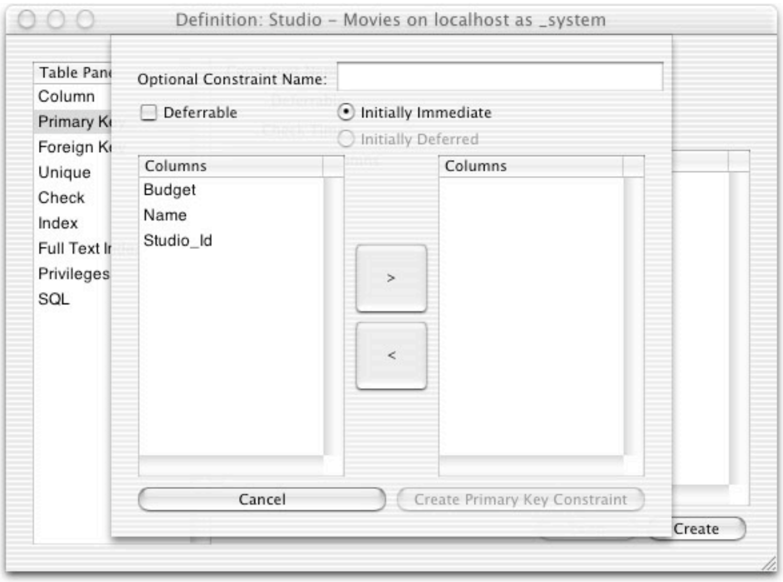
6.2.13.3 Primary Key Pane



Constraint Name: _C0000000009
Deferrable: No
Check Time: Initially Immediate

Studio_Id

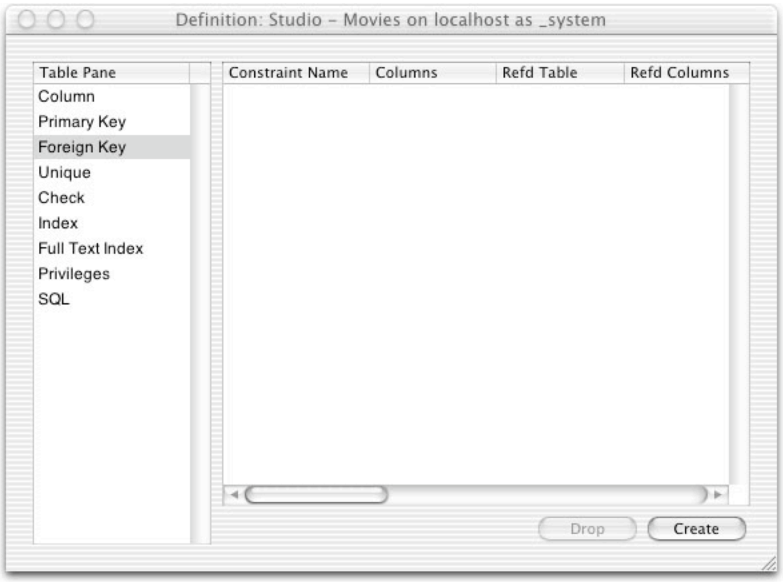
The Primary Key definition pane of the table definition window lets you drop and create primary key constraints. To create a primary key constraint, click the Create button to bring up the primary key creation sheet.


Definition: Studio – Movies on localhost as _system
Optional Constraint Name:
Deferrable
Initially Immediate
Initially Deferred
Columns
Budget
Name
Studio_Id
>
<
Columns
Cancel
Create Primary Key Constraint
Create

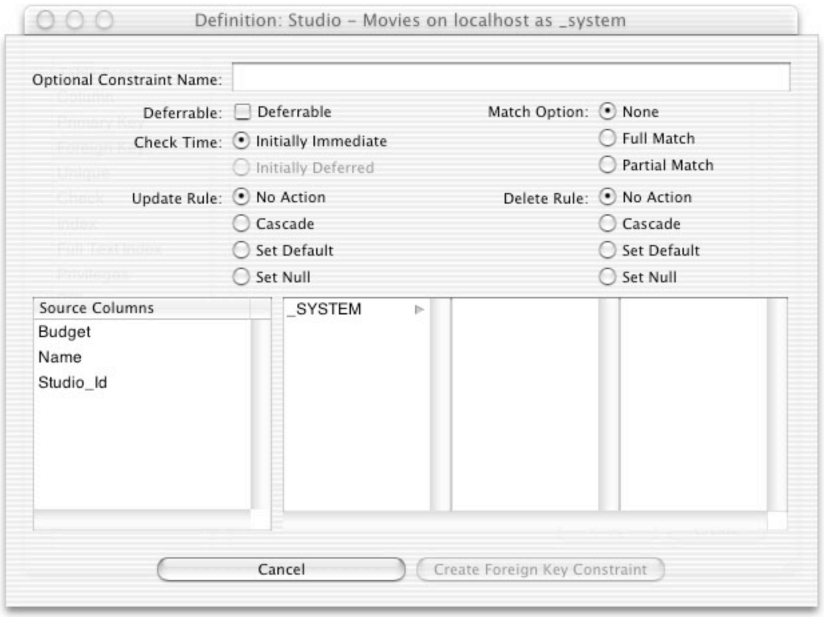
From here you can specify a constraint name (optional), set whether the constraint is deferrable and, if it is deferrable, whether it is initially deferred. You can then specify which columns should participate in the primary key constraint and click the Create Primary Key Constraint button. If your 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.

6.2.13.4 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: Studio – Movies on localhost as _system
Cancel
Create Foreign Key Constraint

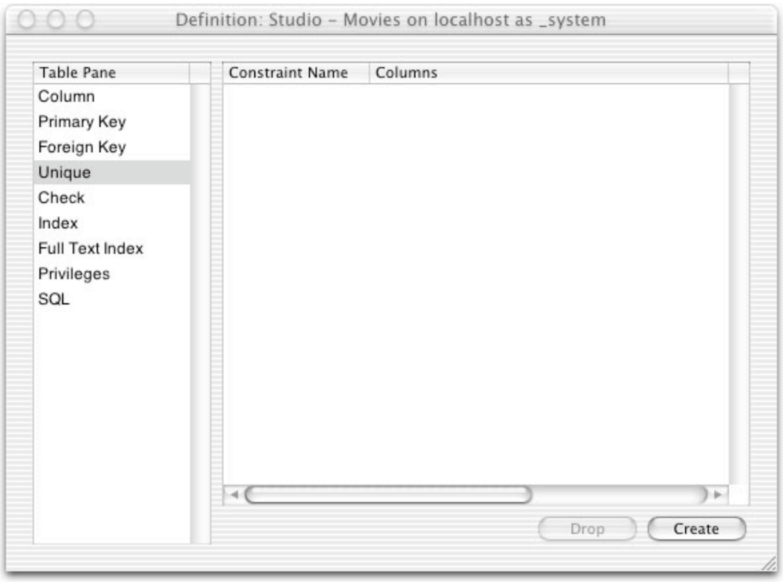
Optional Constraint Name:			
Deferrable:	☐ Deferrable	Match Option:	[✓ None] / Full Match / Partial Match
Check Time:	[✓ Initially Immediate] / Initially Deferred		
Update Rule:	[✓ No Action] / Cascade / Set Default / Set Null	Delete Rule:	[✓ No Action] / Cascade / Set Default / Set Null

Source Columns	
Budget	_SYSTEM
Name	
Studio_Id	

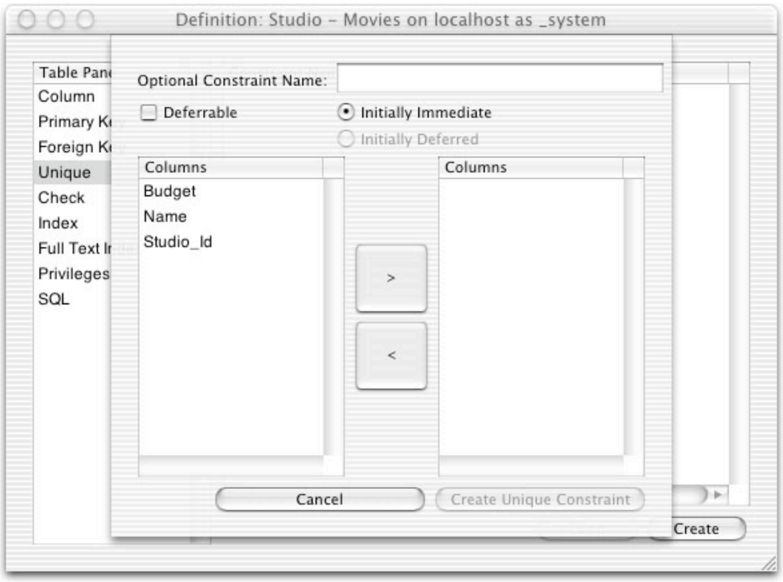
This sheet may look daunting but the only two things you really have to specify are the source column(s) (via the "Source Columns" browser) and the destination columns (via the browser to the right of the "Source Columns" browser). The number of source columns and destination columns must, of course, 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 default behavior.

To drop a foreign key constraint, simply select the constraint and click the Drop button.

6.2.13.5 Unique Pane



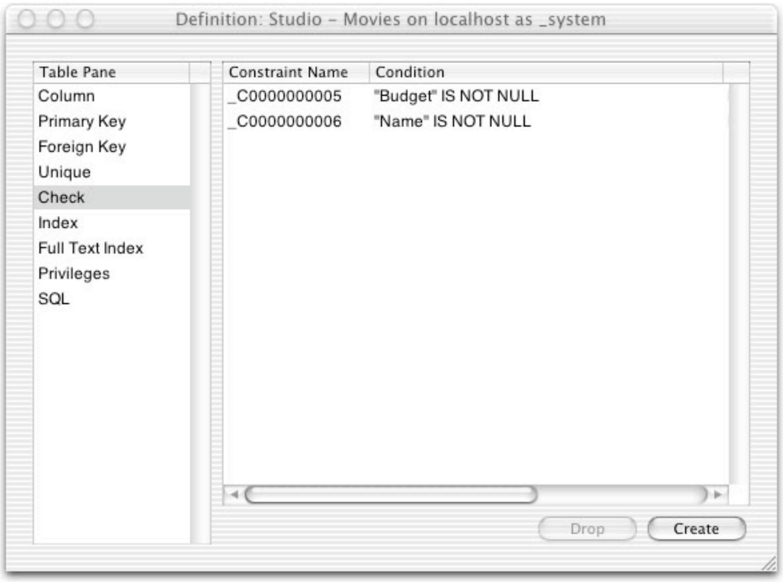
The Unique definition pane of the table definition window presents an interface to create and drop unique constraints. To create a unique constraint, click the Create button to bring up the Unique creation sheet.



From here you can specify a (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.

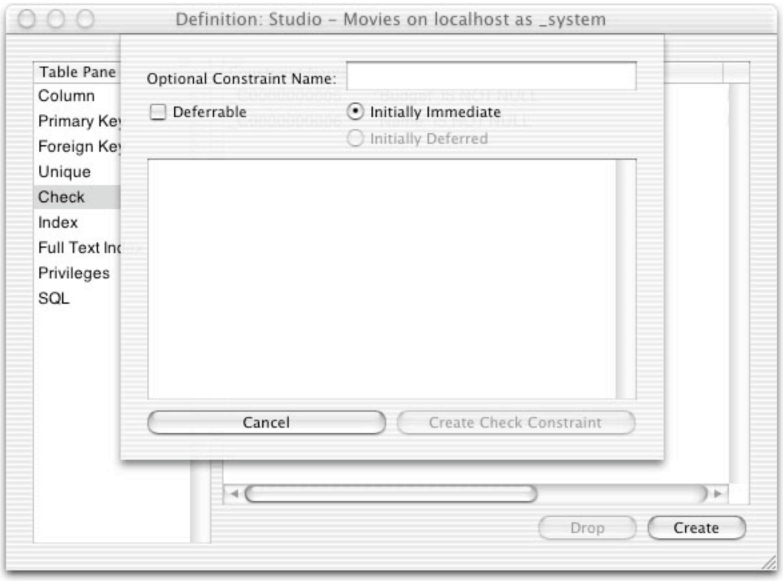
To drop a unique constraint, simply select the constraint and click the Drop button.

6.2.13.6 Check Pane



Constraint Name	Condition
_C0000000005	"Budget" IS NOT NULL
_C0000000006	"Name" IS NOT NULL

The Check definition pane of the table definition window allows you to create and drop check constraints. To create a check constraint, click the Create button to bring up the Check creation sheet.

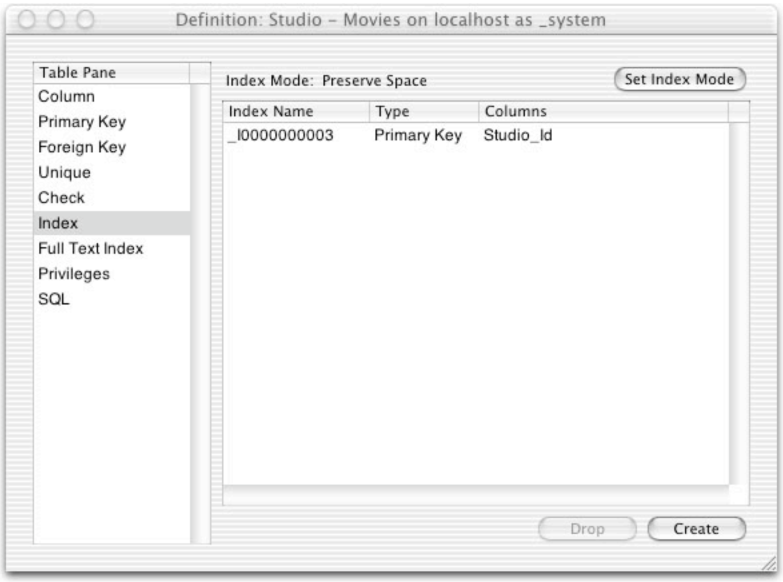


Optional Constraint Name:
[ ] Deferrable
[✓ Initially Immediate] Initially Deferred

From here you can specify a (optional) constraint name, set whether the constraint is deferrable and, if it is deferrable, whether it is initially deferred. You can then specify a free-form check constraints in the definition area. (such as "Budget > 1000000" for an int column 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.

6.2.13.7 Index Pane



Index Mode: Preserve Space

Index Name	Type	Columns
_I0000000003	Primary Key	Studio_Id

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.

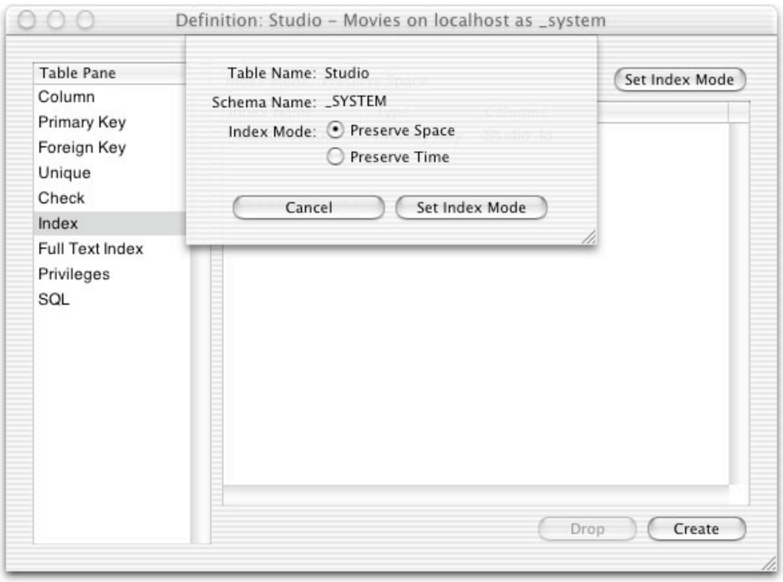
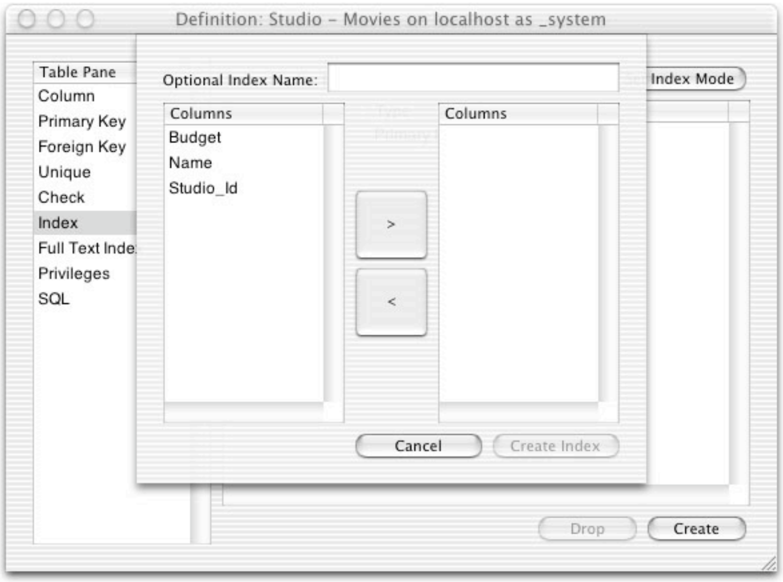


Table Name: Studio
Schema Name: _SYSTEM
Index Mode: [✓ Preserve Space] Preserve Time

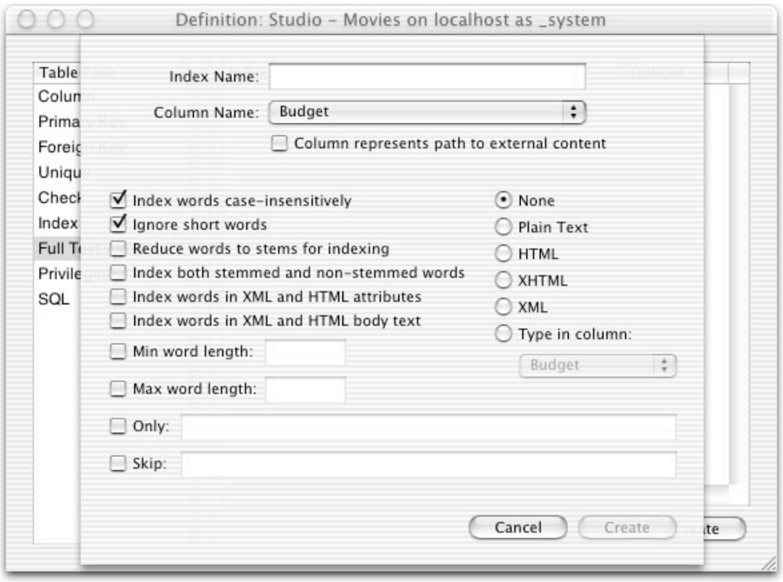
To set the index mode, click the Set Index Mode button and you'll be presented with the choice of either "Preserve Space" or "Preserve Time". Make your choice, click Set Index Mode and FrontBaseManager will change your index mode. You must be in Serializable, Pessimistic mode to change the index settings.

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.

6.2.13.8 Full Text Index

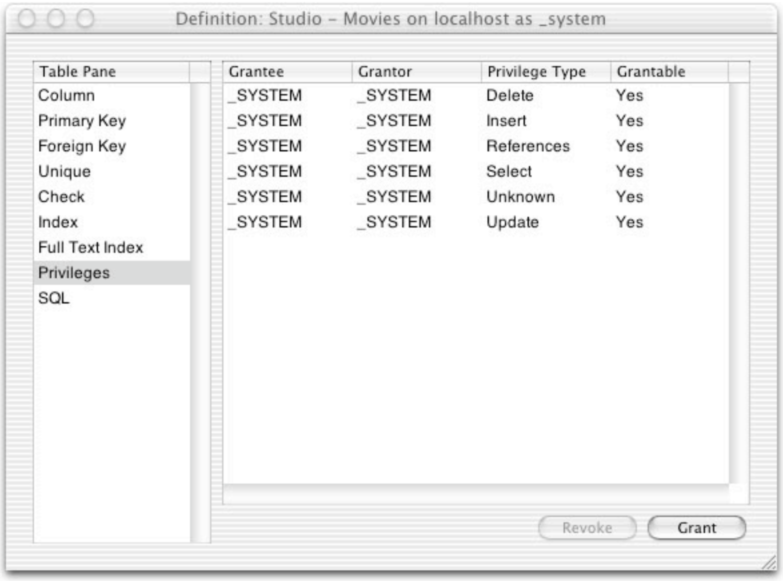


Index Name:
Column Name: Budget
[ ] Column represents path to external content
[✓] Index words case-insensitively
[✓] Ignore short words
[ ] Reduce words to stems for indexing
[ ] Index both stemmed and non-stemmed words
[ ] Index words in XML and HTML attributes
[ ] Index words in XML and HTML body text
[ ] Min word length:
[ ] Max word length:
[ ] Only:
[ ] Skip:
[✓ None] Plain Text HTML XHTML XML Type in column: Budget

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 indexes, 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.

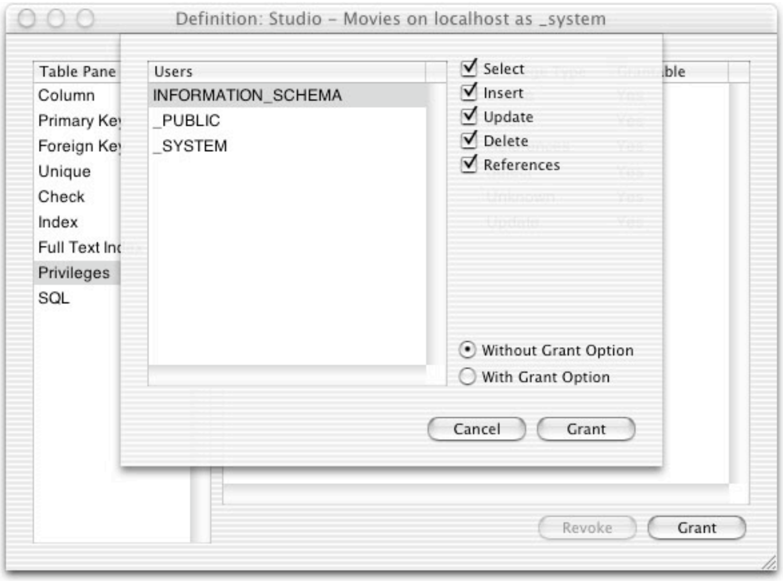
To drop a full text index, simply select the index and click the Drop button.

6.2.13.9 Privileges



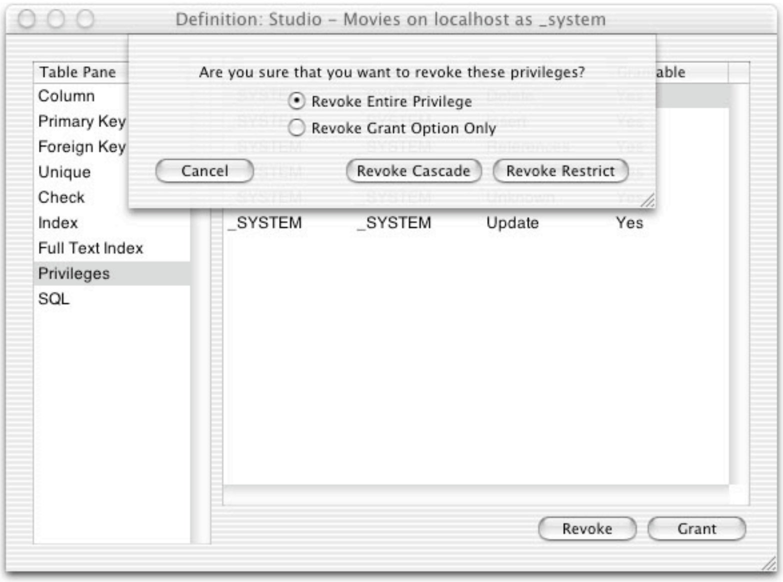
Grantee	Grantor	Privilege Type	Grantable
_SYSTEM	_SYSTEM	Delete	Yes
_SYSTEM	_SYSTEM	Insert	Yes
_SYSTEM	_SYSTEM	References	Yes
_SYSTEM	_SYSTEM	Select	Yes
_SYSTEM	_SYSTEM	Unknown	Yes
_SYSTEM	_SYSTEM	Update	Yes

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.



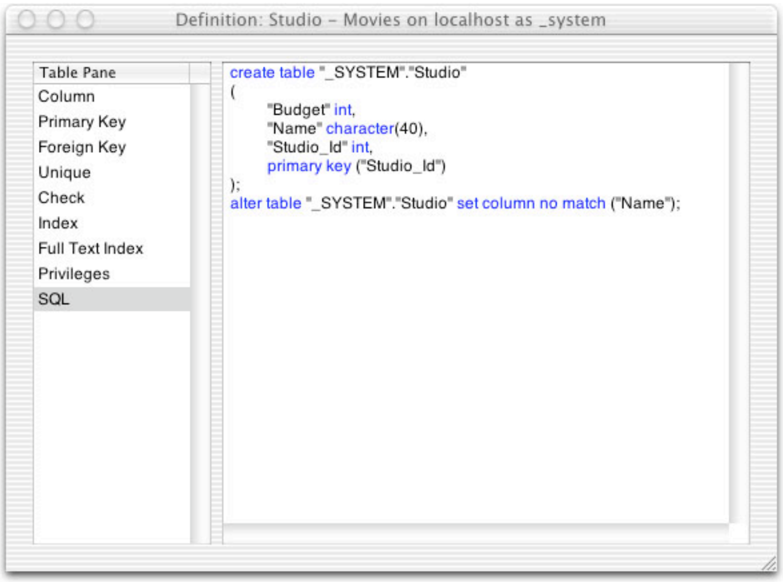
You'll see all the database users in browser on the left. Select the grantee, check all the privilege types you'd like to grant, select whether you'd like to grant without the grant option (the default) or with the grant option, and click the Grant button.

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.)

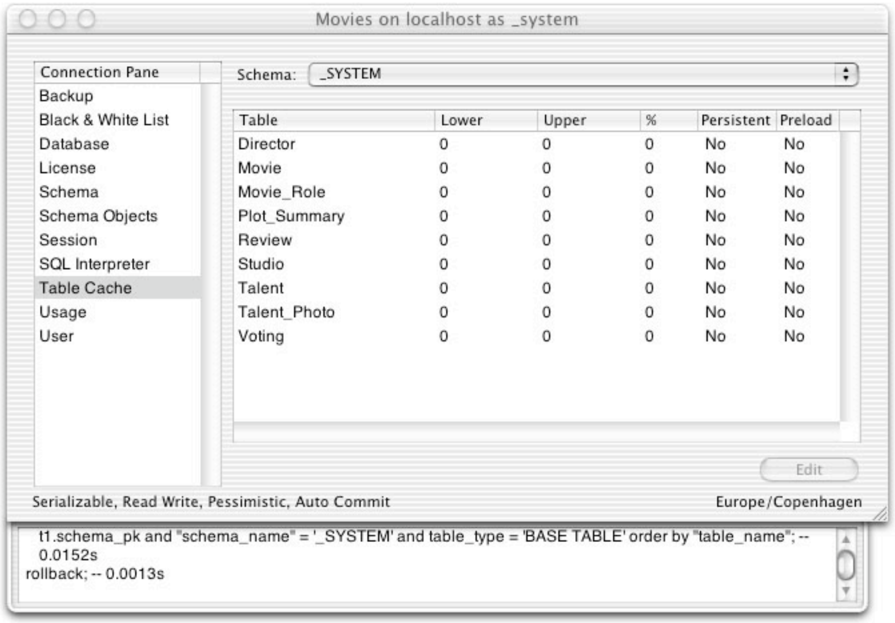
6.2.13.10 SQL



```
create table "_SYSTEM"."Studio"
(
	"Budget" int,
	"Name" character(40),
	"Studio_Id" int,
	primary key ("Studio_Id")
);
alter table "_SYSTEM"."Studio" set column no match ("Name");
```

This pane simply shows the SQL that generated the table.

6.2.14 Table Cache



Schema: _SYSTEM

Table	Lower	Upper	%	Persistent	Preload
Director	0	0	0	No	No
Movie	0	0	0	No	No
Movie_Role	0	0	0	No	No
Plot_Summary	0	0	0	No	No
Review	0	0	0	No	No
Studio	0	0	0	No	No
Talent	0	0	0	No	No
Talent_Photo	0	0	0	No	No
Voting	0	0	0	No	No

Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

```
  t1.schema_pk and "schema_name" = '_SYSTEM' and table_type = 'BASE TABLE' order by "table_name"; --
  0.0152s
rollback; -- 0.0013s
```

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.

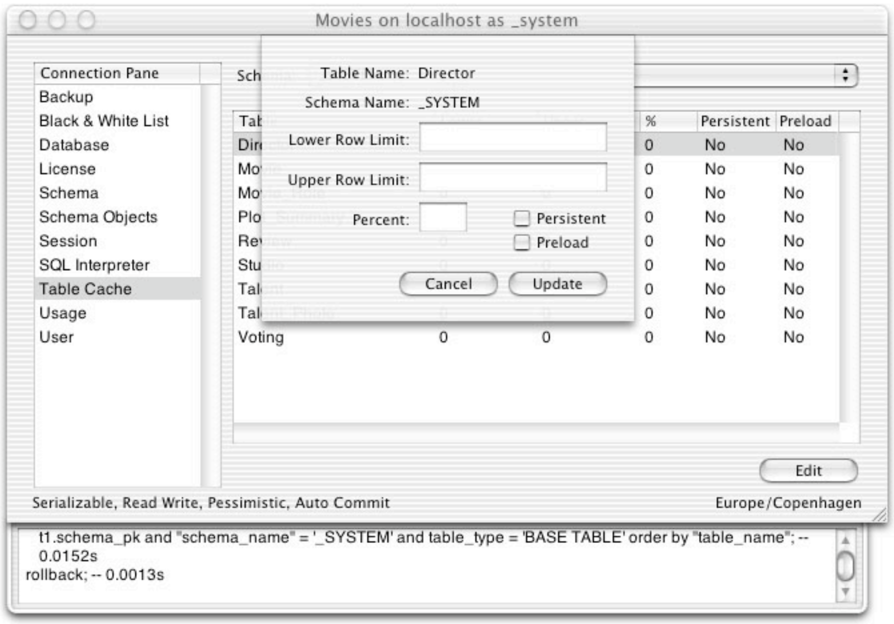


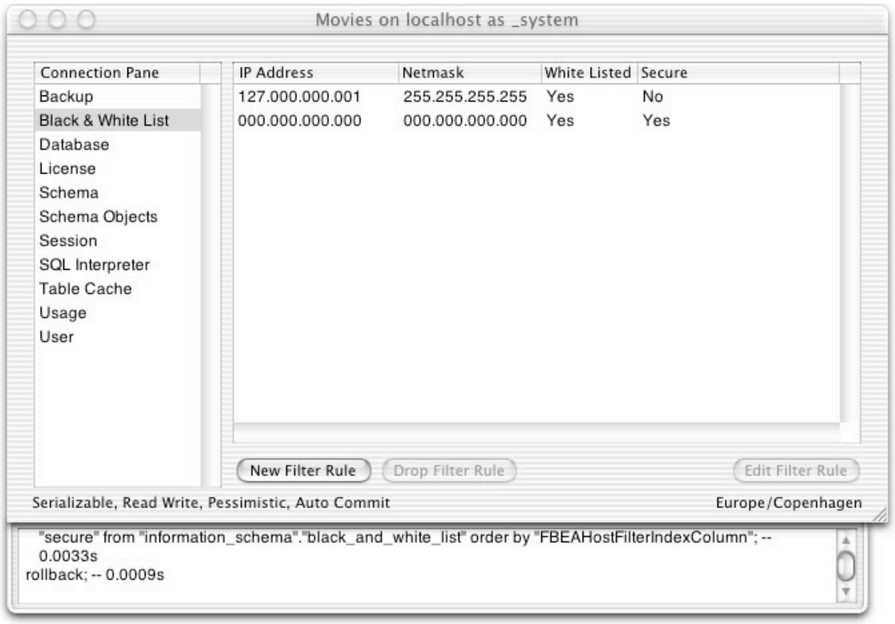
Table Name: Director
Schema Name: _SYSTEM
Lower Row Limit:
Upper Row Limit:
Percent:
☐ Persistent
☐ Preload

Tab			%	Persistent	Preload
Dir			0	No	No
Mo			0	No	No
Mo			0	No	No
Plo			0	No	No
Re			0	No	No
Stu			0	No	No
Tal			0	No	No
Tal			0	No	No
Voting	0	0	0	No	No

Serializable, Read Write, Pessimistic, Auto Commit Europe/Copenhagen

```
  t1.schema_pk and "schema_name" = '_SYSTEM' and table_type = 'BASE TABLE' order by "table_name"; --
  0.0152s
rollback; -- 0.0013s
```

6.2.15 Black & White List



IP Address	Netmask	White Listed	Secure
127.000.000.001	255.255.255.255	Yes	No
000.000.000.000	000.000.000.000	Yes	Yes

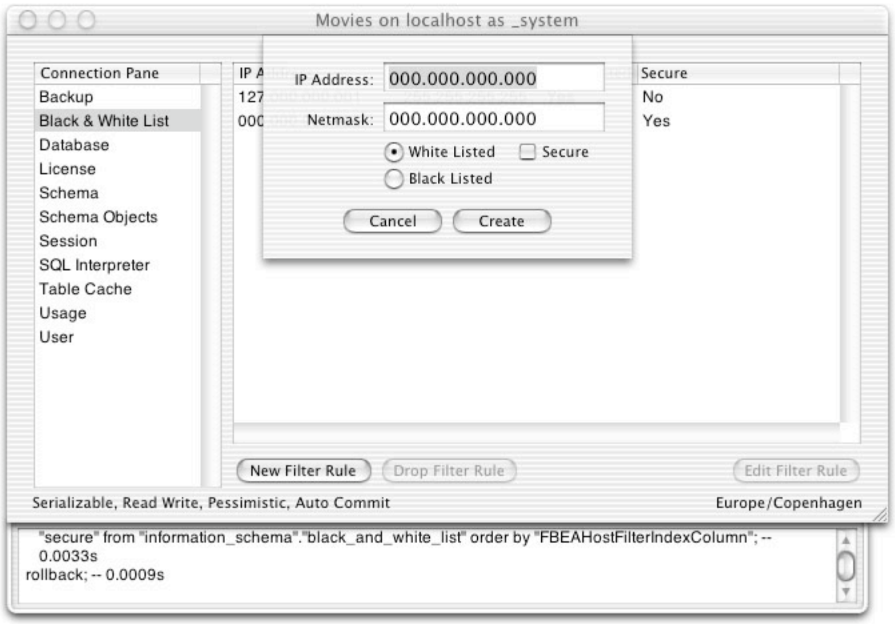
Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

```
  "secure" from "information_schema"."black_and_white_list" order by "FBEAHostFilterIndexColumn"; --
  0.0033s
rollback; -- 0.0009s
```

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 an 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 the secure field, the client is requested to use encryption for all further communication with the server.

Black and white list rules are evaluated in the order of decreasing Netmask value. As soon as a match is found, evaluation stops and the rule is applied.

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 [ ]

IP A	Secure
127	No
000	Yes

Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

```
  "secure" from "information_schema"."black_and_white_list" order by "FBEAHostFilterIndexColumn"; --
  0.0033s
rollback; -- 0.0009s
```

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.

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

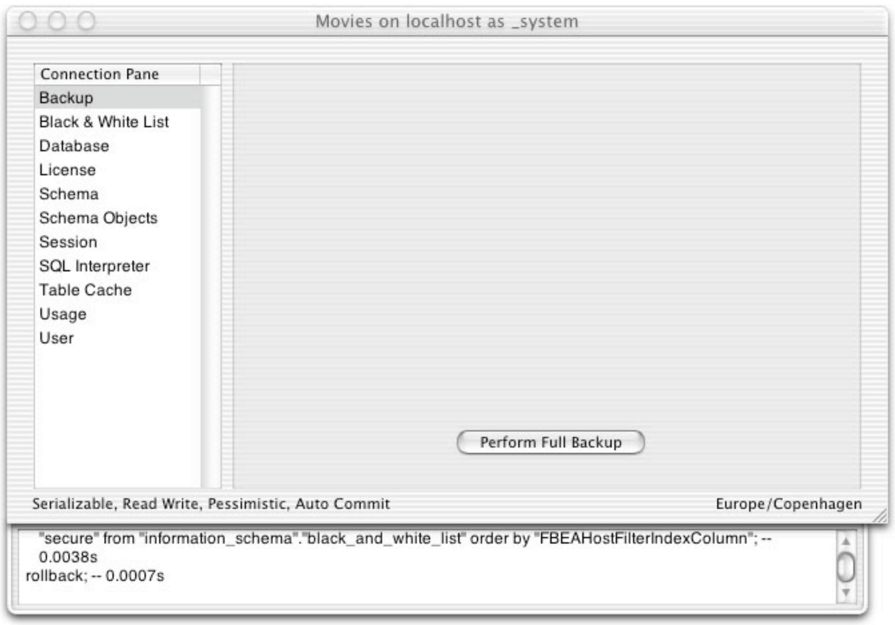
This would lock out the single IP at 24.5.126.145 and would allow connections from every other IP.

IP	Netmask	White 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.

6.2.16 Backup



Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

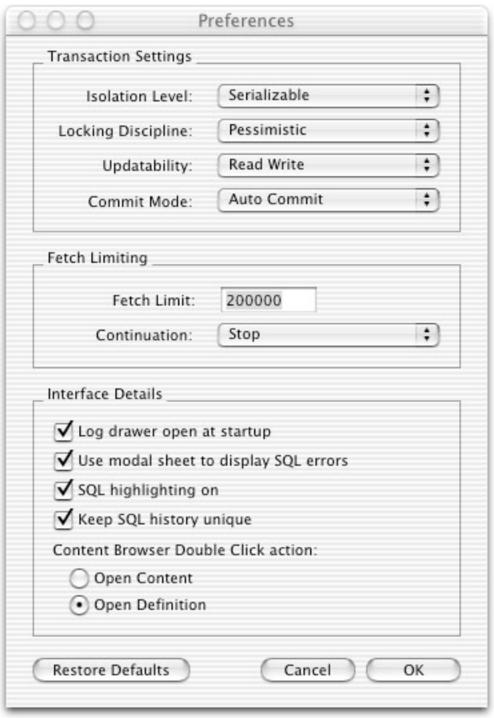
```
  "secure" from "information_schema"."black_and_white_list" order by "FBEAHostFilterIndexColumn"; --
  0.0038s
rollback; -- 0.0007s
```

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 text field if you want to specify a backup file and a checkbox (defaulted to checked) that will instruct the server to compress the backup file.

6.3 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:

6.3.1 Preferences



Transaction Settings

Isolation Level: Serializable
Locking Discipline: Pessimistic
Updatability: Read Write
Commit Mode: Auto Commit

Fetch Limiting

Fetch Limit: 200000
Continuation: Stop

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]

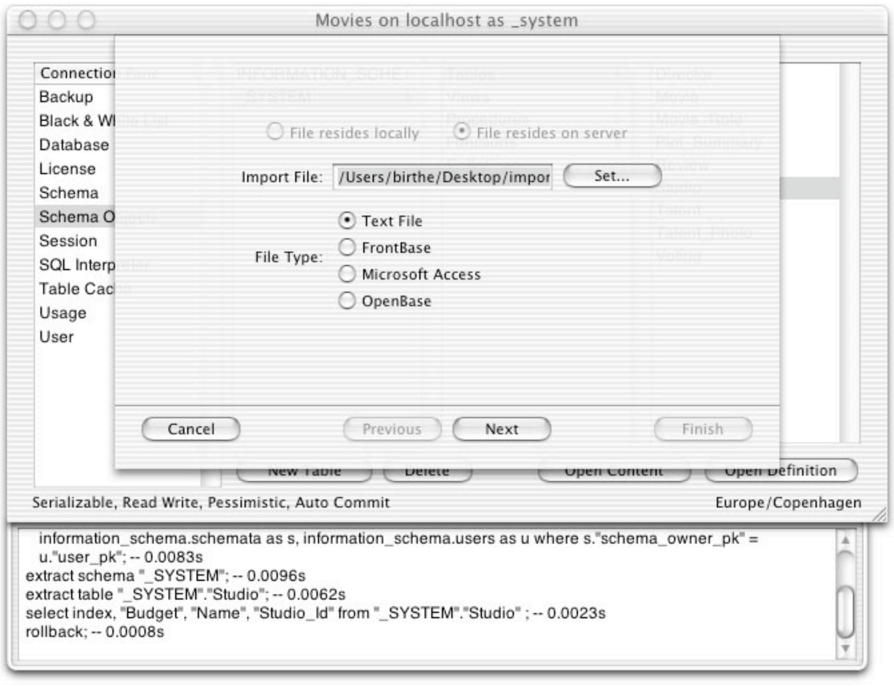
FrontBaseManager's preferences system allows you to specify your start-up transaction settings, fetch limiting, and a few interface details so that you don't have to keep switching from the factory defaults to the setup that you prefer. The transaction settings specified in your preferences will be set 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 SQL 92 syntax errors by showing you which words are SQL 92 reserved keywords. (One benefit of this is reminding you to wrap your column names in quotes if they are named the same as any SQL 92 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.

6.3.2 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*.



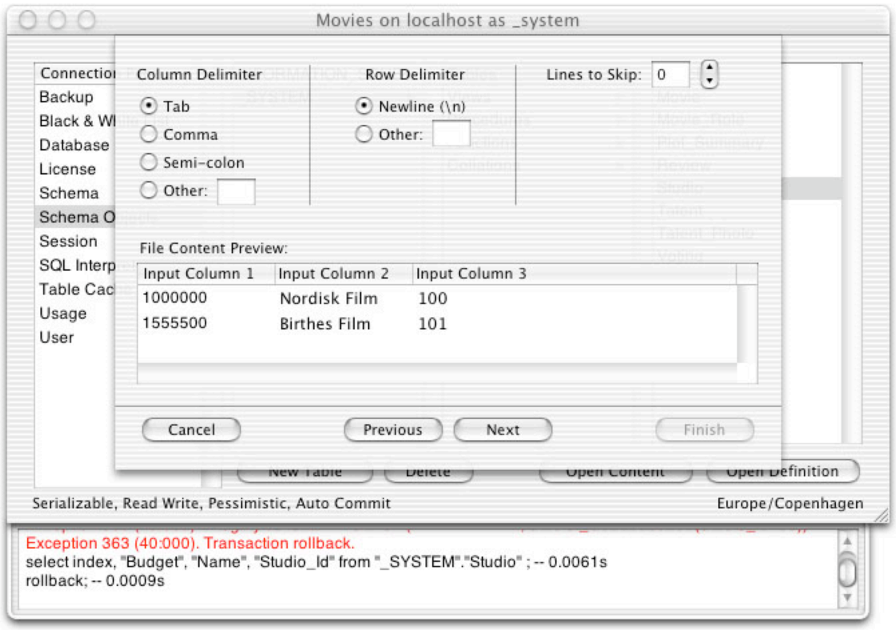
Import File: /Users/birthe/Desktop/impor

File Type: [✓ Text File] FrontBase Microsoft Access OpenBase

Serializable, Read Write, Pessimistic, Auto Commit
Europe/Copenhagen

```
  information_schema.schemata as s, information_schema.users as u where s."schema_owner_pk" =
  u."user_pk"; -- 0.0083s
extract schema "_SYSTEM"; -- 0.0096s
extract table "_SYSTEM"."Studio"; -- 0.0062s
select index, "Budget", "Name", "Studio_Id" from "_SYSTEM"."Studio" ; -- 0.0023s
rollback; -- 0.0008s
```

The first screen allows you to specify where the file is and what general format the filed 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 flat file 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

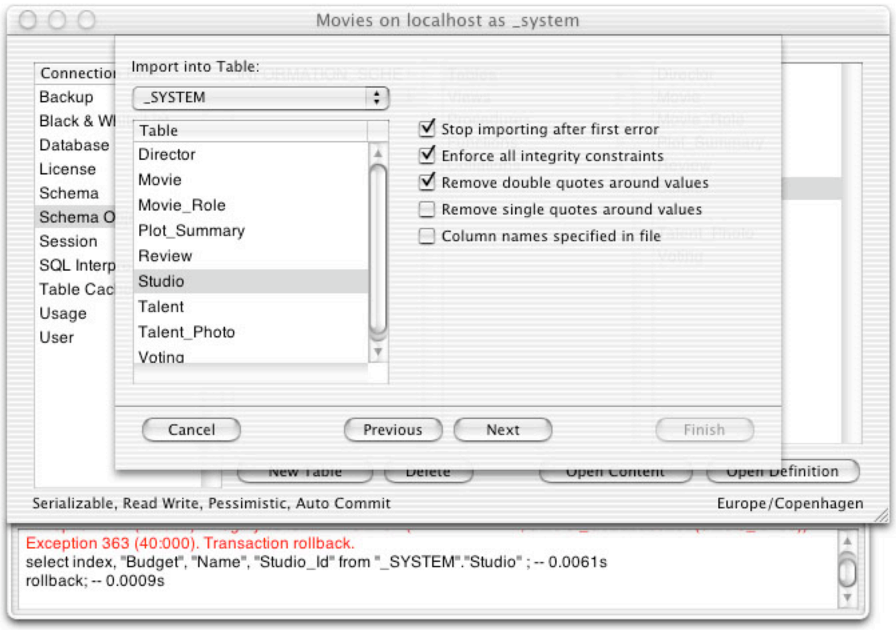
File Content Preview:

Input Column 1	Input Column 2	Input Column 3
1000000	Nordisk Film	100
1555500	Birthes Film	101

Serializable, Read Write, Pessimistic, Auto Commit   Europe/Copenhagen

```
Exception 363 (40:000). Transaction rollback.
select index, "Budget", "Name", "Studio_Id" from "_SYSTEM"."Studio" ; -- 0.0061s
rollback; -- 0.0009s
```

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

```
Exception 363 (40:000). Transaction rollback.
select index, "Budget", "Name", "Studio_Id" from "_SYSTEM"."Studio" ; -- 0.0061s
rollback; -- 0.0009s
```

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 "Column 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.

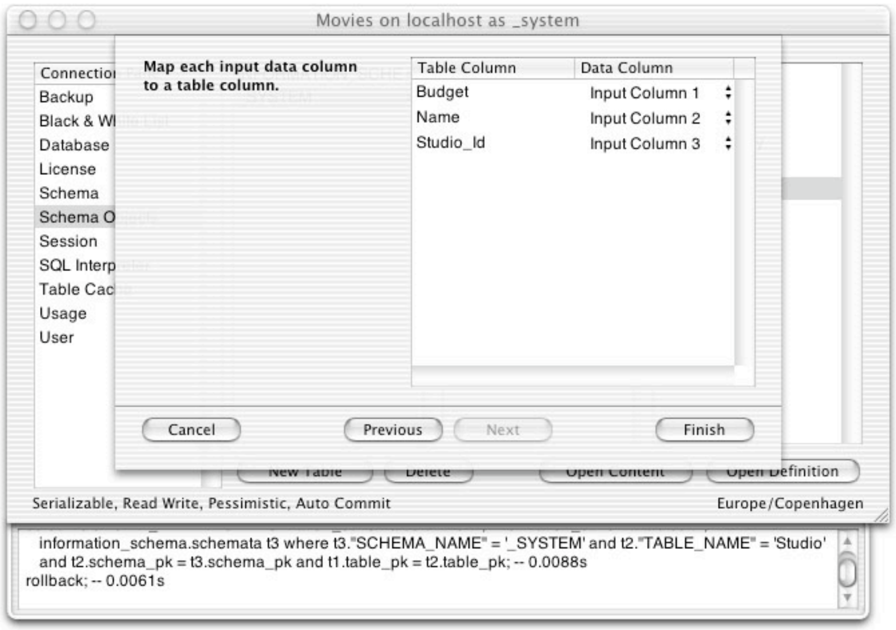


Table Column	Data Column
Budget	Input Column 1
Name	Input Column 2
Studio_Id	Input Column 3

```
  information_schema.schemata t3 where t3."SCHEMA_NAME" = '_SYSTEM' and t2."TABLE_NAME" = 'Studio'
  and t2.schema_pk = t3.schema_pk and t1.table_pk = t2.table_pk; -- 0.0088s
rollback; -- 0.0061s
```

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:

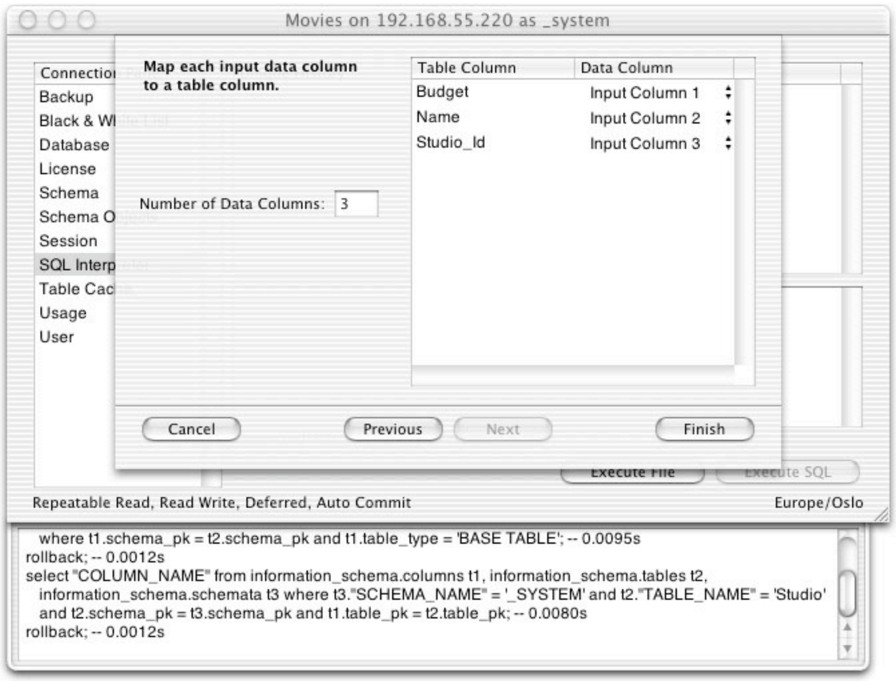


Table Column	Data Column
Budget	Input Column 1
Name	Input Column 2
Studio_Id	Input Column 3

```
  where t1.schema_pk = t2.schema_pk and t1.table_type = 'BASE TABLE'; -- 0.0095s
rollback; -- 0.0012s
select "COLUMN_NAME" from information_schema.columns t1, information_schema.tables t2,
  information_schema.schemata t3 where t3."SCHEMA_NAME" = '_SYSTEM' and t2."TABLE_NAME" = 'Studio'
  and t2.schema_pk = t3.schema_pk and t1.table_pk = t2.table_pk; -- 0.0080s
rollback; -- 0.0012s
```

Notice how changing the number of data columns (3 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.

If the Import helper does not work for your import file, please contact tools@frontbase.com with the details of your problem.

6.3.3 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>Toggle SQL Log or by hitting Command-L. If the Log Drawer is hidden, you can reveal it again by selecting View>Toggle SQL Log or by hitting Command-L. You can clear the SQL Log at any time by hitting Command-K.

6.4 Known Issues

6.4.1 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 database 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>
```

7 sql92

The sql92 command line tool enables administration of FrontBase databases directly from the command line and from scripts.

This chapter contains the following:

- Getting Started on page 177
- Command Syntax on page 178
- sql92 Input on page 179
- Command Line Editing on page 179.
- Commands on page 181

NOTE: Please note that only basic commands are included in this section.

7.1 Getting Started

The sql92 command-line tool is located in the installation bin directory <FB home>/FrontBase/bin. In order to start sql92, do the following:

- Open a terminal application/command-line
- Start sql92 with the command <FB home>/FrontBase/bin/sql92

If you include <FB home>/FrontBase/bin in your $PATH environment variable, you only need to write sql92 to start the tool.

Below is a small example demonstrating how to create a database and connect to it on Mac OS X:

```
bash$ /Library/FrontBase/bin/sql92
sql92#1> CREATE DATABASE db0;
sql92#1> CONNECT TO db0 USER _system;
> Auto committing is on: SET COMMIT TRUE;
db0@localhost#3>

...  Execute your commands

db0@localhost#3> exit;
```

7.2 Command Syntax

```
sql92 <options> [ <filename> [ <arguments> ...] ]
```

If a filename is specified, sql92 reads and executes the SQL 92 statements in the file. If the input file is omitted, SQL 92 statements are read from standard input. All output is written to standard output. When input is read from a file all occurrences of the strings $0, $1, ... in the input are substituted by the corresponding argument, except when occurring inside a string literal. If an argument is not given it is substituted with the empty string.

7.2.1 Options

-a	Autocommit. When a new connection to a database is created do not enable auto commit.
-c	Compare. Perform a test and compare the output with the input. If they are equal write a passed comment; otherwise write a failed comment.
-e	Exit on error. Exit sql92 when an error is encountered. This is the default behavior when input is read from a file.
-i	Ignore. Do not exit when an error is encountered, this is the default behavior when input is read from a terminal.
-l <number>	Fetch limit. At most <number> of rows is fetched from any result set.
-m	Metadata. Print metadata as returned from the server when it has executed sql statements.
-n	Non-comparable. Print a note saying that the output must be inspected as a comparison would fail.
-p	Prompt. Prompt for commands. This is the default behavior when input is read from a terminal.
-q	Quote. Allow control characters in string literals. By default sql92 does not allow control characters in string constants.
-s	Silent. Do not print commands as they are executed. By default, statements are printed when the input read from a file.

-t	Time. Print seconds spent by the server to interpret an SQL statement, or seconds including fetching the results, as well as seconds spent by the server to interpret the fetch statement.
-u <encoding>	Input encoding. The <encoding> argument specifies the encoding of the input. The default is UTF8 that will also accept us-ASCII.
-v	Verbose. Print statements as they are executed.
-w	Warnings. Print warnings.

7.3 sql92 Input

An sql92 command is always terminated by a ';' character. When you type input to sql92 it will not interpret the statement until a ';' character is met. 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 itself. 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 UNIX 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. Results 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 .sql92rc.sql. sql92 searches the current working directory and the current user's home directory.

7.4 Command Line Editing

When sql92 reads input from a terminal it provides command line editing with the following key-bindings:

Left arrow	Cursor left
Right arrow	Cursor right
Down arrow	Previous history line
Up arrow	Next history line
Crtl-A	Cursor to beginning of line
Crtl-B	Cursor left

Crtl-C	Clear line
Crtl-D	End of file exit sql92
Crtl-E	Cursor to end of line
Crtl-F	Cursor Right
Crtl-H	Delete character in front of cursor
Crtl-I	Complete filename
Crtl-K	Delete from cursor to end of line
Crtl-M	Execute line
Crtl-N	Next history line
Crtl-P	Previous
Crtl-S	Enter search in history mode
Crtl-Z	Suspend
Delete	Delete character in front of cursor
Backspace	Delete character in front of cursor
Return	Execute line
Tab	Complete filename

When the command line is in the search in history mode the following key bindings are in effect:

Down arrow	Previous matching history line
Up arrow	Next matching history line
Crtl-D	End of file exit sql92
Crtl-S	Next matching history line
Crtl-H	Delete character in front of cursor
Delete	Delete character in front of cursor
Backspace	Delete character in front of cursor
Crtl-C	Exit search and do not update current line
Esc	Exit search and do not update current line
Crtl-M	Exit search and update current line

When the history search mode is entered, the current line is used as the initial search string. When the search string is changed the search starts from the most recent history line and searches towards the oldest. A history line matches a search string if the search string is a case and space blind prefix of the history line.

When a command is executed it is added to the history as the newest. If the new history line matches a line already in the history the old line is removed. The length of the history is limited to 500 lines.

The h; command prints out the command history, !! recalls the newest history line, !<n> recalls the n'th history line.

When sql92 is reading input from a terminal, pressing CTRL-C will interrupt the execution of the command and return to the command prompt.

7.5 Commands

When sql92 reads commands it will in most cases forward the SQL text to a FrontBase database server, which will interpret the SQL and possibly produce a result. sql92 will read and print the result, in most cases a set of rows. But sql92 does recognize a number of sql92 specific commands, most notably the commands for creating a connection to the database server.

7.5.1 sql92 Specific Command Summary (Alphabetical)

```
ADD CLIENT
ADD MEMBER
agent commands
AUTOSTART
autostart command
clustering commands
CONNECT TO
connection commands
CREATE CLIENT
CREATE CLUSTER
CREATE DATABASE
database commands
default database commands
DEFINE BLOB
DEFINE CLOB
DELETE DATABASE
DISCONNECT
INTERRUPT AGENT
LAPTIME
password commands
REMOVE CLIENT
REMOVE MEMBER
replicator commands
SCRIPT
SET CLUSTER
set commands
SET CONNECTION
SET DATABASE PASSWORD
SET DEFAULT DATABASE
SET FORMAT
SET MEMBER
SET OUTPUT
SET PASSWORD
SHOW CACHES
SHOW CLIENTS
SHOW CLUSTER
show commands
SHOW CONNECTIONS
SHOW DATABASE
SHOW DATABASE LOG
SHOW DATABASES
SHOW DEFAULT DATABASE
SHOW HISTORY
SHOW LICENSE
SHOW LOGS
SHOW MEMBER
SHOW MEMORY
SHOW SCHEMA
SHOW SIZE
SHOW TABLE
SHOW TIMEZONE
SHOW TRANSACTION
SHOW USAGE
SHOW VIEW
SLEEP
START CLUSTER
START DATABASE
START MEMBER
START REPLICATOR
START STOPWATCH
STOP AGENT
STOP CLUSTER
STOP DATABASE
STOP MEMBER
STOP REPLICATOR
STOP STOPWATCH
timing commands
```

7.5.2 Database Commands

7.5.2.1 Create Database

```
CREATE DATABASE <database-name>
   [ON|@|HOST <host-name>]
   [PASSWORD <password>]
   [OPTIONS <option>...];
```

Create a new database named <database-name> on the host named <host-name>. If <host-name> is not specified, localhost 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 prompted for the password, otherwise the statement will fail. The specified options will be used when creating the database.

7.5.2.2 Start Database

```
START DATABASE <database-name>
   [ON|@|HOST <host-name>]
   [PASSWORD <password>]
   [OPTIONS <option>...];
```

Start the database named <database-name> on the host named <host-name>. If <host-name> is not specified, localhost 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 prompted for the password, otherwise the statement will fail. The specified options will be used when starting the database.

7.5.2.3 Delete Database

```
DELETE DATABASE <database-name>
   [ON|@|HOST <host-name>]
   [PASSWORD <password>];
```

Delete the database named <database-name> on the host named <host-name>. If <host-name> is not specified, localhost 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 prompted for the password, otherwise the statement will fail.

7.5.2.4 Stop Database

```
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.

7.5.3 Connection Commands

7.5.3.1 Connect

```
CONNECT TO <database-name> | <port-number>
   [ON|@|HOST <host-name>]
   [DATABASE_PASSWORD <database-password>]
   [AS <connection-name>]
   [USER <user-name>]
   [PASSWORD <password>];

CONNECT TO DEFAULT;
```

The first form establishes a connection to the database named <database-name> on the host named <host-name> and with the optional database password <database-password>. If <host-name> is not specified, localhost is used. The connection is named <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 by the SQL 92 session to establish a session. If the <user-name> is not specified, the login name of the host operating system is used. The <password> specifies the user password. If the database password, the user password, or the user names is not specified but required, sql92 will prompt for the required values.

The second form establishes the default connection.

7.5.3.2 Set Connection

```
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.

7.5.3.3 Disconnect

```
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.

7.5.3.4 Show Connections

```
SHOW CONNECTIONS;
```

Shows all the connections that have been established.

7.5.4 Password Commands

7.5.4.1 9.5.4.1 Set Database Password

```
SET DATABASE_PASSWORD [<password>];
```

Set the database password to <password>. If <password> is omitted, future connections will not require a password.

7.5.4.2 Set Password

```
SET PASSWORD [<password>]
   [USER <user-name>]
   [OLD <old-password>];
```

Set the password for the user with the name <user-name> to <password>. If the user already has a password it must be specified as <old-password>. If <password> is omitted, the user may login without a password.

7.5.5 Default Database Commands

7.5.5.1 Set Default Database

```
SET DEFAULT DATABASE <database-name>
   [ON | @ | HOST <host-name>]
   [DATABASE PASSWORD '<password>']
   [PASSWORD '<password>']
   [HOST PASSWORD '<password>'];
```

Identify the specified database as the "default" database. The default database identifies the master database for replication commands, and a (n arbitrary) cluster member for clustering commands. Any necessary password may be specified.

7.5.5.2 Show Default Database

```
SHOW DEFAULT DATABASE;
```

Show the current default database.

7.5.6 Replicator Commands

7.5.6.1 Start Replicator

```
START REPLICATOR [PASSWORD <password>] [OPTIONS <option> ...];
```

Start a replicator for the default database, using the specified options, if any.

7.5.6.2 Create Client

```
CREATE CLIENT <database-name>
   [ON | @ | HOST <host-name>]
   [DATABASE PASSWORD <password>]
   [PASSWORD <system-password>];
```

Works like Add Client below, except that if the client database does not exist, a copy of the master database is made (which requires a server for the master database to be running), and if a server for the client database is not running, it is started.

7.5.6.3 Add Client

```
ADD CLIENT <database-name>
   [ON | @ | HOST  <host-name>]
   [DATABASE PASSWORDS <password>]
   [PASSWORD <password>];
```

Add the client named <database-name> on the host named <host-name> to the client list of the replicator for the default database. Whenever the replicator and the server for the client is both running, the replicator will keep the client up-to-date. The replicator may need the database password and the password for the _SYSTEM user of the client database, in which case they may be specified

7.5.6.4 Remove Client

```
REMOVE CLIENT <database-name>
```

```
   [ON | @ | HOST <host-name>];
```

Remove the client from the client list of the replicator for the default database. The client is no longer updated, but may later be added to the client list again.

7.5.6.5 Show Clients

```
SHOW CLIENTS;
```

Show the client list of the replicator for the default database, and the status for each client.

7.5.6.6 Stop Replicator

```
STOP REPLICATOR;
```

Stop the replicator for the default database.

7.5.7 Clustering Commands

7.5.7.1 Create Cluster

```
CREATE CLUSTER [OPTIONS <option> ...];
```

Create a cluster consisting of the default database (only). If the default database does not exist, a new database is created. Then a server for the default database is started, and any options needed to start this database must be specified; the -rcluster option is implicitly added.

7.5.7.2 Start Cluster

```
START CLUSTER;
```

Attempt to start servers for all members of the cluster identified by the default database. The servers will be started in the proper order.

7.5.7.3 Stop Cluster

```
STOP CLUSTER;
```

Attempt to stop any running servers for all members of the cluster identified by the default database.

7.5.7.4 Show Cluster

```
SHOW CLUSTER [DESCRIPTOR | ALL];
```

Display properties of the cluster identified by the default database. DESCRIPTOR, if specified, directs certain static information about the cluster to be displayed whereas ALL, if specified, requests further dynamic information to be displayed.

7.5.7.5 Set Cluster Descriptor

```
SET CLUSTER DESCRIPTOR;
```

Distribute the cluster descriptor associated with the cluster member identified by the default database to all members identified by said cluster descriptor. After this operation, all cluster descriptors of all members of a cluster should be consistent (provided that all members were accessible), and that is exactly the purpose of this command.

7.5.7.6 Add Member

```
ADD MEMBER <database-name>
   [ON | @ | HOST <host-name>] [OPTIONS <option> ...];;
```

Add the specified database as a member to the cluster identified by the default database. If the specified database does not exist, a new database is created, and it must be possible to bring the new member up-to-date from existing members of the cluster. A server for the new member is started, and any options needed to start this database must be specified; the -rcluster option is implicitly added.

7.5.7.7 Remove Member

```
REMOVE MEMBER <database-name>
   [ON | @ | HOST <host-name>];
```

Remove the specified database from the cluster identified by the default database.

7.5.7.8 Start Member

```
START MEMBER <database-name>
   [ON | @ | HOST <host-name>] [OPTIONS <option> ...];
```

Attempt to start a server for the specified cluster member. If any options are specified, they overwrite existing options for the database, and the -rcluster option is implicitly added.

7.5.7.9 Stop Member

```
STOP MEMBER <database-name>
   [ON | @ | HOST <host-name>];
```

Attempt to stop a running server for the specified cluster member.

7.5.7.10 Show Member

```
SHOW MEMBER <database-name>
   [ON | @ | HOST <host-name>]
   [DESCRIPTOR | ALL];
```

Display properties of the specified cluster member. DESCRIPTOR, if specified, directs certain static information about the cluster to be displayed whereas ALL, if specified, requests further dynamic information to be displayed.

7.5.7.11 Set Member

```
SET MEMBER <database-name>
   [ON | @ | HOST <host-name>]
   MAJORITY TRUE | FALSE;

SET MEMBER <database-name>
   [ON | @ | HOST <host-name>]
   READONLY TRUE | FALSE;
```

Set the specified property to the specified value for the specified cluster member. The explicit setting of the readonly property to FALSE opens for the possibility to create inconsistencies between cluster members, and should only be done by someone knowledgeable.

7.5.8 Autostart Commands

7.5.8.1 Autostart

```
AUTOSTART
   [ON | HOST <host-name>];
```

With the autostart command the user can control which databases are started on the host named <host-name>, when it is restarted. If <host-name> is not specified, localhost is used. The autostart command has the following sub-commands, and no other commands are accepted until the autostart command loop is exited:

```
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> [<option> ...];
```

Add a database to the list of autostarted databases. The server is started with the specified options.

```
OPTIONS <number> <option> ...;
```

Set the options for the selected database.

```
DELETE <number>;
```

Remove the specified database from the list 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 for the password.

```
EXIT;
QUIT;
```

Exit the autostart command loop.

7.5.9 Show Commands

7.5.9.1 Show Usage

```
SHOW USAGE;
```

Print a summary of the sessions for the database currently connected.

7.5.9.2 Show Database Log

```
SHOW DATABASE LOG <database-name>
   [ON | @ | HOST <host-name>];
```

Show the tail of the database log file for the database named <database-name> on the host named <host-name>. If the host name is not specified the local host is assumed.

7.5.9.3 Show Databases

```
SHOW DATABASES
   [ON | HOST <host-name>];
```

Show the list of databases on the host named <host-name>. If <host-name> is not specified, localhost is used

7.5.9.4 Show Database

```
SHOW DATABASE [FULL];
```

Show information about the currently connected database. If FULL is specified all available information is shown.

7.5.9.5 Show License

```
SHOW LICENSE [FULL];
```

Show license details.

7.5.9.6 Show Logs

```
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 n transaction log files are displayed. The default is 1.

7.5.9.7 Show Memory

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

Show the status of the memory allocated by the database server currently connected. Please refer to the memory use section in the Administration chapter.

7.5.9.8 Show Caches

```
SHOW CACHES;
SHOW CACHES <table-name>;
SHOW CACHES <schema-name>.<table-name>;
```

Show 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 _.

7.5.9.9 Show Transaction

```
SHOW TRANSACTION [FULL];
```

Show the current transaction parameters.

7.5.9.10 Show Schema

```
SHOW SCHEMA <schema-name> [FULL];
```

Show the tables and views defined in the specified schema. If FULL is specified all available information is printed.

7.5.9.11 Show Table

```
SHOW TABLE <table-name> [FULL];
```

Show the definition of the table. If FULL is specified all available information is printed.

7.5.9.12 Show View

```
SHOW VIEW <view-name> [FULL];
```

Show the view definition. If FULL is specified all available information is
printed.

7.5.9.13 Show Size

```
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 specified tables in the specified schemas. The schema name and the table name may contain the SQL 92 wild card character % and _.

7.5.9.14 Show History

```
SHOW HISTORY;
```

Show the command history.

7.5.10 Agent Commands

7.5.10.1 Stop Agent

```
STOP AGENT <agent-number>;
INTERRUPT AGENT <agent-number>;
```

The first form terminates the agent specified by <agent-number>. The second form terminates the operation that the specified agent is currently executing.

7.5.11 Set Commands

7.5.11.1 Set output

```
SET OUTPUT [<file-name>];
```

Direct all output to the specified file. Output lines in the file are written without the leading >. If the filename is omitted all output is written to stdout.

7.5.11.2 Set Format

```
SET FORMAT ROW|COLUMN|XML|SUPPRESS|<'template'>;
```

Controls the format used to print rows received from the server. If ROW is specified one row is written per line, if COLUMN is specified, the default, then one COLUMN is written per line, if XML is specified the output is written in XML format, and if '<template>' is specified each row is formatted according to the template. SUPPRESS means that no output is produced.

The template string may contain format specifiers:

```
<format> :== %{<column>}{.<width>}<flag>
             %*{<width>}<flag>
             %{<width>}[<template>|<template>]
             %%
<column> :== positive integer
<width>  :== positive integer
<flag>   :== c              -- column value
             l              -- column label
             T              -- table name
             S              -- schema name
             N              -- column name
             #              -- the row number in the current result
set
             r              -- the result set number
```

If the column or the width is omitted they are interpreted as 0.

The expanded string is padded with spaces to the width.

%{<width>}[<template>|<template>] iterates over all columns and expands the template for each. If the <column> is not specified inside the iterator the current column is referenced. The first column is expanded using the first template and the following columns by expanding the second.

The %*<flag> form is a shorthand for %[%<flag> | <%flag>].

Other expansions:

%%	%
\n	new line
\r	carriage return
\t	tab
\ddd	to the decimal unsigned byte values
\c	to the character

7.5.12 Timing Commands

7.5.12.1 Start Stopwatch

```
START STOPWATCH;
```

Start the stopwatch.

7.5.12.2 Lap Time

```
LAPTIME [<comment>];
```

When sql92 is started with -c or -n write the elapsed time since the stopwatch was started and the comment to standard output.

7.5.12.3 Stop Stopwatch

```
STOP STOPWATCH [<comment>];
```

When sql92 is started with -c or -n write the elapsed time since the stopwatch was started and the comment to standard output, and stop the stopwatch.

7.5.12.4 Sleep

```
SLEEP <seconds>;
```

Sleep for <seconds> seconds.

7.5.12.5 Show Timezone

```
SHOW TIMEZONE;
```

Show the name of the current time zone.

7.5.13 Other Commands

7.5.13.1 Define Blob and Define Clob

```
DEFINE BLOB <blob-name> LENGTH <length> VALUE {<hex-bytes>};
DEFINE CLOB <clob-name> LENGTH <length> VALUE {<hex-bytes>};
DEFINE CHAR <char-name> LENGTH <length> VALUE {<hex-bytes>};
```

Define a blob (or a clob, or a char string) object with the name <name>, length <length> in bytes, and the value that 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, or char string) object is defined and can be referenced in subsequent SQL 92 statements. A reference has the form @'<name>' and will create the blob (or clob, or char string) object for that particular column. The blob (or clob, or char string) object will disappear when the next commit or rollback is executed.

7.5.13.2 Script

```
SCRIPT <path-to-file>;
```

Read and execute the sql92 commands from the specified file.

7.5.13.3 Exit and Quit

```
EXIT
QUIT
```

Exit sql92. The EXIT and QUIT commands are the only ones that do not need to be terminated with a ';'.

8 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:

8.1 FBExec Invocation

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 <FB home>/FrontBase/bin to your $PATH):

```
FBExec -daemon
```

Be sure to include the –daemon option so that the task doesn't end with your terminal session.

If you are running FrontBase on Mac OS X, you can use the Activity Monitor utility provided by Apple to check whether FBExec is running. 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 start-up.

8.1.1 Files

FBExec uses the following files, all located in the FrontBase installation directory:

FBExec.autostart
Contains the specification of databases to be automatically started when the FBExec is started, which typically happens when the FrontBase server machine is booted. This file is best maintained using FrontBaseManager or the sql92 tool.

FBExec.log
FBExec may be requested to log information concerning most FBExec events to this log file. By default, this mechanism is disabled.

FBExec.passwd
For any FrontBase installation, server authentication may be enabled. This has the form of a password which must be supplied for the FBExec to carry out any operation on a FrontBase database. When enabled, the password is written to this file in digested form.

8.1.2 Options

FBExec obeys the following invocation options:

-autostart	Start databases on FBExec start-up, as specified in the FBExec.autostart file
-console	Relevant for Windows systems only: Allow FBExec to run from a command line
-daemon	Run FBExec in the background, ie without controlling terminal
-log	Enable logging into the FBExec.log file
-nolog	Disable logging into the FBExec.log file. This is the default
-oldpasswd	Remove an existing server authentication password
-newpasswd	Define a new server authentication password
-version	Display the current FBExec version

8.2 FrontBase Invocation

8.2.1 Files

FrontBase writes the following files, all located in the FrontBase installation directory:

Backups/<database-name>**.fb/B_**<yyyy_mm_dd-hh:mm:ss>
The default location for backup files for the named database. The name of the backup file identifies the point in time when the backup was initiated. In reality, backup files may be placed at arbitrary location s in the file system, by specifying their path at the point of initiating them.

Databases/<database-name>**.fb**
The main file for the named database. If no devices other than the default are defined (see Storage Management on page 97) this file contains all data in the database.

Databases/<database-name>**.fb.log**
Log information mainly concerning client connections and session identifications.

Databases/<database-name>**.fb.pid**
The PID of the FrontBase Server process managing the named database. If no Server is currently active for the database, the PID of the latest Server is found here.

Databases/<database-name>**.fb.sql**
All SQL statements presented to the FrontBase Server, provided SQL logging has been enabled.

Databases/<database-name>**.cluster**
The description of the cluster composition, if the named database is part of a cluster setup.

TransactionLogs/<database-name>**/L_**<yyyy_mm_dd-hh:mm:ss>
A Transaction Log directory for the named database. The directory contains the file **transactions.log**, and possibly other files, altogether making up a part of the Transaction Log for the named database. New Transaction Log directories are created for various reasons, and the one with the latest date identification is the current Transaction Log directory.

TransactionLogs/<database-name>**/.lock**
A Lock File enabling FrontBase Servers to ensure that no more than one Server is active for the named database at any one point in time.

8.2.2 Options

FrontBase sports an extensive list of so called options that can be specified when a FrontBase Server is started. The options can be specified on the command line or when using e.g. FrontBaseManager or the sql92 tool to start a FrontBase Server.

autocommit	Automatically COMMIT all transactions after each SQL statement

autocreate	Automatically create new user names as database clients connect
clientLimit	Limit number of concurrent clients
clients	Exit (or not) after database initialisation
console	Run as command-line tool in background (relevant for Windows only)
core	Specify maximum size of core file
create	Create a new database unconditionally
daemon	Run in background (ie, without controlling terminal)
fbexec	Run with or without the FBExec process
index	Specify the default mode for new indexes
init	Initialize database from specified file
keeptlog	Prevent deletion of transaction logs on database creation
key	Specify encryption key for data encryption on disk
localonly	Accept only local connections
logSQL	Log SQL statements to a file as they are received
majority	Override majority property of cluster member
port	Specify a port number to use for connection creation
prvchk	Disable privilege checks
rmaster	Run as replication master
rclient	Run as replication client
rcluster	Run as member of a cluster
rdd	Enable the Raw Device Driver
replicator	Start a replicator for database
restore	Restore from a backup
rlpriv	Enable Row Level Privileges
rollforward	Control database update from transaction log
scomm	Enable client communication encryption
sdisk	Enable data encryption on disk
tlog	Disable transaction logging
transaction	Establish database transaction number
triggers	Enable trigger mechanism
vmlimit	Specify maximum size of process vitual memory
version	Display FrontBase version identification, and exit

8.2.2.1 autocommit

```
-autocommit[=no|yes]      yes is the default
```

FrontBase offers an auto commit feature whereby a transaction is automatically committed after the SQL statement that initiated the transaction has been successfully executed. All new client connections inherits the general setting of the auto commit feature.

An individual client connection may turn its auto commit feature 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 database is started, the auto commit feature is turned off.

8.2.2.2 autocreate

```
-autocreate[=no|yes]      yes is the default
```

By specifying –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 database is started, unknown users will not be allowed to connect.

8.2.2.3 clientLimit

```
-clientLimit=<client number>
```

By specifying –clientLimit, The maximum number of concurrent client connection accepted by the database server is limited to the specified number. Otherwise, this number is given by a system default.

8.2.2.4 clients

```
-clients[=no|yes]      yes is the default
```

By specifying –clients=no, the server will exit at the point in its start-up where it would otherwise permit client connections. The effect of this is that the database has been brought up-to-date with respect to its transaction log.

8.2.2.5 console

```
-console
```

By specifying –console, the server may run as a command line tool, rather than as a service. Relevant for Windows only.

8.2.2.6 core

```
-core=no|yes|<limit in MB>
```

This option controls the creation of core files, should the FrontBase server crash. –core=no disables the creation of core files, –core=yes limits the size of a core file to 1.500 MB, and otherwise the maximum size of a core file may be specified.

If the option is omitted when a FrontBase database is started, the creation of core files is diabled.

8.2.2.7 create

```
-create[=no|yes]      yes is the default
```

By specifying –create=yes, a new database is unconditionally created, overwriting an existing database with the same name. This irreversibly deletes the existing database, so the option should be used with care.

8.2.2.8 daemon

```
-daemon
```

By specifying –daemon, FrontBase executes in the background, decoupled from any controlling terminal. The option –b is synonymous with this option.

8.2.2.9 fbexec

```
-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.

8.2.2.10 index

```
-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.

The mode of the indexes created for a given table can be changed by executing one of the following SQL statements:

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

8.2.2.11 init

```
-init=<filename>
```

The specified file is used for initializing the newly created database. This implies that –init may not be specified for an existing database. The specified file must contain SQL statements (only).

8.2.2.12 keeptlog

```
-keeptlog
```

This option prevents the transaction log from being deleted, in situations where this would otherwise happen (eg, when –create has been specified).

8.2.2.13 key

```
-key=<filename>
```

FrontBase offers a high degree of protection of your data by deploying a unique encryption scheme (see Encryption on page 210). Actual data stored on the hard disk may be encrypted, using the encryption keys found in the specified file

8.2.2.14 localonly

```
-localonly
```

This option specifies that the FrontBase server must 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.

8.2.2.15 logSQL

```
-logSQL
```

This option specifies that the FrontBase server should log all 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 ForewordSQL statements.

8.2.2.16 majority

```
-majority=no|yes
```

This option overrides the implicit calculation of the majority property of the starting FrontBase cluster member – see Provisions for Absent Servers on page 81. Specifying majority to be TRUE (–majority=yes) enables the starting cluster member to commit transactions. If this is in contrast to the result of the implicit calculation, the starting cluster member may commit transactions causing a divergence of the clustered database.

WARNING! This option should be used with extreme caution.

8.2.2.17 port

```
-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. set-ups with a firewall, static and known port numbers may have to be used. By using the –port option, a FrontBase database can be directed to use a specific port number.

8.2.2.18 prvchk

```
-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, and existing apps should work as with FrontBase 1.2.

8.2.2.19 rmaster , rclient, rcluster

```
-rclient
-rcluster
-rmaster
```

These options specify the role of the starting FrontBase database wrt. replication and clustering. –rclient specifies that the database is a replication client (and as such, ,a read-only database for any client but the FrontBase Replicator), –rcluster specifies that the database is a member of a cluster, and –rmaster specifies that the database may serve as a replication master.

8.2.2.20 rdd

```
-rdd[=<size in MB>]
```

The Raw Device Driver offered by FrontBase is an advanced write-through cache mechanism, that also supports use of a raw device (partition) as data storage, hence the name. This mechanism is described in Raw Device Driver (RDD) on page 111.

The size of the cache need only be given the first time the option is specified for a given database or when the size needs to be decreased or increased.

8.2.2.21 replicator

```
-replicator[=”<option string>”]
```

This option specifies that a FrontBase Replicator should (also) be started for the starting database. This implies that the starting database should serve as a replication master. If an <option string> is given, it is passed to the Replicator as the options for its start-up.

8.2.2.22 restore

```
-restore[=<filename>]
```

This option specifies that the starting database should be initialized from a FrontBase backup. If <filename> is specified, it is a full path identifying the backup file, and if not, the backup file is taken to be the newest found in the default location (<FB home>/FrontBase/Backups/<database>.fb). See Backup and Restore on page 85.

8.2.2.23 rlpriv

```
-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 database. The –rlpriv option need only be given when a new database is created.

Refer to SELECTing the Access Privileges for a Row on page 225 for more information.

8.2.2.24 rollforward

```
-rollforward=no|<transaction number>
```

This option controls how far the database is brought up-to-date with respect to its transaction log, if present. If –rollforward=no is specified, no transactions from the transaction log are executed, and otherwise transactions up to and including the specified number are executed.

8.2.2.25 scomm

```
-scomm
```

FrontBase offers a high degree of protection of your data by deploying a unique encryption scheme – see FrontBase and Security on page 209. This option specifies that the communication between clients and the FrontBase database must be encrypted (streaming)

8.2.2.26 sdisk

```
-sdisk
```

FrontBase offers a high degree of protection of your data by deploying a unique encryption scheme – see FrontBase and Security on page 209.. This option specifies that the actual data store on the hard disk must be encrypted (block mode); the encryption keys must be specified with the –key option.

8.2.2.27 tlog

```
-tlog=no|yes
```

This option overrides the status of transaction logging for the duration of this activation of the FrontBase server. If –tlog=no is specified, transaction logging is disabled, and otherwise it is enabled. In normal operation, we strongly discourage disabling of transaction logging.

8.2.2.28 transaction

```
-transaction=<transaction number>
```

This option specifies the starting point of bringing the starting database up-to-date with respect to its transaction log. The transaction number stored in the database is overridden by the specified number.

8.2.2.29 triggers

```
-triggers[=no|yes]      Default is no
```

This option controls the trigger mechanism defined by SQL 99 and described in SQL 99 Triggers on page 96. If –triggers=yes is specified, support for triggers is enabled, otherwise it is disabled.

8.2.2.30 vmlimit

```
-vmlimit=<limit in MB>
```

This option specifies the maximum size of the virtual memory of the FrontBase server process. If this option is not specified, the maximum size is dictated by system constraints.

8.2.2.31 version

```
-version
```

This option makes the FrontBase server display its version number, and then exit. Options --version and –v are synonyms.

8.3 FBReplicator Invocation

8.3.1 Files

TransactionLogs/<database-name>**/L_**<yyyy_mm_dd-hh:mm:ss>
A Transaction Log directory for the named database. The FBReplicator monitors the current Transaction Log directory for the database that it services, and it reads the SQL written to these directories by the FrontBase Server, and distributes this SQL to the replication clients for the named database.

TransactionLogs/<database-name>**/FBRSClientList.txt**
The description of the replication clients for the named database. This file is maintained exclusively by the FBReplicator.

TransactionLogs/<database-name>**/.replic**
A Lock File enabling FBReplicators to ensure that no more than one FBReplicator is servicing the named database at any one point in time.

8.3.2 Options

The Replicator accepts the following options:

-b	Run as a background program (i.e., without controlling terminal).
-s	Silent. The Replicator will not log events to stdout (default).
-v	Verbose. The Replicator will log events to stdout.
-i <interval>	Seconds. The polling interval determines how often the Replicator will examine the transaction log of the master. The Replicator will only sleep the specified amount of time when no transactions are pending replication. The default value is 10 seconds.
-d <transactionlog-directory>	Specifies the transaction log directory to be used for replication. To be used when the transaction log is not in the default location.
-p <portno>	Specifies the port number at which the Replicator will talk to the controlling program (e.g., FBRAccess)
-k <keyfile>	Specifies the file containing the encryption key for an encrypted database. Must be specified when database is encrypted.

8.4 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 passwords for authorization, encryption of communication and data storage, 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.

8.4.1 Password Protection

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 a password to the FrontBase client software, a one-way function is applied to generate a password digest. The function throws 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.

Two layers of password protection of database access are provided: Database password and user passwords.

A different level of password protection is offered by the so-called Server Authentication which, when enabled, requires password authorization of FBExec operations like creating, starting, stopping and renaming FrontBase databases.

8.4.1.1 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 FrontBase server cannot verify the password, the client connection is closed down immediately.

8.4.1.2 User Password

Each database user can have a password, the password is verified by the FrontBase 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 SQL 92 defined protection takes over.

8.4.1.3 Server Authentication

FrontBase has an authentication feature that 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, run:

```
FBExec -newpasswd=secret
```

Henceforth, the creation of databases, starting them, stopping them, etc., will require password authentication. Access to individual databases can be further secured through the use of database passwords and user passwords.

8.4.2 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, and optionally specify that communication channels between the server and its clients must be secure.

8.4.2.1 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, so that blocks with the same contents never generate identical cipher text blocks.

The key used for encryption of data is a 64 bits initialization vector, and 3x56 bits for the DES encryption.

FrontBase also offers a much simpler encryption in the form of XOR'ing a 128-bit key onto data. The XOR operation includes position dependent data, so that identical blocks generate distinct cipher text blocks.

8.4.2.2 Secure Channels

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 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.

8.4.3 Tools and Options

A few tools are provided to support key management, and FrontBase has a number of options related to security.

8.4.3.1 FBKeyGenerator

The key generator is used to generate a set of keys that can be used by FrontBase:

```
FBKeyGenerator des | xor [<key-filename>]
```

FBKeyGenerator may generate keys for DES encryption as well as for a much simpler XOR encryption. If the key-filename is specified, the keys are written to that file, otherwise the keys are written to standard output. Each key set is assigned a key identification.

Example key text:

DES f756cc7e 0e589bc4b68ab745 c80c0621e2472f64 eaecef3047bdad1c ae0af69be877b4c9

The first 4-byte number is the key identification.

8.4.3.2 FrontBase

FrontBase has three security options:

-scomm	If present, communication between the server and its clients is secure
-sdisk	If present, data stored on the disk are encrypted with a specified encryption key
-key=<file-name>	Encryption keys are read from the named file.

If -sdisk is specified the server requires a set of keys in order to operate. If -key 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 have been specified.

8.4.3.3 FBChangeKey

The FBChangeKey tool can be used to encrypt a database that was not previously encrypted, to change the keys used for encryption, or decrypt an encrypted database.

```
FBChangeKey <database-file> [<key-filename> [<key-filename>]]
```

If no key-filenames are provided, one or two key 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 first key, and encrypted with the second.

8.4.4 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 address is blacklisted the connection is refused, if it 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 address is white listed you can specify whether you require a secure communication channel to that IP address. 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.

8.5 Data Types

SQL 92 offers an extensive list of data types all of which are supported by FrontBase. Additionally, FrontBase supports a number of data types from SQL3. Although the list of data types seems long and maybe even confusing, don't worry, many of the names denote the same data type (such is the work that comes from a committee).

8.5.1 TINYINT

Implemented as an 8-bit integer.

```
CREATE TABLE t0(c0 TINYINT, ...);
```

8.5.2 SMALLINT

Implemented as a 16-bit integer.

```
CREATE TABLE t0(c0 SMALLINT, ...);
```

8.5.3 INTEGER, INT

Implemented as a 32-bit integer. Apart from the obvious use, this data type 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.

```
CREATE TABLE t0(c0 INTEGER PRIMARY KEY, ...);
```

8.5.4 LONGINT

Implemented as a 64-bit integer.

```
CREATE TABLE t0(c0 LONGINT, ...);
```

8.5.5 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 data type 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) data types.

```
CREATE TABLE t0(c0 DECIMAL, PROFITS DECIMAL(20,2), ...);
```

8.5.6 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).

```
CREATE TABLE t0(c0 NUMERIC, SALARY NUMERIC(10,2), ...);
```

8.5.7 FLOAT[(<precision>)]

Implemented as a 64-bit integer + 32 bits to hold sign and exponent. Default value for <precision> is 19 (the max.).

```
CREATE TABLE t0(c0 FLOAT, C1 FLOAT(10), ...);
```

8.5.8 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 maximum precision when using FLOAT.

```
CREATE TABLE t0(c0 REAL, ...);
```

8.5.9 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 Mapping of Foundation/Java objects into FrontBase Data Types on page 220 for details.

```
CREATE TABLE t0(c0 DOUBLE PRECISION, ...);
```

8.5.10 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, occupy two bytes.

NOTE: The max. length of a CHARACTER value is 2GB.

```
CREATE TABLE t0(c0 CHAR(1), c1 CHARACTER(100000), ...);
```

8.5.11 NATIONAL CHARACTER, NATIONAL CHAR, NCHAR

As FrontBase supports Unicode exclusively, the NATIONAL CHARACTER data type is mapped into CHARACTER.

```
CREATE TABLE t0(c0 NATIONAL CHAR(1), c1 NCHAR(100000), ...);
```

8.5.12 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 may be specified to be stored only once.

NOTE: Since FrontBase encodes varchars very efficiently, use of variable length strings is in general recommended over fixed length strings.

```
CREATE TABLE t0(c0 VARCHAR(128), c1 CHARACTER VARYING(200000), ...);
```

8.5.13 NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING, NCHAR VARYING

As FrontBase supports Unicode exclusively, the NATIONAL CHARACTER VARYING data types are all mapped into CHARACTER VARYING.

```
CREATE TABLE t0(c0 NATIONAL CHAR VARYING(10), c1 NCHAR
VARYING(10000), ...);
```

8.5.14 BIT

The bit data type is conceptually a string of 1's and 0's, but is implemented as an opaque binary data type, i.e. BIT(8) occupies one byte. See below as concerns EOF and BYTE.

```
CREATE TABLE t0(c0 BIT(32), C1 BIT(256)...);
```

8.5.15 BIT VARYING

As BIT, but with the obvious exception that the bit strings are variable in length.

```
CREATE TABLE t0(c0 BIT VARYING(32), c1 BIT VARYING(256)...);
```

8.5.16 BYTE

A simple wrapper for BIT, i.e. BYTE(n) is identical to BIT(n*8). This data type is not part of the SQL 92 standard, but has been introduced to better support EOF's automatic primary key generation. If you use 12-byte binary keys, EOF can automatically generate a primary key without doing a roundtrip to the database server (and thus cause a transaction to be initiated).

```
CREATE TABLE t0(c0 BYTE(12), ...);
```

8.5.17 DATE

The traditional date data type. 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.

```
CREATE TABLE t0(c0 DATE, ...);
```

8.5.18 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 server's time zone, i.e. the server's time zone is applied to the time value when it is inserted.

```
CREATE TABLE t0(c0 TIME, ...);
```

8.5.19 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.

```
CREATE TABLE t0(c0 TIME WITH TIME ZONE, ...);
```

8.5.20 TIMESTAMP

Holds a complete timestamp value that 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 server's time zone, i.e. the server's 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!

```
CREATE TABLE t0(c0 TIMESTAMP, ...);
```

8.5.21 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 data type is needed if you want to be in complete control over how time zone information is stored and displayed.

```
CREATE TABLE t0(c0 TIMESTAMP WITH TIME ZONE, ...);
```

8.5.22 INTERVAL

INTERVAL is actually two separate data types: a data type called year-month interval and a data type 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, ..);
```

8.5.23 BLOB

A Binary Large OBject is an opaque binary data type, 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 as well as on the client side, i.e. no unnecessary copying. A BLOB value can be up to 2 GB in size.

```
CREATE TABLE t0(c0 BLOB, ...);
```

8.5.24 CLOB

A Character Large OBject is a data type 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.

```
CREATE TABLE t0(c0 CLOB, ...);
```

8.5.25 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).

```
CREATE TABLE t0(c0 BOOLEAN, ...);
```

8.6 Mapping of Foundation/Java objects into FrontBase Data Types

The following is a brief description of the recommended mapping of the most common Foundation/Java objects into FrontBase data types.

8.6.1 String

	Suggested Data Types	**Recommended Data Type**
NSString java.lang.String java.io.Reader (for CLOB)	CHARACTER VARCHAR CLOB	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 data type can be a good choice. The main advantage of the CLOB data type is that the string is sent to the database separately. That is, rather than 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 and decoding taking place on the client side, i.e. handled by the native methods of NSString/java.lang.String.

8.6.2 Integer Numbers

	Suggested Data Types	**Recommended Data Types**
NSNumber java.lang.Number	BOOLEAN, SMALLINT, INTEGER, FLOAT, REAL, DOUBLE PRECISION	INTEGER DOUBLE PRECISION depending on actual use

NOTE: As NSNumber/Number is a cover for the C/Java number data types, the FrontBase type to choose depends on the actual use.

8.6.3 Decimal Numbers

	Suggested Data Types	Recommended Data Types
NSDecimalNumber java.math.BigDecimal	NUMERIC, DECIMAL, REAL, FLOAT, DOUBLE PRECISION	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).

8.6.4 Dates

	Suggested Data Types	Recommended Data Type
NSCalendarDate java.sql.Date java.sql.Timestamp	DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE	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 client's time zone.

8.6.5 Time

	Suggested Data Types	Recommended Data Type
java.sql.Time	TIME, TIME WITH TIME ZONE	Either

NOTE: java.sql.Time is a simple wrapper (or just a plain gruesome hack) around java.sql.Date.

8.6.6 Stream Data

	Suggested Data Type	**Recommended Data Type**
NSData java.io.InputStream	BLOB	BLOB

8.6.7 Primary Key

	Suggested Data Types	**Recommended Data Type**
	INTEGER, BYTE(12)	Either

NOTE: The BYTE(12), which is a cover for BIT(96), data type should be used if the generation of primary key is done by Apple'a Enterprise Objects Framework (EOF) (client side calculation facility). Any FrontBase data type can be used for a primary key and since all numeric data types are represented as exact numeric values, it is safe to use e.g. DOUBLE PRECISION or TIMESTAMP as a data type for a primary key column. However, EOF does not allow certain data types (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.

8.7 Primary Keys and Auto Generation

FrontBase supports, as required by the SQL 92 standard, multiple-column primary keys, but in the case of single-column integer primary keys, FrontBase can help you in generating such keys.

8.7.1 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>;
```

that returns a single row with a single integer column. The SELECT UNIQUE construct can also be used as a scalar sub-query:

```
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 an INSERT statement, the default will be used.

8.8 Row Level Privileges

8.8.1 Defining

FrontBase offers a unique feature called Row Level Privileges, 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.

8.8.2 Deploying

To use the Row Level Privileges feature, a given database has to be initialized with the feature given as an option:

```
FrontBase -rlpriv <database-name>
```

You can also specify the -rlpriv option when creating a database via the FrontBaseManager.

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.

8.8.2.1 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);
```

8.8.2.2 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
```

8.8.2.3 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>';
```

8.9 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.

8.9.1 International Characters

FrontBase implements Unicode and thus supports use of all the so-called international character sets including Kanji, Hangul etc. The positional value 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.

8.9.2 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 for doing case insensitive compares.

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, including compares done when building an index.

Example:

```
INSERT INTO t0(db) VALUES 'frontbase', 'FrontBase', 'FRONTBASE';
COMMIT;

SELECT db FROM t0 WHERE db = 'FrOnTbAsE';     -- returns 3 rows

SELECT db FROM t0 WHERE db LIKE ‘f%’;         -- returns 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;
      --returns 1 row

SELECT db FROM t0 WHERE db LIKE 'F%' COLLATE CASE_SENSITIVE;
      -- returns 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? Easily done:

```
SELECT db FROM t0 WHERE
   db = 'FrontBase'
      AND
   db = 'FrontBase' COLLATE CASE_SENSITIVE;
```

The first WHERE clause will return 'frontbase', 'FrontBase', 'FRONTBASE', while the second WHERE clause will reduce the result to 'FrontBase';

8.10 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 independently 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 is not used.

2) An application or solution-specific license string is to be used. This license string is generic, i.e. it is not tied to a specific IP or MAC address.

3) The FBExec, which is like a DNS service for FrontBase databases on a given host, is not 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.

8.10.1 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/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 housekeeping files incl. files used during bootstrapping of a new database. If a parent application includes a pre-bootstrapped 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.
FBSQLErrors.array	This 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.
.ucm	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.
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.

8.10.1.1 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
```

8.10.2 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>:<FB home>\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>:<FB home>\FrontBase\
```

8.10.3 Starting the FrontBase Server – non-Windows Platforms

The server is simply started as any other background application:

```
<FB home>/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.

8.10.3.1 How to Tell the Server what Port Number to use

The port number is hard-coded in the LicenseString file, any use of the –port <number> option is ignored.