



Bizgres 0.9

User Guide

© Greenplum, Inc. 2006. All Rights Reserved.

Bizgres is a pending trademark of Greenplum, Inc. Use of Bizgres is granted under the following irrevocable license: All users of Bizgres are perpetually and irrevocably granted the fair use right to use the name Bizgres to refer to the Bizgres source code, binaries, and derivative works and products created using that code. The exception to this grant of fair use is that you may not name your company Bizgres or any name containing Bizgres which could reasonably cause confusion with the Bizgres project.

All other trademarks are the property of their respective owners. PostgreSQL is a trademark of Marc Fournier held in trust for The PostgreSQL Development Group. GreenPlum is a trademark of GreenPlum Inc.

Greenplum makes no warranty of any kind with respect to the completeness or accuracy of this manual.

Contents - Bizgres 0.9 User Guide

Chapter 1: Overview	1
About Bizgres	1
What's New in Bizgres 0.9?	1
Bizgres Contents and Features	2
Getting Started with Bizgres	3
Chapter 2: Installation	5
Installing Bizgres Using the Greenplum Installer	5
System Requirements	5
Running the Installer	6
Uninstalling Bizgres	6
Installing Bizgres From Source	7
Before You Begin	7
Building the Bizgres Source	8
Post Installation Steps	9
Configuring Your Installation	10
Designating a Bizgres User Account	11
Setting Bizgres Environment Variables	11
Verifying Your Installation	11
Creating the Database Cluster	11
Starting the Database Server	12
Creating a Database	13
Chapter 3: Working in psql	15
Connecting to a Database	15
Entering SQL Commands	16
psql Meta-Commands	16
Creating Tables	17
Chapter 4: Using Bitmap Indexes	18
About Bitmap Indexes	18
When to Use Bitmap Indexes	19
When Not to Use Bitmap Indexes	19
Bitmap Index Example	19
Creating a Bitmap Index	21
Examining Bitmap Index Usage	21
Maintaining Bitmap Indexes	21
Altering a Bitmap Index	21
Deleting a Bitmap Index	22
Reindexing After Updates/Deletes	22

Chapter 5: Using the Bizgres Loader	23
Overview	23
Before You Begin	23
Running Bizgres Loader	24
Running Bizgres Loader in Regular Mode	24
Running Bizgres Loader in Batch Mode	24
Running Bizgres Loader in Preview Mode	25
Bizgres Loader Log Files	26
Updating Database Statistics After Data Loads	26
Vacuuming the Database After Load Errors	26
Chapter 6: Bizgres Demo Programs	27
Installation Verification Program (IVP)	27
Prerequisites for Solaris Users	27
Installing and Running IVP	27
Removing IVP	29
Bizgres Clickstream	29
Appendix A: Command Reference	30
loader.sh	30
Appendix B: Bizgres Environment Variables	37
Additional Environment Variables	38

1

Overview

This chapter provides overview information about the Bizgres download distribution. It contains the following topics:

- [About Bizgres](#)
- [Getting Started with Bizgres](#)

About Bizgres

Bizgres is the first open source, production-ready database server focused exclusively on supporting Business Intelligence (BI) applications. Bizgres is a product that is driven directly from the efforts of Bizgres.org, a Greenplum-sponsored and community-supported open source project, the mission of which is to build a comprehensive BI database platform on top of PostgreSQL 8.1.3. The Bizgres distribution from Greenplum is a compiled, packaged, documented, and tested release of the Bizgres open source project.

What’s New in Bizgres 0.9?

The Bizgres 0.9 release includes the following new features and enhancements:

- **PostgreSQL 8.1.3 compatability.** The previous release of Bizgres was based on PostgreSQL 8.0.4. This release includes all of the features, fixes, and patches of the PostgreSQL 8.1.3 release. See the [PostgreSQL 8.1 Release Notes](#) for more information.
Previous Bizgres-only features such as bitmap scan, bypassing WAL logging for `CREATE TABLE AS SELECT` commands, `COPY` performance enhancements, and table partitioning are now incorporated into PostgreSQL 8.1.3.
- **Bitmap Index.** The use of bitmap indexes in data warehousing environments can dramatically improve response times for large classes of ad hoc queries and reduce storage requirements compared to other indexing techniques. The advantages of using bitmap indexes are greatest for columns with a low degree of cardinality. For more information on this powerful new feature, see [Chapter 4, “Using Bitmap Indexes”](#).
- **Sort Performance Enhancements.** Several sort performance enhancements planned for the next PostgreSQL release are available now in Bizgres 0.9. These enhancements include:
 - The query executor has been improved to allow information to pass between query plan nodes for complex SQL statements, which removes unnecessary plan steps involving *Materialize* and *Sort*. In particular, Sort/Merge joins run faster as a result.

- The first column of the sort key is now pre-extracted from the rows being sorted, improving sort performance considerably by reducing repeated extractions. This is particularly important for Sort/Merge, Sort/Aggregate, and ORDER BY operations when the columns being sorted are not the leading columns in the rows being sorted. Performance improves the more unique the first column is.
- The sort algorithm now produces a number of intermediate runs limited only by available memory. This removes an earlier limitation in the algorithm that derived from its origins as a tape-sorter. Very large sorts (larger than 5 times the *work_mem* setting) will improve considerably as a result.
- **COPY Enhancements.** The `COPY` command has been enhanced to allow for more flexible escaping in the data input file. The default `ESCAPE` character is a `\` (backslash), however it is possible to specify any other character to represent an escape. It is also possible in Bizgres to disable escaping by specifying the value `'OFF'` as the `ESCAPE` value. This is very useful for data such as web log data that has many embedded backslashes that are not intended to be escapes. For example:

```
COPY web_address FROM 'my_input_file' ESCAPE 'OFF';
```

Bizgres Contents and Features

Bizgres is comprised of the following components and features (in addition to the features described in “What’s New in Bizgres 0.9?” on page 1):

- [PostgreSQL 8.1.3](#)
- [Bizgres Loader](#)
- [Demonstration Programs and Utilities](#)
- [KETL Integration](#)
- [JasperReports Integration](#)
- [Bizgres Clickstream](#)

PostgreSQL 8.1.3

The base engine for Bizgres is the PostgreSQL open source relational database management system (RDBMS). Bizgres includes all of the features and documentation of PostgreSQL 8.1.3, plus enhancements and modifications to support Business Intelligence (BI) applications.

Bizgres Loader

The Bizgres Loader is a Java command-line program to facilitate loading large quantities of data into a database. The functionality of the Bizgres Loader is similar to the PostgreSQL `COPY` command, but provides several additional features such as enhanced load performance, error logging, data batching, and configuration options.

Demonstration Programs and Utilities

The Bizgres distribution contains the following demonstration programs and utilities, which can be found in the `$BIZHOME/demos` directory of your Bizgres installation:

- **IVP** — The installation verification program (IVP) can be used to confirm that your Bizgres system is installed and configured properly.
- Greenplum also provides **Bizgres Clickstream**, an end-to-end business intelligence solution for web log loading, analysis, and reporting. See “[Bizgres Clickstream](#)” on page 3.

KETL Integration

Kinetic Networks (www.kineticnetworks.com) has contributed their Extraction, Transformation, and Loading (ETL) solution for web log analysis. This solution includes the KETL Java libraries, ETL scripts for extracting data from a web log, a scheduling engine, and schema DDL for creating the database tables in Bizgres. See “[Bizgres Clickstream](#)” on page 3 for information about getting the Bizgres Clickstream demo application, which utilizes KETL.

JasperReports Integration

JasperSoft (www.jaspersoft.com) has contributed their JasperReports solution along with several reports for web log analysis. JasperReports is a powerful open source Java reporting tool that has the ability to deliver rich content onto the screen, to the printer or into PDF, HTML, XLS, CSV and XML files. It is entirely written in Java and can be used in a variety of Java enabled applications, including J2EE or Web applications, to generate dynamic content. Its main purpose is to help creating page oriented, ready to print documents in a simple and flexible manner. See “[Bizgres Clickstream](#)” on page 3 for information about getting the Bizgres Clickstream demo application, which utilizes JasperReports.

Bizgres Clickstream

Greenplum, together with Kinetic Networks and JasperSoft, has developed an end-to-end solution for click stream analysis reporting using the KETL solution, Bizgres, and JasperReports. Bizgres Clickstream is one of the industry’s first complete open source business intelligence (BI) development stacks. It is intended to demonstrate how easily data warehousing and analytics solutions can be built on Bizgres. The Bizgres Clickstream application and documentation is available for download from Bizgres.org.

Getting Started with Bizgres

This section provides the high-level steps to help you get your Bizgres system up and running.

1. Install the Bizgres software on your chosen operating system. See “[Installing Bizgres Using the Greenplum Installer](#)” on page 5.
2. Configure your environment after installation. See “[Configuring Your Installation](#)” on page 10.
3. Create a data storage area and initialize the database instance. See “[Creating the Database Cluster](#)” on page 11 for instructions.

4. Start the PostgreSQL database server. See [“Starting the Database Server”](#) on page 12 for instructions.
5. Create your database. See [“Creating a Database”](#) on page 13 for instructions.
6. Access your database and create your database tables using `psql`, the PostgreSQL interactive terminal. See [“Creating Tables”](#) on page 17 for more information.
7. If you already have data you would like to use, then use the Bizgres Loader to load it into the database. See [“Using the Bizgres Loader”](#) on page 23 for more information on using the Loader.

If you do not have data, you can try the demonstration programs provided with your Bizgres distribution. These demo programs will create sample databases and data. See [“Bizgres Demo Programs”](#) on page 27.

8. Additional documentation for PostgreSQL is also provided in this distribution in `$BIZHOME/doc/postgresql/html/index.html`. Refer to the PostgreSQL documentation for more information on using PostgreSQL features.

2

Installation

This chapter provides information about installing and configuring Bizgres on your chosen platform (Linux or Solaris). It contains the following topics:

- [Installing Bizgres Using the Greenplum Installer](#) or [Installing Bizgres From Source](#)
- [Configuring Your Installation](#)
- [Verifying Your Installation](#)
- [Creating the Database Cluster](#)
- [Starting the Database Server](#)
- [Creating a Database](#)

Installing Bizgres Using the Greenplum Installer

System Requirements

Make sure your system meets the following minimum system requirements before running the installer.

Table 2.1 Bizgres System Requirements

System Component	Minimum Requirement
Operating Systems	RedHat Enterprise Linux 3, Update 4 Solaris 10 for x86 MAC OSX 10.3
Privileges	root access
Disk Space ¹	126 MB for installer 50 MB for install destination 100K in /tmp
RAM	500 MB (minimum)
Ports	5432 ² (the default port of the Bizgres database server)
Software and Utilities	GNU tar GNU zip GNU readline ³ GCC runtime libraries (glibc, etc.)

1. This specifies the disk space required to install the Bizgres distribution only. It does not account for table data, indexes, or backups. An empty database cluster takes about 25 MB, databases take about five times the amount of space that a flat text file with the same data would take. Greenplum recommends allocating approximately three times the disk space as primary database data. For more information see the sections on [Database Physical Storage](#) and [Determining Disk Usage](#) in the PostgreSQL documentation.
2. The default port is configurable and also can be overridden at runtime. See the section on [Runtime Configuration](#) in the PostgreSQL documentation for more information.
3. The GNU readline utility and GCC runtime libraries are included by default with RedHat Linux distributions, but not with Solaris 10. The required package names for Solaris are `SFWrlne.bz2` and `SFWgcc341.bz2`, and are available on the Sun Companion Software CD included with Solaris 10.

Running the Installer

1. Go to <http://bgn.greenplum.com> and download the correct installer for your target platform. This will save an installer file to your machine. If needed, copy this file to the machine where you want to install Bizgres.
2. `cd` to the directory where you saved the installer file, and unzip the installer file.
`gunzip bizgres-0_9_GA-x.bin.gz`
3. Run the following command as root to launch the installer:
`sh bizgres-0_9_GA-x-RHEL3.bin`
 OR
`sh bizgres-0_9_GA-x-SOL.bin`
4. Follow the prompts in the installer and enter the requested information.
5. The default install location is `/usr/local/bizgres`, or you can choose an alternate install location. The installer will add the Bizgres binaries directly to the location specified, so enter an absolute path (for example, `/home/mydir/bizgres`). The directory where Bizgres is installed is referred to as `$BIZHOME` in this documentation.
6. After installation, proceed to the instructions for “[Configuring Your Installation](#)” on page 10.
7. Confirm your installation using the IVP program. See “[Installation Verification Program \(IVP\)](#)” on page 27.

Uninstalling Bizgres

To uninstall Bizgres

1. If running, stop the database server. For example:
`pg_ctl -D /BIZDATA -l logfile stop`
2. Go to the following directory:
`cd $BIZHOME/Uninstall_bizgres`
3. Run the following command:

```
sh Uninstall_bizgres
```

4. The uninstaller will remove all of the Bizgres components. It will not remove the *BIZDATA* directory, clear environment variables that have been set, or reset the original shared memory setting on Linux machines.

Installing Bizgres From Source

Before You Begin

Complete the tasks in this section before installing Bizgres.

- “Installing Java for Use With Bizgres” on page 7
- “Removing Ant (RedHat Users)” on page 8

Installing Java for Use With Bizgres

If your operating system has a pre-installed Java executable, you must make sure it is not used, and that you are using the Java 5.0 (1.5.0) or higher SDK available from Sun or IBM. For example, RedHat bundles the GNU gcj, which is a compiler for the Java language, but is not a fully functional JRE. You want to make sure that a JRE that is compatible with Bizgres is called when the `java` command is issued.

For the examples in this guide, Greenplum uses the Sun *Java™ 2 SDK, Standard Edition 5.0 (1.5.0_04)* for Linux.

To install and configure Java for use with Bizgres

1. Download a compatible Java SDK for your operating system. The following URL links to the Sun download pages for *Java™ 2 SDK, Standard Edition 5.0 (1.5.0_04)*.
<http://java.sun.com/j2se/1.5.0/download.jsp>
2. Follow the instructions provided by Sun for installing Java on your operating system. Instructions for installing on Linux are provided here:
<http://java.sun.com/j2se/1.5.0/install-linux.html>
 Instructions for Solaris are provided here:
<http://java.sun.com/j2se/1.5.0/install-solaris.html>
3. Set your `JAVA_HOME` environment variable to point to the installed location of your Java SDK. For example:

```
JAVA_HOME=/usr/java/jdk1.5.0_04
export JAVA_HOME
```
4. Make sure that the Sun Java SDK is listed *first* in your `PATH` environment variable. For example, your `PATH` should look something like this:

```
PATH=/usr/java/jdk1.5.0_04/bin/java:$PATH
```

```
export PATH
```

Note: See “[Bizgres Environment Variables](#)” on page 37 for a list of all environment variables to set for Bizgres.

5. Ensure that the correct Java is called when you issue the `java` command as the `bgadmin` user. For example:

```
$ which java
/usr/java/jdk1.5.0_04/bin/java
```

Removing Ant (RedHat Users)

On many RedHat systems, there is an existing version of Apache Ant installed. This must be removed in order for the Bizgres build to work correctly. Execute the following command to remove ant from your system:

```
rpm -e ant
```

Building the Bizgres Source

1. Download the GNU compressed tar file of the Bizgres source code from www.bizgres.org, and copy it to the machine where you want to install.
2. Uncompress the tar file. For example:


```
gunzip -v bizgres-0_9_GA-x-release.tar.gz
```
3. `cd` to the directory where you want to install, and untar the file. The tar file contains the base directory path of `/bizgres`. For example:


```
cd /usr/local
tar -xvf /home/bgadmin/downloads/bizgres-0_9_GA-x-release.tar
```
4. Set your source properly using the `source-before-install` script. If it finds that build requirements (such as Ant) are missing, the script will attempt to install these as well.


```
source source-before-install.sh
```
5. Build with Ant:


```
ant -Ddebug=false -DinstallRoot=/usr/local/bizgres [-DdevoBuild=true] install
```
6. This will build all components, including the Bizgres Loader and the Installation Verification Program (IVP). All build options start with `-D`, and include:
 - **debug=true|false** — Turns on or off debugging information. Default is `true`, which is not useful for production installations.
 - **installRoot=path** — All Bizgres binaries will be installed under this directory.
 - **port=default_port** — The default PostgreSQL port for Bizgres is set to 5432, which is different from the default port of 5432 for regular PostgreSQL.

- **COPTX**=*compiler_optimizations* — Can be specified to add to the compiler optimizations.
- **minGW**=true — Implements a Windows cross-compile using minGW32 (mini-GNU for Windows).
- **devoBuild**=true|false — Allows you to install a development build if set to *true*. Defaults to *false* (install not allowed for development builds).

Build targets other than **install**:

- **apidoc** — Generate the javadocs
 - **clean** — Clean the project of build materials
 - **compile** — Compile all subprojects of Bizgres
 - **compiledGcommon** — Compile Bizgres common
 - **compiledGloader** — Compile Bizgres Loader
 - **compiledGpostgres** — Compile Bizgres-PostgreSQL
 - **dist** — Prepare a distribution package for Bizgres
 - **distclean** — Clean the project of distribution and build materials
 - **makeDoc** — Generate PostgreSQL documentation (requires special tools)
 - **pgsql-check** — Perform Postgres 8.0 regression testing
7. Once built, all Bizgres components will be available under the install path location. For example: `/usr/local/bizgres`

Post Installation Steps

Kernel Tuning on Linux

PostgreSQL will not work unless the shared memory segment for your kernel is properly sized. Most default Linux installations have the shared memory value set too low. For more information see the section on [Managing Kernel Resources](#) in the PostgreSQL documentation.

The default Bizgres configuration is set to 128 MB for shared buffers, with the operating system set up to enable use of 128 MB in one buffer.

To tune the kernel for Linux

1. Add the following lines to the `/etc/sysctl.conf` file. The `shmmax` value is required by the default GreenPlum configuration.


```
kernel.shmmax = 3221225472
kernel.shmmni = 4096
kernel.shmall = 2097152
kernel.sem = 1000 32000 100 150
```

2. After editing the `/etc/sysctl.conf` file, you can apply your changes without rebooting by issuing the following command as root:

```
/sbin/sysctl -p
```

3. Configuring the `bigpages` value can also be beneficial to performance of your GreenPlum system. Add the following directive to the Linux kernel boot command line:

```
bigpages=4100MB
```

For example, the `/boot/grub/grub.conf` file entry might look like this:

```
kernel /vmlinuz-2.4.9-e.40enterprise ro root=/dev/cciss/c0d0p2 bigpages=4100MB
```

4. After initializing the database cluster, you can sort the memory allocated to each connection. This is determined by the `work_mem` parameter in `postgresql.conf`. See the section on [Runtime Configuration](#) in the PostgreSQL documentation for more information about this parameter.



Note: Kernel tuning is not required for Solaris 10. The default settings are sufficient. However, there are some configurations you can make to enhance performance. See the following section for more information.

Performance Tuning for Solaris 10 x86

Bizgres relies on the file system cache to do most of the buffering activities. The default Solaris 10 x86 configuration is not optimal for Bizgres, especially for repeat queries. Solaris 10 will not keep large database files in the file system cache.

The following changes in `/etc/system` can improve caching of database files. For the `segmapsize` parameter, set file system cache to about 60 to 80 percent of available RAM. For example, if you have 16 GB and want to set about 10 GB for file system cache:

```
segmapsize=10737418240 set ufs:freebehind=0
```

This change allows data files of Bizgres to be in cache and available for repeat usage. It improves I/O performance, especially if the next query uses the same data files (rows).

Configuring Your Installation

After installation, perform the following configuration steps:

- [Designating a Bizgres User Account](#)
- [Setting Bizgres Environment Variables](#)

For more information about post-installation configuration, refer to the [Post-Installation Setup](#) section of the PostgreSQL documentation.

Designating a Bizgres User Account

You cannot run the PostgreSQL database server as root. GreenPlum recommends that you designate a user account that will own your Bizgres installation, and to always start Bizgres/PostgreSQL as this user. For the purposes of this documentation, we will use the user `bgadmin`. The PostgreSQL documentation uses the user `postgres`.

To add a new user, for example, run the following command as root:

```
adduser bgadmin -d /home/bgadmin -s /bin/bash -p password
```

Setting Bizgres Environment Variables

If you installed Bizgres using the Greenplum installer, a `bizgres_path.sh` file is provided in your `$BIZHOME` directory following installation with the following environment variables set. You can source this in the `bgadmin` user’s startup shell profile (such as `~/.bashrc` or `~/.bash_profile`), or in `/etc/profile` if you want to set the environment variables for all users.

For example, you could add a line similar to the following to your chosen profile file (making sure the right Bizgres install path is used):

```
source /usr/local/bizgres/bizgres_path.sh
```

After editing the chosen profile file, source it as the correct user to make the changes active. For example:

```
source ~/.bashrc
```

or

```
source /etc/profile
```



Note: If you installed Bizgres from source, see “[Bizgres Environment Variables](#)” on page 37 for a list of all environment variables to set for Bizgres.

Verifying Your Installation

Greenplum provides an Installation Verification Program (IVP) that you can run to confirm that Bizgres is installed and configured correctly. See “[Installation Verification Program \(IVP\)](#)” on page 27.

Creating the Database Cluster

Before you can begin using Bizgres, you must initialize a database storage area or *database cluster* on disk. A database cluster is a collection of databases that is managed by a single instance of a running database server. After initialization, a database cluster will contain a database named `template1`. This will be used as a

template for subsequently created databases. For more information about database clusters, see the section on [Creating a Database Cluster](#) in the PostgreSQL documentation.

Because the data directory contains all the data stored in the database, it is essential that it be secured from unauthorized access. The initialization program therefore revokes access permissions from everyone but the Bizgres user. However, while the directory contents are secure, the default client authentication setup allows any local user to connect to the database and even become the database super user. Before you start the server for the first time, you can modify the generated `pg_hba.conf` file after running `initdb` to configure authentication methods for the database cluster. See the section on [Client Authentication](#) in the PostgreSQL documentation for more information.

To create the database cluster

1. Create or choose a directory that will serve as your database storage area. This directory should have sufficient disk space for your data and be owned by the Bizgres user. This directory will be referred to as the *BIZDATA* directory in this documentation. For example, run the following commands as root:


```
mkdir /BIZDATA
chown bgadmin /BIZDATA
```
2. For Linux, make sure that your shared memory segment is properly sized. See [“Kernel Tuning on Linux”](#) on page 9.
3. Change to the bgadmin user:


```
su - bgadmin
```
4. Initialize the database cluster using the following command:


```
initdb -D /BIZDATA
```
5. After initialization, you may want to perform some additional configurations before starting the database server. For more information about configuration options for your database cluster, see the sections on [Runtime Configuration](#) and [Client Authentication](#) in the PostgreSQL documentation.

Starting the Database Server

Before anyone can access the database, you must start the database server. The database server program is called `postmaster`. The `postmaster` process must know where to find the data it is supposed to use (using the `-D` option). Make sure to start the server as the user who owns your Bizgres installation (for example, the `bgadmin` user), not as root. The simplest way to start the server is:

```
pg_ctl start -D /path_to/BIZDATA -l logfile
```

This starts the server in the background and puts the output into the named log file.

For more information about starting and stopping the database server, refer to the section on [Starting the Database Server](#) and [Shutting Down the Server](#) in the PostgreSQL documentation.



Note: At this point, you may want to try the Bizgres demo programs. See “[Bizgres Demo Programs](#)” on page 27 for more information.

Creating a Database

Before you can begin using Bizgres to manage your data, you must create one or more databases. A database is a named collection of SQL objects (or database objects). Typically, a database object (tables, functions, etc.) belongs to only one database.

When connecting to the database server, you must specify the name of the database you want to connect to. It is not possible to access more than one database per connection. By default, a database called `template1` was created when you initialized the database cluster. You can use this database to connect to the database server and create your first real database.

For more information about creating and managing databases, see the section on [Managing Databases](#) in the PostgreSQL documentation.

To create a database

1. First, make sure you are logged in as the `bgadmin` user. This user will be the default owner of the database that is created.

```
su - bgadmin
```

2. Connect to the database server using the `template1` database.

```
psql template1
```

3. Create your database with the SQL command `CREATE DATABASE`.

```
CREATE DATABASE database_name;
```

4. To see a list of the current databases, use the `\l` command.

```
\l
```

Your output should look something like this, where `bizdb` is the name of the database you just created:

Name	Owner	Encoding
bizdb	bgadmin	UNICODE
template1	bgadmin	UNICODE

5. Now you are ready to define your schema and tables, and load your data. For more information about defining your data, see the section on [Data Definition](#) in the PostgreSQL documentation.

For instructions on using the Bizgres Loader, see [Chapter 5, “Using the Bizgres Loader”](#).

For a simple overview of creating tables and data, see [Chapter 3, “Working in psql”](#).

For instructions on using the bitmap index feature of Bizgres, see [Chapter 4, “Using Bitmap Indexes”](#).

3

Working in psql

Once you have created a database, you can access it by running the PostgreSQL interactive terminal program, called `psql`, which allows you to interactively enter, edit, and execute SQL commands. This chapter provides the following high-level topics for using `psql`:

- [Connecting to a Database](#)
- [Entering SQL Commands](#)
- [psql Meta-Commands](#)
- [Creating Tables](#)

For more information about using `psql`, refer to the section on [psql](#) in the PostgreSQL documentation. For more information about SQL (Standard Query Language) and how PostgreSQL implements SQL, refer to the section on [The SQL Language](#) in the PostgreSQL documentation. For users who are unfamiliar with SQL and relational databases, you may want to complete the [PostgreSQL Tutorial](#) to learn more about creating and managing databases.

Connecting to a Database

In order to connect to a database you need to know the name of your target database, the host name and port number of the server and what user name you want to connect as. This information can be provided on the command-line using the options `-d`, `-h`, `-p`, and `-U` respectively. If an argument is found that does not belong to any option it will be interpreted as the database name.

All of these options have default values which will be used if the option is not specified. The default host is the local host. The default port number is determined at compile time. Since the database server uses the same default, you will not have to specify the port in most cases. The default user name is your Unix user name, as is the default database name. Note that Unix user names and PostgreSQL user names are not necessarily the same.

If the default values are not correct, you can save yourself some typing by setting the environment variables `PGDATABASE`, `PGHOST`, `PGPORT`, and `PGUSER` to the appropriate values. See the section on [Environment Variables](#) in the PostgreSQL documentation for more information. It is also convenient to have a `~/.pgpass` file to avoid regularly having to type in passwords. See the section on [The Password File](#) in the PostgreSQL documentation for more information.

To connect to a database

Depending on the default values used or the environment variables you have set, the following examples show how to access a database via `psql`:

```
psql -d mydatabase -h myhost -p 5432 -U bgadmin
psql mydatabase
psql
```

Entering SQL Commands

After connecting to a database, `psql` provides a prompt with the name of the database to which `psql` is currently connected, followed by the string `=>` (or `=#` if you are the database super user). For example:

```
mydatabase=>
```

At the prompt, you may type in SQL commands. A SQL command must end with either a `;` (semicolon) or a `\g` (backslash and the letter g) in order to be sent to the server and executed. For example:

```
SELECT * FROM mytable;
SELECT * FROM weather
        WHERE city = 'San Francisco' AND prcp > 0.0\g
```

For more information on SQL commands, see the [SQL Commands](#) reference in the PostgreSQL documentation. You can also type `\h` on the command-line while in `psql` to see the online help for SQL commands.

psql Meta-Commands

Anything you enter in `psql` that begins with an unquoted backslash is a `psql` meta-command that is processed by `psql` itself. These commands help make `psql` more useful for administration or scripting. Meta-commands are more commonly called slash or backslash commands.

For a complete list of meta-commands, refer to the section on [psql](#) in the PostgreSQL documentation. You can also type `\?` on the command-line while in `psql` to see the online help for `psql` commands.

The following list shows some useful commands:

- `\?` for help on `psql` meta-commands.
- `\dt` to show a list of tables in the database.
- `\dts` to show a list of system tables.
- `\du` to show a list of database users.
- `\h` for help on SQL commands.
- `\l` to show a list of databases.
- `\q` to quit `psql`.

Creating Tables

After you have created a database, you must then create the database structures that will hold your data. In a relational database, such as PostgreSQL, the raw data is stored in tables. This section provides a simple overview for creating a table and defining the data that a table will hold. For more information about tables, schemas, and data definition refer to the section on [Data Definition](#) in the PostgreSQL documentation.

A table is comprised of columns, and each column in a table has a data type. The data type constrains the set of possible values that can be assigned to a column and assigns semantics to the data stored in the column so that it can be used for computations. Some of the frequently used data types are integer for whole numbers, numeric for possibly fractional numbers, text for character strings, date for dates, time for time-of-day values, and timestamp for values containing both date and time. For more information on data types, see the section on [Data Types](#) in the PostgreSQL documentation.

To create a table, use the `CREATE TABLE` command. In this command you must specify a name for the new table, the names of the columns and the data type of each column. For example:

```
CREATE TABLE products (  
    product_no integer,  
    name text,  
    price numeric  
);
```

4

Using Bitmap Indexes

This chapter provides information on using the bitmap index feature of Bizgres. Bitmap index is a feature of Bizgres that supplements the indexing options offered by PostgreSQL. For more information about indexes in general, refer to the section on [Indexes](#) in the PostgreSQL documentation.

This chapter contains the following topics:

- [About Bitmap Indexes](#)
- [Creating a Bitmap Index](#)
- [Examining Bitmap Index Usage](#)
- [Maintaining Bitmap Indexes](#)

About Bitmap Indexes

Bitmap indexes are one of the most promising strategies for indexing high dimensional data in data warehousing applications and decision support systems. These types of applications typically have large amounts of data and ad hoc queries, but a low number of DML transactions.

An index provides pointers to the rows in a table that contain a given key value. A regular index stores a list of tuple ids for each key corresponding to the rows with that key value. In a bitmap index, a bitmap for each key value replaces a list of tuple ids.

Fully indexing a large table with a traditional B-tree index can be expensive in terms of space because the indexes can be several times larger than the data in the table. Bitmap indexes are typically only a fraction of the size of the indexed data in the table.

Each bit in the bitmap corresponds to a possible tuple id, and if the bit is set, it means that the row with the corresponding tuple id contains the key value. A mapping function converts the bit position to an actual tuple id, so that the bitmap index provides the same functionality as a regular index. Bitmap indexes store the bitmaps in a compressed way. If the number of distinct key values is small, bitmap indexes compress better and the space saving benefit compared to a B-tree index becomes even better.

Bitmap indexes are most effective for queries that contain multiple conditions in the `WHERE` clause. Rows that satisfy some, but not all, conditions are filtered out before the table itself is accessed. This improves response time, often dramatically.

When to Use Bitmap Indexes

Bitmap indexes perform best for columns in which the ratio of the number of distinct values to the number of rows in the table is small. This ratio is known as the degree of *cardinality*. A gender column, which has only two distinct values (male and female), is a good choice for a bitmap index, and is considered *low cardinality*.

However, columns with higher cardinalities can also have bitmap indexes. For example, on a table with one million rows, a column with 10,000 distinct values is also a good candidate for a bitmap index. A bitmap index on this column can outperform a B-tree index, particularly when this column is often queried in conjunction with other indexed columns.

Bitmap indexes can dramatically improve query performance for ad hoc queries. `AND` and `OR` conditions in the `WHERE` clause of a query can be resolved quickly by performing the corresponding Boolean operations directly on the bitmaps before converting the resulting bitmap to tuple ids. If the resulting number of rows is small, the query can be answered quickly without resorting to a full table scan.

When Not to Use Bitmap Indexes

Bitmap indexes should not be used for unique columns or columns with high cardinality data, such as customer names or phone numbers. B-tree indexes are a better choice for these types of columns.

Bitmap indexes are primarily intended for data warehousing applications where users query the data rather than update it. They are not suitable for OLTP applications with large numbers of concurrent transactions modifying the data.

Bitmap Index Example

To illustrate how bitmap indexes work, let's consider the following example that shows a portion of data from a company's *customer* table:

Table 4.1 Sample Customer Data

customer_id	gender	marital_status	annual_income
1	M	married	A: 29,999 or under
2	F	single	D: 120,000 - 149,999
3	M	married	D: 120,000 - 149,999
4	M		F: 200,000 or higher
5	M		B: 30,000 - 59,999
6	F	married	C: 60,000 - 99,999
7	M	single	E: 150,000 - 199,999
8	F		D: 100,000 - 149,999

The *gender*, *marital_status*, and *annual_income* columns are ideal for a bitmap index since they are all low-cardinality columns — there are only three possible values for *marital_status*, two possible values for *gender*, and six possible values for *annual_income*. Note that unlike most other index types, bitmap indexes include rows that have `NULL` values. Indexing of nulls can be useful for some types of SQL statements, such as queries with the aggregate function `COUNT`.

You would not want to create a bitmap index on *customer_id*, however, because it is a unique column (a B-tree index would be a better choice for this column).

The following table illustrates a bitmap index for the *gender* column:

Table 4.2 Sample Bitmap Index

customer_id	gender='M'	gender='F'
1	1	0
2	0	1
3	1	0
4	1	0
5	1	0
6	0	1
7	1	0
8	0	1

Each entry (or bit) in the bitmap corresponds to a single row of the *customer* table. The value of each bit depends upon the values of the corresponding row in the table. For example, the bitmap for `gender='M'` would look like this. It contains a '1' as its first bit because the gender is 'M' in the first row of the *customer* table:

```
10111010
```

The bitmap for `gender='F'` would look like this. It has a zero for its third bit because the gender of the third row is not 'F'.

```
01000101
```

An analyst for the company might want to know, “Which of our customers who are female are also married?” This corresponds to the following SQL query:

```
SELECT * FROM customers WHERE gender = 'F' AND marital_status = 'married';
```

The bitmaps for `gender='F'` and `marital_status='married'` are used to identify the rows that have the correct bit values, and the results are merged before accessing the actual tuple id in the *customer* table. This is more efficient than scanning the entire table multiple times, as only the exact rows that are needed are retrieved.

Creating a Bitmap Index

Use the [CREATE INDEX](#) SQL command to create a new bitmap index. A new index method named *bitmap* is available in addition to the regular PostgreSQL index methods. See [CREATE INDEX](#) in the PostgreSQL documentation for more information. The complete command syntax is as follows:

```
CREATE INDEX name ON table USING bitmap
    ( { column | ( expression ) } [ opclass ] [, ...] )
    [ TABLESPACE tablespace ]
    [ WHERE predicate ]
```

For example, to create a bitmap index on the *customer* table for the *gender* column:

```
CREATE INDEX gender_bitmap ON customer USING bitmap (gender);
```

Examining Bitmap Index Usage

Examining index usage for an individual query is done with the [EXPLAIN](#) command. See the section on [Using Explain](#) in the PostgreSQL documentation for more information about reading query plans.

The query plan shows the different steps or *plan nodes* that the database will take to answer a particular query, along with time estimates for each plan node. To examine the use of bitmap indexes, look for the following query plan node types in your `EXPLAIN` output:

- **Bitmap Heap Scan** - Retrieves all rows from the bitmap generated by `BitmapAnd`, `BitmapOr`, or `BitmapIndexScan` and accesses the heap to retrieve the relevant rows.
- **Bitmap Index Scan** - Retrieves all rows that satisfy the query predicates from the underlying index. For all rows retrieved, builds an in memory bitmap.
- **BitmapAnd** or **BitmapOr** - Takes the bitmaps generated from multiple `BitmapIndexScan` nodes, ANDs or ORs them together, and generates a new bitmap as its output.



Note: Always run [ANALYZE](#) after creating or updating an index. This command collects table statistics that are used by the query planner.

Maintaining Bitmap Indexes

Altering a Bitmap Index

You can use the [ALTER INDEX](#) SQL command to change the name of a bitmap index or to set the tablespace. For example:


```
ALTER INDEX mf_bitmap RENAME TO gender_bitmap;  
ALTER INDEX gender_bitmap SET TABLESPACE fasttablespace;
```

Deleting a Bitmap Index

You can use the [DROP INDEX](#) SQL command to delete a bitmap index. For example:

```
DROP INDEX gender_bitmap;
```

Reindexing After Updates/Deletes

Bitmap indexes are not intended for OLTP applications, or applications with frequent updates or deletes. Updates or delete operations do not update bitmap indexes. Therefore, if you have deleted rows or updated columns in a table that has bitmap indexes, you will need to rebuild the indexes using the [REINDEX](#) command.

To rebuild all indexes on a table

```
REINDEX my_table;
```

To rebuild a particular bitmap index

```
REINDEX my_bitmap_index;
```

5

Using the Bizgres Loader

This chapter provides information on using Bizgres Loader. Bizgres Loader facilitates the loading of large quantities of data into a running Bizgres database. This chapter contains the following sections:

- [Overview](#)
- [Before You Begin](#)
- [Running Bizgres Loader](#)
- [Bizgres Loader Log Files](#)
- [Updating Database Statistics After Data Loads](#)
- [Vacuuming the Database After Load Errors](#)

For syntax, options, and reference information on the `loader.sh` command, refer to [Appendix A, “Command Reference”](#).

Overview

Bizgres Loader loads data from a text file (or from standard input) into a running Bizgres database. Bizgres Loader can be useful when loading large quantities of data from another RDBMS or other external data source.

Bizgres Loader is a Java command-line program that is called from a shell script, `loader.sh`, which is located in `$BIZHOME/client/loader/bin` of your Bizgres installation. See “[loader.sh](#)” on page [30](#) for complete command syntax and options.

The functionality of Bizgres Loader is similar to the PostgreSQL `COPY` command and uses similar data formatting options. However, Bizgres Loader provides several additional features such as error logging, data batching, and enhanced configuration options.

Before You Begin

Before you can run Bizgres Loader, make sure you have done the following:

1. Make sure your `JAVA_HOME` environment variable is set.
2. You must have the Bizgres software installed and the system initialized and running. See [Chapter 2, “Installation”](#) for instructions.
3. Create your database, schema, and table structures prior to loading data. See the section on [Data Definition](#) in the PostgreSQL documentation for more information.

4. Prepare your data so that it is in a format acceptable by Bizgres Loader. See [“Formatting the Data File”](#) on page 31 for more information.
5. Write your control file. The control file contains one or more `LOAD` commands that specify a table, data file, and specifications for loading the data. See [“Formatting the Control File”](#) on page 31 for more information.

Running Bizgres Loader

Bizgres Loader is invoked by running the `loader.sh` script, which is located in the `$BIZHOME/client/loader/bin` directory. For complete command syntax and options for `loader.sh`, see [“loader.sh”](#) on page 30.

This section explains the different modes for running Bizgres Loader:

- [Running Bizgres Loader in Regular Mode](#)
- [Running Bizgres Loader in Batch Mode](#)
- [Running Bizgres Loader in Preview Mode](#)

Running Bizgres Loader in Regular Mode

When running Bizgres Loader in regular mode the entire data load is treated as one transaction. Regular mode is fine for smaller data loads. When running in regular mode, bad records are still recorded in a *bad file* by default. However, a bad file is not really needed in regular mode, as no data will be loaded if a data error is encountered (the bad file and original data file will be the same).

To disable the creation of the bad file, use the `--noreject` option. This option also disables Bizgres Loader from storing data in memory, as it does by default. If Bizgres Loader stores data in memory, it will batch the data internally if the size limit is reached, which is 8 MB by default or the value specified by the `-l` option.

For large data loads, you may want to use batch mode. See [“Running Bizgres Loader in Batch Mode”](#) on page 24.

To run Bizgres Loader in regular mode

```
loader.sh -f '/path/to/control_file.txt' -h bg_host -p 5432 -d
database_name -u username --noreject
```

Running Bizgres Loader in Batch Mode

By default data batching is disabled, meaning the entire data load is generally treated as one transaction. Batching allows you to divide the data records into batches to be loaded into the database in increments. This is useful if you have large quantities of data. With batching enabled, Bizgres Loader does incremental error checking and loading for each batch of records. If Bizgres Loader encounters data errors, it will not

load a batch and will write the records from that batch into a *bad file* for you to review later. If batching is not enabled and one record has a data error, none of the data will be loaded.

Batching is enabled by using the `-b` option on the command-line when you run Bizgres Loader. This option specifies the number of records or rows that should comprise a batch. For example:

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d
bizdb -u bgadmin -b 200
```

By default, a bad file is created in the current directory the name `BGLOADER-timestamp-table_name.bad`. The bad file contains batch records that were not loaded due to data errors. You can choose your own name for the bad file using the `--badfile` option (only recommended for control files with a single `LOAD` command). For example:

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d
bizdb -u bgadmin -b 200 --badfile /home/bgadmin/bad
```

For very large batch sizes, you may also want to increase the memory allocated for saving a data batch in memory using the `-l` option. The default amount of memory is 8 MB. If the memory limit is reached before the number of rows in the batch has been processed, Bizgres Loader will dynamically reduce the batch size to the allowed size limit and commit the data to the database. For example, with the default limit of 8 MB and an average row size of 100 bytes, the loader will commit a batch after about 80,000 rows even if the user specified batch size was set higher than that. The following example shows how to increase the batch limit:

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d
bizdb -u bgadmin -b 900000 -l 10
```

The `--noreject` option can be used to disable saving batch data in memory, but this will also disable the bad file, which can make troubleshooting load errors more difficult for large data loads.

Running Bizgres Loader in Preview Mode

When running Bizgres Loader in preview mode, no data is actually loaded into the database. There are two options for preview mode: `-n` or `-s`.

The `-n` option will parse the data and print load statistics without actually loading the data. For example:

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d
bizdb -u bgadmin -n
```

The `-s` option will send the generated SQL commands and data to one or more `dataloader_dump#.sql` files in the current directory instead of loading the data into the database. The data and commands in the SQL files are already modified, escaped, and divided into batches. For example:

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d
bizdb -u bgadmin -s
```

Bizgres Loader Log Files

The default log file is called `bgloader.log` and is written to the current directory each time Bizgres Loader runs. This log file contains information about the load such as statistics and errors. If you want to change the default name and location of the log file, you can use `--logfile` option when running Bizgres Loader. For example:

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d
bizdb -u bgadmin --logfile /home/bgadmin/mylog
```

By default, Bizgres Loader also creates a *bad file* if any data errors are encountered during the data load. The bad file contains the records that were not loaded. The bad file is created in the current directory as:

`Bizgresloader-mmddHHMMSS-table_name.bad`. The table name is appended to the file name while the load is in progress. This is done to accommodate control files with multiple `LOAD` commands.

Updating Database Statistics After Data Loads

After loading data, always run the `ANALYZE` SQL command to update the database statistics used by the query planner. `ANALYZE` collects statistics about the contents of tables in the database, and stores the results in the system table `pg_statistic`. The query planner uses these statistics to help determine the most efficient execution plans for queries. For example, to collect statistics on a newly loaded table, run the following command:

```
psql dbname -c 'ANALYZE mytable;'
```

Vacuuming the Database After Load Errors

The `COPY` command and Bizgres Loader will stop a load operation if it encounters an error. When this happens, the target table may already have received earlier rows in a `COPY FROM` or load operation. Although these rows will not be visible or accessible, they still occupy disk space. This may amount to a considerable amount of wasted disk space if the failure happened well into a large copy or load operation. You may wish to invoke the `VACUUM` command to recover the wasted space. For example, run the following command after a load error:

```
vacuumdb dbname [table_name]
```

`VACUUM` reclaims storage occupied by deleted tuples. In normal operation, tuples that are deleted or obsoleted by an update are not physically removed from their table; they remain present until a `VACUUM` is done. Therefore it's recommended to do `VACUUM` periodically, especially on frequently-updated tables.

6

Bizgres Demo Programs

This chapter provides information about the demo programs provided with your Bizgres distribution. The following demo programs are available:

- [Installation Verification Program \(IVP\)](#)
- [Bizgres Clickstream](#)

Installation Verification Program (IVP)

The Installation Verification Program (IVP) tests your Bizgres installation by doing the following:

- Initializes its own IVP database cluster in the current directory and starts its own PostgreSQL database server instance running on port 5432.
- Creates a sample database called `bgtestdb`.
- Generates approximately 145 MB of sample data using a data generator program. The IVP program directory needs approximately 500 MB of space in all.
- Loads the data into the `bgtestdb` database using the Bizgres Loader.
- Runs a sample query.

Prerequisites for Solaris Users

If you are running Bizgres on Solaris, make sure you have the following:

- Make sure that `make`, `tar`, and a C compiler (`gcc` recommended) is in your `$PATH`.
- Verify that your `$USER` environment variable is set.
- Greenplum recommends using the bash shell to run IVP.

Installing and Running IVP

To install and run the IVP program

1. Before you begin, you must have Bizgres installed and have configured your environment. See [“Installing Bizgres Using the Greenplum Installer”](#) on page 5 and [“Setting Bizgres Environment Variables”](#) on page 11.
2. From the directory where you wish to install IVP, untar the IVP tar file. This will be referred to as `IVP_HOME`. Make sure this directory has 500 MB of free space:

```
cd IVP_HOME
```

```
tar -xvf $BIZHOME/demo/IVP.tar
```

Note: tar is located in /usr/local/bizgres/gnu/bin

3. Go into the IVP directory you just untarred.

```
cd IVP
```

4. If you did not install Bizgres in /usr/local, edit the path.sh file and make sure it is pointing to your correct Bizgres install location.

5. If logged in as root, switch to the bgadmin user at this time. You cannot run IVP as root.

```
su - bgadmin
```

6. Source the path.sh file

```
source path.sh
```

(or if using sh)

```
.path.sh
```

7. In the IVP directory, run make:

```
make
```

8. If you encounter any errors or problems, check the IVP.log file.

The IVP program will create the bgtestdb database, generate data, load the data, and run a sample query.

The last lines of example output should look something like this:

```
-----
TOTALS:
```

```
- Time           : 20.364 secs
- Rows Read      : 1000000 (non blank)
- Rows Loaded    : 1000000
- Bytes Loaded   : 142621213
- Load rate      : 6.679149 Mbytes/sec
- Failed Batches: 0
```

```
=====
```

No more control commands.

The result from this next query should be 1000000

```
time psql -p 5432 bgtestdb -c "select count(*) from
```

```

IVP.bigtable1"
count
-----
1000000
(1 row)

real    0m3.207s
user    0m0.004s
sys     0m0.004s
-----

```

Removing IVP

To stop and remove the IVP program

1. From the IVP directory run make clean:

```
make clean
```

This will remove the IVP database cluster and stop the database instance.
2. Remove the IVP directory. For example:

```
cd ..
rm -rf IVP
```

Bizgres Clickstream

Greenplum, together with Kinetic Networks and JasperSoft, has developed an end-to-end solution for click stream analysis reporting using the KETL solution, Bizgres, and JasperReports. Bizgres Clickstream is one of the industry's first complete open source business intelligence (BI) development stacks. It is intended to demonstrate how easily data warehousing and analytics solutions can be built on Bizgres.

The complete Bizgres Clickstream demo application and documentation is available for download separately from Bizgres.org.

The Bizgres Clickstream developer tools are provided with your Bizgres 0.9 distribution. This refers to the following portions of your Bizgres installation:

```

$BIZHOME/demo/solutions/clickstream
$BIZHOME/KETL
$BIZHOME/JasperReports

```


A

Command Reference

This appendix provides references for the command-line tools for Bizgres. For reference information on PostgreSQL and SQL commands, refer to the [PostgreSQL documentation](#).

The following Bizgres commands are available:

- [loader.sh](#)

loader.sh

Loads large batches of data into a running Bizgres database.

Synopsis

```
loader.sh -f 'control_file' -h hostname -p port_number -d database
-u username -P password [-c number_of_threads] [--skipheader
number_of_rows] [--logfile filename] [-n | -s] {[-b batch_size]
[-l batch_limit | --noreject] [--badfile filename]}

loader.sh --help | --version
```

Description

Bizgres Loader is a Java command-line program that is called from a shell script (`loader.sh`). The functionality of Bizgres Loader is similar to the PostgreSQL `COPY` command and uses similar data formatting options. However, Bizgres Loader provides several additional features such as error logging, data batching, and enhanced configuration options.

Bizgres Loader takes as input a control file. The control file contains one or more `LOAD` commands, which are the specifications for loading the data into a table. See “[Formatting the Control File](#)” on page 31 for more information about the control file. In a `LOAD` command, you must specify the location of your data (either a data file or standard input). This data must be formatted in a way that is compatible with Bizgres Loader. See “[Formatting the Data File](#)” on page 31 for more information about preparing the data to be loaded.

Bizgres Loader can be run in batch mode, which means that large quantities of data are loaded into the database in batches. You can specify the number of rows and a size limit that define a batch. If a batch contains data errors, the bad batches are written to a *bad file* so you can go back and troubleshoot the rows that did not get loaded. See “[About Data Batching](#)” on page 33 for more information on batching.

After loading data, always run `ANALYZE` to update the database statistics used by the query planner.

Formatting the Control File

The control file contains one or more `LOAD` commands. `LOAD` commands are not case-sensitive. A `LOAD` command is structured as follows:

```
LOAD [schema.]tablename [(column1,column2,...)]
FROM {'filename' | STDIN}
[ [WITH]
  [DELIMITER [AS] 'delimiter']
  [NULL [AS] 'null string']
  [ESCAPE [AS] 'escape' | 'OFF'] ] ;
```

The following list explains each line of a `LOAD` command:

- Each control command begins with the keyword `LOAD`, followed by a table name (may also be qualified by a schema name), followed by an optional column list in parenthesis.
- The `FROM` clause specifies either a full path and file name of a data file to load or `STDIN` if you want to stream data into the loader. To stream the data into the loader you can include the data in the control file after the `LOAD` command terminated with a `\.` (backslash period). You can also pipe the `LOAD` command and data into Bizgres Loader on the command-line, omitting the `-f` option.
- The `WITH` keyword indicates the beginning of the data format specification section. If not specified, the default values will be used.
- `DELIMITER` is a single character that separates data fields in the data file. The default is a tab.
- `NULL` is the sequence of characters that represent a `NULL` value. The default is nothing in between two delimiters.
- `ESCAPE` is the single character that is used for C escape sequences (such as `\n`, `\t`, `\100`, and so on) and for quoting data characters that might otherwise be taken as row or column delimiters. The default `ESCAPE` character is a `\` (backslash), however it is possible to specify any other character to represent an escape. It is also possible to disable escaping by specifying the value `'OFF'` as the `ESCAPE` value. This is very useful for data such as web log data that has many embedded backslashes that are not intended to be escapes.
- A `LOAD` command must always end with a semi-colon (`;`).

Formatting the Data File

The data file contains the data to be imported into the database. It must follow a specific field-delimited format, and conform to the following syntax rules:

- Header rows may be added at the beginning of a data file. Use the `--skipheader` option when running Bizgres Loader to skip the header rows when loading the data.

- A row is determined by a line feed character (0x0a). Blank rows in the data file (a row with a single line feed character) are allowed and will be ignored by Bizgres Loader.
- The `DELIMITER` character (as specified in the control file) must only appear between any two data value fields. Do not place a delimiter at the beginning or end of a row. For example:

```
data value 1 | data value 2 | data value 3
```
- The order of data columns should correspond to the column list specified in the control file, or if no column list is specified, the same order as in the table metadata.
- `NULL` values are represented by the `NULL` string specified in the control file, or by the default convention of nothing between two delimiters.
- Special characters such as a line feed (0x0a) or the delimiter character should be escaped (using the escape character specified in the control file) when used as data values within a field. If there is no need to escape the data, you can disable escaping by using: `ESCAPE 'OFF'`. See [“About Escaping”](#) on page 32 for more information.

About Escaping

The data file has two reserved characters that have special meaning to Bizgres Loader:

- The designated delimiter character (such as a tab or |), which is used to separate fields in the data file.
- A line feed (0x0a), which is used to designate a new row in the data file.

If your data contains either of these characters, you must escape the character so Bizgres Loader treats it as data and not as a field separator or new row.

By default, the escape character is a \ (backslash). If you want to use a different escape character, you can do so using the `ESCAPE AS` clause in the control file. See [“Formatting the Control File”](#) on page 31. Make sure to choose an escape character that is not used anywhere in your data file as an actual data value.

If there is no need to escape the data, you can disable escaping by using:

```
ESCAPE 'OFF'
```

For example, suppose you have a table with three columns and you want to load the following three fields using Bizgres Loader.

- percentage sign = %
- vertical bar = |
- exclamation point = !

Your designated delimiter character is | (pipe character), and your designated escape character is * (asterisk). The formatted row in your data file would look like this:

```
percentage sign = % | vertical bar = *| | exclamation point = !
```

Notice how the pipe character that is part of the data has been escaped using the asterisk character.

About Data Batching

By default data batching is disabled, meaning the entire data load is generally treated as one transaction. Bizgres Loader will batch the data internally if the size limit is reached, which is 8 MB by default or the value specified by the `-l` option. This dynamic batching feature is enabled by default whether running in batch mode or not. The `--noreject` option disables dynamic batching as well as the creation of a bad file.

Batching allows you to divide the data records into batches to be loaded into the database in increments. This is useful if you have large quantities of data. With batching enabled, Bizgres Loader does incremental error checking and loading for each batch of records. If Bizgres Loader encounters data errors, it will not load a batch and will write the records from that batch into a bad file for you to review later. If batching is not enabled and one record has a data error, none of the data will be loaded.

Batching is enabled by using the `-b` option on the command-line when you run Bizgres Loader. This option specifies the number of records or rows that should comprise a batch.

By default, a bad file is created in the current directory the name `LOADER-timestamp-table_name.bad`. The bad file contains batch records that were not loaded due to data errors. You can choose your own name for the bad file using the `--badfile` option (only recommended for control files with a single `LOAD` command).

For very large batch sizes, you may also want to increase the memory allocated for saving a data batch in memory using the `-l` option. The default amount of memory is 8 MB. If the memory limit is reached before the number of rows in the batch has been processed, Bizgres Loader will dynamically reduce the batch size to the allowed size limit and commit the data to the database anyways. The `--noreject` option can be used to disable saving batch data in memory, but this will also disable the bad file, which can make troubleshooting load errors more difficult.

Options

-f 'control_file'

Specifies the full path and file name of the control file to use. The control file contains one or more `LOAD` commands, which specify the data to import along with various details of the data import operation. If the `-f` option is omitted, you must then pipe the data file and control file into Bizgres Loader on the command-line. Your control file should then specify `STDIN` as the `FROM` value. See the section on [“Formatting the Data File”](#) on page 31 for more information about the control file.

-h *hostname*

Required. Specifies the host name of your Bizgres database server (PostgreSQL). If not specified, `localhost` is used.

-p *port_number*

Required. Specifies the port number of your Bizgres database server (PostgreSQL). If not specified, defaults to 5432.

-d *database*

Required. If not specified, defaults to `template1`.

-u *username*

Required. The name of a database user in PostgreSQL. If not specified, defaults to `username`. For more information about database users, see the section on [Database Users and Privileges](#) in the PostgreSQL documentation.

-p *password*

Required only if using password authentication. The password of the user used to connect to the database. If not specified, defaults to `password`.

-c *number_threads*

Optional. The number of concurrent threads to use when loading the data into the database. Default is 1 (no concurrency).

--skipheader *number_of_rows*

Optional. If the data file to be loaded contains header rows, you must use the `--skipheader` option to declare the number of header rows to ignore at the beginning of a data file. By default Bizgres Loader assumes the data file contains no header rows, and will load all rows as data.

--logfile *filename*

Optional. The path and file name of the log file for Bizgres Loader. By default a log file named `loader.log` is written to the current directory.

-n

Optional. Runs the loader in preview mode. The control file and data are parsed and load statistics are printed, but no data is loaded into the database.

-s

Optional. Sends the generated SQL commands and data to one or more `.sql` files instead of loading into the database. The data and commands in the `.sql` files are already modified, escaped, and divided into batches. This option is overridden by the `-n` option.

-b *batch_size*

Optional. Specifies the batch size (number of rows) to load. Batching is disabled by default. If the batch size is extremely large, you may need to adjust the batch limit specified by the `-l` option.

-l *batch_limit*

Optional. The limit in mega-bytes of batch data to save in memory. Defaults to 8 MB. This is used as a safety mechanism to limit the amount of batch data saved in memory. If the loader has reached this limit, but has not yet reached the number of rows specified by the `-b` option, it will go ahead and commit the data to the database anyways. If running Bizgres Loader with very large batch sizes, increase this value as needed. This option is overridden by the `--noreject` option.

--noreject

Optional. When specified, no memory is set aside for saving batch data. Bizgres Loader will not save failed or rejected batches into a bad file.

--badfile *filename*

Optional. Allows you to specify a path and file name for the bad file. The bad file contains batch records rejected by Bizgres Loader due to data errors. If you specify your own file name for the bad file using this option, make sure a file with the same name does not already exist, and that your control file does not include multiple load commands that load data into different tables.

By default, the bad file is created in the current directory as:

`loader-mmddHHMMSS-table_name.bad`. The table name is appended to the file name while the load is in progress. This is done to accommodate control files with multiple `LOAD` commands.

Examples

To run Bizgres Loader with the minimal options:

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d
bizdb -u bgadmin
```

To run Bizgres Loader in regular mode (no batching):

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d
bizdb -u bgadmin --noreject
```

To run Bizgres Loader when the data file has one header row at the beginning of the file:

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d
bizdb -u bgadmin --skipheader 1
```

To run Bizgres Loader in batch mode. This command divides the data into batches of 200 rows:

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d
```

```
bizdb -u bgadmin -b 200
```

To run Bizgres Loader in preview mode:

```
loader.sh -f '/home/bgadmin/control.txt' -h myhost -p 5432 -d  
bizdb -u bgadmin -n
```

A sample of a LOAD command in a control file:

```
LOAD customers (id, name, address, phone) FROM  
/home/bgadmin/ctl.txt WITH DELIMITER AS '|' NULL AS '' ESCAPE  
AS '^';
```

B

Bizgres Environment Variables

This is a reference of the environment variables to set for Bizgres. If you installed Bizgres using the Greenplum installer, a `bizgres_path.sh` file is provided in your `$BIZHOME` directory following installation with the following environment variables set. You can source this in the `bgadmin` user's startup shell profile (such as `~/.bashrc` or `~/.bash_profile`), or in `/etc/profile` if you want to set them for all users.

For example, you could add a line similar to the following to your chosen profile file (making sure the right Bizgres install path is used):

```
source /usr/local/bizgres/bizgres_path.sh
```

If you installed Bizgres by building from source, you will need to set each of these environment variables in your user's environment.

BIZHOME

This is the installed location of the Bizgres binaries. For example:

```
BIZHOME=/usr/local/bizgres
export BIZHOME

BIZHOME=/home/mydir/bizgres
export BIZHOME
```

JAVA_HOME

Bizgres is installed with a Java Development Kit (JDK) installation that is compatible with Bizgres. For example:

```
JAVA_HOME=$BIZHOME/jdk1.5.0_05
export JAVA_HOME
```

PATH

Your `PATH` environment variable should point to the location of your JDK `bin` directory (listed first), the location of the Bizgres Loader `bin` directory, and the location of the Bizgres database engine (PostgreSQL) `bin` directory. For example:

```
PATH=$JAVA_HOME/bin:$BIZHOME/pgsql/bin:$BIZHOME/client/loader/bin:$PATH
export PATH
```

LD_LIBRARY_PATH

The `LD_LIBRARY_PATH` environment variable should point to the location of the PostgreSQL library files. For Solaris, this also points to the GNU compiler and readline library files as well. For example:

```
LD_LIBRARY_PATH=$BIZHOME/pgsql/lib
```


PGPORT

The default port number of the Bizgres/PostgreSQL database server. For example:

```
PGPORT=5432
export PGPORT
```

MANPATH

The location of the PostgreSQL manual pages.

```
MANPATH=$BIZHOME/doc:$MANPATH
export MANPATH
```

Additional Environment Variables

The following environment variables are not set in the `bizgres_path.sh` file, but you may want to add them for convenience.

PGDATABASE

The name of the default Bizgres/PostgreSQL database to use. For example:

```
PGDATABASE=bizdb
export PGDATABASE
```

PGHOST

The host name of the Bizgres/PostgreSQL database server that clients use to connect to the database. For example:

```
PGHOST=myhost
export myhost
```

PGUSER

The Bizgres user name used to connect to the Bizgres/PostgreSQL database server. For example:

```
PGUSER=bgadmin
export PGUSER
```

Index

B

- batch load: 24
- bitmap index: 18
- Bizgres
 - building from source: 8
 - components: 2
 - features: 2
 - installing: 5
 - overview: 1
 - starting: 12
 - uninstalling: 6
 - user account: 11
- Bizgres Loader: 30
 - control file: 31
 - data file: 31
 - overview: 23
- BIZHOME: 37
- building from source: 8

C

- commands
 - loader.sh: 30
- connecting to a database: 15
- creating
 - data directory: 11
 - database cluster: 11
 - databases: 13
 - tables: 17

D

- database
 - connecting: 15
 - creating: 13
- demos: 27

E

- environment variables: 37

G

- getting started: 3

I

- initializing a database cluster: 11
- installing
 - Bizgres: 5

- Java: 7

- IVP: 27

J

- JasperReports: 3
- JAVA_HOME: 37

K

- kernel tuning, Linux: 9
- KETL: 3

L

- LD_LIBRARY_PATH: 37
- Linux tuning: 9
- loader
 - batch mode: 24
 - log files: 26
- loader.sh: 30
- logs
 - loader: 26

M

- management scripts
 - loader.sh: 30
- MANPATH: 38

P

- PATH: 37
- performance tuning, Solaris: 10
- PGDATABASE: 38
- PGHOST: 38
- PGPORT: 38
- PGUSER: 38
- psql: 15

Q

- queries
 - running: 16

S

- Solaris tuning: 10
- SQL commands: 16
- starting Bizgres: 12
- system requirements: 5

T

tables
 creating: 17

U

uninstalling Bizgres: 6
user: 11
utilities
 loader.sh: 30

V

verifying an installation: 27