Turbo V-8

Database Maintenance

R:BASE Technologies, Inc.

R:BASE Turbo V-8

Database Maintenance

by R:BASE Technologies, Inc.

Welcome to R:BASE Turbo V-8 for Windows

Welcome to the Next Generation R:BASE Turbo V-8 for Windows. R:BASE is an Industrial-Strength, True 32-Bit, Multi-User Relational Database. But R:BASE is not just a Database Management System; it is a total GUI development environment for all Windows desktop and network applications. R:BASE Turbo V-8 for Windows is the ideal Database Management Suite for creating and maintaining your mission critical data with a true graphical user interface. Since its introduction in 1981 as the first PC-based database management system based on Dr. Codd's relational model, R:BASE has led as the first 32-bit DBMS in its class, providing programming-free application development, automatic multi-user capabilities, 4GL (a full-featured programming language in the R:BASE base product) and embedded ANSI SQL. And now with R:BASE Turbo V-8 for Windows, we have added a whole new look and feel to enhance the applications you develop in R:BASE. You can rapidly produce the type of results that previously would have required various third party development tools. Simply using native controls, you can now design cool applications at a fraction of the cost and development time when compared to other database and development tools available.

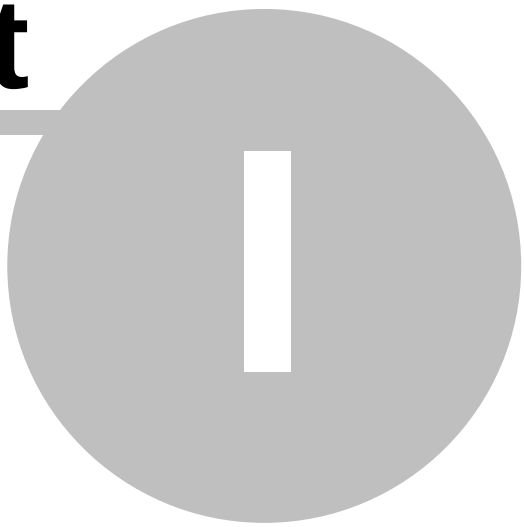
R:BASE Turbo V-8 Database Maintenance

Printed: December 2008 in Murrysville, PA

Second Edition

Table of Contents

Part



1 Maintaining a Database

This document covers several important database maintenance procedures to increase speed and efficiency. After covering this information, you will be able to do the following:

- Check your database for damage
- Fix your database
- Compress your database
- Copy your database files
- Back up your database
- Restore your database files
- Fix a corrupt database
- Improve data integrity

Note: Once you put a database into operation, it becomes another asset of the company to insure and maintain. The insurance is the backup copy of the database. The maintenance plan includes procedures that help decrease the database's response time and ensure the integrity of its data.

1.1 Backups

The data contained in a database is valuable and represents time and money. No one can predict when or how a database will be damaged. If human error or a technical malfunction destroyed or damaged the data, all of the valuable data would be lost. To prevent this, you should make a copy of the database on a regular basis. The copy is referred to as a backup. You can store backups on a tape drive, on a hard disk, or on floppy disk(s).

While it seems like extra work to make regular backups, it is good insurance and not difficult. With a backup, you can restore the data. Without a backup, your database is gone.

If you do not make good, consistent, daily backups then how important is your data? Yes, there is software and techniques available to recover data from damaged databases, but no method is 100% reliable. None is guaranteed to recover 100% of your data, and using such a data recovery method may involve hours to days of down time.

Restoring a current backup and re-entering any missing data is by far the fastest and most accurate method of data recovery. By current we are talking yesterdays backup, not last weeks or last months. With the number of commercial software programs available to do backups there is no excuse for not doing them. If you have a large database, invest in a streaming tape backup. The cost of regular backups saves money. You will recover the cost of the backup and more the first time you use it.

Store backups away from the computer. Use off site backups for the most important, irrecoverable data. Store disks in a locked, fireproof box.

Test your backup procedure to make sure the procedure works. The time to find out that there is a bad disk or the data did not copy is not when you need to restore the backup. Who has not heard of users faithfully executing backup commands, but, until they needed to restore, not realizing that the procedure is not completing successfully? In fact, two different backup methods are best for vital data.

How important is your data? How much time and effort would be involved if you had to recreate the data from scratch, assuming that you can recreate all the data? What state would your business be in if someone breaks into your office and steals your computer? What about a hard disk failure? Do not assume "that will never happen to me". It happens and it only needs to happen once. Backups do not take time; they save you time, and money.

1.1.1 What to Back Up

You need to back up the application files only when the application has changed. However, because the database files contain any changes or additions you make to the database, you should back up these files regularly.

To determine how often to back up the database, you should consider the following:

- How much data is entered into the database and how often?
- How hard would it be to reconstruct the data if lost?

1.1.2 Copying Database Files

If your database is small enough to fit on one floppy disk, you can simply copy the four database files. You can use the COPY command in the "R:BASE R> Prompt" window or the operating system COPY functionality. Copying doesn't change the database files. Always copy the four database files together. The files can be copied to a floppy disk or to another location on your hard drive. You can copy the four database files, .RX1, RX2, .RX3, and RX4, as a set, to one floppy disk when you use an operating system wildcard (?, %, or *).

You should store the backup copy of your database on a storage medium that is different than the one on which your original files are stored. For example, if you are using a hard disk to store the data, you can back up your database to a an external hard drive, a flash drive, a tape drive, a CD-ROM or to floppy disks. With these storage options, you can also copy the backup files back to your hard drive. This is especially useful if your original database files become damaged.

1.1.3 Understanding BACKUP and RESTORE

The R:BASE BACKUP and RESTORE commands are useful tools for backing up small databases and for backing up the database structure elements. The R:BASE BACKUP command makes an ASCII file of R:BASE commands; it does not compress the data in any way. The BACKUP command copies your database into a file that has a different format than your database files. This file cannot be used by R:BASE in place of the .RX1, .RX2, .RX3, and .RX4 files. Before you can use the contents of the backup file, you must return it to the four database files.

Unlike copying the database files, the BACKUP/RESTORE commands can copy a large database to multiple floppy disks if your database files are too large to fit on one floppy disk. For a large database, it is often impractical to have that many disks available. Other third party backup utilities that do data compression require fewer disks.

There are two parts to the R:BASE BACKUP command: STRUCTURE and DATA. The STRUCTURE contains table and column definitions including indexes, view definitions, granted access rights, rules, constraints, auto-number definitions, and comments. STRUCTURE does not contain the form, report, or label definitions. Those are actually stored as data. The DATA contains the data from the user-defined tables as well as data from some system tables (forms, reports, labels, and layouts).

The structure and data of a database are backed up together by using ALL. This is not the recommended method as the structure and data are put into a single ASCII file. Ninety-nine percent of the problems with restoring a database involve a problem with re-creating the structure of the database. By putting both structure and data into one large ASCII file, it is very difficult to edit the file to correct structure problems such as reserved or illegal names.

In addition to backing up structure to one file and data to another file for ease of correcting problems with RESTORE, the structure of a database does not usually change once it is set. Data, obviously, changes on a daily basis. By backing up database elements separately, the backup process is customized to ensure accurate and easily restored backups.

The backup process uses three commands:

OUTPUT filename
BACKUP <parameters>
OUTPUT SCREEN

The OUTPUT command creates the file to contain the backup data; the BACKUP command puts the requested data in the file; the second OUTPUT command closes the file. The created file is an ASCII file containing a series of R:BASE commands-the commands that would recreate the database if they were issued at the R> prompt.

The file always begins with a series of SET commands to set the database environment exactly as it was when the BACKUP command was issued. This ensures that the QUOTE, DELIMIT and NULL settings match those in the commands and data in the backup file. Object names, such as table names, will be enclosed within the

IDQUOTES character, the current setting for DELIMIT is used to separate values, text strings are enclosed in the current setting of QUOTES, and the current value of WIDTH affects the width of data lines in the backup file. When backing up DATE data, R:BASE always sets the DATE FORMAT and SEQUENCE to a four-digit year to ensure that the data is restored accurately. The end of the BACKUP file resets these to the database setting.

The following are three tips that will ensure your database is restored from a back up properly:

- Set the null symbol to -0- (the R:BASE default) before backing up.
- Do not set a special character to the same setting as another special character.
- For best results backing up, keep all the default settings.

To view the current database character settings and operating conditions:

Choose **Setting: Configurations Settings** from the Menu Bar.

Since a backup file is an ASCII file, when using the R:BASE BACKUP command to backup up your database or parts of your database, view the file with any text editor to become familiar with the format.

BACKUP does not back up computed-column values; the values will be computed when the database is restored. BACKUP does not change the data or the structure in the original database. Use the RESTORE command to restore your data. If a BACKUP command is included in a transaction when transaction processing is on, the backup cannot be rolled back. The BACKUP command creates a file with a .LOB extension for binary large objects, and a file for the data and/or structure.

1.1.3.1 Backing Up Database Structure

When you backup database structure, R:BASE puts commands in the file to rebuild the structure from scratch. There is always a CREATE SCHEMA command to create or open the database. The CREATE SCHEMA command automatically creates all system tables in R:BASE. The table and column definitions are backed up to CREATE TABLE commands. Views become CREATE VIEW commands. Indexes are CREATE INDEX commands and so on.

If you are transferring information to a different database, you need to edit or remove the CREATE SCHEMA command. It is better to edit the command line to change the database name instead of removing the line. A DISCONNECT command is included in the backup file so that database setting such as EXTENDED and STATICDB can be set. If the CREATE SCHEMA line is deleted, additional lines must also be deleted.

Many converted databases contain table and column names that are reserved words but not illegal. Many times, even illegal names continue to work, depending on where they are used. The CREATE TABLE command, however, errors out on illegal names and the database does not restore. Often, all that is necessary is to use the setting SET ANSI OFF. This limits the reserved word list. It does not allow illegal names or names like OWNER. If you have backed up your database structure to a separate file it is a relatively simple process to edit the file and change the reserved or illegal name. The name is changed not only in the CREATE TABLE command, but in CREATE VIEW commands, in CREATE INDEX and GRANT commands, in RULES definitions, in AUTONUMBER definitions, in ALTER TABLE commands that define constraints, and in comments.

The following sequence of commands backs up database structure, including forms, reports and labels. Forms, reports, and labels would be backed up again each time they are modified or new ones added.

```
OUTPUT dbname.str
BACKUP STRUCTURE
OUTPUT SCREEN

OUTPUT dbname.frm
BACKUP DATA FROM sysform3
OUTPUT SCREEN

OUTPUT dbname.rep
BACKUP DATA FROM sysreps3
OUTPUT SCREEN

OUTPUT dbname.lab
BACKUP DATA FROM syslabels3
OUTPUT SCREEN
```

This backs up all the database structure elements to separate files making it easy to restore individual elements. Structure can be backed up for individual tables or views. Backing up structure for an individual table includes other structure elements defined for that table such as rules, auto-number and comments. Defined access rights are only backed up with the entire structure.

Note that the created file is an ASCII file and contains the database owner password on the CREATE SCHEMA command. Store your R:BASE BACKUP files in a secure location to protect that password.

1.1.3.2 Backing Up Database Data

Your data is, of course, the most important part of the database. The structure can be recreated, but it may take months to reenter data. Some of the data may go back years and not be recoverable. The BACKUP DATA command creates an ASCII file with SET commands at the beginning to set the environment, then a LOAD *tblname* command followed by the data in ASCII delimited format.

If you backup form, report and label data with the structure, do not use the BACKUP DATA command to backup your data tables. That command includes the data for forms, reports and labels. Instead, use the BACKUP DATA FOR *tblname* command and do each table individually. Some tables, such as lookup tables that do not change frequently, are backed up together. Create command or application files to do the backup process. Setting up a backup routine is a one-time process, but it pays for itself the first time you need to RESTORE.

For example:

```
OUTPUT looktab.dat
BACKUP DATA FOR looktab1
BACKUP DATA FOR looktab2
BACKUP DATA FOR looktab3
OUTPUT SCREEN

OUTPUT tblname1.dat
BACKUP DATA FOR tblname1
OUTPUT SCREEN

OUTPUT tblname2.dat
BACKUP DATA FOR tblname2
OUTPUT SCREEN
```

When backing up data, it is important to set the NULL symbol in the database to -0- or some other character that does not appear in your data. When R:BASE loads the data during the restore, it checks the first four characters of each data field to see if it matches the NULL symbol. If it matches, the field is loaded as a NULL. This is why backing up and restoring forms and reports with the NULL symbol set to a blank causes problems - part of the form and report data is blank lines, these blank lines become NULL if restored with the NULL symbol set to a blank and the forms and reports do not work correctly.

In R:BASE, you will have LOB data in your database. All new forms and reports are stored as binary data rather than ASCII data. The BACKUP command backs up all LOB data to a separate file, called a LOB file. The LOB file has the extension .LOB; the first part of the filename is the same as specified in the OUTPUT command. The ASCII file created by BACKUP refers to the LOB file in the LOAD section where the table data is loaded. Both files are necessary to correctly restore the database. The LOB file is assumed to be in the same location as the backup file. Multiple BACKUP commands cannot be done to the same file when LOB data is being unloaded. Each BACKUP command opens the LOB file, but does not append to it; the LOB data is over written with each BACKUP command.

1.1.3.3 Backing Up a Database

An example to back up the ConComp sample database from your hard drive to a floppy disk:

1. Insert a blank formatted floppy disk into drive A.
2. In the Database Explorer, click the Databases option.
3. Select the "Concomp" from the list and click the "Connect" option. If ConComp is not listed, choose "Databases" from the Menu Bar and choose "Connect to Database...". Navigate to the following folder location: C:\RBTI\RBG8\Samples\ConComp. Select ConComp (ConComp.RX1) from the list of files.
4. Choose **Tools: R> Prompt**, or select the R> button on the Tool Bar (third button from the left).

The "R:BASE R> Prompt" window opens.

5. Type "OUTPUT A:\CONCOMP.BUP" and press [Enter].
6. Type "BACKUP ALL" and press [Enter].

R:BASE starts the backup process to the disk in drive A.

7. Type "OUTPUT SCREEN" and press [Enter].

The database backup is now located in drive A. Two files will be located in drive A with the names CONCOMP.BUP and CONCOMP.LOB.

1.1.3.4 Confirming the Backup File was Created

You can confirm that the backup was completed.

To confirm the backup file was created:

R:BASE opens the "R:BASE R> Prompt" window.

At the R>, type "DIR A:" and press [Enter].

R:BASE displays a line of information for each file on the floppy disk. Each line lists the file name, its size in bytes, and the date and the time it was created. The number of bytes for the backup file is relatively large because the backup file contains the information stored in each of the four database files with the .RX1, .RX2, .RX3, and .RX4 extensions.

1.1.3.5 Restoring a Database

If the original database files become damaged, you can restore the files using a backup file. R:BASE uses the backup file to restore the four database files to the same drive and directory of the backup file location. Make sure the current location of the backup file contains enough free space before attempting to restore the database. The RESTORE command uses the file RESTORE.RMD to prompt the user to enter a diskette. The backup file is then executed using the R:BASE INPUT command. The commands in the file are read and executed, and the database is re-built. The RESTORE command does not overwrite existing database structure or data. It uses the LOAD command for data and appends that data to the end of the table. Remember, the backup file is just an ASCII file of R:BASE commands.

Before restoring an entire database, delete or rename the database files (dbname.RX1, 2, 3, 4). Otherwise, you will receive an error stating the database already exists. Before restoring forms, reports or labels, delete all rows from the appropriate system table. Otherwise, the result will be multiple forms, reports or labels with the same name. Before restoring a user table, delete all rows from the table, or copy the table, and delete rows from the original table. Otherwise, again you will have duplicate entries.

Before you restore the damaged database, change the name of the damaged database files. The *ConComp* database really isn't damaged; we are only testing the backup.

To change the name of the damaged database:

1. In the Database Explorer, click the Databases option.
2. Select "ConComp".
3. Choose "Disconnect" to disconnect the damaged database.
4. Click the Rename option.

You are prompted to enter the new name for the database files.

5. Enter "ConComp2" as the new name.
6. Click the OK button.

R:BASE renames the CONCOMP.RX1, CONCOMP.RX2, CONCOMP.RX3, and CONCOMP.RX4 files to CONCOMP2.RX1,
CONCOMP2.RX2, CONCOMP2.RX3, and CONCOMP2.RX4.

1.1.3.6 Using the Restore command

You can restore a database backup created with the RESTORE command. Remember, the RESTORE command restores the four database files to the same drive and directory of the backup file location. You can either copy the backup files to another location, but right now they are still in drive A.

To restore a database:

1. At the R>, type "A:" and press [Enter].

Right now, R:BASE has navigated to drive A.

2. To check your location, type "DIR" and press [Enter] at the R>.

The R> output console will show the current location and the two backup files.

3. At the R>, type "RESTORE CONCOMP.BUP" and press [Enter].

R:BASE will prompt you to continue or stop the operation.

4. Click the OK button.

Then recognizing the first, or only, backup file, you will be prompted to proceed.

5. Enter a "Y" and click the OK button.

The output console will show the creation and loading of tables.

6. When the process is complete, click the OK button.

Now, check drive A for the database files.

7. At the R>, type "DIR" and press [Enter].

The R> output console will show the current location, the four database files, and the two backup files.

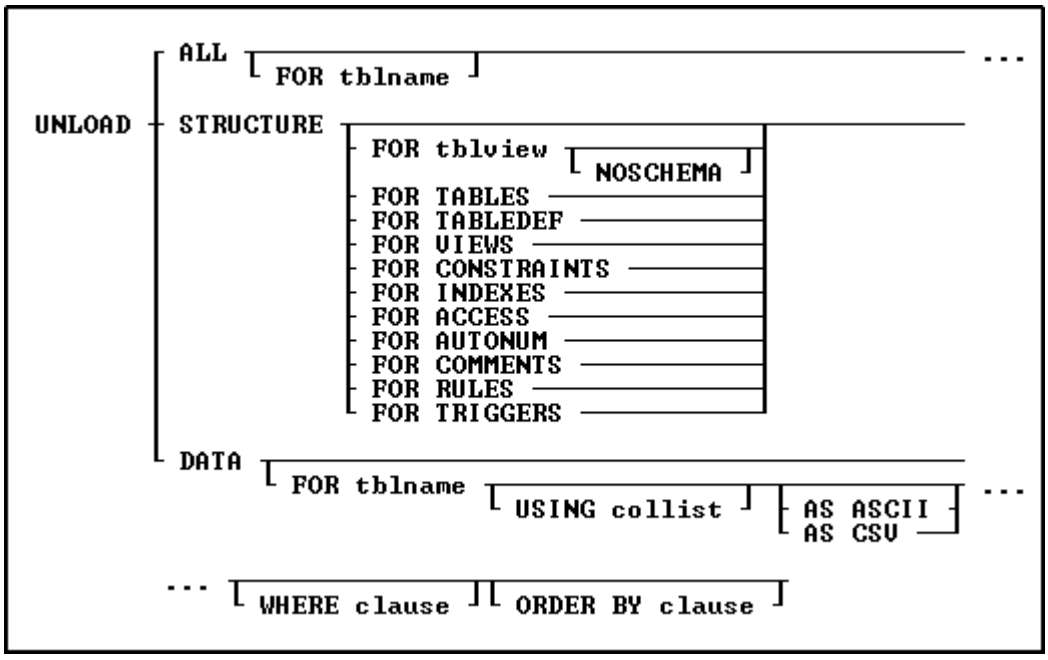
After you are sure that the backup files were successfully restored, you can delete the damaged database.

To delete the damaged database:

1. In the Database Explorer, click the Databases option.
2. Select the damaged database.
3. Click the Delete option.
4. Click the Yes button to verify that you want to delete the selected database.

1.1.4 Using UNLOAD

Use the UNLOAD command to copy the data, structure, or data and structure of a database or table to a specified output device.



Options

ALL
Unloads both the data and the structure.

AS ASCII
Unloads data in ASCII-delimited format. Use only with the UNLOAD DATA command.

AS CSV
Unloads data in a minimally quoted comma separated format. Each field will be separated by the current DELIMIT character (usually the comma). Fields that contain the current DELIMIT character will be enclosed in the current QUOTES character.

DATA
Unloads just the data.

FOR ACCESS
Unloads all current access rights in the database.

FOR AUTONUM
Unloads the AUTONUM column formula for all autonumbered database columns in the database.

FOR COMMENTS
Unloads all comments assigned to tables, views, and columns in the database.

FOR CONSTRAINTS
Unloads all primary key, foreign key, and unique key relational constraints in the database, based on their order of creation.

FOR INDEXES
Unloads all indexes in the database.

FOR TABLES
Unloads the complete table structure for all tables in the database. The output includes the table definition, relational constraints; based on their order of creation, access rights, autonumbered columns, comments, rules, and triggers.

FOR TABLEDEF
Unloads the table definition structure for all tables in the database. The output contains only the SQL command necessary to create all the tables.

FOR tblview
Specifies a single table/view to unload the SQL command necessary to create a specific table/view.

FOR TRIGGERS
Unloads all triggers in the database.

FOR RULES
Unloads all rules in the database.

FOR VIEWS
Unloads the SQL command necessary to create all views.

NOSCHEMA
Omits the character settings and CREATE SCHEMA syntax from the beginning of the output.

ORDER BY clause
Sorts rows of data.

STRUCTURE
Unloads just the structure.

STRUCTURE FOR tblview
Unloads the SQL command necessary to create a specific table/view.

tblname
Specifies the table name to unload the data, structure, or both.

USING collist
Specifies the column(s) to use with the command.

WHERE clause
Limits rows of data.

About the UNLOAD Command

Use UNLOAD to transfer tables or views from one database to another, or to back up a database. You can also use UNLOAD to free up space while using a temporary table.

The UNLOAD ALL and UNLOAD STRUCTURE commands require the database owner's user identifier if the database has had access rights granted with the GRANT command. R:BASE places the owner's user identifier and all the granted access rights in the file created by UNLOAD to ensure that the restored database file continues to be protected. Be sure to protect this file from unauthorized users.

The UNLOAD command creates a file with a .LOB extension for binary large objects, and a file for the data and/or structure.

UNLOAD with a click

From the R:BASE Database Explorer window, you can right click on any individual table, and choose from various UNLOAD options to quickly and easily unload from a table:

- Unload > Structure (Table/View)
- Unload > Definition
- Unload > Definition with Constraints, Indexes, Comments, etc.
- Unload > Structure and Data

Transferring Tables and Views

UNLOAD does not change the data or structure of the original database, and it does not unload computed column values unless you use the AS ASCII or AS CSV option.

You can also unload and transfer the data and structure of a view. To input the data from an unloaded view into a table, create a table to match the view's structure and use the LOAD *tblname* FROM *filespec* command,

since views do not contain data. The UNLOAD command is useful when you want to create a file to transfer data to another database as a table.

To be able to reliably load data back into R:BASE from an unloaded file:

- Set null to -0- (the R:BASE default) before unloading data.
- Do not set a special character to the same setting as another special character.
- Precede UNLOAD with an OUTPUT command to direct the data to be unloaded to a file. You can edit the unloaded file with any ASCII text editor.

To transfer unloaded information back into R:BASE, use the RUN, RESTORE, or GATEWAY commands, if transferring only data. For example, you can use the UNLOAD DATA command to unload data, then the RUN command to transfer the data to a different database.

If the UNLOAD AS CSV syntax has been used you can use the LOAD AS CSV command to restore the data.

Backing up a Database

R:BASE unloads data in ASCII delimited format: values are separated by the current delimiter and all text strings are enclosed in quotation marks. UNLOAD creates a file containing commands that set special characters, such as commas and quotation marks. The setting of the SET WIDTH command affects the width of data lines in the unloaded file

If the database has columns defined as binary or text large objects, then UNLOAD creates two files, one file containing the R:BASE commands, and a second file (with a .LOB extension) containing the large object data. Both files are needed to transfer the information back into R:BASE.

Note: The unloaded file cannot span multiple floppy disks.

The UNLOAD STRUCTURE or UNLOAD ALL commands write all the commands necessary to define the database or table, starting with CREATE SCHEMA AUTHORIZATION *dbname* near the beginning of the file. Before you input or restore the data or structure into a different database, you can use a text editor to change the database name in the unloaded file. If you use UNLOAD STRUCTURE, you can copy the table structure after you change the database name in the unloaded file.

Unloading Temporary Tables

Use the UNLOAD *tblname* command to backup individual temporary tables created when STATICDB is set on-which activates a read-only schema mode. When UNLOAD is used to backup temporary tables, it generates a SET STATICDB OFF command to be executed prior to the CREATE SCHEMA command. For more information about the SET STATICDB command, see the "SET STATICDB" entry.

Note: UNLOAD ALL does not back up temporary tables.

Examples

- The following command lines unload only the data from the *product* table to a file named MYFILE.DBS. The data is in ASCII delimited format. The OUTPUT SCREEN command redirects the output back to the screen and closes the file.

```
OUTPUT myfile.dbs
UNLOAD DATA FOR product AS ASCII
OUTPUT SCREEN
```

- In the example below the a file will be created that contains Comma Separated Values with no headings and no page breaks.

```
SET HEADINGS OFF
SET LINES 0
SET WIDTH 200
OUTPUT myfile.csv
UNLOAD DATA FOR Employees AS CSV
OUTPUT SCREEN
```

The commands above might create the file below. Notice that Jane Dough has Quotes surrounding her address.

This is because the text contains an embedded comma.

```
Robert,Smith,123 Main St,Denver,CO,Support
Jane,Dough,'98 Folk St, Apt 1',Pittsburgh,PA,Sales
Matt,Follows,14 Arrowhead Ln,Portsmouth,RI,Services
```

- The following command unloads the complete structure for the Contact table within the ConComp sample database.

```
R>UNLOAD STRUCTURE FOR contact

SET QUOTES=NULL
SET QUOTES='
SET DELIMIT=NULL
SET DELIMIT=','
SET LINEEND=NULL
SET LINEEND='þ'
SET SEMI=NULL
SET SEMI=';'
SET PLUS=NULL
SET PLUS='+'
SET SINGLE=NULL
SET SINGLE='_'
SET MANY=NULL
SET MANY='%'
SET IDQUOTES=NULL
SET IDQUOTES='`'
SET CURRENCY '$' PREF 2 B
DISCONNECT
SET STATICDB OFF
SET ROWLOCKS ON
SET FASTLOCK OFF
CREATE SCHEMA  AUTHOR ConComp  NONE
CREATE TABLE `Contact`  +
(`CustID` INTEGER  ,  +
 `ContFName` TEXT    (10) NOT NULL  +
('Value for column Contfname cannot be NULL.') ,  +
 `ContLName` TEXT    (16) NOT NULL  +
('Value for column Contlname cannot be NULL.') ,  +
 `ContPhone` TEXT    (12) ,  +
 `ContInfo` NOTE     ,  +
 `ContPhoto` LONG VARBIT   )
ALTER TABLE `Contact` ADD FOREIGN KEY +
 ( `CustID` )+
 REFERENCES `Customer`
COMMENT ON `CustID` IN `Contact` IS +
'Customer identification number'
COMMENT ON `ContFName` IN `Contact` IS +
'Customer contact first name'
COMMENT ON `ContLName` IN `Contact` IS +
'Customer contact last name'
COMMENT ON `ContPhone` IN `Contact` IS +
'Customer contact phone number'
COMMENT ON `ContInfo` IN `Contact` IS +
'Customer contact notes'
COMMENT ON `ContPhoto` IN `Contact` IS +
'Customer contact photo'
COMMENT ON TABLE `Contact` IS +
'Customer Contact Information'
```

- The following command unloads the structure for the Contact table within the ConComp sample database, without any database schema at the beginning.

```
R>UNLOAD STRUCTURE FOR contact NOSCHEMA
```

```
CREATE TABLE `Contact`  +
(`CustID` INTEGER  ,  +
 `ContFName` TEXT    (10) NOT NULL  +
('Value for column Contfname cannot be NULL.') ,  +
 `ContLName` TEXT    (16) NOT NULL  +
('Value for column Contlname cannot be NULL.') ,  +
 `ContPhone` TEXT    (12) ,  +
 `ContInfo` NOTE     ,  +
 `ContPhoto` LONG VARBIT   )
ALTER TABLE `Contact` ADD FOREIGN KEY +
 ( `CustID` )+
 REFERENCES `Customer`
COMMENT ON `CustID` IN `Contact` IS +
'Customer identification number'
COMMENT ON `ContFName` IN `Contact` IS +
'Customer contact first name'
COMMENT ON `ContLName` IN `Contact` IS +
'Customer contact last name'
COMMENT ON `ContPhone` IN `Contact` IS +
'Customer contact phone number'
COMMENT ON `ContInfo` IN `Contact` IS +
'Customer contact notes'
COMMENT ON `ContPhoto` IN `Contact` IS +
'Customer contact photo'
COMMENT ON TABLE `Contact` IS +
'Customer Contact Information'
```

- The following command unloads the indexes for the ConComp sample database.

```
R>UNLOAD STRUCTURE FOR INDEXES

CREATE     INDEX CustState ON `Customer` +
(`CustState` ASC SIZE 2 )
```

- The following command unloads the rules for the ConComp sample database.

```
R>UNLOAD STRUCTURE FOR RULES

RULES 'Value for onhand cannot be less than minimum.' +
               FOR `ProdLocation` SUCCEEDS  +
               WHERE ProdLocation.Onhand >= 1
```

1.1.5 BACKUP Versus UNLOAD

The UNLOAD command is very similar to the BACKUP command. The file it creates contains the same R:BASE commands to rebuild the database. There are three differences:

1. The BACKUP command backs up the data to multiple floppy disks and the user is prompted to put in a new disk when a disk is full. The UNLOAD command backs up to a single file; it does not back up to multiple floppy disks.
2. The UNLOAD command has the "AS ASCII" or "AS CSV" options for creating a file of data only, with no R:BASE commands.
3. The UNLOAD command unloads data from computed columns or expressions.

The AS ASCII option creates a delimited ASCII file of data with no R:BASE commands. There are no SET commands or LOAD *tblname* command. The file is delimited with commas (or the current DELIMIT setting) separating the fields, and text data enclosed in single quotes (or the current QUOTE setting). The AS ASCII option is used to create a file to transfer data to another program, or to a different table. A file created using UNLOAD DATA...AS ASCII is put back into a database using the LOAD command. Only one table can be unloading to a file using the AS ASCII option. There are no R:BASE commands separating the data from different tables.

The BACKUP command assumes that the data is returning to an R:BASE database. BACKUP includes R:BASE commands that set the database environment and identify the table to load the data into. Since the data is loaded back into an R:BASE table, data for computed columns is not unloaded. It is re-computed as the data is loaded. The AS ASCII option includes no R:BASE commands and is just data. As such, UNLOAD DATA...AS ASCII includes the data from computed columns in the output file. In addition, the AS ASCII option allows the use of expressions in the USING *collist*.

For example, the code below creates the file that follows it:

```
UNLOAD DATA FOR employee USING +
empid, (empfname & emplname) , empaddress, + (empcity + ',' & empstate & empzip) AS ASCII
```

```
102, 'June Wilson', '3278 Summit Drive', 'Seattle, WA 98115'
129, 'Ernest Hernandez', '12390 Windermere Dr.', 'Seattle, WA 98115'
131, 'John Smith', '3050 N.E. 41st', 'Seattle, WA 98105'
133, 'Peter Coffin', '4105 29th Ave N.E.', 'Duvall, WA 98004'
160, 'Mary Simpson', '101 West Mercer', 'Redmond, WA 98052'
165, 'Darnell Williams', '8806 88th Street', 'Seattle, WA 98103'
166, 'John Chou', '5001 Main Street', 'Woodinville, WA 98072'
167, 'Sandi Watson', '1002 S. Front Ave.', 'Redmond, WA 98052'
```

The full name and the city, state and zip code are considered single fields. This option makes it easy to format the data for transfer to other software programs.

Both the UNLOAD and BACKUP commands allow specifying a set of columns to unload. A WHERE clause and an ORDER BY clause are used when a single table is unloaded.

The UNLOAD command deals with LOB data the same way as the BACKUP command. A LOB file is created containing all the LOB data in the database. Multiple UNLOAD commands cannot be done to the same file when LOB data is being unloaded. Each UNLOAD command opens the LOB file, but does not append data to it; data is overwritten each time the LOB file is opened.

1.2 Checking the Database

Because your information is important, you want to check the database for errors before making a backup. You don't want to back up a damaged database. In addition, you should check a database after power failures, electrical storms, and any time the database is not exited normally. And errors accessing data such as, "Disk problems. Please check disk and files" should be investigated for possible database damage.

There are two different ways to check your database structure and data pointers; R:SCOPE and AUTOCHK.

1.2.1 R:SCOPE

R:SCOPE checks the database structure and data pointers but incorporates database correction features. The R:SCOPE documentation includes detailed information about R:BASE database file structure and data storage. But you may not have R:SCOPE and you'd like to know the possible damage right away.

R:SCOPE is a tool that lets you examine and modify R:BASE database files outside R:BASE. R:SCOPE can check a database for problems and lets you look in the database files to diagnose a problem. Once you have determined the problem, R:SCOPE allows you to fix the structure of a database and correct data file errors.

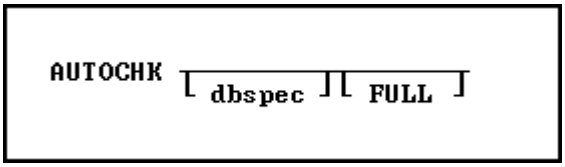
Once you are familiar with the program, you can use R:SCOPE to check your database on a regular basis. You might want to make a habit of checking the database with R:SCOPE before packing the database with the R:BASE PACK or RELOAD commands, or before making a backup of the database with either the BACKUP or Copy commands.

http://www.rbase.com/products/rscope/

For more information on R:Scope, please contact R:BASE Technologies, Inc. by phone at 1+724.733.0053 or through email at sales@rbase.com

1.2.2 AUTOCHK

AUTOCHK is included with R:BASE. AUTOCHK is command designed for use to help diagnose database problems. AUTOCHK checks the structure file (RX1) and the pointers in the data files (RX2, RX4), but does not check the index file (RX3). AUTOCHK can be used at the R> or in a command file that can be easily incorporated into an application. AUTOCHK returns a value to the R:BASE error variable. An AUTOCHK value other than 0 indicates damage to the database. AUTOCHK does check the entire database. AUTOCHK does not correct damaged databases, it only indicates if there is damage. You cannot run AUTOCHK against a database that is currently being used. Everyone using a database must be disconnected from the database or exited from R:BASE in order to check the database.



AUTOCHK Options

dbspec
Specifies a database other than the open database to check; otherwise, the open database is checked.

FULL
Provides detailed information about the processing being performed, and when AUTOCHK encounters an error, it continues processing.

About the AUTOCHK Command

Use the AUTOCHK command to ensure that the connected database is intact before using the PACK or RELOAD commands, or before making a backup of the database with either the BACKUP or COPY commands. If you are using R:BASE for Windows, you can choose **Utilities: Backup/Restore Database**.

Please Note: If any user connected to the database has temporary tables or views created you may receive an abnormal amount of errors. This is expected and is a side effect of having temporary tables active during the check. For completely accurate results, have all users disconnect from the database to be checked. AUTOCHK checks the following:

- The structure-file block sizes and locations.
- The timestamps for all database files.
- The database-file lengths.
- The number of tables and columns.
- The starting and ending pointers for tables.
- The location of columns.
- The File 4 (.RX4) data pointers.
- The data types of columns.
- The size and number of rows in each table.
- The row pointers in the data file.

AUTOCHK does not check indexes.

When you run AUTOCHK, it systematically checks the structure file (DBNAME.RX1) of the open database, and the data files (DBNAME.RX2 and DBNAME.RX4). AUTOCHK only checks the index file (DBNAME.RX3) for the timestamp and length of the file. When opening a database, AUTOCHK ignores any user-identifier protection. AUTOCHK without the FULL option sets the R:BASE error variable to a non-zero value if errors are found.

The results of AUTOCHK with the FULL option are displayed on screen, or the current OUTPUT device. First, AUTOCHK validates the timestamps in the database files, then systematically checks the structure of each table and view in the database, providing a list of columns, constraints, and indexes for each. Any structure errors are noted after each table listing.

Database Statistics

Next, AUTOCHK checks the data for each table, listing active rows and deleted rows. Any problems with data,

such as broken pointers, are listed after the respective tables. Finally, AUTOCHK provides a summary of the database structure, including the number of tables, columns, and indexes, and the actual space that the data occupies in the data file (DBNAME.RB2). AUTOCHK shows the percent of space used for the items in each list to give an idea of how much space has been used, and to indicate the need to recover space in the database files. Any numbers less than 100 percent indicate the need to pack or reload the database using the PACK or RELOAD commands. For more information about the PACK and RELOAD commands, see the "PACK" and "RELOAD" entries.

The following section contains information about using AUTOCHK in application files and capturing the error variables returned. This allows the application developer to prevent users from continuing to use a corrupted database.

```
SET ERROR VAR E1
WRITE 'Checking database for errors...'
AUTOCHK dbname
IF E1 > 40 THEN
   WRITE 'AUTOCHK has found errors in the database!'
   BEEP
ENDIF
If E1 > 0 and E1 < 50 THEN
   WRITE 'AUTOCHK will not run - User Abort or Out of Memory'
   BEEP
ENDIF
IF E1 = 0 THEN
   WRITE 'AUTOCHK successful - No errors found'
ENDIF
PAUSE 2
RETURN
```

If AUTOCHK with no option finds an error, it stops checking the database and displays one error message. If the error message (see list below) indicates that the database is damaged, you might want to start using a backup copy of the database. Alternatively, you might want to use R:SCOPE, a database repair tool available from R:BASE Technologies, Inc.

If AUTOCHK finds no errors, it displays the message "NO ERRORS FOUND." If you press any key while AUTOCHK is checking the database, the program stops and displays the message "USER ABORT." AUTOCHK automatically sets the error variable to the number corresponding to the message returned. For example, if the error "UNABLE TO OPEN DATABASE FILE 2" is returned, the error variable is set to 52.

Example

The following is an example of how to put AUTOCHK results in a file for viewing:

```
DISC
OUTPUT dbname.chk
AUTOCHK dbname FULL
OUTPUT SCREEN
```

You can view DBNAME.CHK in the R:BASE Editor to view the results.

1.2.2.1 Multi-User Databases

Use caution when running AUTOCHK in a multi-user environment. If the database being checked is currently open with MULTI set on, AUTOCHK places a database lock on the database. The database lock remains in effect until AUTOCHK stops checking the database. Database users are unable to make any changes to the data or structure of the database while this lock is in place.

If a user attempts to open a database being checked by AUTOCHK and the database does not have any other users, the user receives an error message indicating that the database is currently open in a mode that makes it unavailable. If other users have the database open with the MULTI set on and the database is being checked, the user attempting to open the database receives a message indicating the user is waiting in a lock queue. If AUTOCHK successfully completes checking the database and finds no errors, it reports that no errors were found and sets the error variable to 0.

Checking continues in multi-user mode (even if a database lock cannot be obtained) if a database is connected by another user; however, row errors in File 2 can occur because of database activity.

1.2.2.2 Error Messages

AUTOCHK displays one of the following messages when it is unable to start checking or complete checking the database or when it finds an error in the database files. AUTOCHK returns *0 No errors found* if the database is okay. Some of these messages indicate that the database is damaged. Either switch to a backup copy of the database, or attempt repair of the database using R:SCOPE, R:BASE Technologies's database repair tool. If AUTOCHK is unable to open File 1 of the database, check that the path you specified to the database is correct; or, if you are trying a multi-user database, check that no other user has the database connected with MULTI set off.

Checking continues in multi-user mode (even if a database lock cannot be obtained) if a database is connected by another user; however, row errors in File 2 might occur because of database activity.

Any of these messages, except the first (code 0), indicates that the database is damaged. Either switch to a backup copy of the database, or attempt repair of the database using R:SCOPE, R:BASE Technologies's database repair tool.

AUTOCHK Error Messages

Number	Code Message
0	No errors found
1	This database is not of the correct version
2	The database filenames must all match
20	Out of memory
40	User Abort
50	Unable to open database file number 1
51	Unable to lock this database
52	Unable to open database number 2
53	Unable to open database number 3
54	Unable to open database number 4
55	Error reading the database information block
56	Error reading the timestamp information
57	Timestamp in file number 2 does not match file 1; run RBSYNC
58	Timestamp in file number 3 does not match file 1; run RBSYNC
59	Timestamp in file number 4 does not match file 1; run RBSYNC
60	Invalid number of tables
61	Invalid number of columns
62	Invalid number of indexes
63	File 1 is too small
70	Error in database structure block
80	Error reading the table list
81	Error reading the column list
82	Error reading the index list
100	Incorrect version flag
101	Error reading Case Folding and Collating tables
110	Error in DBinfo block offset
111	Error in DBinfo block length
120	Error in length of database file 2
121	Error in length of database file 3

1.2.3 Check the Database Before a Backup

You should always check the database before doing a backup to make sure there are no errors in the database. AUTOCHK was designed specifically to run from an application to check the database before a backup or other maintenance procedure is run. If the AUOTCHK result shows errors, you don't want to perform the backup or other maintenance procedure such as PACK or RELOAD.

Applications should include a menu choice or automatic procedure to check the database. In R:BASE, you have the ON CONNECT and ON PACK commands that can be used to check the database before packing or connecting the database. For example, you might add the ON PACK command to the startup block of your application,

```
ON BEFORE PACK RUN autochk.cmd
```

Then, before a PACK command in the application is executed, R:BASE runs the command file AUTOCHK.CMD. The command file checks the database for errors. If no errors, the PACK is executed; if there are errors, a message is displayed and the PACK command is aborted. The file might look like this:

```
SET ERROR VAR verror
AUTOCHK
IF verror <> 0 THEN
  ABORT ON
  PAUSE 2 USING + 'Errors found. Contact database administrator!'
ENDIF
RETURN
```

A similar routine could be run ON FIRST CONNECT, which would check the database when the first person connects each day. The ABORT ON command is used to not execute the CONNECT or PACK command if an error is found.

1.2.4 Database Integrity Routine

Follow the steps below and to validate the integrity of your database.

1. Make a backup copy of the database.

2. Copy the entire database (.RX1-.RX4) to your "local drive" in a separate folder with at least twice as much available free disk space.

3. Start R:BASE using the latest version/update and switch the current folder to the appropriate database folder on your local drive.

4. At the R>, connect to the database:

   ```
   DISCONNECT
   SET MESSAGES ON
   SET ERROR MESSAGES ON
   SET MULTI OFF
   -- CONNECT the database with OWNER password, if any
   CONNECT dbname IDENTIFIED BY owner password
   -- CONNECT the database with no passwords
   CONNECT dbname
   ```

5. Check the connected database CHARacter settings, especially the IDQUOTES. At the R> prompt:

   ```
   CLS
   SHOW CHAR
   ```

   Notice the setting for IDQUOTES (the last item on list).

   If this setting is blank/null, make sure to set the IDQUOTES settings to ` (that is a single reversed quote, on the same key as the ~ tilde). At the R>, type:

   ```
   SET IDQUOTES=`
   ```

6. Now create the unload file with NULL set to -0-. At the R> prompt:

```
CLS
SET NULL -0-
OUTPUT newdb.ALL
UNLOAD ALL
OUTPUT SCREEN
```

   This step will create two files (newdb.ALL and newdb.LOB)

7. DISCONNECT, and then rename this database to some other name. At the R> prompt:

```
DISCONNECT
RENAME dbname.RX? dbnameBK.RX?
```

8. Now, let's rebuild the fresh database and see if it passes the integrity check. At the R> prompt:

```
CLS
RUN newdb.ALL
```

   Watch the activity and all messages on the screen. Don't fall asleep. You might miss an important warning. Completion time may vary based on size of the database (number of tables/records/indexes)

If there were no warnings or error messages, you've got a fresh/healthy database. Give yourself a pat on the back.

Errors?

1. If there are any warning or -ERROR- message(s), take them seriously. In that case, disconnect and delete the bad database (built using RUN newdb.ALL). At the R> prompt:

```
DISCONNECT
DELETE dbname.rx?
DELETE newdb.ALL
DELETE newdb.LOB
```

2. Rename and connect to the previously saved database (Step 7).

```
RENAME dbnameBK.RX? dbname.RX?
CONNECT dbname
```

3. CORRECT all -ERROR-s accordingly and then repeat Step 6.

4. Do not quit or give up until you see a completely fresh database without any warnings or errors.

If there is a considerable amount of errors with both rebuilding the structure and/or data, please refer to the Database Repair Routine.

1.3 Database Corruption

What happens when the database is corrupt or damaged? The first thing to do is evaluate the damage and determine if you need to restore a backup copy or fix the database. Sometimes the database can be fixed by just rebuilding indexes. Usually you need to look at all the errors in a database to determine the next step.

Part of evaluating database damage is deciding if reentering data into a backup copy is quicker than trying to fix a database. That is often the case, particularly if you are not familiar with R:SCOPE yourself and would need to call in a developer or send the database for repair. In addition, don't forget that whenever you correct a table with R:SCOPE, you need to rebuild the indexes for that table. Then, you can use the PACK INDEX indexname command to singly rebuild just the indexes in the affected table. After making corrections to a table, you must rebuild the indexes using PACK.

Remember, damage to a database or computer system most often happens due to accident, oversight, or negligence. Document and have ready backups of all your database and application files. It is not an easy or

quick process to plan for disasters happening to your database, and then to decide what to do when the unthinkable does happen, but being prepared can make all the difference between being down for four hours or for four days.

There are varying degrees of database corruption that can be determined on what type of functionality R:BASE allows.

Level	R:BASE Behavior
High	R:BASE will error and close when opening a table, UNLOADing the entire database, or will not let you connect to the database at all
Medium	R:BASE will error and close when UNLOADing the structure or data for an individual table, "broken pointer" error messages are displayed, strange ASCII characters are displayed when using LIST ALL at the R>, strange ASCII characters are displayed in rows of data, duplicate, integer or blank tables may appear in the table list
Low	R:BASE displays errors during normal processing and does not close, duplicate output is returned when querying data, form or report corruption preventing use of he designers

1.3.1 HIGH

With high levels of database corruption, R:BASE will error and close when opening a table or when you are UNLOADing the entire database into an output file. It some cases, you not even be able to connect to the database with R:BASE, but with R:SCOPE, it may be an easy repair. With the ability to connect to the database, your first two priorities is to save the structure and then the data.

These different examples do not include all forms of high level corruption, as corruption occurrences can differ.

Problem	Suggestion
R:BASE closes when performing an UNLOAD ALL on the database	You may be forced the UNLOAD command to capture the different portions of the structure individually. Use UNLOAD DATA to capture all of the data. Refer to the UNLOAD command within the R:BASE Command Index in the main Help file.
R:BASE closes when opening tables or other modules	You may be forced the UNLOAD command to capture the different portions of the structure individually. Use UNLOAD DATA to capture all of the data. Refer to the UNLOAD command within the R:BASE Command Index in the main Help file.
R:BASE closes when connecting to the database	You must use R:SCOPE to find where the database problems exist.
You cannot connect to the database	You must use R:SCOPE to find where the database problems exist.

1.3.2 MEDIUM

With medium levels of database corruption, R:BASE will error and close when UNLOADing the structure or data for an individual table, or you may receive "broken pointer" error messages during data processing. Any form of strange ASCII characters displayed in the table records or at the R> is another example. And, with some structural corruption, you may see duplicate, integer or blank table names appear in the table list.

These different examples do not include all forms of medium level corruption, as corruption occurrences can differ.

Problem	Suggestion
R:BASE closes when performing an UNLOAD of a table's structure or data	You may be forced to UNLOAD STRUCTURE of the structure into one file to rebuild the database structure. Use UNLOAD DATA to capture all of the data. Refer to the UNLOAD command within the R:BASE Command Index in the main Help file.
Strange ASCII characters at R> output console	Based upon where the ASCII charcaters are being displayed within the database structure, e.g. RULES, COMMENTS, use the UNLOAD command to capture just that portion of the database structure.
Strange ASCII characters within table records	Save any understandable data for the row, then delete the row from the table.
R:BASE Pointer Error Message: *"Table tablename has an incorrect number of rows."* *"Table tablename has a bad last row pointer."* *"A table's starting pointer is out of range."* *"A table's ending pointer is out of range.*	You must use R:SCOPE to edit the database pointers. Be sure to review the R:SCOPE documentation before working on the database.
Integer value table name, e.g. *4875E596*	This can occurs when processing for a table was abnormally aborted leaving a duplicate table, only with an integer value for the name. It is common that the table name is the same as the R:BASE temporary file name that was used during the processing, **4875E596.$$$.** First confirm that the originating table contains the correct number of rows. Then, rename the integer value table to a text value. Now, you can permanently delete it.
Blank table appears in the table list	This occurs when processing for a table, a PACK, or RELOAD was abnormally aborted leaving a blank table name. To repair, UNLOAD the structure and data into two separate files. Then, edit the structure file to remove the blank table and save the file. To rebuild the database, run the structure file, then the data file.
Duplicate table name in table list	This can occurs when processing for a table was abnormally aborted leaving a duplicate table name. To repair, UNLOAD the structure and data into two separate files. Then, edit the structure file to rename one of the duplicate tables and save the file. You will also need to edit the data file to avoid loading the data twice from both tables that were unloaded. To rebuild the database, run the structure file, then the data file.

1.3.3 LOW

With low levels of database corruption, R:BASE can display error messages during normal data processing and will not close. Or, it is possible for duplicate output to be displayed when querying data at the R> and the results differ from what is stored in the table. And, with some form or report corruption, you will be prevented from accessing the Form or Report Designers.

These different examples do not include all forms of low level corruption, as corruption occurrences can differ.

The following is a list of possible low level forms of corruption and how you may be able to correct the problem:

Problem	Suggestion
R:BASE Error Message: *"I/O problems - check for full disk"*	Try to PACK the database indexes. Refer to the PACK command within the R:BASE Command Index in the main Help file.
R:BASE Error Message: *"Incorrect file legnth"*	Use R:Scope to adjust the file legnth of the database file.
R:BASE Error Message: *"Database File are out of sync"*	Use the RBSYNC command to reset the database file date and time stamps.
Form Designer Error Message: *" "char" expected on line ###"* *"Invalid binary value on line ###"* *"Invalid string constant on line ###"*	The raw form data has been slightly altered. You can repair the form by editing the SYS_DATA data within the SYS_FORMS3 table. In order to edit this raw data, Form Compression within your Form Designer settings must be set OFF prior to the corruption.
Duplicate data appears in queries	Try to PACK the database indexes or just the indexes within the table. Refer to the PACK command within the R:BASE Command Index in the main Help file.
Data Browser Error Message: *"Invalid argument to date encode"*	Invalid date formats exist in a DATE data type column. This problem stems from an invalid date which was enetred into the database from a previous version of R:BASE. Return to the previous version and review the table data in all DATE columns.

1.3.4 Database Repair Routine

The following routine will rebuild your database at 100%. It is recommend to use this routine if the Database Integrity Routine results in errors during the rebuild process. The routine can also be used if after using the PACK or RELOAD commands and there is a loss of table data.

1. Make a backup copy of the database.

2. Copy the entire database (.RX1-.RX4) to your "local drive" in a separate folder with at least twice as much available free disk space.

3. Start R:BASE using the latest version/update and switch the current folder to the appropriate database folder on your local drive.

4. At the R>, connect to the database:

```
DISCONNECT
SET MESSAGES ON
SET ERROR MESSAGES ON
SET MULTI OFF
-- CONNECT the database with OWNER password, if any
CONNECT dbname IDENTIFIED BY owner password
-- CONNECT the database with no passwords
CONNECT dbname
```

5. Check the connected database CHARacter settings, especially the IDQUOTES. At the R> prompt:

```
CLS
SHOW CHAR
```

Notice the setting for IDQUOTES (the last item on list).

If this setting is blank/null, make sure to set the IDQUOTES settings to ` (that is a single reversed quote, on the same key as the ~ tilde). At the R>, type:

```
SET IDQUOTES=`
```

6. Check that your NULL setting is set to "-0-" or some other value besides being set to blank.

```
R> SHOW NULL

  NULL symbol -0-
```

   If the setting is blank, set it to -0- using the following command:

```
SET NULL -0-
```

7. Unload the database structure into an output file using the following commands:

```
OUT DB_STR.STR
UNLOAD STRUCTURE
OUT SCREEN
```

8. In the current directory, open the DB_STR.STR output file with R:BASE Editor.

```
RBEDIT DB_STR.STR
```

9. Navigate to the location of the file where the last TABLES is created and "before" any VIEWS are created. Search for "CREATE VIEW", which should bring you to the first view listed in the structure file. At this point, save the top portion of the file as DB_STR1.STR and the bottom portion as DB_STR2.STR.

10. Now, it is time to get the data out. This step may take a considerable amount of time with large databases. Unload the database data using the following commands:

```
OUT DB_DATA.DAT
UNLOAD DATA
OUT SCREEN
```

    This step will create two files (DB_DATA.DAT and DB_DATA.LOB)

11. Next, disconnect and rename the problem database in this folder.

```
DISCONNECT
RENAME dbname.RX? TO db_bckup.RX?
```

12. Now, the rebuild process begins. First, the initial database table structure must be rebuilt. During this step, we will capture any possible errors into a LOG file.

```
OUT STR1_ERR.LOG
RUN DB_STR1.STR
OUT SCREEN
```

13. The database should be recreated with just tables containing no data rows. Check the contents of the error log to see if there were any issues.

```
RBEDIT STR1_ERR.LOG
```

    If any errors are listed, likely causes include:

    - common columns in the same table
    - common columns with different data types in different tables

    The extent of the error(s) will decide how to fix the database table structure. To insure the database is rebuilt error free, you can:

    - (few errors) make the necessary fixes in the DB_STR1.STR file. After the fixes are made, you must disconnect from the database, delete this new "structure-only" database and repeat Step #8.

    - (many errors) edit the original database table structure. After the fixes are made, you must delete this new "structure-only" database and start from the beginning at Step #1.

14. With an error free table structure it is time to load the data into the database tables. During this step, we

will once again capture any possible errors into a LOG file. Use the following commands to do so:

```
OUT DATA_ERR.LOG
RUN DB_DATA.DAT
OUT SCREEN
```

15. During this process all the data, forms, reports, and labels will be loaded. Once the process is complete, check the error log.

    ```
    RBEDIT DATA_ERR.LOG
    ```

    If any errors are listed, likely causes include:

    - computed TEXT columns that have not been assigned a large enough length
    - invalid value for a column data type
    - incorrect number of values for a table

    Fix any errors that prevent the data from being loaded successfully by performing the fixes on the original database. After the fixes are made, you must delete this new "structure with partial data" database and start from the beginning at Step #1.

16. Finally, load second half of the database structure. Use the following commands to do so:

    ```
    OUT STR2_ERR.LOG
    RUN DB_STR2.STR
    OUT SCREEN
    ```

17. During this process the views, constraints, indexes, access rights, autonums, rules, and comments are loaded into the database. Once the process is complete, check the error log.

    ```
    RBEDIT STR2_ERR.LOG
    ```

    If any errors are listed, likely causes include:

    - views based on tables that no longer exist
    - duplicate index names

If this step completes with no errors, you have a rock solid database in your hands!

Why?

You may be asking why did the instructions split the structure output into two files. The reason is that normally you would direct the complete database structure and data output into a single file using UNLOAD ALL. With UNLOAD all, the output is listed in the following order:

1. CHAR Settings
2. CURRENCY Settings
3. SCHEMA
4. TABLES
5. DATA
6. DATE Format
7. VIEWS
8. CONSTRAINTS and INDEXES - based on order of creation
9. USERS
10. PRIVILEGES
11. AUTONUM
12. COMMENTS
13. RULES
14. TRIGGERS

So, with a corrupt database where running the UNLOAD ALL output results in errors, by breaking the structure file into two parts, you are less likely to have problems adding constraints where data already exists, than loading data where constraints already exist. Of course, this depends on the level of corruption errors you are encountering and the integrity of your data.

1.4 Compressing the Database

When you delete rows or remove columns or tables from a database, the information is gone, but the disk space it occupied remains unusable because R:BASE still regards it as assigned space. Also, as you add or modify data in tables, R:BASE stores the data wherever space is available on the disk. Over time, the rows of a particular table can become scattered over the files, causing R:BASE to spend more time looking for the rows when you use the table.

1.4.1 Using PACK

If you have not added, changed, or deleted many rows in your database, use the Pack In Place option. Pack In Place eliminates the space occupied by deleted rows, columns, or tables within the existing database. This option cannot reorganize rows on the disk. Pack In Place also has options that allow you to restructure indexes or compress the database structure.

Before you compress your database, make sure you have a current backup of it. If your database is large, the compression can take a considerable amount of time, sometimes several hours. If the operation is interrupted for any reason, your database will be damaged or destroyed.

Pack In Place has three options:

Schema - eliminates unused space from the database structure only
Keys - packed the entire .RX3 file and rebuilds indexed columns
All - eliminates unused space from both database structure and database data and rebuilds indexed columns

Using PACK, you can also compress the SYS_PASSWORDS system table, compress an individual table, or an individual index.

In the following exercise, you'll compress the database using the Pack in Place... menu option.

To compress the database files:

1. Choose **Database: Connect to Database...**
2. In the "Select a Database" dialog box, select a database to compress.
3. Click the OK button.
4. Choose **Utilities: Pack in Place...**
5. In the "Pack In Place" dialog box, select "All."
6. Click the OK button.

R:BASE displays the following warning message: "You should back up this database before you PACK it. Continue with an in-place PACK of your database?"

Click the Yes button to continue with the process because you already have a backup copy of the database.

R:BASE displays all the tables, columns, and indexes as it packs them. Pack In Place rebuilds the indexes for all indexed columns. If messages are set off in this database, you will not see these messages.

1.4.1.1 PACK TABLE using SET RECYCLE

Platform: R:BASE 7.5, 7.6 and V-8 Turbo for Windows
Builds: R:BASE 7.5 (7.5.24.30214 or higher), R:BASE V-8 Turbo (8.0.12.30214 or higher)

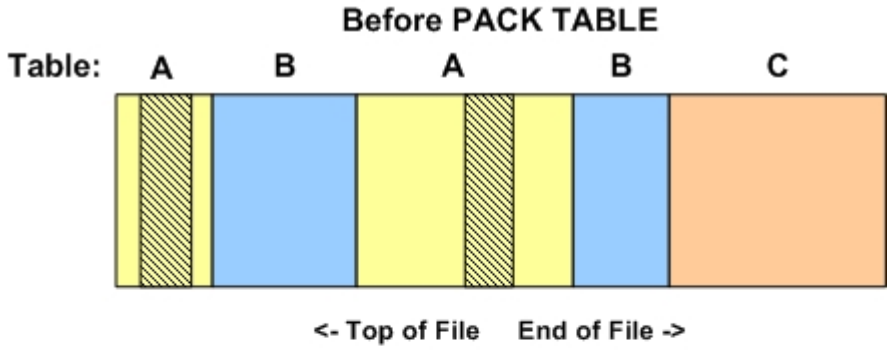
Background:

Tables in a database that have a significant number of DELETEs over time will contain space representing the deleted rows that normally cannot be reclaimed without doing a PACK or RELOAD. This presents a problem in a multi-user environment: to reclaim the space requires that all users except one disconnect the database. In a live, 24x7 environment it can be difficult to
obtain the downtime required for this database maintenance.

PACK TABLE tablename

The PACK TABLE tablename command supports packing a single table when MULTI is set ON. This command will remove the dead space from the table as shown in Figure 1. In the diagram, three tables exist – A, B and

C. Table A is shown in yellow, with the cross-hatch area representing deleted rows. Table B is blue, table C is orange.



After the PACK TABLE command is executed, the unused space appears at the end of the blocks originally allocated to table A. Now that the space has been freed up, we wish to use that space. To accomplish this we need to SET RECYCLE ON.

RECYCLE ON tells R:BASE to attempt to reuse free space. Each time an INSERT requires that a new block be allocated to the table, a routine is called that looks for free blocks in the file rather than appending to the file. To be used:

• The block must not be in use by any other table
• The block must exist farther down in the file than the current last block for the table

In Figure 1, after the PACK TABLE has been executed the unused space shown can be used by table A since it falls after the last block for the table. It cannot be used, however, for either table B or table C. In the case of table B, blocks are already allocated farther down in the file than the free space. In the case of table C, the free space occurs before the first block of the table. If a suitable free block is found it is allocated to the table. If not, a new block is allocated at the end of file 2.

Considerations:

SET RECYCLE ON (the default is OFF) must be done for all users to be effective. If only some users have RECYCLE ON, others may cause new blocks to be allocated at the end of file 2, defeating the use of the setting.

SET RECYCLE is best used in the configuration file:

- RBENGINE75.CFG
- RBENGINE76.CFG
- RBENGINE8.CFG
- OTERRO35.CFG

- OTERRO8.CFG

When RECYCLE is ON there will be a slight overhead each time a new block is required. This only occurs when new rows are being added to the table.

RECYCLE will only have an impact when used in conjunction with PACK TABLE since dead space must be freed up before it can be reused.

1.4.1.2 RECYCLE Setting

RECYCLE is an operating condition to recycle file space on the RX2 data file. When adding new rows that requires a new block, a new routine is called which searches for a suitable "unused" block rather than always adding a new block to the end of the file.

Syntax: SET RECYCLE ON/OFF

Default: OFF

The criteria for such a block are:

- No other table uses it
- The block is further down the file than the current last block of the table

PROS
If a suitable block is found, the block will be allocated to the table requiring the additional space and File 2 will not grow as a result of this allocation. The main benefit of using RECYCLE is that the growth of File 2 will be minimized. Over time this can add up to significant savings on disk space and backup media.

CONS
Since a new routine is being called to search for a suitable block, there will be a slight performance penalty. The penalty will only be incurred when an INSERT requires a new block.

Considerations
For RECYCLE to be effective, all users should have the setting ON. Do this in the RBENGINE76.CFG configuration file. RECYCLE will only have an impact when used in conjunction with PACK TABLE. Dead space in File 2 must first be freed up before it can be reused. RECYCLE will not be of benefit if your database does not end up with lots of deleted rows over time, providing the opportunity to recover dead space.

Conclusions
Periodic use of PACK TABLE tablename in conjunction with RECYCLE ON will retard File 2 growth and reduce fragmentation. Use of PACK INDEX FOR tablename will keep the index statistics fresh and query optimization results maximized. The need for planned downtime will be reduced.

1.4.2 Using RELOAD

If you have added, changed, or deleted many rows in your database, use the Reload option. Reload copies the open database table by table to a new database name that you specify, preserving the original database. As it duplicates the database, Pack To places the rows for each table in a single area of your database file .RX2, thereby eliminating unusable disk space. Reload does not have options that allow you to compress indexes or the database structure only.

To use the Reload option, you must have room for a copy of your database on your hard disk or your database must be small enough to fit on one floppy disk. If you do not have enough space on your hard disk, or if the database is too large for one floppy disk, use the Pack In Place option to compress the database instead.

After the process is complete, you will see the new database with the name you specified listed in the Database Explorer. Once you are sure that your new database is working properly, you can use the Database Explorer to erase the old database files and rename the new files to the original database name.

1.4.3 PACK Versus RELOAD

Once you have ensured that your database files are intact, you can compress your database. R:BASE has two options for compression, Pack In Place and Reload, which are compared in the table below.

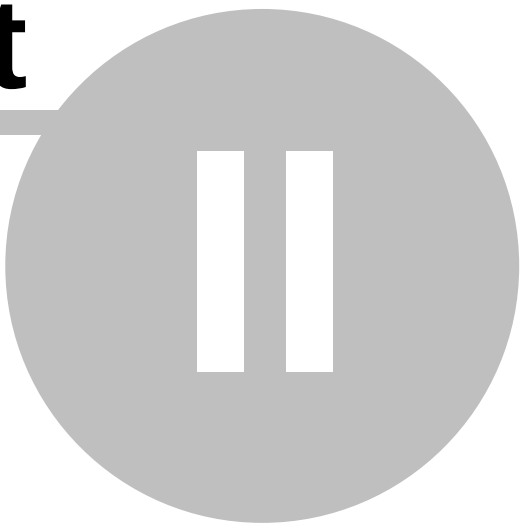
Pack In Place	**Reload**
Eliminates unusable disk space	Eliminates unusable disk space
Does not reorganize rows	Reorganizes rows
Does not change response time	Might improve response time
Operates row by row within the open database	Copies the open database to new files
Does not require extra disk space	Requires that a copy of your database fit on your hard disk or on one floppy disk

1.5 Data Integrity Features

More Data Integrity Features R:BASE include a number of data integrity features in addition to **ON CONNECT** and **ON PACK** that can help you protect your data:

- **SET MIRROR** - When MIRROR is set ON, a copy of the database is automatically maintained in another, specified location. Additions, changes, and deletions are simultaneously written to the working database and to the mirrored database.
- **SET WRITECHK** - You can set WRITECHK to ON to have R:BASE automatically verify each write to disk. This guarantees that data is actually written to the disk and not cached in memory.
- **PACK INDEX** - Use the PACK INDEX indexname command to rebuild just one index. This can save considerable time over having to rebuild all the indexes in the database.

Part



2 Feedback

Suggestions

From time to time, everyone comes up with an idea for something they'd like to add. If you come across an idea that you think might make a nice addition to this document, your input is always welcome.

Please submit your suggestion to our R:BASE Developers' Corner Crew (R:DCC) email address at: rbg8rdcc@rbase.com. Describe what you think might make a nice enhancement.

Unless additional information is needed, you will not receive a direct response.

Index

- A -

- B -

- C -

- D -

- E -

- F -

- I -

- K -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- U -

- W -

Back Cover