RECOVERING A CRASHED SQLBASE DATABASE

SQLBASE CRASH RECOVERY TECHNIQUES

by Cory D. Wiegert

Database crashes may not be as serious as some people believe. If the proper recovery techniques are used, many times the database can be recovered without any data corruption or loss.

Introduction

Depending on the state of the database which crashed, a different recovery technique will be needed to restore the database to a consistent state. The author will attempt to cover most known inconsistent states and the recommended techniques to recover either the entire database, or as much of the uncorrupted data as possible. This paper is not intended to state the only way Centura Technical Support will tell you how to recover the data, but rather the Centura Technical Support standards for database recovery. All methods described here are for non-partitioned databases only and are not intended for recovery of a partitioned database. Recovery should only be performed or authorized by the database administrator (DBA). There are certain presumptions made that untrained or inexperienced users may not fully understand.

Contents

Diagnosing the problem

When a database crashes, the first thing to determine is the cause of the crash. By looking at the FAIL.SQL the DBA may be able to determine the type of recovery needed and which of the scenarios listed below is best suited for the instance of the crash. Providing the operating system did not crash, an error will be written to the FAIL.SQL, found in the root of the SYS:\ volume on Netware and in the directory where the server executable resides on Windows NT, when the Database server crashes. The FAIL.SQL contains all the errors posted by the server, and the processes running on the server at the time of the error. What is important to the DBA is the error code and description. The process information, although helpful to engineering and someone who knows the internals of the database, is not relevant to the crash recovery process. Because the FAIL.SQL is not deleted each time the server is started, there may be a number of errors listed in the file which are not pertinent to the situation at hand. Each time the server is restarted, the file should either be deleted or the file should be edited, noting the server was restarted. If the file is deleted, a log containing the date and reason for the crash should be kept for historical reasons. The FAIL.SQL can be a helpful tool in noticing trends leading up to a crash, therefore some record of the file should be kept.

Normally the last error listed in the file will be the one which caused the crash. Previous errors may have led to the final one and may be helpful in determining why the last listed error occurred. Once the cause of the crash is determined, recovery can begin based on one of the six scenarios listed.

Load Procedure

Loading a database is the same regardless of the crash scenario. Typically the DBA will want to do an unload and load soon after a database crashes. There may be many pages marked as unused, or the crash may have happened due to some corrupt data, which should be removed from the database. After the unload, the load process can be optimized to minimize down time. The following steps should be used to optimize the load and are what will be referred to later in the paper as the Load Procedure. All commands should be issued by a SYSADM in a SqlTalk session; actual SQL is in bold.

1. **set server < servername>/<password>;** servername is the name of the database server, password is the optional password set for the server
2. **create database < database name>;** database name is the name of the new database
3. **connect < database name>;**
4. **set recovery off;** There is no point creating log files during a load process.
5. **set bulk on;** Once the outmessage buffer fills up, the transactions will be applied. This set command helps reduce network traffic.
6. **Lock database;** This is only valid if using version 6.0 or higher. Locking the database will speed the load porcess.
7. **set outmessage 8000;** Setting the outmessage to 8000 pages. The 8000 is optional and depends on the amount of RAM available. The maximum number of pages is 32000. Description on p. 2-97 of the *SqlTalk Command Reference*
8. If the database is version 6.X or higher use **load sql <volume:\path\file> on server;** On server implies the unload file resides on the server and will be read from there. If using a Novell engine, the volume and path must be from the server's perspective, not the client machine. I.e. db:\unload not j:\unload. The DBA may wish to add the **log <volume:\path\filename>** to the end of the load statement to monitor what is happening in the load process. This eliminates network traffic and frees the client machine to do other tasks while the load is taking place. If the database is version 5.X use **load sql < drive:\path\file>;** See p. 3-111 of the *SqlBase SQL Language Reference* for a description of the Load command.
9. After a message comes back that the load is complete, issue a **commit;** to complete the load process.
10. **Set bulk off;** Turns the bulk transaction process off.

11. **Set Recovery on;** Starts transaction logging
12. **Set outmessage 2000;** Sets the outmessage buffer back to the default

Recovery Process

The following five steps are the common procedure which will be called the Recovery Process, variations will be explained wherever necessary.

1. Restart the database server
2. Be sure the database in question reinstalls itself. If it does not, connect to the server, and reinstall the database (**Install database <database name>;**)
3. Connect to the database.
4. The first connection will start the recovery process on the database.
5. Wait while the database rolls back all open transactions and tries to restore itself to a consistent state.
6. If the database reestablishes a consistent state, do a check database and backup the new database.

Rollforward procedure

Only the log files which remain intact in a sequential numeric order can be rolled forward. If there is a break in the numbering of the log files, only the logs up to the break can be rolled forward. Before beginning the process, check in the logdir to find which logs exist. The highest number log file will be of interest if an error occurs in the rollforward process. Using the steps outlined below, a database can be rolled to the current log file. As long as the database checks complete, normal operation can continue, if the database does not check, further recovery is needed. Only one method of roll forward will be described. If more information is necessary, the recovery process can be found on pages 8-10 through 8-16 of the *Database Administrator's Guide*.

1. Restore the database from backup
2. **Set server < server name>/<password>;**
3. If necessary, **Install database < database name>;**
4. **Rollforward < database name>;**
5. There may be an error which states a restore must be done first. Check the log number versus the logs which resided in the directory prior to the rollforward. If the log reported in the error is one greater than the log which resided, ignore the error. If on the other hand the log reported in the error is less than the logs which exist in the logdir, there is a corruption in the transaction process and data will be lost. The DBA should note the date of the last valid log rolled into the database, and check for lost data since that date. Due to the break in the log files, no further transactions can be rolled into the database.
6. This statement initiates the rollforward process and sets an internal flag that the rollforward process has begun.
7. **Rollforward <database name> to end;**
8. An error stating that a log file can not be found will probably appear in the output section. Ignore the error as it is only a warning that all transactions have been applied and the log file for continuing the open transaction in the last available log is not available.
9. **Rollforward < database name > end;**
10. The output window should state that rollforward has been completed.
11. Now the database is in a recovered consistent state and a **Lock database** should be issued. The lock database does not allow any user except SYADM to log on.
12. **Check database**. A consistency check should be done
13. Depending on the outcome of the check, either the database can be unlocked using **unlock database**, or further data recovery will need to be done.

14. If the database checks, a backup should be done to insure there is a recoverable, consistent backup if, in the future, a restore is needed.

Typical Crash Scenarios

1. Crash with NO RECOVERY set.
2. Crash due to insufficient disk space
3. Crash with corrupt row pages
4. Crash with corrupt index pages
5. Crash with hardware failures or Virtual I/O (VIO) errors
6. Out of memory at the database server

Crash with No Recovery set

At this point the database which crashed is unrecoverable. By definition, no log files were being created. Restore a valid backup, and all data from the point of the backup forward will be lost.

There may be some hope for data which was not being touched in the transaction which crashed the database. The DBA can use the rollforward procedure described above. The commands should reset the recovery flag, and he should be able to connect to the database after completing the rollforward technique. If the DBA knows which tables were involved in the transaction which caused the database to crash, he can check those tables for consistency. If, on the other hand, the transaction impact is not known, the DBA should check the entire database to be sure there are no inconsistencies which may cause the database to crash again. No users will be able to connect until the recovery global is set. All subsequent connections will have recovery turned on.

It is not recommended to run a database with recovery off. The only time you may want to set recovery off is during a bulk load of a database, such as a load after an unload, or during a very large update when performance is of utmost concern.

Setting recovery off tells the database not to create transaction log files, thus increasing performance because disk writes for the log files are not necessary. However, there will be no record of the open transactions and the state of the database at any given time limiting the database's ability to keep track of what is going on at any one point. A backup should be done just prior to a bulk load, and recovery should be turned off to optimize the performance of the database, and limit the crash recovery process. **Be warned—all data inserted after the backup will be lost if the database crashes with recovery off.**

Crash due to insufficient disk space

The three places a database can run out of space are on the database, log, or temp volumes. Each of the instances will be described separately. The presumptions that the database server is a dedicated server and the crashed database is either the only database or other databases can not be affected are taken for the following descriptions. If other databases can be moved to free disk space, do so as moving the databases off the affected volume is the safest way to recover disk space for the affected database.

Insufficient disk space on the log volume

The best scenario for insufficient disk space is if a database crashes because it ran out of space on the log volume. If the database was in the middle of a long running transaction, more logs will be needed for recovery, both to read from and to write to. Scan the log volume for unnecessary files which do not pertain to the database or any other active database. If there is a development or test database which creates logs on the same volume, deinstall it, to create room for the crashed database. Since there is no way of knowing which log file contains the start of the transaction, all the log files for the crashed database should be kept for crash recovery.

If there is no way to create space on the current log volume and another volume has been added, or there is room on another volume, the sql.ini can be edited to write log files to another directory. Open the sql.ini for the server, find the applicable server section and modify the LOGDIR entry to point to another directory. i.e. LOGDIR=logs:\logfiles;log2:\logifles The example is for Netware volumes, if the NT engine is being used, the log and log2 should be substituted with a drive letter.

At this point, the database server can be restarted and the recovery process can begin.

Insufficient disk space on the temp volume

The second and least serious of this type of crash is if the tempdir runs out of space. The way to find this is the case is if there is enough space on the dbdir and the logdir, but the database crashes for no apparent reason. Cartesian joins or long running queries sometimes need to create temporary files which are stored in the tempdir if it is specified in the Sql.ini, or in the directory specified by the tempdir environment variable or in the database directory if neither of the above is specified. When the database crashes, the temporary file will be left on the disk and is visible until the database engine is restarted. If this situation is encountered, the DBA should go to great lengths to discover and eliminate whatever caused the database to create such a temp file. Once the database server is restarted, the recovery process can begin.

Insufficient space on the database volume

This scenario is the worst of the three. A database can not be used again until there is enough disk space for it. Something like this can happen on bulk loads, very large updates, or if the database has grown to the size of the database volume. If the database crashes during a bulk insert or update, the database will go through a normal recovery process when it is restarted, as long as there is enough space on the log volume. Most likely the database will be deinstalled by the server and the user will not be able to connect to the database. Using the following steps, the database can be brought back to a consistent state and an unload can be done.

1. Connect to the server and reinstall the database. From a client start a SqlTalk session and issue **Set Server < servername>/<password>;** where <servername> is the name and <password> is the optional password of the database server. Type

the command **install database < database name >;** where <database name > is the name of the database which crashed

2. Connect to the database which was just reinstalled. This will initiate the crash recovery process. Since there is no insert or update activity, the database should recover to a consistent state.
3. Once the database has recovered, unload the database to recover the pages of the database marked as unused. Being in a consistent state, the database will unload only the pages marked as used. This is not a necessary step, but should be done when time permits shortly after the database is recovered.
4. If there is no valid backup of the database, do a backup and drop the database.
5. Start the Load Procedure.

Crash due to a Corrupt Row Page

All of the following steps may be done on a current database file or a restore from backup. If the current database file is used, there will be no need to roll the database forward after cleaning the corrupt data. However, if a restore from backup is used, the log files associated with transactions since the backup will need to be rolled forward (rollforward procedure) if the data associated with the transactions is not going to be lost.

Listed below are some of the most common errors associated with corrupt row pages and the meaning of the acronym found in the error code. If the error number appears in the FAIL.SQL, the database should be stopped and recovered as soon as possible.

Database Error	Corresponding SqlBase Action Failed
803 ROW UEP	Row - unexpected end of page
808 ROW UFA	Row - unexpected failure to alloc slot
809 ROW BPT	row page type field is bad
810 ROW BSN	row page has bad slot number
811 ROW FSB	row page has bad free slot field
811 ROW FSB	row page has bad free slot field
812 ROW BSO	row page has a bad slot offset
813 ROW BSL	row page has a bad slot length
814 ROW BRH	row page has bad row header item
815 ROW MRH	row is missing row header
816 ROW URH	row has unexpected row header
817 ROW BIT	row page has bad item type
818 ROW BIL	row page has bad item length
819 ROW BEM	row page has bad end marker
820 ROW TMI	row page slot has too many items
821 ROW TFI	row page slot has too few items
822 ROW IOS	row page items overflow slot
823 ROW SDT	row page space doesn't tally
824 ROW FDT	row page free space doesn't tally
825 ROW BPL	row page link is bad
826 ROW BEN	Row - bad end page number in TDA
827 ROW BEB	Row - bad ebo page number in TDA
828 ROW BRC	Row - bad row count
831 ROW BTP	Row - bad page type in TDA page

832 ROW BRP	Row - bad page type in RCT page
835 ROW MHV	row is missing hash-value column
839 ROW CNL	row page corrupt: could not locate all selected columns
840 ROW CND	row page corrupt: could not delete all columns
841 ROW CNU	row page corrupt: could not update all columns
842 ROW PPN	Row - internal error: primary page not allocated
843 ROW PEB	Row - page after ebo has bitmap
844 ROW BFP	Row - bad ffp number
845 ROW BLP	Row - long data corrupt: bad page type in long page
846 ROW BLS	Row - long data corrupt: bad long data length

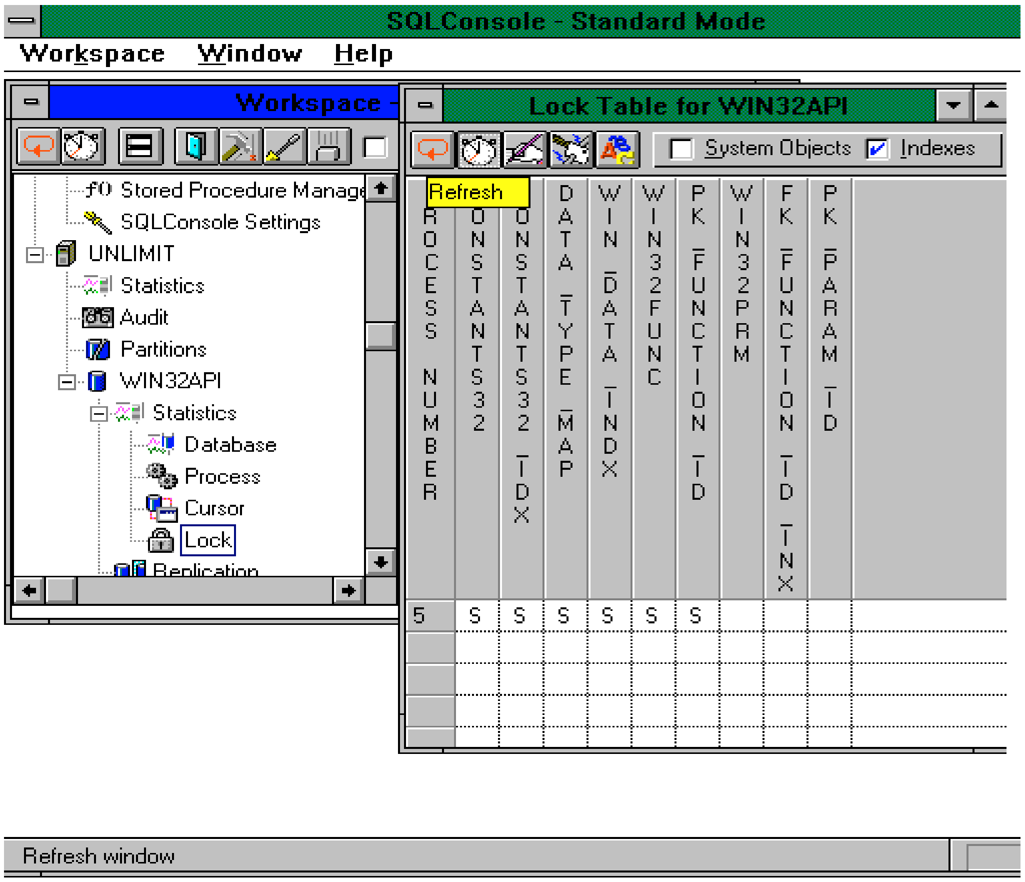
Not all of the above errors will occur while the database is running. Most of the errors listed are Check Database errors, but all represent problems with data pages. When the database is reinstalled, the DBA needs to find out which data is corrupt and drop that data from the database. (see section *Finding the corrupt data*) Since a DBA can not edit the data pages directly, **there will be data loss**— how much is dependent on the amount of corruption.

Finding the corrupt data

With the aid of SqlConsole, the DBA can fairly easily find which table is corrupt. After the database is installed, start SqlConsole and connect to the corrupt database as SYSADM. For purposes of this paper, the different versions of SqlConsole need to be distinguished. Versions prior to 6.1 will be referred to as 6.X and version 6.1 will be referred to as 6.1.

Finding the corruption with SqlConsole is the same no matter which version is used, just the interface to the display is different. If version 6.X is going to be used, click on the Display Locks icon in the tool bar. After the initial display is shown, turn off the automatic refresh by clicking on the alarm clock tool button. The screen should not be automatically refreshed as the display may be cleared when the check database finds the corruption and releases the locks on the tables. If version 6.1 is being used, a connection to the server will need to be established in using the connection manager. Once the connection is made, double click on the database in question, double click on the statistics option, finally double click on the locks section. A diagram of the display can be found on page 7-2 of the 6.1 SqlConsole Guide.

Once the lock screen is displayed, a check database can be started. Preferably the check should be done from another machine so that the machine running SqlConsole will respond quickly to the mouse clicks necessary to keep refreshing the Display Locks screen. After starting the check database, the DBA should monitor the locks displayed by SqlConsole, refreshing every thirty seconds or so. Setting automatic refresh on may result in the lock not being displayed if the screen refreshes after the check database fails. A shared lock will be kept on each table the check database has been through and is working on. If the shared lock is kept, the table is in a consistent state and the check database will continue. When the inconsistent data is found on a data page, the check database will report the error and stop. Before anything is done in the SqlTalk session running the check database, refresh the SqlConsole lock display **The last shared lock displayed will be the corrupt table.** Once the corrupt table is found, a commit can be issued in the SqlTalk session.



ROCESS NUMBER	ONSTANTS32	ONSTANTS32_IDX	DATA_TYPE_MAP	WIN_DATA_INDX	WIN32FUNC	PK_FUNCTION_ID	WIN32PRM	FK_FUNCTION_ID_INX	PK_PARAM_ID
5	S	S	S	S	S	S			

Figure 1. Lock screen from SqlConsole 6.1 showing the indexes and tables during a check database. Notice the shared (S) locks being held as the tables are being checked. At the time of the picture, PK_FUNCTION_ID was being checked. Indexes are noted by the X at the end of the name.

Now that the corrupt table is found, the corrupt data has to be dealt with. The following section will give several ways to detect and deal with corrupt data.

Managing corrupt data

Depending on the size of the database and the time constraints the DBA is under, the corrupt table may be dealt with in one of two ways. The first way described below is the 100% guaranteed way to recover the offending table, the second way has worked in some instances but is not guaranteed to restore the table to a consistent state. In the first case there should be a sequential unique index on the table. If there is, finding the corrupt row is easy. If not, the DBA will need to set a column to an imaginary unique index. Do not try to create a unique index, as a serial read of the data will not be possible, but rather use a column whose data is somewhat sequential as the “indexed” column. After explaining what the index is used for, the imaginary unique index will become more evident.

If there is a unique index, issue the following statement from SqlTalk.

```
Set spool <filename>;
Select * from <table> order by <unique index>;
```

The statement forces the database to do a table scan on the rows while the rows are being read into a result set. Once the corrupt row is hit, the database will crash. Creating a spool file will allow the DBA to see what was the last row returned from the database before the crash. Since the spool file is under control of the client, the rows in the result set will be written to the file regardless of what happens at the server. The DBA should look at the spool file for the last row returned from the database. The row following the last row returned is the start of the corruption in the table, but not necessarily the last corrupt row.

After reinstalling the database, the DBA should issue another select like the following to determine where the corruption stops.

Set spool <new filename>;
select * from <corrupt table> where <unique index> > <corrupt row identifier + 2> order by <unique index>;

Notice the where clause in the statement. Selecting the corrupt row identifier is the last row returned in the spool file. Adding two, will tell the select to jump over the next two rows when reading from the database. Two is used because the identifier + 1 will give the corrupt row, and the row following the corruption is the desired starting place.

If the select brings back the remaining rows in the table, there is only the single corrupt row page. On the other hand, if more corruption's are found, the previous SQL statement will need to be issued until the last row in the table is returned. Once the last row of the table is returned, the good data can be moved to a temporary table and the offending table can be dropped.

Create a new database table using the schema from the old table. If no script for the current database schema exists, create one by doing an unload of the schema.

Unload schema < filename>;

After unloading the schema, the file will need to be altered to just create the temporary table for moving the uncorrupted data. Find the table name in the schema file. Copy the entire portion of the file relating to the creation of that table, and paste it into the input window of a SqlTalk session. Change the name of the table being created to something like TEMP_HOLDER, and issue the create table statement. "Table created " should appear in the output window. After creating the table and committing the transaction the data can be moved to the temporary table with the following command:

insert into < temporary table> select * from < offending table> where < unique index> < <corrupt row>;
insert into < temporary table> select * from < offending table > where < unique index > >= < next good row identifier>;
commit;

The previous commands insert all the good data rows from the corrupt table to the new temporary table. The **next good row identifier** is the first non-corrupt row found after the corrupt row. The last statement may need to be issued several times, changing the where clause each time, if there were a number of corrupts sections found in the table.

After all the data is moved, the old table can be dropped and the temporary table can be renamed to the old table name.

Drop table < offending table name>;
commit;
Alter table < temporary table name> rename table < old table name>;
commit;

If multiple tables were involved in the corruption, please see fast facts number 4316 entitled *Using SQL to Generate SQL*.

Run a check database to insure consistency. If the database passes, backup the database to insure a clean backup.

If time is of the essence and the DBA would like to try a quick and dirty trick, he may try unloading and reloading the offending table. There is no guarantee the data will be restored to a consistent state, but this method has worked in some instances. Again, this call is up to the DBA and the amount of time he has to work with. The steps are the same as the unload/load of a database but the procedure is done on a table.

1. Using the procedure described above, find the corrupt table
2. Unload the table. **Unload SQL <file name> <table name> (on server/client);** the (on server/client) portion is optional. If the database version is 5.X this will not be an option as the on server clause was not introduced until version 6.0.
3. Rename the offending table. Do not delete it in case the unload/load procedure does not work the new table can be dropped and the original table can be renamed and worked with. **Alter table < table name> rename table < archive table name >;**
4. **commit;**
5. Reload the table. **Load SQL < file name> (on server/client);**
6. **commit;**
7. **check table < table name>;**
8. If the table check passes, the data is recovered and the archive table can be dropped. If not, the procedure outlined in the previous section will need to be executed on the current table. Only after the data has been recovered should the old archive table be dropped.

Crash due to corrupt index

Databases which crash due to corrupt indexes may not be recoverable if the corruption is on a system index. Most other indexes can be dropped and recreated to correct any index corruption. Depending on which index is corrupt, a DBA may not even be able to connect to the database. If there is a system index corrupt, the likelihood of being able to connect is very slim, but any other index can be dropped and recreated.

Any of the following error are indicative of a corrupt index:

Database Error	Corresponding SqlBase Action Failed
807 ROW IIC	index is corrupt
829 ROW BRN	bad row number
830 ROW USL	unused slot
833 ROW RIF	bad row index function
9601 IDX BPT	Index - bad page type
9602 IDX BFV	Index - bad flag value
9603 IDX BLV	Index - bad level value
9604 IDX BMK	Index - bad keylength value
9605 IDX BKL	Index - bad key length
9606 IDX BEO	Index - bad entry order
9607 IDX DKU	Index - duplicate keys in unique index
9608 IDX SDT	Index - space does not tally

9609 IDX KIF	Index - key in free space
9610 IDX WNE	Index - wrong number of entries
9611 IDX LNS	Index - levels not same
9612 IDX BOA	Index - bad entry order in adjacent pages
9613 IDX NLC	Index - non-leafed page is chained
9614 IDX IRF	Index - error: index replace failed
9615 IDX IDF	Index - error: index delete failed
9616 IDX RTY	Index - tell caller to retransverse down
9617 IDX BKR	Index - current high key greater than parent high key

If there is a system index corrupt, the database is unrecoverable. A restore and rollforward procedure must be done to retrieve any lost data. If the check database reveals a corrupt data index, drop the index and recreate it.

Finding the corrupt index

In the same way corrupt data can be found, corrupt indexes can be found using SqlConsole and a check database. Notice in Figure 1, the system object check box is not checked. If the system indexes are in question, the indexes and system objects check boxes should be checked before the check index is started. A shared lock will be put on the indexes as well as the tables being checked. The last shared lock held when the check database stops is the index which is corrupt.

An alternative to using SqlConsole, is checking each index individually. This is a time consuming process and should be put in a SQL script which can be executed automatically. The syntax for checking each index is **check index < index name>;**

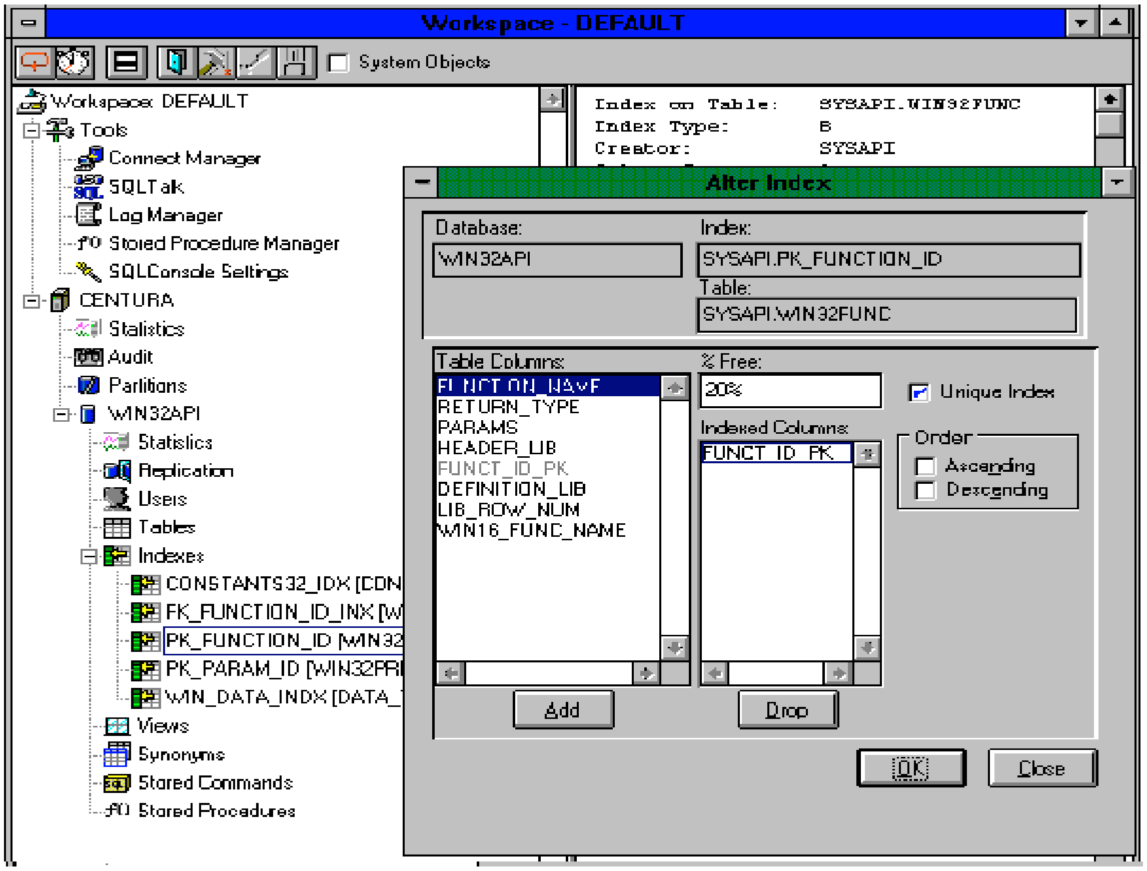
Dealing with corrupt indexes

Before the index can be dropped, the syntax to recreate the index must be known. Using SqlConsole 6.X or above, the index can be observed in a graphical fashion. In SqlConsole 6.1, connect to the database, double click on the index branch, right click on the offending index, click on the alter menu item. A graphical description of the index is shown including the columns and the uniqueness of the index. (See Figure 2 below) If SqlConsole is not an option, unload the schema and look in the unload file for the index in question. The syntax for creating the index will be one of the statements in the unload file.

Now that the syntax for creating the index is known, the index can be dropped and recreated.

1. **Drop index < index name>;**
2. **set refintcheck off;**
3. Paste the create index statement from the schema file, or recreate the index in the SqlConsole create index dialogue.

The picture below displays the alter index dialogue. Notice the tree on the left side of the picture is expanded to the index level. The dialogue shows the definition of the PK_Function_ID index. The indexed column is funct_id_pk, with 20% free and it is a unique index.



```
Index on Table:   SYSAPI.WIN32FUNC
Index Type:       B
Creator:          SYSAPI
```

Database: WIN32API
Index: SYSAPI.PK_FUNCTION_ID
Table: SYSAPI.WIN32FUNC

Table Columns: FUNCTION_NAME, RETURN_TYPE, PARAMS, HEADER_LIB, FUNCT_ID_PK, DEFINITION_LIB, LIB_ROW_NUM, WIN16_FUNC_NAME

% Free: 20%
[✓] Unique Index
Indexed Columns: FUNCT_ID_PK
Order: [ ] Ascending [ ] Descending

Figure 2: Alter index screen from SqlConsole 6.1. Notice the included columns and the Unique Index check box.

In SqlConsole 6.X, the display of the index is the same but the road to get to the display is a bit different. Connect to a server, and highlight the database to be worked with. Select the Server->Database Object Manager… from the menu. A display will appear with the database installed on the server. Select the database, double click, double click on the indexes tree, double click on the index in question. The display of the index will appear just as the small dialogue appears above.

4. Recreate the index. This step may take a couple of minutes as an index must be recreated on all the rows in the table. The DBA should not stop this process! If the client or the server are restarted during this process, the database may end up in a corrupted state.
5. **set refintcheck on;** Turns the referential integrity check back on.
6. **check index < index name>;**

Dealing with a primary key requires as additional step. If the corrupt index was on the primary key, the key will need to be dropped and rebuilt after the index has been built. Page 3-14 of the *SqlBase SQL Language Reference* outlines the steps for dropping and adding primary keys.

1. **alter table < table name> drop PRIMARY KEY ;**
2. **alter table < table name> add primary Key (< column names separated by comma's >);**

If there is no means to graphically display the index, or there is no schema to look at, the create index statement is described on pages 3-45 through 3-52 of the *SqlBase SQL Language Reference.*

Crash with hardware failures or Virtual I/O (VIO) errors

VIO errors are the hardest to diagnose. There is no hard and fast reason for the errors. They may be caused by hardware failures or by the database itself. If you encounter any of the following errors, contact your local support center to help diagnose the problems:

Database Error	Corresponding SqlBase Action Failed
701 VIO OCP	Out of cache pages
702 VIO PNO	page not owned
703 VIO IPN	invalid page number
704 VIO DUN	crashed with recovery off, database unrecoverable
705 VIO NFT	no free transaction available
706 VIO CPO	no CPO'S available
707 VIO TBO	transaction bit map overflow
709 VIO BVC	bad read-only version chain
710 VIO ILP	invalid log page
711 VIO MEM	missing end marker in transaction list
714 VIO ILT	invalid low transaction
717 VIO THR	threshold failure
720 VIO UCO	page use count exceeded maximum
722 VIO MIP	attempted to write partially modified page
724 VIO PMC	page modification conflict
725 VIO ITP	invalid temporary page
729 VIO CPG	corrupt page
730 VIO BPN	invalid page number
732 VIO MZU	modified page has zero use count
733 VIO RZU	released page has zero use count

Most of these errors will be cause by internal workings of SqlBase. Normally an unload/load process will eliminate the error. However if the errors continue to happen, an investigation of the hardware is in order.

The DBA should work closely with the Centura support group if these errors continue to happen. They are spurious errors which should not be manifesting themselves on a regular basis. If the errors are happening on a regular basis, they could be indicative of an internal problem with the database engine itself and should be addressed by the Centura engineering staff as soon as the problem can be diagnosed.

Out of memory at the database server

This error is a graceful shut down of the database engine, and is not indicative of a crash. The simple answer to this problem is increase the amount of RAM on the database server. Ignoring the acquisition of more memory as a resolution, the cause of the problem should be addressed. There is no hard and fast fix for the problem, but limiting the number of cursors and processes that can execute on the database is the first solution. Even though the database has gone down, there is no reason to suspect it is damaged. Cycling the server should bring the database back up and allow work to continue as before the crash. The typical resolution listed from the ERROR.SQL is to decrease the CACHE settings to free more memory. Of course the DBA should look at the cache settings as the cache memory uses available RAM, but if the settings have not been touched, or are not out of the ordinary, there is another problem here.

Typical sortcache and database cache settings are going to depend on the type of processing happening on the server. If there are mainly large queries and sorts happening, the database cache should be bumped up, typically to 12000 or 16000, providing there is enough RAM on the server. Remember, the 12000 is 12Meg, plus the 12M for the operating system and enough RAM left over for normal operation of the server. In the case of a 12000 page cache, the server should have at least 48M of RAM and preferably 64M. If the primary operation is small selects, inserts and updates, there is no reason to have a sortcache or database cache greater than the default 2000 pages. However, if large result sets are being returned and/or sorted a sortcache may need to be bumped up to 8000 pages and the database cache may need to be moved up to 8000 or 12000 pages, realizing there must be enough memory on the server to allow for the adjustments.

As with the VIO errors, the support center should be notified if the out of memory errors keep occurring. There may not only be problems with the configuration of the database server, but there may be an internal problem which needs to be addressed.

Listed below are a number of errors and a brief explanation for each error.

Database Error	Corresponding SqlBase Action Failed
234 DBO CAI	Database operation-cannot allocate input message
243 DBO CIB	Database operation-cannot allocate initialize database buffer
256 DBO OOM	Database operation-out of memory
257 DBO CAL	Database operation- cannot allocate log record buffer
344 EXE CAS	Execute- cannot allocate sort space
407 DBA CAD	Database Access - cannot allocate database space
443 DBA CAC	Database Access - cannot allocate control block
502 WSP CAN	cannot allocate work space
706 VIO CPO	no CPO'S available
707 VIO TBO	transaction bit map overflow
712 VIO CAT	cannot allocate transaction
713 VIO CAF	cannot allocate 'gfp' entry
727 VIO CAL	cannot allocate locks
1503 SRT WTS	Sort - work area is too small
1506 SRT CSS	Sort - cannot allocate sort space
1808 LKM CAL	Locks - cannot allocate locks
1809 LKM LPB	Locks - cannot allocate lock promotion bitmap
1812 LKM IWS	Locks - insufficient work space
1917 SYS CAS	System - cannot allocate structure
2001 LOD IWS	Load - insufficient work space
2106 FIL OOM	out of disk space
2411 RST CRS	cannot allocate result set space

2509 SRV OOM	Server - out of memory
2602 RSI CAM	Remote Server interface - can't allocate message buffer
2703 SCD NES	Scheduler - not enough stack space available
2708 SCD OOM	Scheduler - out of memory
2716 SCD CCT	Scheduler - cannot create thread
2726 SCD CCP	Scheduler - cannot create process
2733 SCD CAN	Scheduler - Out of memory at database computer
2802 RSW NEM	Remote Server Interface (windows) - not enough memory
3004 CFM OOM	configuration management - out of memory
3204 BKP UAB	Backup - unable to allocate buffer
3207 BKP CAC	Backup - cannot allocate backup control block
3303 RES CAC	Restore - cannot allocate restore control block
3305 RES UAB	Restore - unable to allocate restore buffer
3602 DBW CAM	DBWindows - cannot allocate message structure
3710 CFF OOM	Configuration File Management - out of memory
3903 LOG CAL	Logs - cannot allocate log control structure
3904 LOG CAB	Logs - cannot allocate log buffers
3913 LOG OOM	Logs - out of memory
4001 REC CAR	Recovery - cannot allocate rollback log buffer
4009 REC CRL	Recovery - cannot allocate recovery log buffer
4014 REC CRC	Recovery - cannot allocate recovery control structure
4101 BMO OOM	Buffer Manager - out of memory
4301 LDP CAL	Load Procedure - can not allocate load buffer
4407 UNL CUB	Unload - cannot allocate buffer
4507 CSV OOM	Commit Server - out of memory
4603 DTM CAC	Distributed Transaction - cannot allocate control block
4801 SLK CAS	Semaphore Lock - cannot allocate structure
4810 SLK CAB	Semaphore Lock - cannot allocate buffer
4902 HST CAT	Hash Table - cannot allocate hash table
4904 HST CAE	Hash Table - cannot allocate entry
8004 FIO OOM	File I/O - out of memory
10103 DSM OOM	Display Manager - out of memory
10425 SQL CAX	No error defined
10706 PFSCOV	Partitioned File - cannot allocate open file vector
10707 PFS COD	Partitioned File - cannot allocate open file descriptor
10108 PFS CFV	Partitioned File - cannot allocate new file vector
10109 PFS CFD	Partitioned File - cannot allocate new file descriptor
10713 PFS CEE	Partitioned File - can not allocate extent map
10716 PFS CEE	Partitioned File - can not allocate new extent map
10722 PFS CEV	Partitioned File - can not allocate extent vector
10801 SVC CAT	Audits - No error defined
11001 HSH CSS	Hash Table - adding overflow page
11101 APA CNA	Access Path (Optimizer) - cannot allocate work memory
11102 APA OOM	Access Path (Optimizer) out of memory
11203 GFS CAD	Global File System - cannot allocate descriptor
11313 CAC OOM	Catalogue Buffer - out of memory
11406 DFS OOM	out of memory extending file descriptor list
12301 DMN CAD	Daemon - cannot allocate descriptor
12602 SEQ OOM	out of memory

Conclusion

This paper is not designed to give every situation and fix for database crashes. It was truly intended to give the user a place to look for instructions on restoring a crashed database. Any time a database crashes, the support center should be notified, if not for help, at least to keep track of the number of times customers are reporting crashes. If there are internal problems with the database, the fastest way to get the problem fixed is to report the problem in the first place.

About the Author

Cory has been working with SQLBase versions 5.2.0 through 6.1.0. Both in the roles of Assistant DBA, his position before joining Centura, and as Technical Team Leader of the Platinum Support Team for Centura Software Corporation, he has dealt with a number of different database crash scenarios. He can be reached via email at Cory.Wiegert@centurasoft.com or on Compuserve at 102761,3351.