

# Top 10 Cool New Features In SQL Anywhere 10

By Breck Carter



A lot of folks watch the Oscars, not to find out what the best movies are, but to see if their favorites got picked. The same is true of any “Best Of” or “Top 10” list like this: folks like to see if they've picked the winners, and they get upset if something really really good got left out.

So be it. Let's deal with that first; here are some of the SQL Anywhere 10 new features that got left out of this Top 10 list, and why.

### VERSION 10 IS REALLY REALLY BIG

First of all, this is a Top 10 list, not a Top 100 or Top 1000. Version 10 is a huge release, far more extensive than any previous release of SQL Anywhere or any other product in the history of iAnywhere Solutions, and that's counting the Watcom days. So yes, intra-query parallelism is a really big deal, especially when it helps speed up a rogue OLAP query that would otherwise bring an OLTP server to its knees... and no, it didn't make the Top 10.

Why not? Well, intra-query parallelism isn't cool in the same way that fuel injection isn't cool; both are great for performance, but mention them to most people and watch their eyes glaze over. Mention cup holders, or 21-inch spinners, or hot failover (which is on the Top 10 Cool list), and the reaction is completely different.

It might not be fair, but coolness has never been about fairness.

Well, how about those cup holders? If all the new enhancements to Sybase Central aren't cool, what is? What about the Task List, the ER Diagram tab, the deadlocks display? What about table editor undo? Text completion? The answer is, there are two kinds of database designers in the world, those who use GUI tools like Sybase Central to work on the schema, and Old School folks who use text editors to write SQL scripts. You guessed it, I'm a Notepad user... well, actually, I've upgraded to Wordpad in recent years.

But seriously, there have been an enormous number of enhancements made to Sybase Central, enough to convince me to try using it as part of my regular workday. And you will see some Sybase Central screen shots in the Top 10 Cool list.

### THE TROUBLE WITH 10

The real trouble with the number “10” is that it isn't bigger. If it was a bigger number, like “Top 20” or “Top 30”, there might be room for some of these cool features in the Top 10 list:

- Control over how much BLOB data is stored inline versus separate extension pages.
- BLOB sharing, where two rows that contain the same BLOB value only cause one copy to be stored in the database.
- Index sharing: no more warnings about redundant indexes because it's not a problem any more.
- MobilLink “Model Mode”, and why I don't want anyone to find out about this.
- Unicode data and the NCHAR data type.
- The CREATE TABLE column COMPRESSED keyword...
- ... and the CREATE TABLE ENCRYPTED keyword.
- The CREATE EXISTING TABLE ... EXISTS TRUE clause produced by dbunload to avoid reload errors.
- Applying all the transaction logs in a folder, without naming them, plus leaving the engine running when recovery is complete.
- Passing the DEFAULT keyword as an argument in a CALL.
- SELECT \* FROM sa\_transactions() to see which connections have uncommitted transactions running.
- Web service sessions for maintaining state, plus keep-alive and connection pipelining.

## TOP 5 WEIRD NEW FEATURES

Just like with the Oscars, you have to sit through a comic monologue before hearing who won. And just like with the Oscars, this comic monologue doesn't actually have to be funny... you be the judge... here's the list of the Top 5 Weird New Features in SQL Anywhere 10:

### New passwords are case-sensitive, old ones are not.

When you create a new user id in SQL Anywhere 10, or assign a new password to an old user id, that password is case-sensitive even if the database isn't; in other words, the user ids DBA and dba might be the same but the passwords SQL and sql are different. That might qualify for a Top 10 Most Annoying New Features, but weird it isn't; this is weird: If you upgrade an older database, any old passwords brought over to the new database remain case-insensitive. So SQL is the same as sql... for a while... until you change it. The reason for this? Well, let's say you set up a password like sEcReT back in 1996, and now you've just migrated that database from 5.5 to 10. For ten years it hasn't mattered if you typed in SECRET or secret or SeCrEt, and you surely don't remember what the original was... you wouldn't want to be locked out of your new database, would you? So it's a good thing... weird, but good (and the scary part is, it didn't get fixed during testing, the engineers thought about it from the beginning.)

### What language \*is\* that?

The demo database uses "GRUPO" as the owner name for all the sample tables like GRUPO.Contacts,

GRUPO.Customers and so on. That's the letter "O", not a zero. For several weeks I thought it was a zero. Then I thought it sounded Spanish, but that would be "GRUPO". There are 270,000 hits on "grupo" in Google but I still don't have a clue what it means. Or why "DBA" wasn't good enough. So it's not a bad thing, but not a good thing either, just weird.

### You can look at ISYSTAB but you can't see anything.

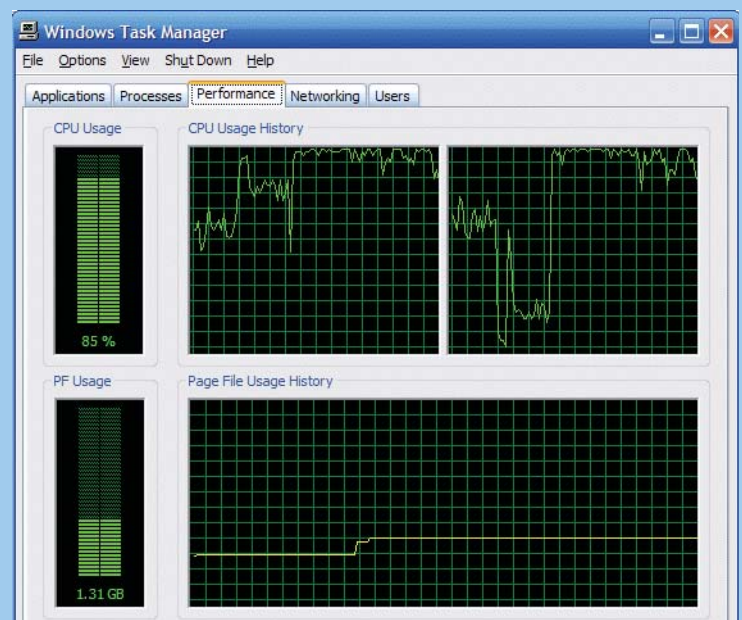
The system catalog tables have been redesigned (that's in the Top 10 Cool list) and renamed from SYS.SYS\* to SYS.ISYS\*. In some cases, the renaming went further: SYS.SYSTABLE is now SYS.ISYSTAB. So far, so good... the weird part is, you cannot do `SELECT * FROM SYS.ISYSTAB`. Doesn't matter if you have DBA authority, doesn't matter if you hide it in a view. You can't do it. That would be very bad, except they've also given us a whole set of views that do work; just drop the "I" from "ISYS\*" as in `SELECT * FROM SYS.SYSTAB`. And how are those views defined? Exactly like the view I tried to create, but wasn't allowed: "create view SYS.SYSTAB as select \* from SYS.ISYSTAB".

### The way you upgrade databases

The first step in upgrading a database to SQL Anywhere 10 is to shut down all database engines that might be running (and I mean all of them), and the second step is to run dbunload... and that's it, there is no third

step, no other steps at all, except to start the new database when dbunload is done. The dbunload step does all the work, and it does it fast: 8 minutes to upgrade a 400MB database on a 3.4Ghz laptop, that's not bad in my opinion. Pretty good in fact... but don't plan on using the computer to do much else while this is going on, as this Task Manager display indicates:

concentrate on improvements rather than legacy support) I prefer to call it a "gotcha". There is a little bit of legacy built in, however: the version 10 dbunload program can, in fact, unload directly from a version 9 database file. To do that, it uses an undocumented program called dbunlspt.exe, or the "Adaptive Server Anywhere Database Unload Support Engine". In the Beta, it

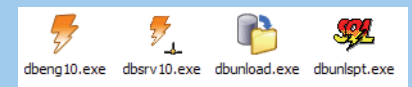


DBUNLOAD Makes The Most Of CPU And RAM

### A Little Bit Of Legacy

Speaking of upgrading databases, one of the new "features" in SQL Anywhere 10 is that you cannot run an old database file with the new database engine. Some would call that a "bug", but because the benefits outweigh the costs (the engineers will be able to

is marked as version 9.0.2.3290 rather than 10.0.0.2089, and it doesn't have one of the cool new icons like the other engines, it still has the older "SQL" with lightning bolt:



## TOP 10 COOL NEW FEATURES

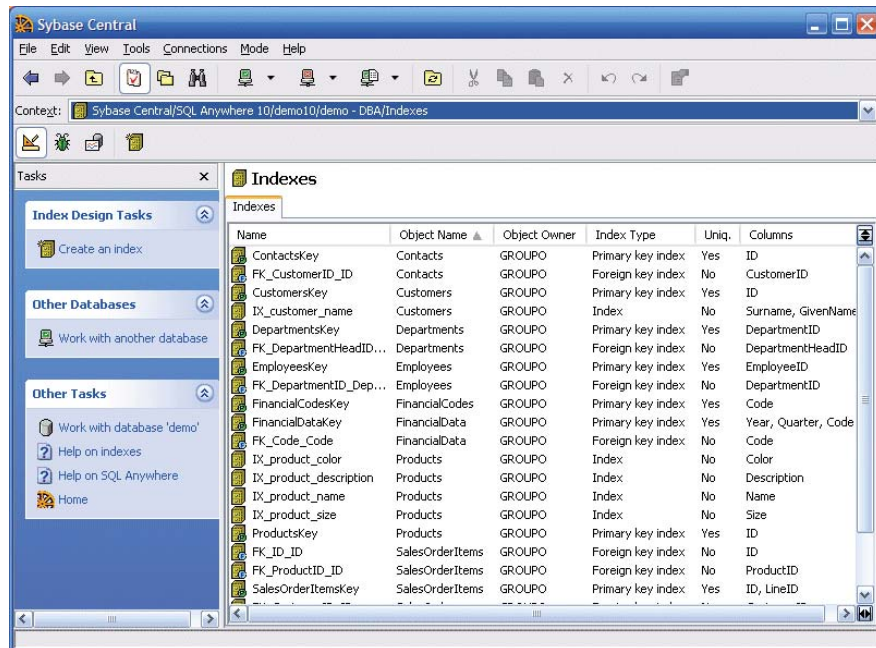
Here are the Top 10 Cool New Features In SQL Anywhere and MobiLink list, in NO PARTICULAR ORDER...

### 1. REDESIGNED SYSTEM CATALOG TABLES

This one's pretty subtle, and just because it's Number 1 in this list doesn't mean it's the coolest feature...this list is randomly ordered because SELECT TOP 10 WHERE cool = 'Y' was hard enough, ORDER BY coolness DESC was just plain impossible (please, no letters about TOP 10 needing an ORDER BY to work properly, software might have to be deterministic, but there's no such requirement for wetware).

Most folks don't care about the system tables, but they might notice one result: Figure 1 shows that primary and for-

FIGURE 1:  
Sybase Central  
Shows All The  
Indexes



oreign key indexes are now included in the Sybase Central Indexes tab. In earlier versions, information about primary and foreign key indexes was hidden away in dark recesses of the system tables, and not displayed by Sybase Central, but in SQL Anywhere 10 the new ISYSIDX table holds information about all the indexes.

### 2. SCRIPT-DRIVEN MOBILINK UPLOAD

There are many reasons why MobiLink is better than SQL Remote for synchronizing databases. One of those reasons is stability in the face of schema changes, at least on the consolidated database. And one of the reasons for MobiLink's improved stability is that the download stream is generated by scripts you write rather than by a synchronization agent that analyzes the consolidated database transaction log. Log-based synchronization is very easy to set up, but it isn't very flexible, and it does have stability issues (like throwing a hissy fit when you lose a log file).

The trouble with all that is, until now MobiLink still generated the upload stream by analyzing the remote database transaction log, just like SQL Remote, so losing that log file is still a huge deal, and so is changing the schema on the remote.

Starting with SQL Anywhere 10, however, you have the option to write stored procedures to generate the upload stream instead of relying on the transaction log. Yes, I have wanted this for a long time. No, I haven't actually tried it yet...did I say log-based synchronization was easy to set up? That means script-based synchronization is...not so much. But I still think it's (going to be) very cool.

### 3. APPROXIMATECPUTIME AND LOCKCOUNT CONNECTION PROPERTIES

Have you ever wanted to know, "Which connection is taking all the CPU time?" I sure have, and now we can get the answer by using the "ApproximateCPUtime" connection property. Here is a query that displays all the connections in decreasing order by CPU time:

```
SELECT Number                               AS connection_number,
       CONNECTION_PROPERTY ( 'Name',   Number ) AS connection_name,
       CONNECTION_PROPERTY ( 'Userid', Number ) AS user_id,
       CAST ( Value AS NUMERIC ( 30, 2 ) )   AS approximate_cpu_time
FROM sa_conn_properties()
WHERE PropName = 'ApproximateCPUtime'
ORDER BY approximate_cpu_time DESC;
```

Here's what the output looks like after "LongRunner" ran a huge SELECT with no WHERE clause:

connection_number	connection_name	user_id	approximate_cpu_time
33	SQL_DBC_5071260	LongRunner	97.20
35	SQL_DBC_3aa4a78	DBA	2.09
12	Sybase Central 1	DBA	1.09
37	SQL_DBC_504b0f0	LotsOfLocks	0.00

How about the answer to, "Which connections have all the row locks?" Here is a query that uses the new "LockCount" connection property to display all the current connections in decreasing order by the number of row locks they are holding:

```
SELECT Number                               AS connection_number,
       CONNECTION_PROPERTY ( 'Name',   Number ) AS connection_name,
       CONNECTION_PROPERTY ( 'Userid', Number ) AS user_id,
       CAST ( Value AS NUMERIC ( 30 ) )       AS lock_count
FROM sa_conn_properties()
WHERE PropName = 'LockCount'
ORDER BY lock_count DESC;
```

Here's what the output looks like after "LotsOfLocks" ran an UPDATE with no WHERE clause:

connection_number	connection_name	user_id	lock_count
37	SQL_DBC_504b0f0	LotsOfLocks	255
35	SQL_DBC_3aa4a78	DBA	0
33	SQL_DBC_5071260	LongRunner	0
12	Sybase Central 1	DBA	0

Now you know who to go after:

```
DROP CONNECTION 33;
DROP CONNECTION 37;
```

#### 4. MOBILINK NAMED SCRIPT PARAMETERS, GLOBAL SCRIPT VERSION, SYNCHRONIZATION ID AND REMOTE ID

These four enhancements are standing in for all the other cool new features in MobiLink that the marketing folks might not talk about but are really neat nonetheless.

Named script parameters let you dispense with all those positional “?” placeholders that you used to be forced to use in all your MobiLink scripts. Now instead of writing this:

```
WHERE last_modified >= ? AND emp_name = ?
```

You can write this:

```
WHERE last_modified >= {ml s.last_table_download} AND emp_name = {ml s.username}
```

Since named parameters are not positional, you can write them in any order you want:

```
WHERE emp_name = {ml s.username} AND last_modified >= {ml s.last_table_download}
```

You can also refer to the same parameter more than once in a script, or leave one out like this (which was hard to do before when it was the first “?” you wanted to leave out):

```
WHERE emp_name = {ml s.username}
```

The global script version allows you to create a new MobiLink script version that contains a new version of only the connection scripts you have changed, without copying all the other connection scripts that haven't changed.

The synchronization id is a number assigned to each synchronization session; it starts at 1 when the MobiLink server starts and goes up to 4,294,967,295. The synchronization id appears in the MobiLink diagnostic log so you can easily tell which synchronization is which.

The remote id gives us something we thought we had all along, but really didn't: a single value that uniquely identifies a remote database. We used to think the “MobiLink user name” did that, and most of the time we were right, but it is possible to have more than one MobiLink user name on one remote database.

**Tip:** To simplify administrative duties when defining a straightforward one-user-per-remote database MobiLink setup, use the same number 1, 2, 3, for all three MobiLink identifiers on each remote database. And there are now three of them:

```
CREATE SYNCHRONIZATION USER "1" ... ; -- the MobiLink "user name"
SET OPTION PUBLIC.GLOBAL_DB_ID = '1'; -- the partition number for DEFAULT GLOBAL AUTOINCREMENT
SET OPTION PUBLIC.ml_remote_id = '1'; -- the MobiLink "remote id"
```

#### 5. MATERIALIZED VIEWS

Before SQL Anywhere 10, a view was just a shorthand version of a SELECT statement, just a kind of “macro” to be plugged into a query when you reference the view. No data was stored in the database for a view, just the SELECT statement itself, and every single time you used a view, the result set was computed from scratch.

Now, however, you can create a “materialized view” where actual data will be stored in the database. This is useful when you have an expensive query that is repeatedly executed; if you create a materialized view for some or all of that query, the optimizer will use the data that has been stored for the materialized view instead of recomputing the entire query just as you have written it.

Application developers do not have to know anything about the materialized views; in particular, they do not have to explicitly SELECT FROM any materialized view. If the optimizer sees that the SQL for a materialized view matches part or all of an application query, the optimizer will automatically use that view if it will improve performance. This is particularly useful when the materialized views contain only a small number of rows when compared with the base tables in the original query.

And if that isn't cool enough, you can also create indexes on materialized views.

## 6. SNAPSHOT ISOLATION

The coolest thing about snapshot isolation is that folks who grew up thinking “database” means “Oracle” will feel a bit more comfortable when their employers decide to stop spending all that money and switch to SQL Anywhere.

Simply put, snapshot isolation is when you select a row that has been updated by someone else but not committed, you see the old (snapshot) version of that row before it was updated. Before SQL Anywhere 10, you had two choices. First, the default was isolation level zero where you would see the updated row, also known as the “dirty row” because you are looking at a change that might be rolled back. The alternative, higher isolation levels that prevent retrieval of dirty rows, meant that your connection would be blocked until the other connection committed or rolled back the update.

Oracle developers might not believe the world is flat, but many of them do believe that snapshot isolation is the only kind to have. The rest of us now have another choice, one that is more efficient in some situations and less so in others.

## 7. LOAD TABLE ROW DELIMITED BY

Before SQL Anywhere 10, LOAD TABLE interpreted '\xod\xoa' as the end-of-row delimiter for the input file. That meant if you wanted to store multi-line blocks of text in a single row, you had to jump through hoops to pre-process the text file before it hit the LOAD TABLE, or manipulate the data after it was loaded one-line-per-row.

Now you can specify a different delimiter for each row, for example, if your input file contains a separate line of '=====' between each block of text, you can use the ROW DELIMITED BY option to store one block per row regardless of the number of embedded line breaks inside each block.

Here is an example of code that loads such a file into a table, and then writes the same data back out to a file that is identical to the input file:

```
CREATE LOCAL TEMPORARY TABLE raw_text (
    block_number          BIGINT NOT NULL DEFAULT AUTOINCREMENT,
    block_text            LONG VARCHAR NOT NULL DEFAULT '',
    PRIMARY KEY ( block_number )
    NOT TRANSACTIONAL;

LOAD TABLE raw_text ( block_text )
FROM 'c:/temp/xxx.txt'
DEFAULTS ON DELIMITED BY '' ESCAPES OFF QUOTES OFF ROW DELIMITED BY
'\x0d\x0a=====\x0d\x0a' STRIP OFF;

UNLOAD SELECT STRING ( block_text, '\x0d\x0a=====' )
FROM raw_text
ORDER BY block_number
TO 'c:/temp/yyy.txt'
DELIMITED BY '' HEXADECIMAL OFF ESCAPES OFF QUOTES OFF;
```

## 8. NEW ARCHITECTURE FOR PROFILING AND TRACING

Just for fun, I threatened to leave this one off the list. Woohoo! Sure got the reaction I expected! Because folks at iAnywhere have put a great deal of effort into this, and if I left it off the list I'd have to go into hiding. So it's a good thing I agree that it should be on the list, and so will you, when you get past the initial shock.

By shock, I mean that you may have to go through “The Five Stages Of Application Profiling and Diagnostic Tracing” (with apologies to Elisabeth Kubler-Ross):

- Denial: I can't get this thing to run at all!
- Anger: If it isn't one thing, it's another!
- Bargaining: I ran it, now how do I get it to show anything?
- Depression: Holy Cow! What am I going to DO with all this data?
- Acceptance: This thing is GREAT!

To make a long story short, the application profiling and diagnostic tracing feature does many things, but one of the coolest is that it can capture enormous amounts of information from a running production database (call it the “source” database) and send it via TCP/IP directly to a separate “tracing” database. The tracing database can be on a separate computer, and when you start analyzing all this data you won't be affecting performance of the source database.

Some of this functionality was delivered earlier, as described in the article [LogExpensiveQueries: A Little Bit Of Jasper In 9.0.2](#). In SQL Anywhere 10, however, it has become much easier to use as well as covering a much broader scope of diagnostic data.

There isn't room in this article to touch on the “broader scope,” but the following step-by-step demonstration will show the “easier to use” part. These steps show how to manually create and use a tracing database via commands and SQL scripts, then use Sybase Central to analyze the results.

**8.1** Start the “source” database from which you want to capture profiling and diagnostic data.

```
"%SQLANY10%\win32\dbeng10.exe" -x tcpip sniffer.db
```

**Tip:** When starting your source database for the purposes of application profiling or diagnostic tracing, be sure to specify -x tcpip. Do the same if you are creating and starting your own separate “tracing” database manually. If you are using the Sybase Central wizard to create a tracing database, do not specify -gd none when starting your source database. But don't try to memorize all of that, just remember this: When you're trying to do some profiling or tracing and nothing works, come back and re-read this tip.

**8.2** Create the “tracing” database as an empty copy of the source database (all schema, no data).

```
"%SQLANY10%\win32\dbunload.exe" -an sniffer_tracing.db -c "ENG=sniffer;DBN=sniffer;UID=dba;PWD=sql" -k -n
```

**8.3** Start the tracing database.

```
"%SQLANY10%\win32\dbeng10.exe" -x tcpip sniffer_tracing.db
```

**8.4** Run these commands on the source database to set the tracing level:

```
CALL sa_set_tracing_level ( 3 );
UPDATE dbo.sa_diagnostic_tracing_level
    SET trace_condition = 'absolute_cost',
        value = 500
WHERE trace_type = 'plans_with_statistics';
COMMIT;
```

**8.5** Run this command on the source database to start a “tracing session”:

```
ATTACH TRACING TO 'ENG=sniffer_tracing;DBN=sniffer_tracing;UID=dba;PWD=sql';
```

**8.6** Run a workload on the source database; e.g., run some time-consuming queries.

**8.7** Run this command on the source database to stop the tracing session and save the data to the tracing database:

```
DETACH TRACING WITH SAVE;
```

**8.9** Look at the tracing data in Sybase Central:

- Start Sybase Central and double-click on “SQL Anywhere 10” in the Plug-ins folder;
- The Mode menu item will appear in the menu bar.
- Click on Mode - Application Profiling.
- The Application Profiling menu item will appear in the menu bar.
- Click on Application Profiling - Open Analysis File or Connect to a Tracing Database.
- Select “In a tracing database” and click on Open.
- Connect to your tracing database. Note that you do not have to connect to the source database from Sybase Central, all the data required for analysis has been copied to the tracing database, just one of the things that makes this feature so cool!
- Choose the Database Tracing Data tab (at the bottom), then the Details tab.

- Select a query you're interested in; Figure 8.1 shows that a query taking over 19 seconds has been selected.
- Click right mouse - View More SQL Statement Details for the Selected Statement
- Choose the Query Information tab (at the bottom); Figure 8.2 shows the graphical plan that was actually used for the long running query on the source database.

FIGURE 8.1:  
A Long-Running  
Query Selected In  
Sybase Central

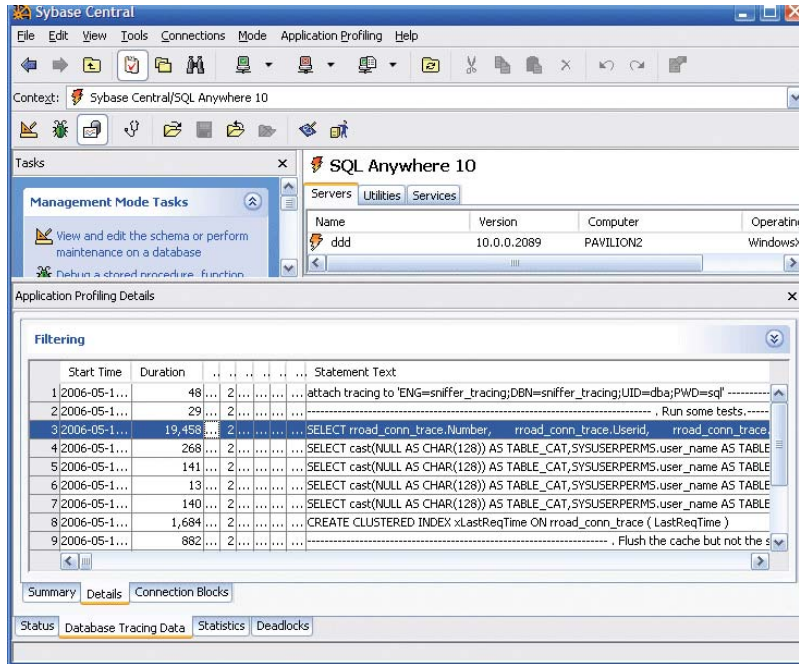
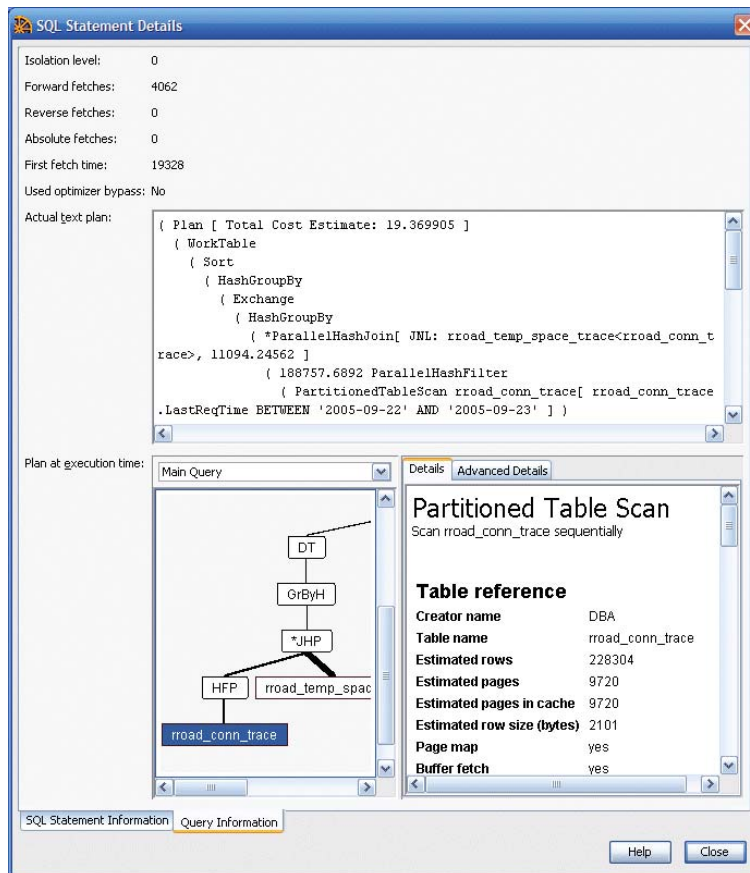


FIGURE 8.2:  
The Graphical Plan  
For The Long-  
Running Query



## 9. ROWID() FUNCTION

Oh, boy, I'm in trouble over this one. Nobody wants to talk about ROWID, it isn't even described in the Help (as of the first Beta). The fact that ROWID is on this Top 10 list is proof that I don't work for iAnywhere...and never will, now, even if I applied for a position as part-time unpaid midnight-shift apprentice understudy.

The ROWID function can be used to answer questions like, "Which rows in table t1 are locked for update?" The sa\_locks procedure returns a new column, row\_identifier, which uniquely identifies the row that's locked, and the ROWID function lets you SELECT that row without knowing the primary key value:

```
SELECT t1.*
FROM DBA.t1,
      sa_locks()
WHERE sa_locks.lock_class = 'Row'
      AND sa_locks.lock_duration = 'Transaction'
      AND sa_locks.table_name = 't1'
      AND sa_locks.lock_type = 'Write'
      AND ROWID ( t1 ) = sa_locks.row_identifier;
```

**Danger, Will Robinson!** Finding locked rows is the **only** reason ROWID was added to SQL Anywhere 10. It is not intended for other purposes; in particular, ROWID values should not be copied and saved in other columns. To do so would be to violate a fundamental principle of relational databases, Rule Number 8 of Codd's 12 Rules for Relational Databases: Application programs must not be affected by changes in the physical data representation. And there is no guarantee that a row's physical row\_identifier will not change over time; e.g., after a database unload/reload.

Nevertheless, finding locked rows **is** really cool.

## 10. HOT FAILOVER (DATABASE MIRRORING)

Somehow, this one got shuffled to the end of the list. I didn't mean to do that, honest...saving the good part until the end is not really in my nature. If this list was called "Top 1 Cool New Features," hot failover would still be on it. Not because it's something you will work with every day, or something that will let you easily create more sophisticated applications, but because it kicks the door open for SQL Anywhere to enter the world of "Enterprise Databases." That door has been ajar for several years, but now it's been slammed back against the wall and has fallen off its hinges.

Hot failover is one of the big features that Marketing talks about, it's Top 1 on their list too. It's also one of only two features in this article that's getting the full code-and-demo treatment.

Three computers were used in testing hot failover for this article. It is possible to get away with two computers, or even one, but that misses the point of "failover": when one computer fails, you switch over to another.

Two of these computers were set up to run identical copies of a database, using the same configuration of the network engine dbsrv10.exe. These two servers are called "partners" or "mirror partners" in the lingo, but only one of them is visible to the outside world as the "primary server" that accepts client connections. The other (non-primary) server is called the "mirror server" in the documentation, but I find that confusing because it sounds like "mirror partner," which applies to both servers. You won't find "mirror server" mentioned again in this article; if I have to give it a name, I'll use "secondary server."

The terms "primary server" and "secondary server" are actually role names. When a failover occurs, the secondary server automatically becomes the primary and it starts accepting connections. All current connections to the old primary server are lost, as are any uncommitted operations, but clients can immediately make new connections and when they do, those connections go to the new primary server.

The client network connection parameters don't have to change depending on which mirror partner is the primary server: there is only one combination of server name and database name visible to the outside world even though there are two servers behind the scenes.

The third computer is used to run an “arbiter” server, which coordinates the other two and decides which one becomes the primary when Bad Things Happen. The arbiter server also uses dbsrv10.exe, but doesn’t run a database or accept client connections.

The following table shows that the arbiter server was run on a computer called PAVILION2, and the two partner servers were named server1 and server2 and they were run on RRHOST and TSUNAMI respectively. As far as client connections were concerned, the ENG=mirtest\_server and DBN=mirtest connection parameters remained constant.

```
dbsrv10
```

server name	Computer Name	Operating System	Client ENG=	Client DBN=
arbiter	PAVILION2	XP Media Center SP2	-	-
server1	RRHOST	2003 Server SP1	mirtest_server	mirtest
server2	TSUNAMI	2000 SP4	mirtest_server	mirtest

The following sections are a step-by-step demonstration of failover using these three computers.

### 10.1 Create the second copy of the database

Perhaps the simplest way to copy an existing database is to use dbbackup.exe to take a full image backup of both the database and transaction log files. Here are the commands used for this demonstration; the first one starts the original database on the RRHOST machine, and the second command sends a full copy to the TSUNAMI machine:

```
"%SQLANY10%\win32\dbsrv10.exe" -c 100M -x tcpip mirtest.db
"%SQLANY10%\win32\dbbackup.exe" -c "ENG=mirtest;DBN=mirtest;UID=dba;PWD=sql" -r -y
\\TSUNAMI\C\mirtest
```

After copying the database, the original server was shut down and the real work started with the next step.

### 10.2 Start the arbiter server

It doesn’t matter what order the three servers are started (the arbiter and two mirror partners), but to me it makes sense to start the arbiter first because it’s the boss. Here is the command line that was used to do that on the PAVILION2 computer, plus the contents of the “configuration file” used to simplify the command line by keeping the options in a separate file:

```
"%SQLANY10%\win32\dbsrv10.exe"
@arbiter_config_file.txt

# arbiter_config_file.txt
-n arbiter
-o C:\mirtest\dbsrv10_log_arbiter.txt
-os 10M
-qn
-x tcpip
-xa auth=dJcNj8nUx3LiJoa8;dbn=mirtest
-xf C:\mirtest\arbiter_sync_state.txt
```

Here is a description of each option and sub-parameter:

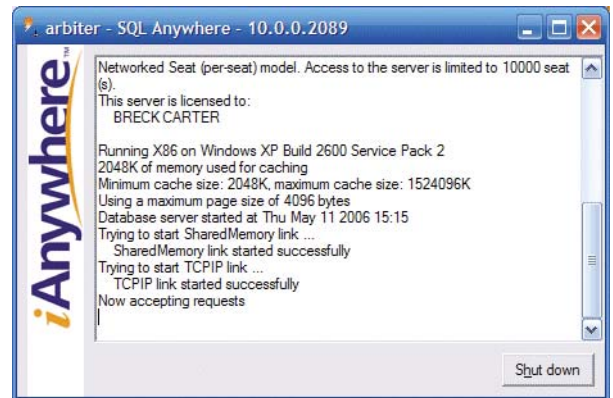


FIGURE 10.2: Arbiter Console on Startup

```

#      comment line
-n      Server name for use by connections from the mirror partner servers; required.
-o      Where to put the server console display text file; suggested.
-os     When to rename and restart the server console display text file; optional.
-qn     Don't minimize the console window on startup; handy for demonstrations.
-x      Protocol; required.
-xa     Authentication list...
auth    Authentication string to be matched against both mirror partners; required.
dbn     The DBN= database name to be used when making client connections; required.
-xf     Where to put the arbiter "state information" text file; required.

```

Some of the entries marked "required" actually have defaults, but everything is clearer when you give them explicit values.

Figure 10.2 shows what the arbiter server console window looked like when it was started before the two mirror partners. It said "Now accepting requests," but that doesn't mean client connections since it isn't running an actual database; it just means it's waiting to talk to the mirror partners.

### 10.3 Start the "server1" partner server

In this example, server1 on RRHOST was the first mirror partner to be started. Here is the command line that was used to do that, plus the contents of the configuration file; note that the options are arranged alphabetically, except that "server options" must appear first, followed by the database file spec C:\mirtest\mirtest.db, and finally the "database options":

```

"%SQLANY10%\win32\dbsrv10.exe" @server1_config_file.txt

# server1_config_file.txt
-c 100M
-hx
-n server1
-o C:\mirtest\dbsrv10_log_server1.txt
-os 10M
-qn
-x tcpip(dobroadcast=no)
-xf C:\mirtest\server1_sync_state.txt
C:\mirtest\mirtest.db
-n mirtest
-sn mirtest_server
-xp
partner={eng=server2;links=tcpip{host=TSUNAMI;timeout=1}};mode=sync;auth=dJCnj8nUx3Lijoa8;
arbiter={eng=arbiter;links=tcpip{host=PAVILION2;timeout=1}}

```

Note: The "-xp ..." line is shown wrapped onto two lines at "arbiter=..." but in real life it has to be all on one line.

Here is a description of each option and sub-parameter:

```

-c      Initial RAM cache size; optional.
-hx     Needed for the Beta, won't be needed in the final version.
-n      Server name for use by connections from the other partner; required.
-o      Where to put the server console display text file; suggested.
-os     When to rename and restart the server console display text file; optional.
-qn     Don't minimize the console window on startup; handy for demonstrations.
-x      Protocol; required, but "(dobroadcast=no)" won't be needed in the final version.
-xf     Where to put the partner "state information" text file; required.

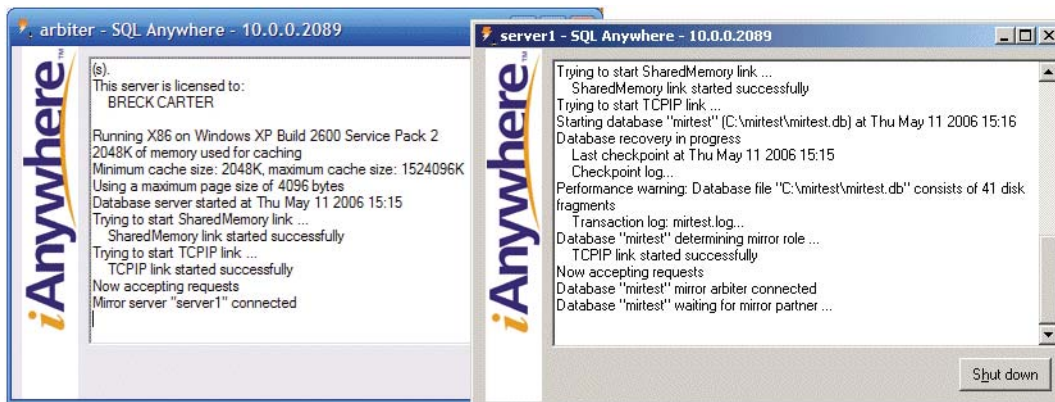
```

```

C:\mirtest\mirtest.db The database file to be started; required.
-n The database name; required.
-sn The alternate server name to be used when making client connections via ENG=; required.
-xp Partner and arbiter list; see sub-parameters. . .
partner Connection string to locate the other mirror partner; required.
mode Mirror synchronization mode, with "sync" being the safest; required.
auth Authentication string to be matched against the arbiter; required.
arbiter Connection string to locate the arbiter server; required.

```

Figure 10.3 shows what the arbiter and server1 console windows look like when the two servers have contacted each other. The arbiter window shows that server1 has connected with it, and the server1 window makes two statements: that it has connected with the arbiter, and that it is waiting to hear from the other partner.



**FIGURE 10.3:**  
**Arbiter and Server1 Consoles After Server1 Started**

Note: Figure 10.3 might look like a screen capture from one computer but it's not. The two console windows were displayed by two different computers, as you can see by the different styles (New XP on the left, Windows Classic on the right). They appear squished together here only through the magic of Paintshop Pro.

#### 10.4 Start the "server2" partner server

In this example, server2 on TSUNAMI is the second mirror partner to be started. Here is the command line that was used to do that, plus the contents of the configuration file:

```

"%SQLANY10%\win32\dbsrv10.exe" @server2_config_file.txt

# server2_config_file.txt
-c 100M
-hx
-n server2
-o C:\mirtest\dbsrv10_log_server2.txt
-os 10M
-qn
-x tcpip(dobroadcast=no)
-xf C:\mirtest\server2_sync_state.txt
C:\mirtest\mirtest.db
-n mirtest
-sn mirtest_server
-xp
partner={eng=server1;links=tcpip{host=RRHOST;timeout=1}};mode=sync;auth=dJCnj8nUx3LiJoA8;
arbiter={eng=arbiter;links=tcpip{host=PAVILION2;timeout=1}}

```

The option and sub-parameter values are all exactly the same as in the previous section, except the names "server1" and "server2" are interchanged and the "RRHOST" is substituted for "TSUNAMI" as the location of the other partner.

Figure 10.4 shows all three console windows when everything has connected and things have settled down. The arbiter window shows that server2 has connected with it, the server1 window shows that it is now officially the “primary server,” and the server2 window indicates that it has successfully synchronized with the primary server.

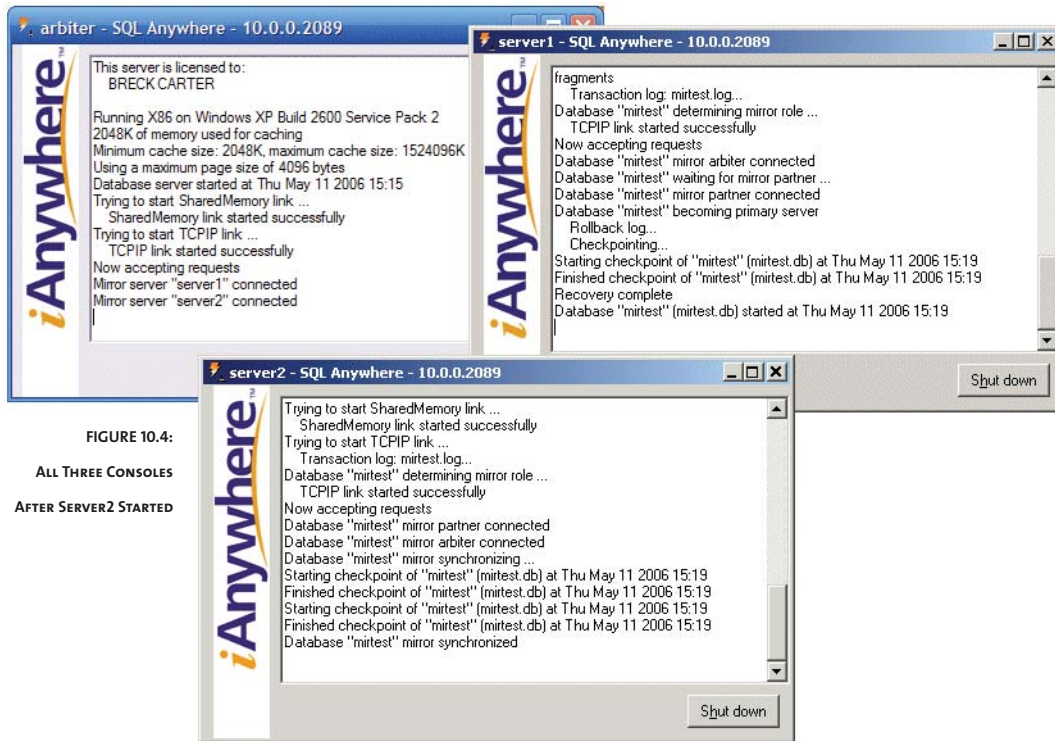


FIGURE 10.4:  
ALL THREE CONSOLES  
AFTER SERVER2 STARTED

### 10.5 Connect to “mirtest\_server” with ISQL

Here’s the ISQL command line used as a client in this article; note that ENG= specifies the alternate server name “mirtest\_server” defined earlier by the -sn option, rather than either of underlying server names “server1” or “server2”:

```
"%SQLANY10%\win32\dbisql.exe" -c
"ENG=mirtest_server;DBN=mirtest;UID=dba;PWD=sql;LINKS=TCPIP"
```

The following SQL commands were used to create a simple table, insert one row, commit the insert, and then run a SELECT to show two things: what the underlying server name is (“server” or “server2”), plus the current contents of the table:

```
CREATE TABLE t ( c VARCHAR ( 100 ) );
INSERT t VALUES ( 'First row' );
COMMIT;
SELECT PROPERTY ( 'ServerName' ), * FROM t;
```

Figure 10.5 shows that server1 is the primary server; remember that “primary server” is the one that actually accepts client connections like this one from ISQL:

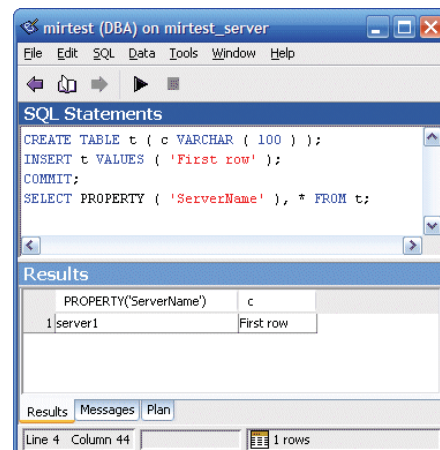


FIGURE 10.5:  
ISQL SHOWS THAT  
SERVER1 IS THE  
PRIMARY

## 10.6 Stop server1

At this point, the question might arise, “Does table t exist on the secondary server, along with its data?” The answer is “yes,” both the CREATE TABLE and committed INSERT were immediately synchronized to the secondary server, even before the SELECT had a chance to run. That’s what “-xp mode=sync” means in the earlier sections: primary server transactions are immediately sent to the secondary server and must be acknowledged by the secondary server before the primary can proceed.

To prove this point, the server1 process was stopped, effectively killed the ISQL session started earlier. Figure 10.6 shows the arbiter and server2 console windows after server1 vanished from the scene; the arbiter window shows that server1 has disconnected, and the server2 window indicates that it has now become the primary server.

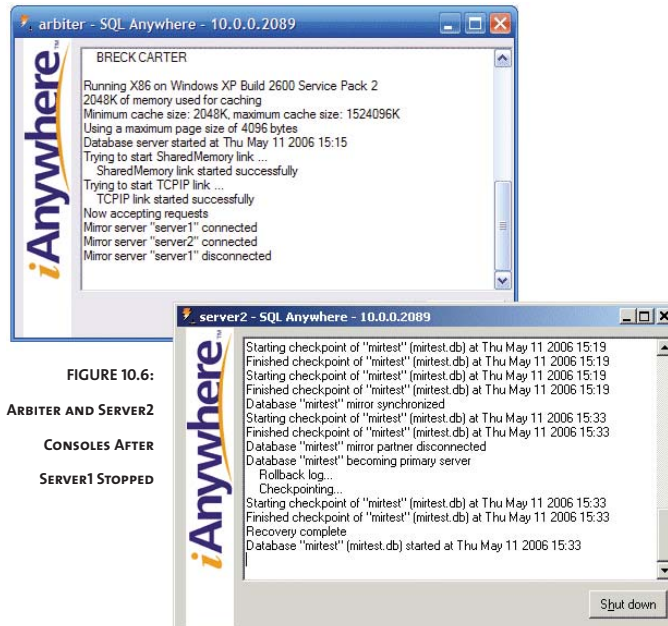


FIGURE 10.6:  
ARBITER AND SERVER2  
CONSOLES AFTER  
SERVER1 STOPPED

## 10.7 Reconnect with ISQL

The ISQL command line from section 10.5 was used again to connect to “mirtest\_server,” and these commands were used to insert more data and display what’s there:

```
INSERT t VALUES ( 'Second row' );  
COMMIT;  
SELECT PROPERTY ( 'ServerName' ), * FROM t;
```

Figure 10.7 shows that everything is fine: the table exists, so does all the data, and it’s all on the new primary server “server2.” That’s it, that’s all there is to it: nobody had to do anything to switch over to the second server except reestablish client connections, otherwise everything happened automatically.

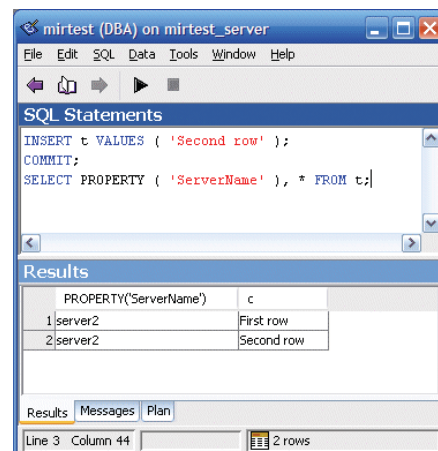


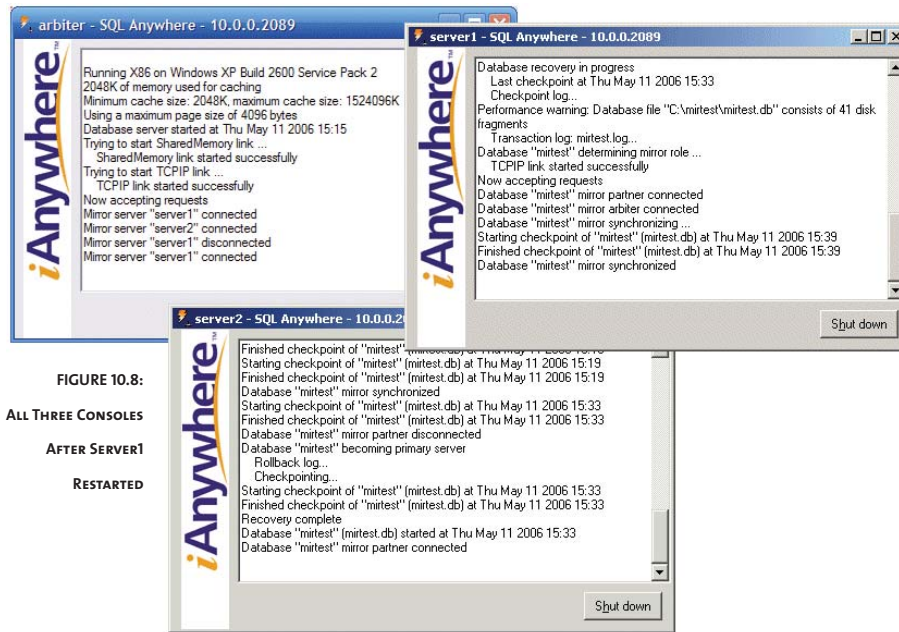
FIGURE 10.7:  
ISQL SHOWS THAT SERVER2 HAS BECOME THE PRIMARY

### 10.8 Restart server1

The next obvious question is, “How do I get back to normal?” With only one partner server running, there is no more failover protection, and you need to get both partners running again.

The answer is as simple as “Start server1 again.” OK, that might sound glib, especially if your primary server has just been wiped out by fire or flood, but once you do have a computer back up and running, and a copy of the current database available (after running dbbackup if necessary), you just run the command line from section 10.3 again.

Figure 10.8 shows all three console windows after server1 on RRHOST was restarted. The arbiter window shows that server1 has connected again, the server1 window shows that it has successfully synchronized with the primary server (which is still server2), and the server2 window shows that it has connected with its partner.



**FIGURE 10.8:**  
ALL THREE CONSOLES  
AFTER SERVER1  
RESTARTED

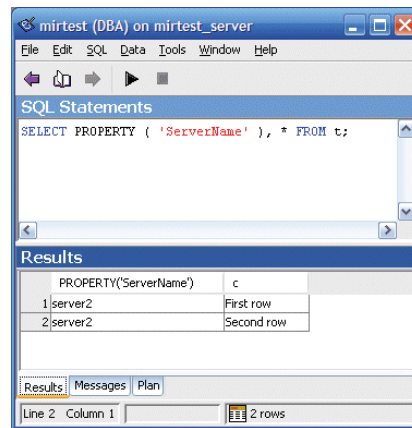
### 10.9 Reconnect with ISQL

At this point, just because server1 has appeared on the scene again doesn't mean it immediately becomes the primary server; that would require all the current client connections to be dropped and reestablished, a complete waste of time.

To prove this, the current ISQL session was manually terminated and then reestablished, and the SELECT shown earlier was run again. Figure 10.9 shows that server2 was indeed still the primary.

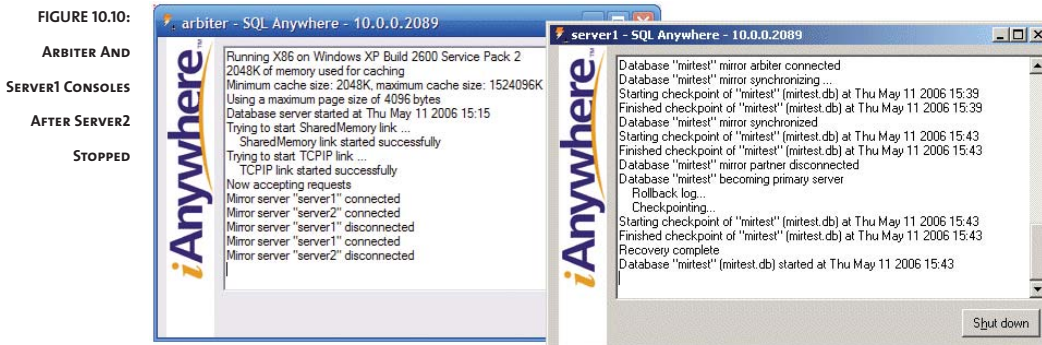
### 10.10 Stop server2

What happens if server2 fails? The same thing as when server1 failed earlier, when it was the primary server: current client connections are lost, but not any of the committed transactions, and the remaining mirror partner becomes the primary server.



**FIGURE 10.9:**  
ISQL SHOWS THAT SERVER2  
IS STILL THE PRIMARY

Figure 10.10 shows the console windows after server2 was stopped: the arbiter console shows that server2 has disconnected, and the server1 console indicates that it has now become the primary server.



### 10.11 Reconnect with ISQL

Figure 10.11 shows that a new ISQL connection goes to the new primary server, which once again is server1. That's what's so cool about hot failover, the data just keeps flowing back and forth as you bounce the servers.

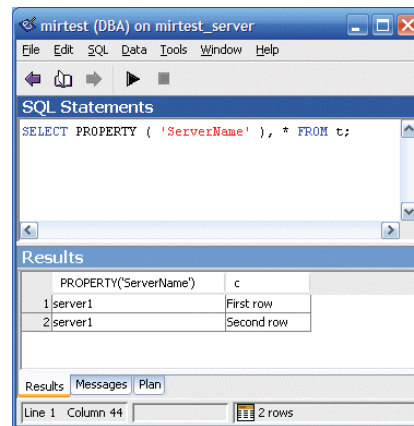
### AND IN CONCLUSION...

There you have it, just **some** of the cool new features in SQL Anywhere 10. The Help file has a long and well-written section "What's New in Version 10.0.0" that I urge you read. I **guarantee** that you will find at least 10 other features that should be in the Top 10 list but aren't even mentioned in this article.

... just like with the Oscars, where everyone has an opinion about movies that weren't even nominated, but were better than the winners.

Breck Carter is principal consultant at RisingRoad Professional Services, providing consulting and support for SQL Anywhere databases and MobilLink synchronization with Oracle, DB2, SQL Server and SQL Anywhere. He is also author of SQL Anywhere Studio 9 Developer's Guide, now available in English, Japanese and Chinese; see here [for more information](#).

Breck can be reached at [breck.carter@risingroad.com](mailto:breck.carter@risingroad.com).



**FIGURE 10.11:**  
ISQL SHOWS THAT SERVER1 HAS AGAIN BECOME THE PRIMARY

iANYWHERE SOLUTIONS, INC.  
WORLDWIDE HEADQUARTERS  
ONE SYBASE DRIVE  
DUBLIN, CA 94568-7902  
U.S.A.

CONTACT\_US@iANYWHERE.COM  
NORTH AMERICA  
T 1-800-801-2069  
1-519-883-6898  
EUROPE, MIDDLE EAST, AFRICA  
+44 1628 597100  
ASIA PACIFIC  
+852 2506 8700  
JAPAN  
+81 3 5210 6380

[www.iAnywhere.com](http://www.iAnywhere.com)

iANYWHERE SOLUTIONS IS A SUBSIDIARY OF SYBASE, INC. COPYRIGHT © 2006 iANYWHERE SOLUTIONS, INC. ALL RIGHTS RESERVED. iANYWHERE, SYBASE, AND THE SYBASE LOGO ARE TRADEMARKS OF SYBASE, INC. OR ITS SUBSIDIARIES. ALL OTHER TRADEMARKS ARE PROPERTIES OF THEIR RESPECTIVE OWNERS. ® INDICATES REGISTRATION IN THE UNITED STATES OF AMERICA. LOXXXX-0606

