Surviving Client/Server: InterBase Or SQL Server?

by Steve Troxell

All Delphi developers have had the opportunity to work first hand with at least one SQL server backend: Local InterBase Server. For many of you, this is your first look at a client/server system. Also, as I've stated many times in the course of this column, functionality of client/server systems varies widely from vendor to vendor. For these reasons, this month I want to take a close look at InterBase (which should be good common ground for everybody) and Microsoft SQL Server (because it's the only other client/server RDBMS I have direct experience with!) in order to illustrate just how different two seemingly similar products can be once you look under the hood.

This is not an exhaustive comparison by any means. We will focus primarily on the features having the greatest impact on day to day development. The basis of our comparison will be InterBase Workgroup Server v4.0 and SQL Server v6.0. At the time of writing, SQL Server v6.5 has just been released and the next InterBase is under development.

ANSI SQL92 Conformance

Despite the fact that SQL is a supposedly "standardized" language, in reality it's about as portable from vendor to vendor as C. Concepts such as indexes, triggers, and stored procedures are not even included in the SQL 92 standard. In actuality, the standard can be implemented at three levels: entry, intermediates and full. Each level builds upon the previous levels and adds functionality. As such, vendors' claims of conformance to any SQL standard should not be taken too seriously.

Neither InterBase nor SQL Server conform to the SQL 92 standard at the full level. They both really only conform at the entry level, with bits and pieces of the higher levels thrown in (along with some vendor-specific enhancements as well). Let's take a look at the major areas where InterBase and SQL Server differ. Features indicated below by (SQL92) are part of the higher levels of the SQL 92 standard, other features are vendor enhancements.

Domains (SQL92) Both products support domains. These are just user-defined data types for column definitions in tables. However, SQL Server allows domains to be used with stored procedure parameters and local variables whereas InterBase restricts them to table column definitions only.

Flow Control Both products provide simple flow control statements such as `IF THEN ELSE` branching, `WHILE` loops and exits from the middle of a stored procedure. InterBase's flow control statements are only valid within the body of a stored procedure or trigger definition. SQL Server's flow control statements can also be used within script files. For example, a script that creates a stored procedure could check to see if the stored procedure already existed and drop it before the `CREATE PROCEDURE` statement is run. This allows the same script file to create new procedures or update existing ones. Without flow control, the `CREATE PROCEDURE` statement would produce an error if the procedure already existed, and the `DROP PROCEDURE` statement would produce an error if the procedure did not exist. With InterBase you would have to switch between the `CREATE PROCEDURE` and `ALTER PROCEDURE` statements, so no single script would work for both creating and updating the same procedure.

Cursors (SQL92) Declared cursors allow you to navigate forwards through a result set with SQL and are available in both InterBase and SQL Server. These can be either read-only or modifiable. SQL Server cursors can also be scrollable (you can move backwards, to first, to last, or jump to a specific row in the cursor) and can be made sensitive or insensitive to changes made by other users to the underlying rows.

Datatypes Both servers support most of the standard SQL 92 datatypes: `CHAR`, `VARCHAR`, `INTEGER`, `DATETIME`, `SMALLINT`, `FLOAT` and `REAL`. SQL Server extends the available datatypes to include `TINYINT` (one byte unsigned integers), `BINARY` and `VARBINARY` (hexadecimal strings), `MONEY` and `SMALLMONEY` (special instances of the numeric datatype with currency related display formatting), `BIT` (binary on-off value), and `TIMESTAMP` (special auto-incrementing value used for optimistic locking schemes).

Arrays InterBase tables can include arrays wherein any column can be designated as an array of up to 16 dimensions of any single data type. Individual array elements, or array slices, can be accessed though SQL statements. However, referencing such structures through third-party tools or even a Delphi program is problematic at best. SQL Server does not support array structures.

BLOB Support (SQL92) SQL Server supports BLOB data fields through separate `TEXT` and `IMAGE` datatypes. However, it allocates these fields in fixed 2048 byte increments. InterBase supports both text and binary BLOB fields through a single datatype called `BLOB` and gives you explicit control over the disk allocation of the field.

InterBase also provides BLOB filters which are DLL functions you write and link into InterBase (except for Novell servers). With

these filters you can perform tasks such as converting a bitmap from the application into a JPEG compressed image in the database and vice-versa. SQL Server does not provide BLOB filters.

Temporary Tables (SQL92) A temporary table can be created such that it is invisible to other users and hence there would be no chance of collision if other users happen to use a temporary table with the same name. Alternatively, temporary tables can be made visible to other users to share data. In either case, temporary tables automatically drop themselves at the end of the process or session.

SQL Server supports temporary tables, although not with the SQL statements specified by the SQL92 standard. InterBase does not support temporary tables at all.

While the `CREATE TABLE` statement can be used to create temporary tables, SQL Server provides a convenient shortcut with the `SELECT` statement. Figure 1 shows how you can add the `INTO` clause to a `SELECT` query and have the server automatically create a temporary table containing the result set of the query. This results in a temporary table named `#TempTable` containing three columns: `Address`, `City` and `State`. Other users can also have temporary tables named `#TempTable` (of completely different structures if desired) without fear of colliding with each other.

Within SQL Server, temporary tables are stored in a special database called `TempDB`. This includes temporary work tables created during the server's normal course of processing queries. SQL Server allows you the option of storing the `TempDB` database within server RAM, greatly improving performance of these activities.

Functions (SQL92) When it comes to providing built in functions to assist the SQL developer, InterBase pales in comparison to SQL Server. InterBase provides a paltry 8 built-in functions, while SQL Server provides more than 75. Some of the more interesting additions to SQL Server's library include the `LOWER`, `LTRIM/RTRIM` and `SUBSTRING` string functions (part of SQL 92) as well as functions to add and subtract datetimes, extract datetime portions (month, day, year, hour etc), a whole slew of math functions, substring searching (similar to `POS` in Delphi) and functions to compute and compare Soundex values.

User-Defined Functions Both servers provide the ability to introduce user-defined functions into the SQL language. Within InterBase (except for Novell servers), you can import a function from a DLL and use it as you would any other system-supplied function in the SQL context. The function can have zero or more input values and can return one output value of any type as the function result. This capability allows users to overcome the spartan built-in function library, but you have to write the functions yourself.

SQL Server provides a similar capability through extended stored procedures. Here a stored procedure call can be made to an external DLL procedure. The interface to SQL is just like any other stored procedure call. While this is not as clean as the true function interface of InterBase, it does allow for multiple output parameters in the stored procedure argument list and for returning multi-row result sets. Some extended stored procedures that are provided by SQL Server allow you to execute any Windows NT command string (returning any output as rows of text), as well as send and read e-mail through the SQL Server mailbox.

Alerts

InterBase triggers or stored procedures can raise user-defined "event alerters" which then notify other applications that have registered their interest in the alert with the InterBase server. This is a simple notification message; no data can be transmitted to or from the listening clients.

SQL Server does not offer built-in support for inter-application communication like this. However, it does have an alert manager which can monitor the NT event log looking for any particular event (which can be fired from an SQL query).

```
SELECT Address, City, State
  INTO #TempTable
  FROM Authors
```

➤ *Figure 1: Temporary tables in SQL Server*

The alert manager can also be tied into the NT Performance Monitor to trigger an alert when a selected performance threshold is crossed. However, the only thing the alert manager can do in response to the event is to send an e-mail, a pager notification, or execute a task such as an SQL script file or NT program.

Triggers

InterBase provides more flexible trigger support than SQL Server. Both servers provide the standard abilities to bind triggers to insert, update, and delete operations on a table. InterBase allows you to have multiple triggers bound to the same event and table; all firing in a specified sequence. One advantage to having separate triggers for distinct tasks on the same operation is that it is easier add and remove individual tasks than to modify the trigger code if it were all in one place.

InterBase also allows triggers to be bound either before or after the data operation occurs. For example, you can have a trigger on a table firing before an update that validates the field contents prior to changing the row in the table. You can also have another trigger on the same table firing after an update that adds a row into a different table, possibly a change history table.

SQL Server only allows one trigger per table per operation (insert, update and delete) and they fire after the operation has been applied to the table. However, although the table is modified before the trigger fires, if an error is raised within the trigger the table change can be rolled back.

InterBase also has a more convenient syntax for examining and modifying columns within a trigger. InterBase provides two context variables called `Old` and `New` that contain the columns of the affected row before and after the

change (Old is not available for inserts, New is not available for deletes). You can manipulate the row contents by directly examining or modifying these context variables. Figure 2 illustrates how this is done in InterBase.

SQL Server provides two virtual tables, Deleted and Inserted, that serve the same function as InterBase's Old and New variables. However, because these are actually tables, you must use regular SQL statements to manipulate them. What this means is that you have to issue SELECT and UPDATE statements to read or change column contents. Figure 3 shows how this is done with SQL Server. Since the rows have already been added to the table, we use an UPDATE statement to modify the affected rows by joining with the Inserted table.

Another difference between the two concerns when a given SQL statement affects more than one row (for example, an UPDATE with a WHERE clause for all the employees within one department). In this case, an InterBase trigger fires once per row affected, an SQL Server trigger fires once per set affected. This makes SQL Server triggers a bit tough to work with: you must allow for the possibility that the Inserted and Deleted tables may contain more than one row. Notice the UPDATE statement in Figure 3: because of the join in the WHERE clause, this will modify all rows that exist in the Inserted table.

```
CREATE TRIGGER OnEmployeeInsert FOR Employee
  BEFORE INSERT
AS
BEGIN
  /* Set the DateEntered field to the current date */
  New.DateEntered = Today;
END
```

➤ *Figure 2: InterBase trigger*

```
CREATE TRIGGER OnEmployeeInsert ON Employee
  FOR INSERT
AS
BEGIN
  /* Set the DateEntered field to the current date */
  UPDATE Employee SET DateEntered = GetDate()
    FROM Inserted
    WHERE Emp_No = Inserted.Emp_No
END
```

➤ *Figure 3: SQL Server trigger*

```
CREATE PROCEDURE Get_Employees_ByDept(Dept char(3))
RETURNS(Emp_No smallint, First_Name varchar(15),
        Last_Name varchar(20), Phone_Ext varchar(4))
AS
BEGIN
  FOR
    SELECT Emp_No, First_Name, Last_Name, Phone_Ext
      FROM Employee
      WHERE Dept_No = :Dept
      INTO :Emp_No, :First_Name, :Last_Name, :Phone_Ext
  DO SUSPEND;
END
```

➤ *Figure 4: InterBase stored procedure*

```
CREATE PROCEDURE Get_Employees_ByDept(@Dept char(3))
AS
BEGIN
  SELECT Emp_No, First_Name, Last_Name, Phone_Ext
    FROM Employee
    WHERE Dept_No = @Dept
END
```

➤ *Figure 5: SQL Server stored procedure*

Stored Procedures

Procedures that do not produce result sets (ones that simply add or change data or return simple output parameters) are handled substantially the same way by both vendors, although there is a slight difference in the calling convention for output parameters.

Stored procedures returning results are significantly different. With InterBase, each column to be returned must be declared as a separate output parameter passed back by the stored procedure. As would be expected, within the body of the procedure each output parameter must be explicitly set with the corresponding column value. Figure 4 shows an example of such a stored procedure.

In contrast, SQL Server allows for "implicit result sets" from stored procedures. In this case, no output parameters are necessary to pass back column values: the stored procedure itself has the result set implicitly bound to it. The stored procedure behaves transparently, there is no difference in the output of the stored procedure than if the underlying SQL queries were sent standalone. Figure 5 shows the equivalent SQL Server stored procedure for the InterBase version shown in Figure 4.

Not only does the InterBase convention negate the use of returning all columns of a table with SELECT * FROM, it involves an additional layer of stored procedure maintenance by requiring much more code in terms of parameters and mechanisms to populate those parameters. However, although SQL 92 does not cover stored procedure, they are planned to be introduced in the next standard currently under development (SQL3). InterBase conforms more to the proposed standard than does SQL Server. We covered this issue in more detail in the March 1996 Issue (No 7).

Other differences include how transaction control is handled through stored procedures. InterBase does not allow transaction control statements (START

```
DECLARE @N INT
DECLARE @Dept CHAR(3)
SELECT @Dept = '623'
SELECT @N = COUNT(*) FROM Employee WHERE Dept_No = @Dept
IF @N = 0
BEGIN
  RAISERROR ('No employees found for department %s', 16, -1, @Dept)
  RETURN  /* RAISERROR doesn't terminate by itself */
END
```

➤ *Figure 6: SQL Server error handling*

```
CREATE EXCEPTION NoEmployees "No employees found for this department.";
...
DECLARE VARIABLE N INTEGER;
DECLARE VARIABLE Dept CHAR(3);
Dept = '623';
SELECT COUNT(*) INTO :N FROM Employee WHERE Dept_No = :Dept
IF (N = 0) THEN
  EXCEPTION NoEmployees;
```

➤ *Figure 7: Basic InterBase error handling*

```
CREATE EXCEPTION DupKey "Don't enter something twice!"
...
BEGIN
  INSERT INTO Table1 VALUES (20, 512);
  WHEN SQLCODE -803 DO
    EXCEPTION DupKey;
END;
```

➤ *Figure 8: InterBase's structured exception handling*

TRANSACTION, COMMIT or ROLLBACK) within stored procedures, the transaction must be handled by the calling application. However, any exception or error raised within the transaction will automatically cause a rollback of the entire transaction. With SQL Server, transaction control statements can appear anywhere (within the calling application or within a stored procedure) and transactions can be nested. This gives you greater flexibility in encapsulating code within stored procedures.

Finally, there are significant differences in error handling capabilities within stored procedures. With SQL Server, the error handling mechanism is the RAISERROR statement, which will send a user-defined message back to the client. The message may be defined in code, or may be defined in a lookup table and referenced by number. RAISERROR also provides the convenience of substitution parameters to allow you to insert text relevant to the specific instance of the error (see Figure 6). If used within a stored procedure or trigger, RAISERROR does not in and of itself end execution of the code, you must explicitly call RETURN. While this is a bit clunky, it does give you a means of placing debugging statements within stored procedures. You can also use RAISERROR to post messages in the NT event log (which would not necessarily be an action you would want to terminate execution of the code).

In InterBase, the text of an exception message is statically defined and bound to a name. You then raise the exception in a block of code (usually within a stored procedure): see Figure 7. Unlike SQL Server, the mere act of raising the exception terminates the block of code and returns to the caller. However, within stored procedures InterBase provides structured exception handling similar to the Try-Except block in Delphi. When an exception is raised (or an SQL statement produces an error), execution of the begin-end block is terminated and any actions performed within it are rolled back. It then looks for a when-do statement to handle the error. Figure 8 shows how you can trap the native SQL error produced when attempting to add a record with a duplicate primary key and return your own message in place of the default server message.

Begin-end blocks can be nested, each having when-do error handlers for specific errors. If an inner block produces an error, InterBase progresses through the levels until it finds a matching when-do handler for that error – just like Delphi's multiple levels of Try-Except blocks.

Integration With Delphi

InterBase works quite well with Delphi, which is not surprising since they both come from the same company. However, there is no direct support in Delphi for array structures and some configurations of NUMERIC and DECIMAL fields translate in Delphi as integers instead of real numbers (truncating the fractional part). Also, you cannot use the TStoredProc component for stored procedures returning more than one row, you must use SELECT * FROM <stored procedure> in a TQuery.

The big problem with SQL Server and SQL Links v2.51 (shipping with Delphi 1) is that it only supports SQL Server v4.21A and won't directly support new features in versions 6.0 or 6.5. For the most part, this can be overcome by encapsulating v6.x-specific functions within stored procedures. Stored procedures called through TStoredProc do not work when connecting though ODBC because of a change to the parameter checking in SQL Server v6.0 (that the BDE violates because it is only designed for SQL Server v4.21A). However, you can still call stored procedures using straight SQL syntax in a TQuery. Although I have not had a chance to work with it at the time of writing, SQL Links v3.01 (shipping with Delphi 2) supports SQL Server v6.0, but messages on the CompuServe fora indicate that it is notoriously slow.

Development Tools

When comparing the development tools provided with these two

products, SQL Server clearly wins hands down. SQL Enterprise Manager is your one-stop resource for nearly all the development and administration functionality needed to manage as many concurrent SQL Servers as you can access from the workstation. This includes multiple servers on the LAN and even remote servers accessible via modem through NT's Remote Access Services.

Enterprise Manager not only performs SQL queries to manipulate data or metadata, it allows you to set up users and security, perform backups and restores, and monitor server connections, activity, and locking for any server it has access to. Tasks can be scheduled to execute automatically at certain intervals. These can include any NT application or SQL statement and e-mail or pager notifications can be sent to any user upon the success or failure of any task.

All of the functions of the GUI front end can also be performed as SQL queries. SQL Server encapsulates a great number (over 180 actually) of routine functions and services within system-supplied stored procedures (basically extending the SQL command set). For example, adding or deleting users or even changing a user's password can all be performed through SQL by calling system-supplied stored procedures. This opens the door for you to write and deploy your own system administrator application to seamlessly integrate these functions in a turnkey database system. InterBase does not supply any SQL statements or procedures to access its administration functions.

The basic query writing features of SQL Server include a fully scrollable and resizable text editor for writing any number of queries at one time (the server can perform multiple queries in one execution, producing multiple result sets if needed).

Separate query windows can be opened for different servers and each window can have multiple query workspaces for the same or different databases. The text editor provides standard text file load/save functions so handling SQL script files is as easy as a click on a load or save button. There is no syntactical difference between SQL code contained in a script file and interactive SQL code.

Within any query window, you can highlight any portion of the code and execute only the highlighted code. This is a great aid to development as you can have a workspace with more than one query and quickly run any of them by simply highlighting and executing the one you want. As another example, you could have a script file containing several related stored procedures, load it into the query editor, change one of them, and select only that one to execute without having to execute the entire script file.

The equivalent capabilities for InterBase can be found in its Windows ISQL and Server Manager programs. Server Manager covers the administration functions: backups and restores, users, security, etc. There is no facility to interactively monitor server activity or schedule automated tasks.

To execute SQL queries with InterBase you use the Windows ISQL program. You can only run one query at a time in a tiny 1 inch by 5 inches non-resizable edit window. There is no provision for multiple simultaneous query windows or workspaces. However, you can call up previously executed queries through a `Previous` button. Script files must be written with an external editor and run through ISQL separately. If you're debugging the script, you must constantly switch back and forth between ISQL and the editor.

Compounding this irritant, the file open dialog used to load the script file isn't even smart enough to remember the last directory you loaded from. You cannot selectively run portions of the script file – it's all or nothing. Because of the need to redefine SQL terminator characters in script files, the syntax of a SQL query within a script file isn't quite the same as an interactive SQL query.

Working with queries or scripts between SQL Server and InterBase is just a night and day difference in ease of use. Here at TurboPower we have had the opportunity to work with both products and consider productivity to be significantly higher with SQL Server's tools than with InterBase's. This is strictly from the developer's point of view, before you even get into the need for such tools at the database administrator's level. Delphi 2's Database Explorer overcomes a lot of the limitations of InterBase's Windows ISQL, but it won't perform the administration tasks and it won't work with script files.

A significant difference in development between these two products is the handling of changes to table structures. InterBase's `ALTER TABLE` statement allows you to add or drop any column from an existing table. If the new columns don't allow nulls, InterBase fills it with a reasonable default value (zeros for numeric columns, empty string for character columns). This is extremely convenient for online changes to tables (and begs for a front end GUI tool to change table structures). It allows structures to be modified without losing existing data, but problems may still arise because the existing data won't have proper values in the newly added columns.

SQL Server's `ALTER TABLE` statement only allows you to add columns, and the columns you add must allow nulls because SQL Server makes no assumptions about what data belongs in the new column. To remove columns you must drop the entire table and recreate it from a script with the columns omitted.

This is further complicated by the fact that in order to drop a table, you must first remove all foreign key references to it. Obviously, you must then restore the foreign key references after recreating the table. The same issue exists with InterBase, except it provides better ways of working around the problem. Because of this, you may not want to physically define foreign key relationships until well into the development process when the table structures have stabilized.

Transportability

In developing a database, there probably comes a time when you need to move it somewhere. For example, you may need to work on the project at home, on a laptop, or deliver it to another site for demo purposes. With Interbase, this is extremely easy: you just copy the .GDB file to whatever machine needs it. With SQL Server, it's another story entirely. The preferred mechanism is the SQL Transfer Manager program, which moves a database (or parts of it) from one database to another on the same or different SQL Servers (through a LAN or modem). Unfortunately, Transfer Manager is a bit buggy. For example, it usually refuses to move triggers. Other than that, it seems gets the job done but it depends on there being a direct link between the two machines.

A backup and restore operation is more reliable, however there are problems with that too when moving between two different SQL Servers. Because of how SQL Server manages user lists, the user access and permissions from your source database won't necessarily coincide with the user list in the target SQL Server. What this means is that you usually have to go through some manual steps or batch processes to keep the user lists in sync.

Locking Schemes

There is a drastic difference in how these servers handle record locking. In SQL Server, the lowest granularity of locking is the page (2Kb). Record locking per se is not possible: if a lock is imposed at least the whole page containing the row is locked (and consequently all other rows on that page). However, true record locking is supposed to available with version 6.5.

Locks occur when reading or modifying data. `SELECT` statements will hold a lock until all the rows selected have been read and returned. This will not prevent other users from reading the same rows, but will prevent them from modifying the rows (see the *TTable Revisited* sidebar). While modifying data, locks are placed that prevent other users from modifying the same pages and from reading them. All this is done to ensure consistency of data and that no reads pick up a partially modified (and therefore possibly inconsistent) result set. The problem is that because the locks are placed at the page level, unrelated data may be blocked as well.

InterBase employs a Multi-Generational Versioning Engine which I understand is an exclusive in the industry. Contention is handled by making separate copies of affected rows in the database and managing them for each user manipulating the same rows. This ensures there is virtually no blocking of processes. There is still an optimistic locking scheme such that if user A changes a row and user B tries to change the same row, user B will get an error message to the effect of *"record has been changed by another user"*. So although there are multiple copies of the rows they are being managed such that data is not lost by multiple simultaneous changes.

Nothing is free, however. All these extra copies of rows eventually become "dead" and must be cleaned up. InterBase has a "garbage collection" routine that periodically goes through and discards these dead rows. It's possible that this garbage collection could put a noticeable drag on your system when it occurs. You can configure the interval at which this happens, but it is based on transaction load not clock time, so you can't force it to happen at a specific time when the system is not in use. You can disable the garbage collection altogether and perform it manually through Server Manager.

Documentation

Both products provide manuals covering administration, installation, SQL programming, API references and tools. SQL Server's manual set is certainly a lot more voluminous (by a factor of 2), which is partly explained by its much larger feature set and administration capabilities.

While any evaluation of documentation quality is going to be subjective at best, I have to say that Microsoft does a much better job in this department. Administration tasks are self-contained within a single task-oriented manual. Tasks are laid out step by step with ample screen shots, and every field, switch, radio button and option is meticulously documented. SQL programming tasks are self-contained in an alphabetical reference manual covering all SQL statements and concepts, including over 180 system-supplied stored procedures. The text is very clear and examples are abundant. The examples usually reference the supplied example database, so you can experiment freely. SQL Server also provides a *Database Developer's Companion* which is an excellent and comprehensive introduction to client/server programming and design issues. Finally, the complete SQL Server documentation set can be installed to hard disk (or read from CD-ROM) for online access.

InterBase's documentation falls a bit short in completeness and organization. The *Language Reference* details the syntax and briefly covers the usage of each SQL statement, but you usually have to refer to the *Data Definition Guide* for a complete explanation (minus the complete syntax). In the *Data Definition Guide* you'll find information about declaring arrays within your tables, but you'll have to turn to the *Programmer's Guide* to find out how to move data between the array and an application. There is lots of example code, but it rarely refers to the sample databases supplied with InterBase. In a few cases the usage defined in the text doesn't quite coincide with the usage in the examples. There seems to be some amount of duplication and dispersal of material between the manuals and you have to read them all to get complete coverage of any one topic. For example, complete information about BLOB filters is spread across three manuals.

Conclusion

There is much more to both of these products than I have covered here. My intention was to show Delphi client/server developers

who may only be familiar with one platform just how diverse the client/server world is. Both products have their strengths and weaknesses and which one is right for you will depend on what kind of project you're undertaking.

The table at the right summarises the capacities of the two systems. More detailed info can be found on the World Wide Web at:

SQL Server:
http://www.microsoft.com/sql

InterBase:
http://www.borland.com/Product/DB/InterBase/ibasmain.html

Next month I plan to take a look at the new database features in Delphi 2.0 from a client/server perspective.

Steve Troxell is a software engineer with TurboPower Software where he is developing Delphi client/server applications for the casino industry. Steve can be contacted at stevet@tpower.com or on CompuServe at 74071,2207

	InterBase v4.0	SQL Server v6.0
Rows per Table	2 billion	Limited by table size
Columns per Table	16,000	250
Row Size (excl. BLOBs)	64Kb	1962
Size of Table	Limited by resources	1Tb
Number of Databases	Limited by resources	32,000
Number of Tables/DB	64,000	2 Billion
Number of Active Users	Limited by resources	32,000
Memory per User	250Kb	40Kb

➤ *Server capacities*

TTable Revisited

Back in the January issue, we talked about the usage of `TTable` in client/server. Because with `TTable` you don't have control over the queries being used to fetch data from the server and because of the locking scheme used by SQL Server (and most other servers besides InterBase), a particularly nasty problem can arise. Let's say you're using a `TTable` and a `TDBGrid` to provide a browser for a customer table. When you open the `TTable`, an unrestricted `SELECT` statement is sent to the server which selects all columns and all rows in the table. This will place a lock on at least part of the table until the query is closed or flushed (all rows are sent back to the client). This will block other users from updating (but not reading) the locked part of the table. Because SQL Server uses page-level locking, rows that aren't even being directly viewed by the browser could be locked. While part of this problem is due to the architecture of the locking scheme, it is also due to the architecture of `TTable` and browsers in general within a client/server RDBMS.