Borland® InterBase® and Microsoft® SQL Server™: A Technical Comparison

This comparative analysis is designed to help developers select the right database for their particular requirements.

A Borland White Paper

by Bill Todd

September 2003

Borland®

Contents

Introduction

Borland® InterBase® is a powerful, SQL-compliant database that is often considered for embedding in applications and for application-specific uses. Savvy developers and application architects who take the time to examine InterBase closely find that it offers substantial advantages over Microsoft® SQL Server.™ Those advantages include:

- Greater concurrency in a mixed read/write environment
- More flexible trigger support
- Faster crash recovery
- Easier event management
- More deployment options
- Cross-platform support
- Smaller size
- Lower system requirements
- Shorter training time
- Lower training cost
- Lower licensing cost

This white paper discusses those advantages in greater detail.

Today's embedded and distributed database applications create an environment that consists of both the short read and update transactions typical of data entry systems and the long read transactions required for reporting and data analysis. When data analysis requires a consistent view of data at a point in time, the locks required by Microsoft SQL Server prevent other users from accessing the data being analyzed.

The InterBase database engine is specifically designed to provide a consistent snapshot of your data at a point in time without blocking reads or updates by other users. With InterBase, readers never block writers and writers only block writers when two transactions try to update the same row at the same time. In addition, the InterBase concurrency model is completely

predictable. With InterBase you never have to worry, as you do with Microsoft SQL Server, about locks being arbitrarily escalated and denying access to entire tables.

InterBase has all the key attributes that are critical for embedded applications and applications that are deployed at remote sites. It is easier for your development team to learn and use, easier to deploy, and requires no maintenance in production. InterBase is one sixth of the minimum installation size for Microsoft SQL Server yet it provides SMP support, cross-platform support, a sophisticated cost-based query optimizer, row level locking, immediate recovery from a server crash, transaction support with isolation levels that guarantee a consistent snapshot of your data at any point in time, a rich SQL dialect, both before and after triggers, full control of trigger firing order, stored procedures, on-line backups, on-line metadata changes, replication and the ability to trigger events in the client application from triggers in the database.

InterBase lowers the cost of your projects three ways.

1. InterBase license costs are lower than Microsoft SQL Server.

2. The time required to train your staff is much lower. It takes 22 days of classroom training to become a Microsoft SQL Server 2000 certified database administrator (DBA). InterBase is so easy to learn that its training course, designed to take complete database novices to the point that they can confidently support InterBase databases, is just five days. Those already familiar with the basics of database administration find that occasional reference to InterBase documentation is all they need.

3. Your developers do not have spend time finding creative ways to code around database limitations such as lock conflicts, lock escalation, the lack of “before” triggers, limited control of trigger firing order and lack of events.

Choosing the right database for your project

It is not difficult to create a checklist of attributes that your database project needs. However, evaluating databases against a simple feature list is not enough. The most critical step in choosing the right database for your application is to examine the behavior of the databases you are considering in the real-world situations that your application will encounter. This paper examines the behavior of InterBase and Microsoft SQL Server in a number of situations that you are likely to face in today's business environment.

Data consistency versus concurrency

Let's say you have an inventory management system that tracks items in many warehouses. Your application must produce an inventory valuation report that is accurate as of the time it is run. Using read committed transaction isolation, the following sequence of events can occur.

1. A read transaction that will total the value of items grouped by warehouse begins.

2. The read transaction scans the records for the Abilene warehouse.

3. Another user starts an update transaction that moves 1,000 gold bars from Abilene to San Antonio.

4. The update transaction commits.

5. The read transaction reads the records for San Antonio.

The read transaction has now counted the 1,000 gold bars twice, once in Abilene and again in San Antonio.

You can easily avoid this problem in Microsoft SQL Server by using serializable transaction isolation for the read transaction. Serializable isolation guarantees that your transaction will not see changes made by other users during the life of the transaction. However, to accomplish this Microsoft SQL Server will either lock the entire table or place a range lock on

an index that will cover all of the values spanned by the WHERE clause[1]. While range locks are less restrictive than table locks, they can still prevent inserts over a much greater range than necessary. Consider the following SELECT statement.

SELECT * FROM CUSTOMER WHERE CITY

>= 'Charleston' AND CITY <= 'Los Alamos'

Assume that the nodes in the City column's index that cover the range of the WHERE clause are Abilene, Detroit, and San Francisco so these nodes are locked. Unfortunately, this means that you cannot insert a row whose value is greater than Abilene but less than Charleston, even though this row would not be included by the WHERE clause, because it is in the range covered by the range lock. You could not insert a row whose value was greater than Los Alamos but less than San Francisco for the same reason[2] .

While this provides the data consistency you need, it hurts concurrency because other users cannot insert or update records in the range of keys covered by range locks or in the tables covered by table locks. When analyzing or summarizing large amounts of data, as in this example, inserts and updates by other users are blocked until the read transaction ends.

With InterBase, this problem does not exist thanks to the InterBase versioning engine. Returning to the inventory valuation example, here is what happens with InterBase.

1. The read transaction begins using snapshot transaction isolation.

2. The read transaction scans the records for the Abilene warehouse.

3. Another user starts an update transaction that moves 1,000 gold bars from Abilene to San Antonio.

---

1 Inside Microsoft SQL Server 2000, Microsoft Press, pp 799
2 Inside Microsoft SQL Server 2000, Microsoft Press, pp 776

4. The update transaction sees that an earlier transaction is active and creates new record versions for the records that are updated.

5. The update transaction commits.

6. The read transaction arrives at the updated record for gold bars in San Antonio. The read transaction sees that the current version of the record was created by a transaction that started after it did, so it scans back through the record versions until it finds a version that was committed at the time the read transaction started and it reads that version.

By reading only record versions that were committed at the instant that the read transaction started InterBase provides a logically consistent view of the data without preventing updates.

Deadlocks

Suppose that user A executes a SELECT statement that reads all of the rows in the Parts table. User B also SELECTs all of the rows in Parts table. Both users must have a stable consistent view of the data they select. User A attempts to update the record for part 100. User B attempts to update the record for part 101.

With Microsoft SQL Server using either repeatable read or serializable transaction isolation, this scenario causes a deadlock even though the two users are updating different rows. How can this happen? User A obtains a shared lock on the table when the records are read. User B also obtains a shared lock on the table when he reads the records. When user A attempts to update part 100 he must obtain an exclusive lock on the row and convert his shared lock on the table to an intent exclusive lock. However, his intent exclusive lock is incompatible with user B's shared lock[3] and cannot be granted. When user B attempts to update part 101 he

3 Inside Microsoft SQL Server 2000, Microsoft Press, pp 799

cannot convert his shared lock on the table to an intent exclusive lock due to user A's shared lock. This is called a conversion deadlock[4].

Lock conversions do not occur in InterBase. Interbase locks individual rows only and only locks rows at the time they are updated. The only type of deadlock that can occur in InterBase is the cyclic deadlock, common to all databases, in which user A updates and locks row 100 then tries to update row 200 which as already been locked and updated by user B. In the meantime, user B tries to update row 100, which has already been locked by user A. In this case one of two things will happen. If the users specified the NoWait option on their transactions, user A will get an immediate error when he tries to lock the row that user B has locked and vice versa. If both users have elected to wait for locked rows then the lock manager will detect the deadlock and select one transaction to roll back.

Lock conflicts

Consider the case when user A updates a row and goes to lunch without committing the transaction. User B executes a SELECT that includes the locked row.

Using Microsoft SQL Server, User B's transaction will wait until the lock held by user A is released[5]. "By default, there is no mandatory time-out period, and no way to test if a resource is locked before locking it, except to attempt to access the data (and potentially get blocked indefinitely).[6]" To prevent an indefinite wait you can set a lock timeout period using the SET LOCK_TIMEOUT command, however, this setting affects all transactions for the connection[7].

4 Inside Microsoft SQL Server 2000, Microsoft Press, pp 776

5 MSDN – Locking Architcture at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/architec/8_ar_sa2_2sit.asp

6 Microsoft SQL Server 2000 Books Online, LOCK_TIMEOUT Setting

7 Microsoft SQL Server 2000 Books Online, SET LOCK_TIMEOUT

InterBase is less restrictive and more flexible. Using snapshot transaction isolation you will always read the latest version of the row that was committed at the time your transaction started. Using read committed transaction isolation InterBase gives you three choices.

1. You can read the latest committed version of the row even though an uncommitted version exists.

2. You can wait until the uncommitted version either commits or rolls back and then read the row.

3. You can receive an immediate exception warning that an uncommitted version of the row exists.

These settings are available at the transaction level so the choice you make for one transaction does not limit your choices for other transactions on the same connection.

Lock escalation

Imagine you are developing an order entry system. The Order Items table will contain several million rows. You must be able to generate a report showing sales by item within customer type and sales territory for a specified period. The report must be based on a consistent view of the data. Running this report for a year will select over a million records in the Order Items table.

"Microsoft® SQL Server™ 2000 automatically escalates row locks and page locks to table locks when a transaction exceeds its escalation threshold. For example, when a transaction requests rows from a table, SQL Server automatically acquires locks on those rows affected and places higher-level intent locks on the pages and table, or index, which contain those rows. When the number of locks held by the transaction exceeds its threshold, SQL Server attempts to change the intent lock on the table to a stronger lock (for example, an intent

exclusive (IX) would change to an exclusive (X) lock). After acquiring the stronger lock, all page and row level locks held by the transaction on the table are released, reducing lock overhead."[8] The result of lock escalation is that the entire table is no longer available to other users. In this example, the shared locks on records will be escalated to a shared lock on the table. This will prevent any other user from updating any row in the table. If you must update a large number of rows in a table, the exclusive locks required for the update may be escalated to the table level preventing other users from reading or updating the table.

The InterBase versioning architecture does not require locks of any kind when reading rows. A consistent view of the data is provided through the record versions without placing locks and without blocking updates. Since InterBase locks only at the row level and only when a row is being updated, the concept of lock escalation does not exist.

Trigger flexibility

When a new customer is added to your database, the system must automatically assign a sales territory and a sales representative. To determine the correct sales territory, the application must search the State table for the user's state and then search the Sales Rep table using the sales territory and the customer's category code. The ideal solution is to add a Before Insert trigger to the Customer table which adds the sales territory and sales rep codes to the record before it is inserted.

Microsoft SQL Server does not provide triggers that fire before the event. Instead, triggers fire after the INSERT, UPDATE or DELETE event to which they are attached.[9] To solve the problem described here, you could use an instead-of trigger. This trigger fires instead of the event to which it is attached. In this example, the trigger would fire but the new customer record would not be inserted into the database. Therefore, the trigger would not only have to determine the sales territory and sales rep, but would also have to execute an INSERT

8 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/acdata/ac_8_con_7a_5ovi.asp, Lock Escalation
9 Inside Microsoft SQL Server 2000, Microsoft Press, pp 657

statement to add the new customer to the database. This makes the trigger more complex and time consuming to write.

Unfortunately, instead-of triggers have other limitations. You can only have one instead-of trigger for each event. You cannot use multiple triggers to modularize your code for easy maintenance. Instead-of triggers cannot be combined with foreign keys that include the cascade option for the event to which the trigger is attached. For example, if a table has a foreign key with the cascade delete option you cannot have an instead-of trigger attached to the delete event.

InterBase provides both before and after triggers for INSERT, UPDATE and DELETE events. There is no limit to the number of triggers you can attach to an event and there are no conflicts between triggers and foreign key constraints.

Attaching multiple triggers to the same event lets you modularize your code for easy testing and maintenance. However, Microsoft SQL server provides limited control of the order in which the triggers execute. You can designate the trigger that fires first and the one that fires last but that is all. Since triggers must be independent of firing order they are more complex to write and less flexible.

InterBase lets you specify the exact order in which triggers will execute. You can easily insert new triggers at any point in the firing order at any time.

Handling errors within triggers is difficult with Microsoft SQL Server. Since triggers fire after the event, the only way to undo the INSERT, UPDATE or DELETE that fired the trigger is to call ROLLBACK. A rollback within a trigger, because of a fatal error or a call to ROLLBACK, aborts the entire batch of SQL commands, not just the changes made by the trigger. [10]

10 Inside Microsoft SQL Server 2000, Microsoft Press, pp 687

Raising an exception in an InterBase before the trigger will abort the INSERT, UPDATE, or DELETE before it takes place. InterBase also allows you to create named savepoints during trigger execution. You can rollback to any savepoint at any time. For example, you can create a savepoint in a before update trigger and rollback to that savepoint in the after update trigger, thereby undoing any changes made after the savepoint was created including the change made by the UPDATE that fired the triggers. Savepoints are available at any point in a transaction, not just in stored procedures and triggers.

Recovery speed

The database server in your regional office in Dallas crashes due to a power failure. When the server is restarted the database must recover to a consistent state automatically and quickly.

Time to recover on restart of Microsoft SQL server depends on the checkpoint frequency established by the DBA because the transaction log must be processed to rollback the active transactions since the last checkpoint.[11] This is typically takes a few minutes but may be longer depending on the tradeoff the DBA has made between frequent checkpoints and reduced performance.

Recovery on restart is immediate with InterBase because no changes to the database are required to rollback active transactions. Instead, InterBase simply changes the status bits for each active transaction to rolled back and leaves the record versions created by the rolled back transaction in the database. The versioning engine will automatically ignore these record versions and they will automatically be removed by the garbage collector as the records are visited during normal database use.

11 Inside Microsoft SQL Server 2000, Microsoft Press, pp 953

Events

Your order entry application must relieve the inventory of each item as it is added to the order. If relieving the inventory causes the quantity on hand to drop below the reorder point, the inventory control program (which is running on another computer) needs to be notified.

Using SQL server you must write a COM object that handles communication with the inventory control program, and then use the sp_OA family of stored procedures to load and call methods of the COM object.

In InterBase, you just add the POST_EVENT command to your trigger. In the inventory control application, you add an event handler component and set its properties to register interest in the event. All of the communications and callback mechanisms are built into InterBase. All your team has to code is the business logic. By the way, don't worry about the inventory system getting erroneous information if the user rolls back the order entry transaction after adding several items. InterBase does not fire the events you post until the transaction commits.

Training time

The time required to train your development team is a significant component of the total cost of ownership and the time it takes to deliver your project. It is also a good measure of product complexity.

Microsoft offers certification for Microsoft SQL Server 2000 database administrators. "The certification is appropriate for individuals who derive physical database designs, develop logical data models, create physical databases, create data services by using Transact-SQL, manage and maintain databases, configure and manage security, monitor and optimize databases, and install and configure SQL Server."[12] "MCDBA on SQL Server 2000

12 http://www.microsoft.com/traincert/mcp/mcdba/mcdba.asp

candidates should have at least one year of experience working with SQL Server."[13] In addition, certification requires 22 days of classroom instruction and four examinations.[14] SQL Server DBA's require extensive training and experience because SQL server is a complex product.

InterBase is so easy to learn and administer that the most comprehensive training course for it, designed to train complete database novices to the point that they can confidently administer InterBase databases, lasts just five days. Those already familiar with database administration often find that they require no training, and that occasional reference to InterBase documentation is all they need.

Cross platform support

Unlike the desktop PC market, the application server market is fragmented. In addition to Windows, Sun has a significant position, particularly in the Web server segment, while Linux ® continues to increase its market share at an amazing rate. It is likely that you may have to support server platforms other than Windows at some time.

Microsoft SQL Server, of course, only runs on Windows.® InterBase, on the other hand, is available on Sun™ Solaris,™ Linux and Windows to give you flexibility now and in the future. All it takes to move your InterBase database to a new platform is a simple backup and restore.

InterBase also provides a variety of connectivity options to ensure that developers can use their favorite development tools and deploy for their particular environment. These include ODBC, JDBC,® ADO.NET drivers and native components for Borland® Delphi™ and Borland® C++Builder. ™

13 http://www.microsoft.com/traincert/mcp/mcdba/mcdba.asp

14 http://www.microsoft.com/traincert/mcp/mcdba/requirements.asp

Cost

Licensing cost becomes a significant part of project cost when you use a database in an application that will be distributed to many sites or that will support many users. The following table compares the cost of InterBase to Microsoft SQL Server Standard Edition. InterBase is a better value in every category.

Users	CPU's	Borland® InterBase®	Microsoft® SQL Server™ 2000 Standard Edition
20	1	$2,300	$4,498
20	2	$3,300	$4,498
50	1	$3,700	$4,999
50	2	$4,700	$9,998
50	4	$6,700	$11,245
100	1	$4,199	$4,999
100	2	$5,199	$9,998
100	4	$7,199	$19,996
200	1	$4,199	$4,999
200	2	$5,199	$9,998
200	4	$7,199	$19,996

Resource requirements

One way to minimize deployment costs is by distributing your application electronically. In addition, you may also want to minimize the cost of hardware upgrades at the deployment sites. How do Microsoft SQL Server and InterBase compare on this front?

Microsoft SQL Server requires "95–270 MB of available hard disk space for the server; 250 MB for a typical installation."[15] SQL Server also requires 64 megabytes of memory on Windows 2000 and 128 megabytes of memory on Windows XP.[16]

A full installation of InterBase requires less than 40 MB. If you do not install the documentation and examples InterBase requires less than 15 MB of disk space. Furthermore, InterBase requires just 32 megabytes of memory on all platforms.

Deployment

Suppose you want to integrate the installation of the database server and its client software into your application's installation to make installation easier.

You must use the Microsoft-supplied installer to install Microsoft SQL Server. "The SQL Server installation program makes multiple entries to the Registry, changes system paths and various system settings, creates program groups and user accounts, and performs basic initial configuration for its operation. The installation program is far from a glorified file-copy program. It would not be realistic for solution providers to create their own programs to install SQL Server as part of their services or products."[17]

InterBase, by contrast, gives you complete flexibility. You can use the Borland installer, use any commercial installer, or write your own installation program.

---

15 http://www.microsoft.com/sql/evaluation/sysreqs/2000/default.asp
16 http://www.microsoft.com/sql/evaluation/sysreqs/2000/default.asp
17 Inside Microsoft SQL Server 2000, Microsoft Press, pp 166-167

MSDE®

You are considering MSDE® as an alternative to Microsoft SQL Server to deploy your application.

MSDE is SQL Server 2000 and has all of the features and limitations of SQL Server 2000. MSDE also has three additional limitations. "It has a managed concurrency workload governor that limits up to five concurrent batch workloads for optimal performance." "As more batch workloads are submitted beyond the five-workload limit, the concurrency governor continues to slow down the system."[18] A batch workload is a connection so if each user requires two connections, MSDE will only support two users. MSDE requires 44 megabytes of disk space, 64 megabytes of memory on Windows 2000 and 128 megabytes of memory on Windows XP. [19]

"MSDE 2000 supports up to 2 gigabytes (GB) per database."[20] Note that this limit is two gigabytes per database, not per table. This means that all of your data, metadata, indexes, temporary tables, stored procedures and triggers must fit within the two gigabyte limit.

"MSDE 2000 includes several command prompt utilities that can be used to administer instances of MSDE 2000."[21] None of the GUI tools that come with Microsoft SQL Server for designing and creating databases, administering the server, testing queries, and tuning the server's performance are provided with MSDE.

"Several Microsoft product licenses convey the right to use and redistribute Microsoft SQL Server 2000 Desktop Engine (MSDE 2000)."[22] The rights you get are complex and depend

18 http://www.microsoft.com/sql/msde/productinfo/features.asp
19 http://www.microsoft.com/sql/evaluation/sysreqs/2000/default.asp
20 http://www.microsoft.com/sql/msde/productinfo/features.asp
21 http://www.microsoft.com/sql/msde/productinfo/features.asp
22 http://www.microsoft.com/sql/msde/howtobuy/default.asp

on which Microsoft product you purchase to obtain MSDE. Some products do not include redistribution rights. Others include redistribution rights if various conditions are met.

Features

This table compares the features of Borland InterBase and Microsoft SQL Server. This list is not exhaustive but instead focuses on features that are important to embedded applications that are deployed to remote sites and applications that consist of a mixture of update and long read transactions.

Feature	Borland® InterBase®	Microsoft® SQL Server™
Transaction support	✔	✔
SQL support	✔	✔
User-defined functions	✔	✔
Cost-based optimizer	✔	✔
Provides a consistent snapshot of data without blocking updates	✔	
Row level locking	✔	✔
No lock escalation	✔	
No conversion deadlocks	✔	
Snapshot transaction isolation	✔	✔

Feature	Borland® InterBase®	Microsoft® SQL Server™
Two phase commit	✔	✔
Multi-user and single user support	✔	✔
SMP support	✔	✔
Stored procedures	✔	✔
Before triggers	✔	
After triggers	✔	✔
Control of trigger firing order	✔	
Fire client events	✔	
Automatic key generation	✔	✔
Supports the null state for all data types	✔	✔
Declarative referential integrity	✔	✔
Online backup	✔	✔
Online metadata changes	✔	✔
Immediate crash recovery	✔	
Replication	✔	✔

Feature	Borland® InterBase®	Microsoft® SQL Server™
Zero maintenance	✔	✔
Memory required	32 MB	64 MB on Win 2000 128 MB on Win XP
Disk footprint	40 MB with manuals and examples; 15 MB without	250 MB average install

Conclusion

With InterBase you get:

- Greater concurrency in a mixed read/write environment
- More flexible trigger support
- Faster crash recovery
- Easier event management
- More deployment options
- Cross-platform support
- Smaller size
- Lower system requirements
- Shorter training time
- Lower training cost
- Lower licensing cost

InterBase gives you the features you need to create a robust application at minimum cost in minimum time.

Bill Todd is president of The Database Group, Inc., a database consulting and development firm based near Phoenix. He has co-authored four database programming books and is the author of over 100 articles. He has presented over two dozen papers at developer conferences in the U.S. and Europe.