

THE INTERBASE & FIREBIRD DEVELOPER MAGAZINE

FYRACLE



**TPC-C based tests:
who is the winner?**

**Bonus: Comprehensive
Repairing Guide, Part 1**

**#/1/4
2006**

www.ibdeveloper.com

THE INTERBASE & FIREBIRD DEVELOPER MAGAZINE



Credits

Alexey Kovyazin,
Chief Editor

Dmitri Kuzmenko,
Editor

Helen Borrie,
Editor

Lev Tashchilin,
Designer

Editorial Office

IBase IBDeveloper, office 5,
1-st Novokuznetsky lane, 10
zip: 115184

Moscow, Russia
Phone: +7495 6869763
Fax: +7495 9531334

Email:
ibdeveloper@ibdeveloper.com

www.ibdeveloper.com

© Copyright 2005-2006 by
IB Developer. All rights reserved.

No part of this publication may be reproduced or transmitted in any form of or by any means, electronic or mechanical, including photocopy or any information storage and retrieval system, without permission.

For promotional reprints, contact reprint coordinator Alexey Kovyazin, **editor@ibdeveloper.com**.

IBDeveloper reserves the right to revise, republish and authorize its readers to use the articles submitted for publication. All brand and product names used in on these pages are trade names, service marks or trademarks of their respective owners.

Contents

Editor notes

by Alexey Kovyazin

Money for nothing 4

Oldest Active

by Helen Borrie

RTFM — regarding those free manuals 5

Community

by Carlos Cantu

About Dolphins and Birds 6

Cover Story

by Paul Ruizendaal

The story of Fyracle 8

Development area

by Paul Ruizendaal

Common Table Expression in Fyracle 12

Products Overview

IBSurgeon Products 17

Development area

by Paul Ruizendaal

Morfik's WebOS: Innovating beyond LAMP 18

by Vlad Horsun

Global Temporary Tables in Fyracle 23

TestBed

by Alexey Kovyazin

TPC-C based tests results 26

Bonus

by Alexey Kovyazin

Comprehensive Repairing Guide. Part 1 35

Money for nothing

I guess some of our readers have become a bit tired of waiting and wondering when the new issue would be released. I am really sorry about the long delay, but there have been some serious financial and time issues holding up production.

I want to begin with sincere thanks to people who made donations to our magazine. Orlin Chankov and Steve Kramer, thank you, guys, for your help. Your donations were much appreciated.

We've come to realise that producing the magazine with only advertising and sponsors to support it is impossible. We've decided to change our business model slightly. We are going to run special proprietary editions of each issue with bonus articles. These special editions will be available in electronic (printable PDF) and paper versions. From this issue forward, the latest issue will be available only for purchase for a period—say one or two weeks from publication—and will be fully released afterwards for everyone, minus the bonus articles.

It's regrettable to have to move in this direction but it seems the only way to make the magazine viable. Orlin and Steve, for their generosity, will have free, unlimited subscriptions to the electronic version of IBDeveloper magazine.

Borland

On February 8, Borland published plans to spin off the developer product line, including Delphi and InterBase. It looked so peculiar at the time that many developers said "Borland is dead", and probably several less printable expressions. It was really strange and, for months, all we had were rumors and fuzzy press-releases.

Recently, however, the chief Borland executives of the new "DevCo" (David I, Jason Vokes and others) went on a worldwide tour to clarify events and plans for the developer community. I met with Jason in Moscow on May 23 at

a «Borland Developer Studio» conference and a lot of things got clearer.

In essence, Borland is splitting into two companies: one is solution-targeted (ALM) and the other is oriented on developer tools. As I understand, the Developer Company (alias "DevCo") guys were the initiators of the spin-off because they considered the IDE product line too different from the ALM business. I can even guess that Delphi and the other IDEs were the financial props for ALM! Jason did not say so, of course, but it seems quite obvious to me.

According to their statements, there's no problem with Delphi sales—Delphi 2006 is a bestseller—and no problem with its development. Jason demonstrated an impressive roadmap for Delphi up to 2009, with many exciting features, and things of interest for JBuilder and C++ Builder too.

According to the team, InterBase sales are good and its roadmap is interesting, too. There is no indication that DevCo intends to drop all these profitable products and, personally, I feel much better about it all now. We can only await the completion of the spin-off.

Just one other thing—why isn't there any name for "DevCo" yet? One guess is that the "Borland" brand is one of the most precious assets of the company and both the IDE and ALM sides want to have it. Anyway, Delphi and InterBase are here to stay, regardless of whether the branding is "Borland" or "DevCo"!

Jim

Jim Starkey was one of the main newsmakers since the previous issue. Jim is a living legend and becomes more and more popular (I think he should start to sell autographs via eBay!). MySQL bought Jim's company Netfrastructure and hired Jim to make a proprietary database engine for them. In this issue Carlos H. Cantu comments on all these events in his article "From Birds to Dolphins". As editor I should emphasize that this article is his personal

By *Alexey Kovyzin*,
editor@ibdeveloper.com



(rather interesting) point of view. Jim's departure from active development on Firebird provided an excellent opportunity to promote Firebird. Now everyone knows where successful companies are looking to headhunt developers and get good ideas :)

Fyracle

This issue is mostly devoted to Fyracle. I'd like to express great thanks to Paul Ruizendaal, Vlad Horsun and Eugeni Putilin for their help with creating this issue. Fyracle is on the cutting edge of new Firebird technologies. Many things that are already implemented there are planned for inclusion in Firebird 3.0, so you can download the future right now. Read on about Fyracle's exciting features and give it a try!

Unfortunately, we couldn't include the article about External Procedures in Java and .NET in this issue. We'll try to ensure that it makes it in time for the next issue.

TPC-C based test

And yes, we've finished the TPC-C series testing. There are some very interesting results, so don't miss the article about it in the TestBed section of our magazine!

Bonus article

Our bonus material for this issue is the first part of "The Comprehensive Repairing Guide for InterBase and Firebird". This part is "Corruption reasons, Part 1". If you are keen to know where the pitfalls and traps for your database are, order the bonus version of "The InterBase and Firebird Developer Magazine"!

RTFM - Regarding Those Free Manuals

By **Helen Borrie**
 helebor@tpg.com.au



A quaint side effect of the Firebird “wanna-be” culture is the oft-sung lament, entitled “Where Are My Free Manuals?” I’m undoubtedly a bit over-sensitive to the assertion that users of a totally free database engine have some indisputable right to expect to pay nothing, ever, for up-to-date user documentation. As the author of a Very Fat Book about Firebird, I doubt I will ever get happy about those list messages and private emails that criticise Greedy Me for getting it published commercially instead of just giving it to you as a gift. It took a year and I’m not even close to making any profit out of it. I never expected I would. But—ouch!—it hurts.

Contrary to popular belief, Borland did not release the IB 6.0 documentation set sources in 2000, nor any other form of documentation. One of the few things Borland was prepared to be clear about was that it would act against anyone in Firebird’s community who tried to re-publish or update any documentation to which it claimed copyright. Fortunately for those who demand free documentation beyond the basics, unauthorised downloads of these seven PDF books have remained available around the Web to this day.

What that means to us as a community is that every bit of Firebird published documentation has to be written on blank paper. It takes time, care, effort, skills and bodies. That’s why, almost six years on from the opening of the InterBase code, the Firebird Project still doesn’t have its own complete, integrated, free documentation set.

It does have Quick Start Guides for its two released versions and a few papers on topics of interest to newbies, experienced users, or both. It does have three complete sets of detailed release notes that document every bug-fix, change and enhancement and tell users how to install and configure that particular Firebird version. Those prepared to reach past the end of their noses can

discover that full, free-beer documentation is available and can be at your fingertips with minimal effort.

Besides that, and top of the agenda currently, Firebird-Docs also has in hand a large volume of user documentation that is work-in-progress, around 20 weighty chapters that began life in 2002, as a Firebird 1.0 ebook manual that was distributed to subscribers of the IBPhoenix CD service. The Volume I manual, “Using Firebird”, was given to the Firebird Project in 2004 as Framemaker 6 source code under open document licensing. Converting the arcane Adobe sources to the Firebird Docs project’s custom Docbook XML format took a lot of work for a couple of people.

Since then, updating those chapters has fallen largely to Paul Vinkenoog, the very same guy who picked up the unfinished, unusable Docbook system begun by David Jencks and refactored it into the sophisticated, multi-language system that today can build our release notes and all of the project’s currently released user docs in a variety of formats and languages. Some of

the original chapters need more work than others, while some others need to be laundered to eliminate content that seems too Borlandish for peace of mind. Paul has a couple of chapters of “Using Firebird” almost ready to go now. As the various chapters become ready for use with Firebird 2.0 and 1.5, so they will be built and published.

That’s where your free manuals are: in the hands of the willing few. The Docs project (see <http://www.firebirdsql.org/index.php?op=devel&sub=doc>) urgently needs some reasonably accomplished writers of English, with Firebird skills, to pick up the XML sources of these chapters and upgrade them to Firebird 2.0, taking in Firebird 1.5 along the way. Waiting in the wings are Spanish, Japanese, Russian and Brazil Portuguese translators. There are tasks aplenty here for those with some time to commit and the will to participate in documenting Firebird.



**29 July
2006**

**Piracicaba,
Brazil**

One day full of Firebird information. Three international speakers, along with big names of the Brazilian community. All this with a extreme low cost subscription.

www.FirebirdDevelopersDay.com.br

* All talks in portuguese. Translation will be available on international talks.



About Dolphins and Birds...

It seems to me that we are living in unprecedented times. As never before, Open Source software has traditional companies like Microsoft, Oracle and IBM running scared. If you think about Linux vs. Windows, it has been the case for a while already. Now, it seems to be carrying over to database server systems.

Those who follow some of the technology news sites are probably aware of recent announcements from the “big guys”: Oracle and IBM both released free versions of their respective database server products. Microsoft already had MSDE, along similar lines.

Those free versions do come with a variety of limitations, such as restricting RAM/CPU or database size, and, of course, although free, they are NOT Open Source. They notably lack one of the base benefits of open source, the ability to get your hands on the source code and make your own, custom builds. And the question of whether those free versions will be maintained or updated by the vendors remains unknown.

The important thing to note is that things are changing! Two or three years ago, it would have been hard to imagine Oracle or IBM releasing free versions of their flagship software. In the intervening years, open source database engines have been growing in features, power and market space conquest. The threat of being overtaken is now forcing the “big guys” to change the way they do business.

MySQL Under Attack

Recently Oracle bought Innobase, the Finland-based manufacturer of InnoDB, the most fully featured engine used by MySQL database. Next, Oracle acquired Sleepycat, the company behind BerkeleyDB, another database engine used by MySQL. It is clear to me that Oracle’s actions are attempts directed at affecting market confidence in MySQL. Remember that MySQL has an impressive number of installations all over the world and claims to be the most-used database for Internet applications. MySQL 5 prom-

ises a lot of new features, including support for stored procedures, triggers, referential integrity, etc., introducing more advanced capabilities that could make MySQL suitable for than more just simple data repositories.

So what does all of this have to do with Firebird? Firebird is Open Source software, making the Firebird community direct or indirect players in this game... but there is more than that.

From Birds to Dolphins

Jim Starkey, creator of the original InterBase and main developer of the Vulcan fork of Firebird, announced on February 18 that MySQL AB had bought his company (Netrastructure) and he was moving on with it (see announcement box). Ann Harrison, who is a partner in the Firebird/InterBase support company IBPhoenix and also Jim’s wife, is to work part-time for MySQL AB. Ann has stated publicly that she has no intention of leaving the Firebird Project.

Before talking about how this can affect the future of Firebird development, let’s check why MySQL did this.

Jim has a great reputation as a software architect and for his first-hand knowledge about RDBMS development and architecture. Transactions, stored procedures, concurrency and all the other powerful relational database features are no mystery to him. Acquiring Netrastructure and bringing Jim aboard along with it could be viewed as MySQL’s answer to Oracle’s attacks. They want Jim’s brain so they can start developing their own fully featured database engine for MySQL. It won’t surprise me to find a lot of similarities between Firebird and the next generations of MySQL.

What About Firebird?

If you are worried about Firebird’s future, calm down! The project loses nothing from the recent events. Actually, it can even win, as I will explain.

Jim Starkey had no direct involvement in the development of Firebird 1.0, 1.5 and 2.0. All three versions were the

By **Carlos H. Cantu**
carlos@firebase.com.br



work of the Firebird development team, a group of people from many places around the world. Jim jumped into the scene almost three years ago, when SAS contracted IBPhoenix to fork Firebird 1.5 and develop a systemically multi-threaded version for its own purposes. IBPhoenix hired Jim to do the work. From that point, he became an active participant in the Firebird discussion lists and involved himself with the team. When Jim’s contract with SAS ended, SAS gave Vulcan to the Firebird Project.

I’m sure Jim would not get directly involved with Firebird if he wasn’t being paid to do his work. The Vulcan project was based on a forked version of Firebird 1.5 code and remains a totally separate project from Firebird 2.0. The next major Firebird release, Firebird 3.0, is planned to be a merger of Vulcan and Firebird 2.0 into a single code base.

The merger process is being conducted by the Firebird team. Jim has stated that he would continue to lurk in the Firebird lists from time to time, so I don’t doubt that he will be around if his advice is needed during the merger process. However, the Firebird team comprises a highly competent group of developers with a thorough understanding of the code and the requirements. The InterBase 6.0 code was released with no access to internal documentation and, in almost six years, the Firebird developers have been digging and figuring out for themselves how to build, change and clean that buggy code. These guys are really good and I’m sure they will do a great job in the merge process!

What does Firebird win from this?

Jim’s announcement attracted a lot of attention to Firebird. FireBirdNews, one of the first sites to spread the notice

about the Starkey move, got a lot of links from many other technology sites and blogs all over the internet. That provided opportunities for potential new users, who previously had no contact with Firebird, to discover how good and special the product is. A special paper was created to introduce Firebird to those people: "Get to know Firebird in 2 minutes" is now available at least eight languages, with the intention of presenting Firebird to outsiders in a simple, concise way.

I'm sure Firebird is getting a lot of new users who can be persuaded to become Firebird Foundation members and so start to contribute to the project funding. Some of them might join the development team and help with bug fixing and implementing new features. Others might contribute to documentation and testing.

In other words, Firebird is out there in the media and the community is getting the kind of exposure it needs to get bigger! Consider my line of thinking: why should you wait years until MySQL can finish a new engine, if Firebird offers all the features of a real RD-BMS right now?

What if Oracle wants to buy Firebird?

It would be a very difficult (should I say "impossible"?) thing to make happen. Firebird has no owner and there is no individual or company that owns or controls its inherent rights and assets. Oracle (or anybody else) can take the code for free and roll their own Firebird, as long as they comply with the open source licensing, but nobody can come and buy Firebird. We don't need to worry about this. Firebird belongs to the community!

Conclusion

I hope this article provided some clarification of recent events and was able to show how Firebird can gain from the extra exposure. Firebird is a great product, and it is getting better and better at each new release.

You can find some more information about this by checking the posts at www.FirebirdNews.org.

Announcement box:

My company, Netfrastructure, Inc., has been acquired by MySQL, AB. As part of the agreement, I will be working full time for MySQL. I expect to lurk on the architecture list from time to time and may contribute the occasional wolf-o-gram, but I will not be taking an active part in Firebird development. Although Ann will work for MySQL, part time, translating from wolf to English, she will continue to be active in the Firebird project.

My decision to join MySQL has almost nothing to do with Firebird and everything to do with Netfrastructure. The Netfrastructure platform represents what I feel about contemporary computing hardware and future application requirements, and has been the center of my technical heart and soul for six year. Some aspects of Netfrastructure technology have already been contributed to the Vulcan project, but Firebird and Netfrastructure are architecturally incompatible. An attempt to integrate the tech-

nologies would be unlikely to meet the goals of either project.

MySQL and Firebird have never seen each other as competitors and I doubt this will change in the future. The projects have different open source philosophies, different technologies, different customer bases, and different sweet spots. The ideas behind the two projects are, happily, public and available to all. If MySQL and Firebird compete, it is only competition in offering the best possible support to their respective customers.

I am pleased to have had the opportunity to finish the Vulcan project. The combination of Vulcan SMP and architecture combined the rich feature set of Firebird 2 will make a solid release and a superb platform for future development.

I wish the Firebird project all the best in years to come. And if you need an opinion, please feel free to call.

Jim Starkey

Customer satisfaction is our top priority! Integrating the Enterprise!



better office is a German company specialized in Borland and Microsoft development environments

With offices in Oldenburg (head office),
Frankfurt, Berlin, Pennsylvania (USA)
and a Team of appr. 25 associates.

better office provide development tools,
custom made software development,
software consultancy,

Project management, training and support.

They have extensive experience in developing client-server
and web based database systems.

Benefits:

Borland Delphi WIN32 and .Net
Microsoft .Net Framework
(C# Visual Studio)
JAVA JBuilder and Eclipse
Borland InterBase
Microsoft SQL-Server
IBM DB/2/40 (iSeries)

Head Office: Oldenburg
Stau 19 – 6.0G -
26122 Oldenburg

Tel.: +49 (0)441 926740

Fax: +49 (0)441 2488675

E-Mail: info@better-office.com

The story of Fyracle

The birth of Oracle-mode Firebird

Some of the best conversation takes place over dinner. Picture yourself in Brussels, February 2003. It was the weekend of the annual FOSDEM conference and I had just attended Ann Harrison's talk about Firebird. For several years I had been looking for a database to accompany Phoenix Object Basic, a cross-platform work-alike of Visual Basic. It needed something equivalent to Microsoft's Jet database engine, only it had to run on both Windows and Linux and it had to be less fragile than Jet. It looked like Firebird could fit the bill.

About half a year earlier I had been searching for an accounting package and had come across something called "Compiere". Compiere is an open source ERP package, written in Java, running on top of a Jboss application server. With several hundred thousand downloads, Compiere was getting noticed. It had one big drawback: it only ran on Oracle. The author of Compiere, Jorg Janke, had tried to port his software to Postgres, but had given up. I asked him what caused him to abandon the project, and he said that Postgres lacked three essential features: sequences, embedded transactions (he meant savepoints) and PL/SQL. This reply intrigued me and I wondered whether Firebird would be up to the job.

With these two issues in mind I had e-mailed Ann and asked if I could invite her to dinner to talk about Firebird during the FOSDEM conference. Ann, probably cautious not to have dinner alone with a complete stranger asked if Paul Beach could be there and as it happened most of the IBPhoenix crew joined the dinner. So there we were, enjoying hearty Flemish food and discussing where Firebird could go.

Using Firebird as a better Jet for Phoenix Object Basic was not much of a topic. It was quickly concluded that it had nearly all features on my wish list, with one exception. I would have liked for the possibility to link to external ODBC database tables. It was suggested that

this could perhaps be modeled on the existing code for external flat file tables. This, I think, was a good idea and remains on my list of future projects for Firebird. Today, I would look at generalizing the external table mechanism to a variety of sources, the most important being another Firebird instance. The toughest hill to climb would be to figure out how to deal with transactions.

The more elaborate part of the dinner conversation was about Compiere and how Firebird could support it. Sequences were not an issue, generators fit the bill perfectly. Embedded transactions was a bigger topic. At the time I did not understand all that much and it took a while before we could figure out that what was needed were user level savepoints. Around that time, these had just been added to the Firebird 1.5 development branch, so that hurdle seemed taken as well.

Remained the issue of PL/SQL and syntax differences between Firebird's SQL and Oracle's SQL. Various approaches were discussed. One such option was converting PL/SQL to the equivalent PSQL text and working from there. There are some products on the market that attempt to do this for other conversions (e.g. Oracle to DB2, etc.). None of these appear to work very well, because if they did, database system migration would not be seen as such a major task.

A second approach was suggested by Paul Beach. He reminded everyone at the table of the old BLR interface to Firebird that still existed. It was once the main connection API into the engine, in a time when client programs used pre-processing instead of dynamic SQL to communicate with the database engine. BLR is an acronym for Binary Language Representation. It is a low level language that high level relational languages like SQL, GDMML and QUEL can be compiled to. Paul suggested that it might be possible to write a compiler that mapped PL/SQL to BLR. The issue of accepting Oracle's flavour of SQL could be handled in a similar way.

By Paul Ruizendaal
pnr@janus-software.com



After tossing the idea around for about an hour, everybody agreed that it might work. In theory. Covering every nook and cranny of Oracle's sometimes weird behaviors would be hard. To make a general solution would be a major amount of work. It would always be out of date with the latest release of Oracle. All very sensible comments. But I only want to run one specific program, I countered. The problem set is defined and not unbounded. Skeptical faces remained. With one exception, Paul Reeves. With a deep, thoughtful look he agreed: it actually might work.

The challenge was on the table. It looked like there was a plan that could give Firebird an oracle-compatibility mode. But who was going to do it? The IBPhoenix crew looked rather exhausted from the 2000-2002 experiences and had enough struggles of their own to deal with. They were not going to shoot off on a tangent just because of a pleasant dinner. So, during the dinner I resolved to look into how hard it would really be and Ann kindly agreed to answer questions I may have. Paul Beach promised to dig up a copy of an old Interbase 3.3 manual for BLR for me, which he did.

So, Fyracle was born in a pub-eatery in Brussels. If the project ever gets to be famous, I will have to go back to the place and put a plaque on the wall. Not sure I can still find it, though.

Building the architecture

In March of that year work began on exploring the work needed to make Compiere run on Firebird, with minimal, or even better no change to Compiere. Work started with getting the table definitions across and moving the data. As it was obvious that a full-blown SQL compiler would be needed further down the road, the first item to

FIREBIRD



FYRACLE

Fyracle is an enhanced version of Firebird 1.5, designed for corporate use.

Fyracle adds support for hierarchical queries, for temporary tables and for java stored procedures.

It also adds support for Oracle style SQL (joins with (+), etc.) and full PL/SQL.

Fyracle is available for both Windows and Linux and installs with just a few clicks.

The evaluation version is free, the full version costs Euro 49.95. Orders placed in October get a **20% discount** when using the coupon code **IBD1031**

www.janus-software.com



**JANUS
SOFTWARE**

be written was a lexer and a simple recursive descent parser that could handle the essential oracle-style DDL statements. Rather than writing out BLR, the back-end of the compiler wrote out a text file with Firebird-style SQL. This was completed about a month later.

The compiler was then extended with the ability to parse a subset of Oracle's DML language, covering the most common constructs. The hardest part in this early phase was figuring out how to handle Oracle's join syntax. As most of you probably know, Oracle has a non-standard syntax for joining tables, using the (+) operator.

```
SELECT tb11.a, tb12.x FROM tb11, tb12 WHERE tb11.a=tb12.x(+);
```

What the compiler does is analyzing the WHERE clause and find the sub-expressions that use the (+) operator. These expressions are then ordered and potential cycles are detected. Cycles are illegal and an error results. The WHERE clause is then rewritten to contain only the other parts. The sub-expressions using the (+) operator are moved to the JOIN ON clauses.

```
SELECT tb11.a, tb12.x FROM tb11 LEFT JOIN tb12 ON tb11.a=tb12.b;
```

With this work done, some of the basic elements were in place, but Fyracle was still a text-to-text tool. For Fyracle to support Compiere it would need to work together with the Jaybird JDBC driver. Ann kindly introduced me to Roman Rokytskyy, who was kind enough to help and has been providing invaluable input ever since. After some discussion, it was decided that Fyracle should expose a standard Firebird client API, so that the JDBC driver could connect to it as a Type II driver. Roman added code to load "fyracle.dll" when an oracle-mode connection was specified and I wrapped the text-to-text tool in a dynamic library exposing the standard API. So, the connection stack would look like the below graphic:

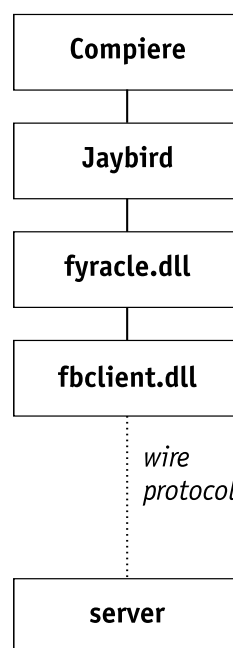


figure 1

At this point we had some basic concepts in place. We could take an Oracle DDL file and use it to define the database structure in Firebird. We could connect a Java client and use basic oracle-style SQL. This SQL was translated on-the-fly to firebird-style SQL and passed to the engine. The result set, if any, was passed back to the Java client in the usual way.

Working with PL/SQL

In parallel work had taken place on compiling PL/SQL into something that Firebird could digest. With the help of Paul's manual and Ann's input, I was beginning to understand the concepts behind BLR. It wasn't long before I had to accept that there were some real challenges here: BLR is not a natural match with PL/SQL. It was designed to be a low level representation of relational languages with some extensions added to help it handle procedural elements like you would find in for instance the "qli" command line tool.

PL/SQL in contrast has a very different design philosophy. It started life as a client side scripting language in Oracle extensions like the "Forms" package. In its concepts it is - according to Oracle - modeled on the ADA programming language, which is rooted in the Algol family of programming languages. The most well known member of this family is Pascal. Being in the Algol family means that PL/SQL supports something called lexical scoping, which in turn means that procedures can be defined inside of other procedures. The variables of the current instance of the outer procedure are available in every

instance of the inner procedure.

At first I tried to figure out how the requirements of PL/SQL could be mapped to regular BLR. At a few points I was considering to add significant new features to it, even revamping it completely. The execution engine for BLR evolved from the co-routine structure that Jim had once developed for Datatrieve and it had evolved so much during 20 years that a major redesign would make sense. In the end I decided that such a task would go beyond my will and skill and chose another route.

The solution was to create a simple byte code interpreter for an Algol-like procedural language, which would be separate, but have tight integration with the relational engine. The design for this byte code engine is similar to the byte code engines that used to support microcomputer Pascal back in the early eighties. We have probably all written such an engine back in university, as an assignment in compiler construction class. For lack of a more creative name, I call it the "btc engine", for byte code engine. Often I refer to the btc engine as the "procedural engine", to distinguish it from the blr engine, which I refer to as the "relational engine".

As in PL/SQL, this engine started life as a client side test implementation. Using the lexer and SQL parser that was already taking shape, code was added to parse the various expressions and control structures. A semantic module was added that verified data types and various other checks. A back-end was added that took the parse tree and generated code for the btc engine. From simple beginnings, the parser, the semantic checks and the btc engine gradually grew in capabilities.

At this point, the btc engine was still a client side only entity, much like PL/SQL itself had been in its earliest incarnations. It needed an implementation inside the engine and a way for the relational engine to call into the procedural engine and vice versa. The first thing to look at was the existing UDF code, which enables the relational engine to call into external code. It seemed like this would work, but it offered no way for the external code to call back into the engine, using the same connection

and transaction.

New code had to be created to build a superset of the UDF mechanism. The engine was modified to accept new syntax that called an external function and provided that external function with the means to call back into the engine. The code was modeled on both the UDF code and the EXECUTE STATEMENT code that had just been added to the Firebird 1.5 development branch. Much debate took place on the Architecture mailing list about how external procedures should call back into the engine. The discussion revolved on the best design of the API and the proper architectural layering of code. Fyracle needed to keep moving, so it took a shortcut: the API was kludged together and the layering was broken. The latter didn't mean much as Borland had already broken the layering and proper refactoring would not occur before Firebird 3.

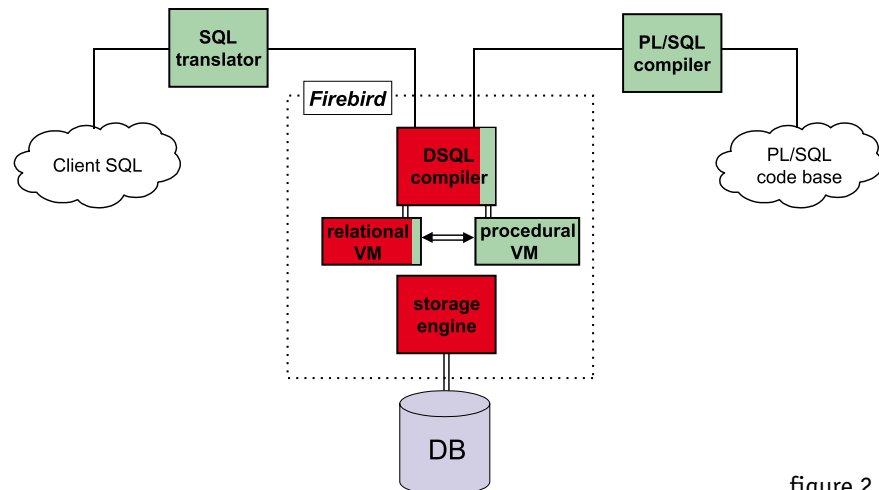


figure 2

By now time had progressed to about November 2003 and all the basic elements seemed in place. Fyracle could connect to a Java client and accept oracle-style SQL statements. There was a compiler to take the source of a PL/SQL procedure, function or trigger and compile it to byte code and the Fyracle build of Firebird could execute that byte code (see below picture).

You can imagine our joy when the Compiere application came up whilst connected to Fyracle for the first time.

Step-wise refinement

The joy soon turned to a long grind of bug fixing. Almost every function in Compiere failed because a built-in oracle function had not been implemented yet, because the PL/SQL compiler had made an error or because the SQL translator did not cover a wide enough range of the full syntax.

The harder cases were those where Firebird did not have a certain bit of functionality that was required by Compiere. On such a case was Oracle's CONNECT BY syntax, which allows hierarchical queries. Another round of debate on the Architecture list ensued. In those discussions it became clear that hierarchical queries are best conceptualized as a special form of union and that the standard's WITH syntax was a clearer way to express that (see separate article about WITH in this issue on page ...). So, the Fyracle build of Firebird was extended with support for recursive WITH and the oracle connector used this syntax to express CONNECT BY statements.

By March 2004, almost exactly one year after that dinner in Brussels, Fyracle was able to run large parts of the Compiere application. As it happened, I had to be in the USA that month and took the opportunity to fly out to Massachusetts and meet with Ann and Jim, proudly showing the result. On the whiteboard the component architecture was drawn out (similar to figure 2) and it was agreed that it was a good way to do things. The client-side SQL-connector, of course, is not such a nice part and it would be better if it moved to the server-side.

In discussions it came up that it would be so much nicer if the architectural layering of Firebird would be restored. This would mean that Firebird's SQL compiler (module "dsql" in the source tree) would move from the Y-valve component to the core engine component. This would enable the use of dynamic SQL in the core engine, whereas it currently relies on preprocessing. I argued that if the way this was done provided a clean API for pluggable SQL compilers, I could swap in my oracle-mode SQL compiler for clients connecting in oracle-mode.

Jim did implement such a structure in the Vulcan branch of Firebird. I had promised to look into cleaning up the code in a specific core engine module which I never found the time to do. As a result of this mal performance, Jim promised to string me up in the tree next to his house. Luckily, he informed me last November that the wind had blown over the tree he had had in mind.

The "coming out party" for Fyracle was the Firebird conference in Fulda in May 2004. Most of the summer little happened in further development of functionality, as the initial goal had been reached. Towards the end of the summer, Marius Popa managed to get Fyracle discussed on Slashdot and interest slowly but surely started to rise. For one, the "oracle-mode" thing was picked up by Computer Associates ("CA") as a nice marketing gimmick for the open source release of Ingres. CA announced the "million dollar challenge", which amongst other projects promised \$600K for a working migration tool for oracle-based applications. After some thought and a look at Ingres, I decided to stick with Firebird and not take part in the competition. Also in that Summer, EnterpriseDB got started on oracle-mode Postgres, although they did not come out of "stealth mode" until May of 2005.

So, motivated by competition, work started to make Fyracle a real product. In order to create an easy to use package, the Fyracle configuration was cleaned up and a graphical installer for both Windows and Linux was added. A guy named Arek Heldt showed up with

a reasonable cross-platform GUI admin tool and agreed to create a special Fyracle plug-in for it. Tim O'Reilly was nice enough to allow me to include some of his manuals into the developer kit. Fyracle 0.8.0 was born in November 2004 and has gone through 10 dot-releases since.

Two of those release contained major functionality provided by other team members. Vlad Horsun had worked out how to add global temporary tables to Firebird, but his code was too late to be included in the FB2 release. Vlad kindly agreed to back-port his code from FB2 to FB15 and from there it was included in Fyracle in May 2005.

Eugeney Putilin had been working on creating stored procedures written in Java for a long time, probably starting as early as 2003. By 2005 he had figured out the issues and gotten to working developer builds. Roman Rokytssky had modified the Jaybird driver to also support usage inside the server, rather than only from a client. Eugeney, Vlad and Roman worked hard late summer 2005 and delivered working code early in the autumn. It was included in Fyracle in October 2005. Soon after, Carlos Guzman added a plugin for dotNET based languages and the CLR. This plugin and the dotNet driver will be included in the next release of Fyracle.

Outlook

What once started as a one-off project to see if it could be done has evolved into a long-term project. Much has been achieved, but much more remains to be done.

- The PL/SQL supported by Fyracle is roughly Oracle 8i with some 9i extensions. Some things are glaringly missing and will be added soon, such as table functions and collections. Other features to be added are the object-oriented features of PL/SQL that were mostly added in 9i. The language enhancements in version 10g are not all that significant.

- On the relational side, Firebird needs to get some upgrades for future releases of Fyracle. Things like statement and database level triggers, deferred triggers, materialized views, 10-base floats

and a 128-bit data type come to mind.

- On the infrastructure side, Fyracle needs to implement a few of the more commonly used predefined packages, including integration with a web server.

- Last but not least, it would be nice to resurrect the clustering code that was once written for VAX clusters and use it to allow using Firebird on a Linux cluster.

Fyracle has been fun for the team that created it. I hope that it has been fun for you, the reader and user as well.

Special thanks

I would like to use this opportunity to thank everybody who helped making Fyracle a reality, by giving advice, contributing code or simply moral support. Special thanks go to, in alphabetical order:

Tiberiu Adorei

Paul Beach

Arno Brinkman

Carlos Guzman

Ann Harrison

Arek Heldt

Vlad Horsun

Holger Klemt

Pascal Legrand

Marek Mosiewicz

Eugeney Putilin

Roman Rokytssky

Nikolay Samofatov

Jim Starkey

Claudio Valderrama

Dmitry Yemanov



Common Table Expressions in Fyracle

Introduction

Whilst building the oracle-to-firebird translation module for Fyracle, it became clear that Firebird does not have support for hierarchical queries. To Oracle users, such queries are known as CONNECT BY queries, after the syntax that Oracle uses to express them. The SQL standard does not support oracle's CONNECT BY syntax, but does support a different syntax known as recursive common table expressions. This syntax is implemented by DB2 and by SQLServer in its latest release (2005).

In order to be able to support CONNECT BY statements in Fyracle's oracle-mode, Firebird had to be enhanced to support hierarchical queries. In order to stay close to the standard, it was implemented using recursive common table expressions.

In this article, I will explain the basic syntax of common table expressions, of recursive common table expressions and the relation to Oracle's CONNECT BY syntax.

Syntax definition

Common table expressions (CTE's) are define using a new syntax element around the WITH keyword. Essentially, the normal SELECT syntax is extended with a preamble that defines common expressions. The WITH syntax specifies a temporary named result set. One can think of the named result set as a named derived table. It is derived from a simple query and defined within the execution scope of a single SELECT statement. A common table expression can include references to itself. This is referred to as a recursive common table expression and in this usage the keyword RECURSIVE is obligatory.

Syntax

```
<with statement> ::=
WITH [RECURSIVE] <common table expression> [ , <common
table expression>... ] <select_statement>

<common table expression> ::=
    expression name [ ( column name [ ,...n ] ) ]
    AS
    ( CTE query definition )
```

Expression name

This is a valid identifier for the common table expression. An expression name must be different from the name of any other common table expression defined in the same WITH <common table expression> clause, but expression name can be the same as the name of a base table or view. Any reference to expression name in the query uses the common table expression and not the base object.

Column name

Specifies a column name in the common table expression. Duplicate names within a single CTE definition are not allowed. The number of column names specified must match the number of columns in the result set of the CTE query definition. The list of column names is optional only if distinct names for all resulting columns are supplied in the query definition.

CTE query definition

Specifies a SELECT statement whose result set populates the common table expression. The SELECT statement for CTE query definition must meet the same requirements as for creating a view, except a CTE cannot define another CTE

Specifying more than one WITH clause in a CTE is not allowed. For example, if a

By Paul Ruizendaal
pnr@janus-software.com



Expert's note

Fyracle ... Yes!

Fyracle ... Yes, it has lot of Oracle specific features.

Really, InterBase and Firebird sometimes used as "small Oracle's brothers". Main office uses Oracle, sub-units uses Firebird.

Want to move from Firebird or Interbase to work with >250 gigabytes database? Go to Oracle.

But, now Fyracle can be used not only as Oracle-like RDBMS.

It's on the edge of new technologies, like Java Stored Procedures, Global Temporary Tables, and all other cool things.

Try Fyracle, if you want to dig in to the future.



Our expert



Dmitri Kuzmenko is the chief expert of "The InterBase and Firebird Developer Magazine" with 20 years of InterBase and Firebird experience

e-mail: kdv@ib-aid.com



IBSurgeon repair services

If you need fast and secure repair service "on-demand" you can sign up to IBSurgeon repair services.

Using remote administration tools (like Terminal Server) we can repair even the largest databases within a few hours.

If you have a corrupted database, contact us and we will help you!

Free investigation and time/price estimation
You pay only for successful recovery

Average bill is
USD\$799
(after discounts)

Contact us now:
support@ib-aid.com

www.IBSurgeon.com

CTE query definition contains a subquery, that subquery cannot contain a nested WITH clause that defines another CTE. The recursive form of WITH must contain exactly two query definitions, an anchor member and a recursive member, joined together by a UNION ALL operator.

Setting up an example

To start, we will define and fill an employee table. The table has an employee ID, which is the primary key, a manager ID, which links the employee to his manager, and name and salary columns:

```
CREATE TABLE emp(empid INTEGER NOT NULL PRIMARY KEY,
                  name  VARCHAR(10),
                  salary DECIMAL(9, 2),
                  mgrid  INTEGER);

INSERT INTO emp VALUES ( 1, 'Winkel',    30000, 10);
INSERT INTO emp VALUES ( 2, 'Henkel',    35000, 10);
INSERT INTO emp VALUES ( 3, 'Bassinger', 40000, 10);
INSERT INTO emp VALUES ( 4, 'Lessig',    38000, 10);
INSERT INTO emp VALUES ( 5, 'Korning',   42000, 11);
INSERT INTO emp VALUES ( 6, 'Beagle',    41000, 11);
INSERT INTO emp VALUES ( 7, 'Reilly',    36000, 12);
INSERT INTO emp VALUES ( 8, 'Smith',     34000, 12);
INSERT INTO emp VALUES ( 9, 'Boot',      33000, 12);
INSERT INTO emp VALUES (10, 'Monroe',    50000, 15);
INSERT INTO emp VALUES (11, 'Schindler', 52000, 16);
INSERT INTO emp VALUES (12, 'King',      51000, 16);
INSERT INTO emp VALUES (13, 'Jones',     54000, 15);
INSERT INTO emp VALUES (14, 'Scott',     53000, 16);
INSERT INTO emp VALUES (15, 'Mills',     70000, 17);
INSERT INTO emp VALUES (16, 'Gaastra',   80000, 17);
INSERT INTO emp VALUES (17, 'Gates',     95000, NULL);
```

Looking at the inserted rows, it is apparent that employee 'Gates' is the top manager for whom 'Gaastra' and 'Mills' work. 'Scott' in turn works for 'Gaastra' and has no employees of his own. 'Monroe,' on the other hand, manages 'Henkel,' 'Bassinger,' and 'Lessig' and is an employee of 'Mills.'

Using common table expressions

Common table expressions can make complex SELECT statements more legible. For example, the following example shows the number of employees reporting directly to each manager:

```
WITH DirReps(mgrid, dirreps) AS
(
    SELECT mgrid, COUNT(*) as dirreps
    FROM emp
    WHERE mgrid IS NOT NULL
    GROUP BY mgrid
)
SELECT mgrid, dirreps
FROM DirReps
ORDER BY mgrid;
```

The resulting output is:

MGRID	DIRREPS
10	4
11	2
12	3
15	2
16	3
17	2

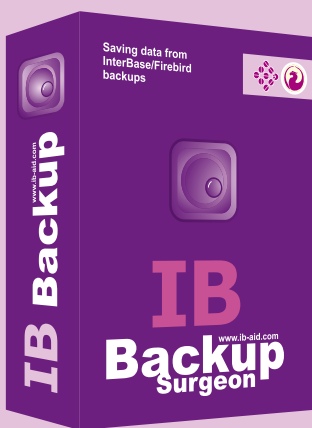


IB Backup Surgeon

Saving data from InterBase/Firebird backups

IBBackupSurgeon is a tool to read and save data from corrupted Firebird or InterBase backup files. With this tool you can browse a backup file, select tables you need and then extract them to a new or existing database.

Supports InterBase 5.x7.x and Firebird 1.x2.x backups.



www.ib-aid.com

And the following example shows the maximum number of employees reporting to a single manager:

```
WITH DirReps (mgrid, dirreps) AS
(
    SELECT mgrid, COUNT(*) AS dirreps
    FROM emp
    GROUP BY mgrid
)
SELECT MAX(dirreps) AS "Max. Number of Direct Reports"
FROM DirReps
WHERE dirreps >= 2 ;
```

The resulting output is:

```
Max. Number of Direct Reports
=====
4
```

Although common table expressions can be used as a hint to the optimiser about how a subquery is used, and to make it explicit that a single subquery is used at more than one place in a query, such optimisations are currently not implemented.

A simple recursive query

A reasonable question to ask is: "Who works directly or indirectly for 'Gaastra'?" To answer the question of who works for 'Gaastra,' an Oracle developer might write the following query:

```
SELECT name
FROM emp
START WITH name = 'Gaastra'
CONNECT BY PRIOR empid = mgrid
```

START WITH denotes the seed of the recursion while CONNECT BY describes the recursive step. That is how to get from step n to step $(n + 1)$. Since it is important to distinguish between the n th and the $(n + 1)$ th step during name resolution, PRIOR is used to show that empid belongs to the n th step while mgrid belongs to step $(n + 1)$ th. So with empid being 16 for step 1, mgrid must be 16 as well and hence step 2 produces 'Scott,' 'King,' and 'Schindler.' Their empids will now serve as PRIOR to step 3, and so on and so forth.

The Oracle syntax is very concise. The SQL standard WITH syntax uses regular SQL to describe the exact same relationships. As you will see, it is more verbose, but equally straightforward:

```
WITH RECURSIVE n(empid, name) AS
(
    SELECT empid, name
    FROM emp
    WHERE name = 'Gaastra'
    UNION ALL
    SELECT nplus1.empid, nplus1.name
    FROM emp as nplus1, n
    WHERE n.empid = nplus1.mgrid
)
SELECT name FROM n;
```

What makes this CTE special is that it is referred to within its very own definition. This is what distinguishes a regular CTE from a recursive CTE. Here I named the CTE n to correlate to the recursive steps. A recursive CTE consists of two parts combined with a UNION ALL:

- The seed or step 1 of the recursion. This is what is described in Oracle using START WITH. In a recursive CTE, it is simply any query providing a set of rows. In this case we query the emp table and filter for 'Gaastra.' We select the name of course and also the empid, because we need it for the recursive step.
- The recursive step going from n to $(n + 1)$. Here we refer to step n (the CTE n)

and join in step (n + 1) using the same predicate used in CONNECT BY. Instead of PRIOR, regular correlation names are used to distinguish n from (n + 1). It is noteworthy to look at the output of this query and realize that the recursive process is a depth first recursion:

```
NAME
=====
Gaastra
Schindler
Korning
Beagle
King
Reilly
Smith
Boot
Scott
```

Before moving on to pseudo columns and more complex examples, I want to briefly explain where the WHERE predicate of an Oracle recursion needs to be placed. We will modify the example to return all employees, including their salaries, working for 'Gaastra' who earn more than 40,000.

```
SELECT name, salary
FROM emp
WHERE salary > 40000
START WITH name = 'Gaastra'
CONNECT BY PRIOR empid = mgrid
```

It is interesting to note how the WHERE clause preceded the recursive specification. For a non-oracle developer, it might seem that the predicate belongs to the set of rows being considered to begin with. This, however, is not correct. Instead, the WHERE filters the final result and belongs at the end of the matching CTE based query:

```
WITH RECURSIVE n(empid, name, salary) AS
    (SELECT empid, name, salary
     FROM emp
     WHERE name = 'Gaastra'
    UNION ALL
     SELECT nplus1.empid, nplus1.name, nplus1.salary
     FROM emp as nplus1, n
     WHERE n.empid = nplus1.mgrid)
SELECT name, salary FROM n WHERE salary > 40000;
```

This query results in the following output:

NAME	SALARY
=====	=====
Gaastra	80000.00
Schindler	52000.00
Korning	42000.00
Beagle	41000.00
King	51000.00
Scott	53000.00

The LEVEL pseudo column

The most well-known of the pseudo columns is LEVEL. The purpose of this column is to show the number of the recursive step n that produced the row. In our example, it indicates the levels of management between 'Gaastra' and the employee plus 1 (because LEVEL starts with 1). Here is the original Oracle example enhanced with LEVEL:



FIBPlus components

Direct access to all InterBase and Firebird features in Delphi, C++ Builder and Kylix applications!



- No middleware
- Easy porting from IBX
- Improved performance and optimized network traffic

MultiProfile tool

Launch Delphi, C++ Builder and C# Builder with different IDE settings for separate projects. With MultiProfile you can install different versions of the same components simultaneously without any conflicts!

See online flash-demo!

Special offer for IBDeveloper readers:

Two products for one price!

Visit ibd.devrace.com and buy FIBPlus and MultiProfile for 235 Euro!

Supercharge your database!

New service from IBSurgeon



Supercharging means increasing performance of your database application(s).

When you think that your InterBase or Firebird application works slow, or you need to make it work faster, and when you are looking for the easiest, fastest and cheapest way to increase its performance - you need our supercharge service.

Pay only for result



www.ibsurgeon.com

```
SELECT LEVEL, name
FROM emp
START WITH name = 'Gaastra'
CONNECT BY PRIOR empid = mgrid
```

The SQL standard saw no need to add syntax for this feature because it can be expressed using regular SQL:

```
WITH RECURSIVE n(lvl, empid, name) AS
  (SELECT 1 lvl, empid, name
   FROM emp
   WHERE name = 'Gaastra'
  UNION ALL
   SELECT n.lvl + 1, nplus1.empid, nplus1.name
   FROM emp as nplus1, n
   WHERE n.empid = nplus1.mgrid)
SELECT lvl, name FROM n;
```

The output for this query is:

LEVEL	NAME
1	Gaastra
2	Schindler
3	Korning
3	Beagle
2	King
3	Reilly
3	Smith
3	Boot
2	Scott

All I have done here is to introduce a level column, which starts with 1 and increments by 1. Of course, any semantics is possible, but this one happens to provide the same semantics as LEVEL.

Conclusion

In this article I provided generic mappings from the Oracle style CONNECT BY recursive query syntax to Firebird's standard compliant recursive common table expressions using UNION ALL. While the Oracle syntax is less verbose because it provides keywords for various common semantics, this also means that it is less expressive since little new semantics can be added without changes to the DBMS.

The InterBase and Firebird Developer Magazine is looking for talented authors.

We will be glad to publish articles regarding
InterBase and Firebird,
including reviews of related products
and services.

authors@ibdeveloper.com

www.ibdeveloper.com

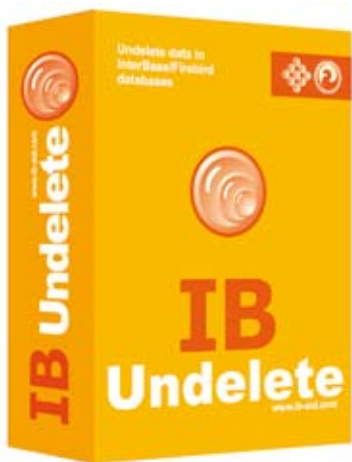
IBSurgeon Products



IBFirstAID

IBFirstAID is a tool that can be used for automatically diagnosing and repairing corrupted Firebird or InterBase databases. It can fix up to 80% of often corruptions.

Supports InterBase 5.x-7.x and Firebird 1.x-2.x databases.



IBUndelete

IBUndelete is a tool which can undelete occasionally deleted records in InterBase or Firebird databases. It uses unique IBSurgeon core engine for direct work with data inside database.

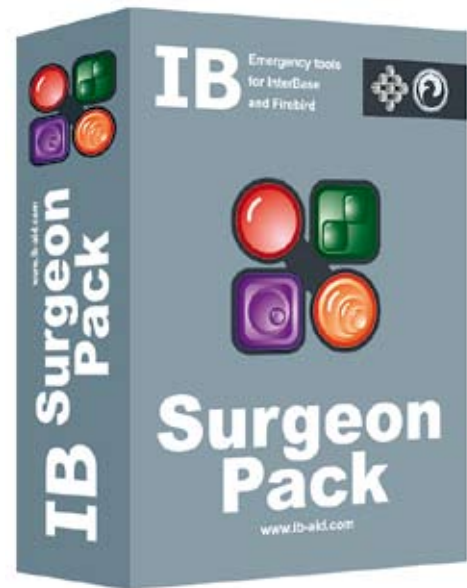
Supports InterBase 5.x-7.x and Firebird 1.x-2.x



IBAnalyst

IBAnalyst is a tool that assists a user to analyze in detail Firebird or InterBase database statistics and identify possible problems with database performance, maintenance and how an application interacts with the database.

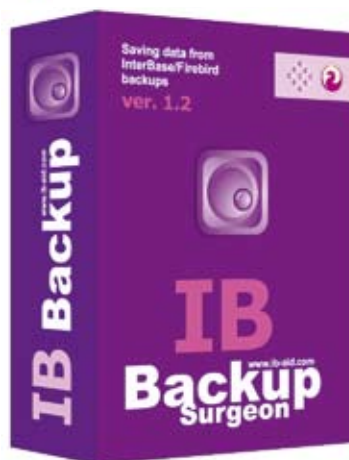
Supports InterBase 5.x-7.x and Firebird 1.x-2.x databases.



IBSurgeon Pack

IBSurgeon Pack is a comprehensive set of tools to repair InterBase and Firebird databases and backups, undelete occasionally deleted records and foresee database problems.

**Buy IBSurgeon Pack
for EUR 399 and save 197 EUR!**



IBBackupSurgeon

IBBackupSurgeon is a tool to read and save data from corrupted Firebird or InterBase backup files. With this tool you can browse a backup file, select tables you need and then extract them to a new or existing database.

Supports InterBase 5.x-7.x and Firebird 1.x-2.x backups.

MORFIK's WebOS:

Innovating beyond LAMP

Last month saw the first public beta of a revolutionary new tool to develop web applications. Several developers have commented that using WebOS for the first time gives that same sensation of excitement that they felt when they first worked with Delphi 1.0 back in the early nineties. WebOS comes with a bundled Firebird database server.

By **Paul Ruizendaal**
pnr@janus-software.com



Introduction

In less than two decades since its humble beginnings, the World Wide Web has not only permeated a fair portion of our lives but has also become the subject of much discussion and speculation as a viable alternative to traditional platforms for business applications. For many years the web community has tried to overcome the limitations of web browsers and their related internet protocols by extending the capabilities of browsers and servers in a variety of ways. Some have tried to extend the functionality of browsers using plug-ins and applets, while others have tried to push the entire computing task to the server.

If taken to either of these extremes, the status of the browser is essentially reduced to that of a "dumb terminal". With applets and plug-ins, the browser is merely a host for another application which runs inside a box. With server-side computing, the browser simply displays what it receives like a slide-show. While centralized systems, such as corporate server farms or applications hosted by ASPs (application service providers), are believed to need little more from the browser than the functionality of a dumb-terminal, in real life the browser has steadily grown in functionality to the extent that it is now a viable alternative platform for both the web and desktop applications – irrespective of their on-line or off-line state.

Recently, developments such as Asynchronous JavaScript And XML (AJAX), have captured the imagination of developers worldwide and have allowed us to re-examine the capabilities that have existed in web browsers for some time. Applications such as Google Gmail,

Google Maps and Flickr have shown that the user experience in a browser can rival that of desktop applications, with the browser powered by nothing more than JavaScript, HTML/XML and XMLHttpRequest.

However, for AJAX to succeed in the long term, it must be supported by professional development tools that are specifically designed for creating web applications and also incorporate the design methodologies and features offered by products such as Visual Studio® and Delphi®. Ideally, developers should be able to leverage their existing language skills and use a familiar Integrated Development Environment (IDE) to develop AJAX applications without the need for learning JavaScript or for hand-coding HTML.

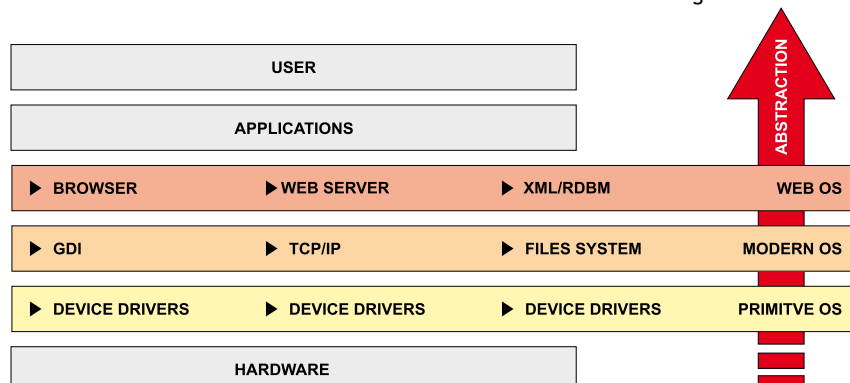
This is precisely what Morfik's WebOS AppsBuilder offers. It allows programmers to implement the business logic of their application in a high-level object oriented language of their choice and develop the presentation layer of their application using a visual design environment. WebOS AppsBuilder compiles the project's code, resources and objects into an AJAX application that can run both on-line and off-line.

A new role for traditional operating systems

The browser is already the undisputed platform of choice and indeed the driving force behind a new class of applications which are centered around the user and their social interactions. Examples include e-mail, blogs, CRM, collaboration software, discussion forums, e-commerce and community portals.

With the advent of AJAX, browsers have demonstrated an ability to provide a user experience that rivals the desktop enabling the browser to claim new territory within the corporate and business applications sphere. Today, many accounting and financial applications – inventory control, sales and marketing, production planning, human resources management and payroll to name a few – can not only be implemented as browser applications but can also benefit from the portability and wider connectivity that the browser provides.

As the browser continues its march in the direction of both traditional desktop territory and new and innovative landscapes, traditional operating systems will gradually take a back seat and revert to their original mission of



providing a hardware abstraction layer. This will allow traditional applications to be gradually superseded by a new class of applications that fully utilize this new operating system and are designed from the outset to be user-centric.

A new approach

“Our objective is to bring web applications to the desktop and take desktop applications to the web”

Can we use the browser as a platform for real-life business applications? Will such applications provide a user experience similar to conventional desktop applications? Will these applications work both on-line and off-line?

The answer to all these questions is a resounding yes! The capabilities required for this have been natively available in browsers for some time, yet we have been hampered by the difficulty of writing JavaScript code and distracted by inadequate attempts to extend the browser's functionality.

By combining the following elements in a single stand alone package, web applications can be brought to the desktop and desktop applications taken to the web:

1. HTML - for the user interface
2. JavaScript - for the application logic
3. XMLHttpRequest - for asynchronous browser requests
4. Web Server - for handling the browser requests
5. Database - for managing data

This concept is not new and appears deceptively simple, yet its implementation presents enormous technical challenges. Unless these challenges are addressed and overcome, the concept shall remain on the drawing board.

Morfik has identified and addressed these technical challenges and in the process has created a unique Integrated Development Environment for developing business applications on the browser platform. The following is a technical account of these challenges and their solutions.

• Limitations of a page-centric architecture

Traditional web applications are comprised of a number of web pages that are separate from one another both spatially and computationally. Unlike the user interface of conventional desktop applications, the browser content changes in a disjointed fashion – sometimes with unacceptable time-delays. This spatial separation of web pages has a detrimental effect on the user experience. Recent developments, such as AJAX, have proven successful in enhancing the user experience, and applications such as Google Gmail and Google Maps are examples of this.

Those who develop for the web are forced to scatter the business logic across the application space and find workarounds for managing the application state in an otherwise stateless environment. This computational separation of web pages has a detrimental effect on the reliability and scalability of the application. In contrast, conventional software engineering emphasizes a unified computational model to ensure reliability and scalability.

One option is to push all of the computing to the server-side in order to unify the computational space and make the application more reliable. But this has three undesirable side-effects: first, it further limits scalability; second, it is dependant on both the availability of band-width and the reliability of the connection; and third, the application is no longer available when it is unplugged.

Morfik has the following solution to limitations of a page-centric architecture: WebOS AppsBuilder applications are not page-centric. The browser content “morphs” according to the requirements of the application. This simplifies state management and works hand-in-hand with an application-specific AJAX engine. WebOS AppsBuilder creates this AJAX engine from the business logic written in a high level language of choice using its unique and patented JavaScript Synthesis Technology (‘JST’). This approach unifies the computational space of the application across the server and the client.

WebOS AppsBuilder applications do not use applets, plug-ins or cookies. The result is an application comprised purely

of HTML and JavaScript, which rivals desktop applications in user experience and computational integrity.

• Limitations of HTML page layout

HTML pages are designed to display their content in a fluid layout much like a word processor. Fluid layouts take the shape of their container. Changing the size of the browser's window will change the alignment of page components. Web designers have managed to find work-arounds to address many undesirable side effects of this fluid model.

With the advent of Cascading Style Sheets, browsers gained the ability to display the elements of the page in a predetermined fixed position – irrespective of the shape or size of the containing window. This fixed layout is similar to conventional desktop applications. But since neither the size of the browser window nor the resolution of the client display is controlled by the developer, the CSS model has some undesirable side-effects also. The ideal model is a plastic layout that combines the strengths of fluid and fixed models. Such a plastic layout is particularly useful when database reports and tabulated data are displayed or page elements change their size and position due to user interaction or business logic.

WebOS AppsBuilder's plastic layout offers the strengths of both fluid and fixed layouts. In Morfik applications, the content of the browser is a hierarchy of heterogeneous nodes. Each node is aware of the existence, state and behaviour of other nodes and can respond to layout changes at run-time according to the rules set out by the programmer at design-time. In other words, at design-time, the programmer can define both the fixed position of each element as well as its run-time plasticity. This uniquely incorporates the best of both worlds in layout design.

• Limitations of JavaScript

The difficulties of writing extensive and yet coherent JavaScript code is the Achilles' heel of AJAX and if not addressed could eventually slow down its widespread uptake. Few acknowledge or recognize that JavaScript's capabilities extend beyond that of a mere scripting language for light programming tasks.

However, market acceptance of JavaScript for implementing large-scale applications faces the following practical challenges:

a) Syntax - JavaScript borrows its syntax from C. While C/C++/C# and Java programmers feel at home with case-sensitive short-hand syntax, the large number of developers who have mastered other languages such as Visual Basic and Delphi face a frustrating transition.

b) Semantic Design - JavaScript is a prototype-based (instead of class-based) object-oriented language. This also presents developers with the challenge of adopting a whole new mindset.

c) Lack of rigor - JavaScript does not provide the developer with a rigorous programming model. For example, it does not support type declarations, nor can functions receive their parameters by reference, to name a few. This limitation is a disadvantage in large scale programming.

d) Interpreted - Interpreted languages do not offer the benefit of compilers in rigorous enforcement of application integrity at compile time. Small changes to an otherwise perfectly working program can cause unpredictable run-time crashes. For example the most common error messages in existing web-page scripts are "object expected" or "object does not support this property".

Morfik solves these problems with its revolutionary JavaScript Synthesis Technology ('JST'). Using WebOS AppsBuilder, programmers implement the business logic of their application in a high-level object oriented language of their choice (e.g. C++, C#, Java, Delphi). WebOS AppsBuilder then compiles this code into a JavaScript AJAX engine.

The process is a true compilation and avoids boilerplates or code snippet libraries. The source code written in the object-oriented, strongly typed language of choice is first passed through a parser that includes a tokenizer and syntax analyzer. The output of the parser is passed through a semantic map builder which builds a detailed semantic map of the entire application. This map conveys the full "meaning" of the application logic. This semantic map is then compiled into JavaScript code that is semantically identical to the original code written by the programmer and conveys the exact same "meaning".

This process offers several advantages:

- Mastering a programming language is a long and arduous process. Switching to a new language requires much more than learning a new syntax. It requires an idiomatic shift in the programmer's thought processes. This skill takes time to acquire. By allowing developers to program in the language they have already mastered, Morfik not only encourages the uptake of AJAX and improves the quality of the output, but also reduces the development cost of AJAX applications.

- The clean separation of the parser, semantic analyzer and compiler allows Morfik to compile from a variety of source languages into a variety of target languages. The only condition is that the source language is either object-oriented (e.g. C++/C#/Java) or has been specifically modified to support object-oriented constructs. (See Appendix A for further details on) "multi-language support in Morfik".

- Strongly typed compiled languages follow a rigorous programming model that has made compilers the preferred

tool for building large and complex applications.

- Compiled code is, by definition, more reliable and scalable than interpreted code. Although the output of the Morfik compiler is interpreted JavaScript, the compilation process nevertheless ensures rigorous type checking and enforcement of referential integrity in producing reliable and scalable applications.

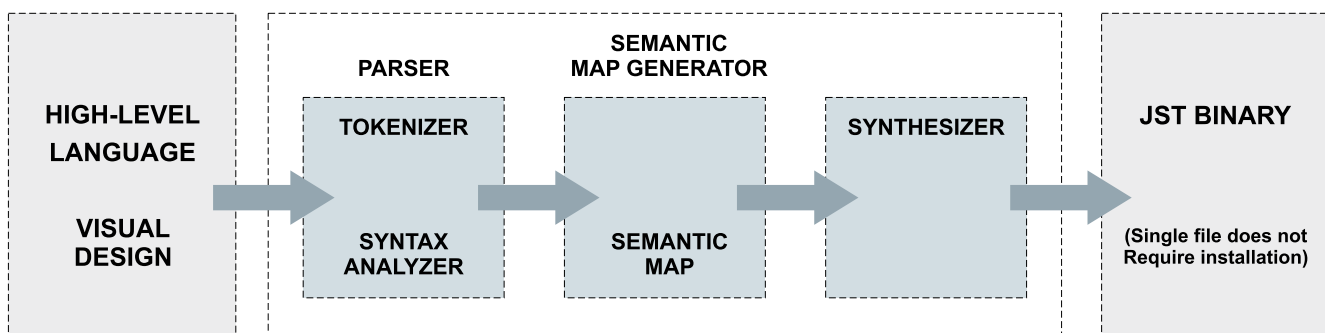
- The AJAX engine created through the Morfik process is an exact semantic equivalent of the high level source code which reflects the programmer's intentions without imposing limitations. Code-snippet libraries and pre-fabricated component frameworks, by their very nature, limit the programmer's ability to choose the appropriate level of abstraction.

Since the output of this process is neither an executable in machine-code, nor a one-to-one translation of source code, nor a collection of predefined code snippets, WebOS AppsBuilder's process is referred to as JavaScript Synthesis Technology ('JST'):

Client-side handling of browser requests

Browsers are designed to work with servers using a stateless request-response model. Normally, the browser and the server do not co-exist on the same hardware, and browsers need to stay connected to the web in order to maintain a dialog with the server. Of course, this is natural and the way of things on the web. However, to bring the web applications to the desktop and enable them to work after they are unplugged from the web, one must be able to handle the browser requests locally.

MORFIK JST COMPONENTS



One obvious solution is to install an HTTP server on the local host. This will also fulfill the second part of the objective, namely taking desktop applications to the web. Many mature and stable examples of HTTP servers are freely available. Some are stand-alone applications whilst others are small embedded systems. Alternatively, the relevant subset of HTTP server protocols can be implemented within the application so it can natively communicate with the browser and process browser requests.

More importantly, the browser vendors have recognized the benefits and possibilities that local handling of browser requests can offer, and they are planning to add this ability to the browser itself – effectively merging the HTTP server and the browser into a unified system.

WebOS AppsBuilder has an open and highly flexible architecture which can support the local handling of browser requests in a variety of ways. To provide a mature, stable and well tested option that can allow its applications to run under Windows, Linux and OS X, WebOS AppsBuilder tightly integrates an embedded open-source Apache server into Morfik applications by default. This is not an exclusive option and will not limit the developer's choice of other methods

Database needs

Business applications are predominantly data-driven and depend heavily on a relational database system. In the corporate world, data is centralized and database servers do not normally co-exist with the client applications on the same hardware. Consequently, the client needs to stay connected to the network in order to maintain a dialog with the server.

To bring data-driven web applications to the desktop and enable them to work after they are unplugged from the network requires a local database engine. This will not only fulfill the second part of the objective, namely taking desktop applications to the web, but also will facilitate the implementation of large scale distributed databases and distributed computing.

WebOS AppsBuilder has an open and highly flexible architecture and can integrate local database engines or provide remote database connectivity in a variety of ways. To provide a mature, stable and well tested option that can allow its applications run under Windows, Linux and OS X, WebOS AppsBuilder tightly integrates a Firebird server into its applications by default. This is not an exclusive option and will not limit the developer's choice of other database engines.

Productivity

AJAX applications have created a lot of excitement but until professional Integrated Development Environments become available, developers will find it difficult to justify the effort – and therefore the cost – associated with AJAX programming.

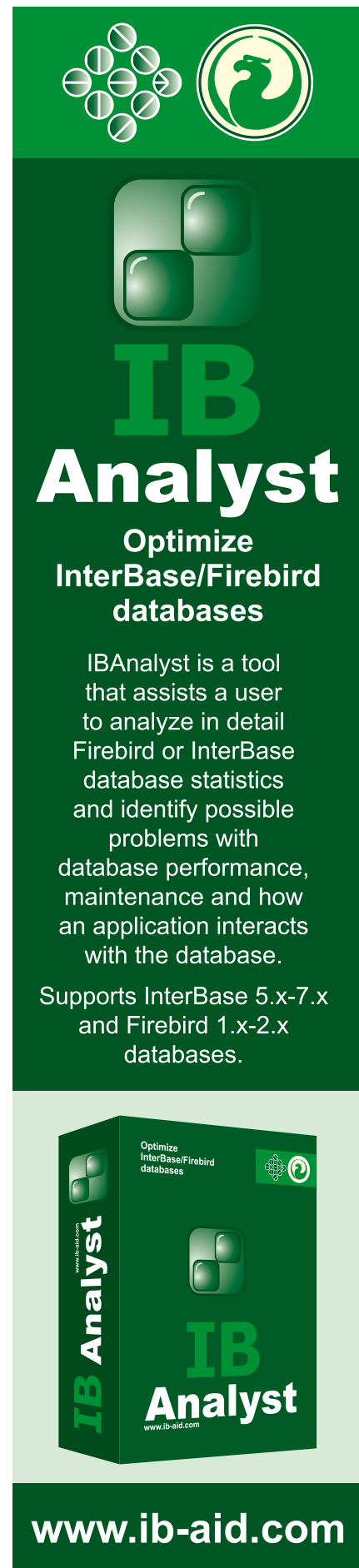
It is Morfik's belief that designing an IDE around code libraries and component frameworks is a repetition of past mistakes. Such tools make it deceptively easy for developers to get simple applications up and running quickly only to have them hit the wall as their projects grow in size and scope.

To ensure reliability, scalability and maximum productivity, JavaScript Synthesis Technology leverages the power of high level languages and the existing skills of programmers. WebOS AppsBuilder is a world class system which incorporates the features that are required for a truly professional AJAX-based Integrated Development Environment.

Multi language support in WebOS

At the core of the WebOS AppsBuilder Integrated Development Environment for AJAX lies the time-tested and proven technique of using a strongly typed object-oriented language and a compiler, which strictly enforces referential integrity, high levels of modularity, refactoring and polymorphism within the source code, to develop applications that are reliable, scalable and easy to maintain.

The output of a compiler is invariably in a language different from that of the source code – usually machine code but



The advertisement for IB Analyst is set against a green background. At the top, there are two circular logos: one with a cluster of dots and another with a stylized horse head. Below these is a large, 3D-style icon of a cube with rounded corners. The main title 'IB Analyst' is prominently displayed in large, bold, white letters. Underneath the title, the text 'Optimize InterBase/Firebird databases' is written in a smaller, white font. A paragraph of text describes the tool's capabilities, followed by supported database versions. At the bottom, there is a 3D rendering of the software's product box, which mirrors the design elements of the advertisement. The website address 'www.ib-aid.com' is printed at the very bottom in white.

IB Analyst
Optimize InterBase/Firebird databases

IBAnalyst is a tool that assists a user to analyze in detail Firebird or InterBase database statistics and identify possible problems with database performance, maintenance and how an application interacts with the database.

Supports InterBase 5.x-7.x and Firebird 1.x-2.x databases.

www.ib-aid.com

sometimes an intermediate language. A good example of this is the Microsoft compiler that takes the source code in C#, VB.NET or J# and produces code in Microsoft Intermediate Language (MIL). When examining a snippet of sample code in Microsoft Developers Network (MSDN), one is usually given the option of seeing it in one of these languages. Yet if these code snippets are carefully compared, one realizes that semantically all three languages are identical! Therefore, the MIL code generated is also the same.

To achieve this, in the case of Visual Basic, Microsoft had to take it to the next level, and developed the strongly typed object oriented VB.NET. Although unpopular with some VB programmers this move was nevertheless a technical necessity.

Morfik has used this strategy in developing Morfik Basic, Morfik C#, Morfik Java

and Morfik Object Pascal (with more to come) for the AJAX platform. Programmers choose their favorite syntax (or even mix and match a number of them) to develop their application. This allows them to benefit from the rigorous software engineering techniques and system design methodologies inherent in the object-oriented programming paradigm. WebOS AppsBuilder then synthesizes the source code into semantically equivalent JavaScript code and packages it into an AJAX application.

In this model, the synthesized JavaScript in WebOS AppsBuilder is the equivalent of MIL in .NET. In addition, WebOS AppsBuilder does not prevent extending the JavaScript output by linking-in external hand written JavaScript code libraries. However, such external code would not benefit from the same rigorous checks as native WebOS AppsBuilder code.

WebOS AppsBuilder has a flexible and

highly modular architecture. There is a clean separation of the syntax parser, semantic analyzers and the target code synthesizer. This enables WebOS AppsBuilder to compile from a variety of source languages into a variety of target languages. The only condition is that the source language must be strongly typed and either object-oriented (e.g. C++/C#/Java) or has been specifically modified to support object-oriented constructs.

Free download

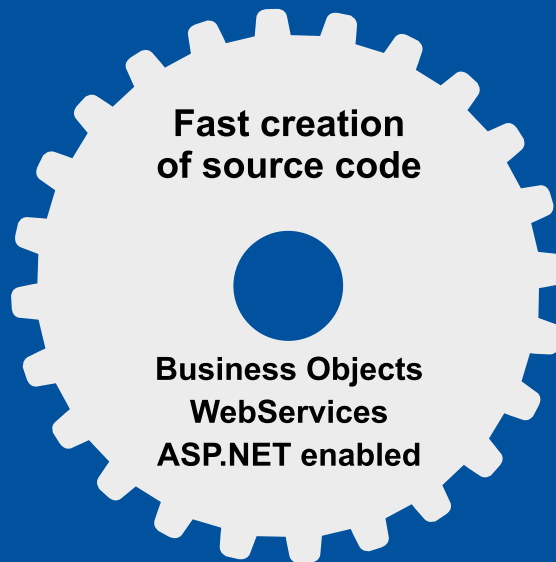
The beta releases of Morfik's WebOS AppsBuilder is a free download.

Please check <http://www.morfik.com> to download your own evaluation copy and get a feel for the future of web application development.

Move over LAMP, WebOS is here!



BDPSourceCodeProducer for InterBase



More information can be found on our web site
www.better-office.com



www.better-office.com
info@better-office.com

better office
Oldenburg
Stau 19
26122 Oldenburg
Tel.: +49(0)441 92674-0
Fax: +49 (0)441 2488675

Global Temporary Tables in Fyracle

By **Vlad Horsun**, hvlad@users.sourceforge.net

Programmers sometimes need to store data that is temporary in nature: for example, to create a data set, work with it and remove it. If it is an infrequent or one-off necessity, just creating and dropping a table is feasible.

But what if the need for temporary tables arises regularly? In practical terms, constantly creating and dropping tables is not good from the security point of view—end-users don't usually have the appropriate rights—and performing DDL on production databases is against all recommendations, anyway. Thirdly, such an approach is far from perfect for productivity.

All that apart, if the same sets of temporary data are needed for several users simultaneously, each would be compelled to assign them unique names, making such tables unworkable with stored procedures.

The “Service Field” Approach

One widespread approach to eliminate these problems is to create the table as a permanent table with an additional service field populated during the life of the working data set with `CURRENT_USER` or with a private value from a generator.

Although this approach is free from the aforesaid deficiencies, it is not without deficiencies of its own, not least being the accumulation of many back versions as the subsequently unwanted rows are deleted. Since the additional field is usually indexed, the index remains interesting to all queries on these tables. The removed records are never visited again, to make the back versions eligible for garbage collection, and this garbage will sit in the database until the a sweep is run.

Fortunately, the SQL standard defines a solution for such problems. True—it doesn't always promise a quiet life for

the ordinary programmer! :). The solution comes as various kinds of temporary tables. In this article we'll consider global temporary tables, first as they were implemented in IB 7.5; then, the Fyracle implementation, that is to be included in Firebird soon, after version 2.0.

A Definition of GTTs

Global temporary tables (GTTs) are tables with permanent metadata, stored in the system catalogue, but with the temporary data.

GTT's may be associated with data having two kinds of persistence: within the lifetime (scope) of the connection in which the given GTT was referenced and within the lifetime (scope) of just the referencing transaction. The data in GTTs from different connections or transactions, respectively, are isolated from one other, but the metadata of the global temporary table is shared by all connections and transactions.

Syntax and semantics

The statement syntax for creating the metadata of a temporary table in the system catalogue is as follows:

```
CREATE GLOBAL TEMPORARY TABLE <table_name> <table_elements>
[ON COMMIT {PRESERVE | DELETE} ROWS]
```

The `ON COMMIT` clause sets the scope of the data persistence for the temporary table:

`ON COMMIT PRESERVE ROWS` causes data left in the table after a transaction ends to remain in database until the end of the connection

`ON COMMIT DELETE ROWS` causes the data to be deleted from database immediately after a transaction ends

If the optional `ON COMMIT` clause is not specified, then `ON COMMIT DELETE ROWS` is used by default.

`CREATE GLOBAL TEMPORARY TABLE` is an ordinary DDL statement that is processed by the engine the same way as a `CREATE TABLE` operation, so creating and dropping a GTT within a stored procedure or trigger is similarly prohibited. The SQL standard has other kinds of temporary tables that can be allowed such treatment, the most suitable for this purpose being the `DECLARED LOCAL TEMPORARY TABLE`—but that's another story.

A GTT differs from a permanent table by way of two new flags in `RDB$RELATIONS`. `RDB$FLAGS`. Because it involves no change to the on-disk structure (ODS), it was possible to include GTTs in Fyracle, which is based on Firebird 1.5.x. A GTT may have indexes, triggers, field→ level and table level constraints, just like ordinary tables.

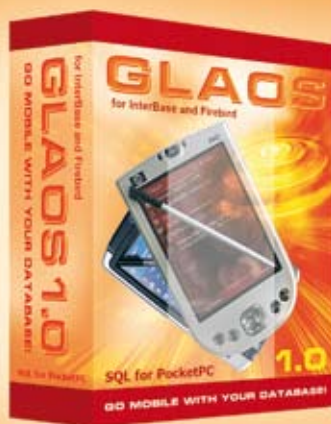
However, constraints between temporary and persistent tables follow some specific rules:

- references between permanent and temporary tables are forbidden
- a GTT with `ON COMMIT PRESERVE ROWS` cannot refer to a GTT with `ON COMMIT DELETE ROWS`

GLAOS

Go
Mobile
with
your
database!

Glaos for
InterBase/Firebird
and MySQL



www.ibsurgeon.com

c) a domain constraints cannot refer to a GTT.

The following tabulation simplifies how the rules apply:

master\detail	persistent	preserve rows	delete rows
persistent	allow		
preserve rows		allow	
delete rows		allow	allow

Implementation details

A GTT instance—a set of data rows created by and visible within the given connection or transaction—is created when it is first referenced, usually at statement prepare time. Each instance has its own private set of pages on which data and indexes are stored. Data rows and indexes have the same physical storage layout as permanent tables. When persistence ends (the connection detaches or the transaction ends, depending on the scope) all pages of a GTT instance are released immediately. Although this is similar to when you do DROP TABLE, of course the metadata remains in the database.

The clearing of a GTT is much quicker than the ordinary row-by-row delete and its associated garbage collection. Each data (index) page contains rows (keys) from the same GTT instance, meaning the connection or transaction should read/write fewer pages than they would if GTT instances were isolated by the service field. With the “service field” design, in contrast, data pages would continue to contain rows from old transactions and dead connections until it was garbage collected.

It can happen that the GTT design requires more space than the “service field” design. For example, 10 instances of a GTT, each with one row, would occupy 10 data pages, while, under the “service field” design, only one page would be used, as long as it could accommodate 10 rows. This difference doesn’t really become evident except with a large number of small tables, so we can accept that as a reasonable price for the benefits of speed and almost instant cleanup.

In Fyrcle, the data pages of all of the GTT instances are placed in the database file. In Firebird 2+, with better performance the objective, these pages will be placed in separate files to allow temporary data to be written to another hard disk. This also enables temporary files always to be opened with forced writes off, regardless of the database setting. And—yes—there are plans to consider this mechanism for implementing tablespaces. (No, I didn’t write this :)

There is no limit to the number of GTT instances active in the database. If you have N transactions active simultaneously and each transaction has referenced some GTT then you’ll have N instances of that GTT.

The InterBase implementation of GTT has one “featurebug”. If you have two transactions in one connection and both transactions insert some rows into a GTT ON COMMIT DELETE table and then one transaction commits, the data in the second transaction’s GTT instance will disappear. Firebird’s implementation of GTT is free of this problem.

Examples

```

- create usual (permanent) table :
CREATE TABLE PERSISTENT (
    ID INT NOT NULL
);
- create temporary table with scope of transaction
CREATE GLOBAL TEMPORARY TABLE GTT_DELETE (
    ID INT
) ON COMMIT DELETE ROWS;

- create temporary table with scope of connection
- try to reference on permanent table (this is forbidden!)

```

```
CREATE GLOBAL TEMPORARY TABLE GTT_PRESERVE (
  ID INT
  CHECK (EXISTS(SELECT * FROM PERSISTENT))
) ON COMMIT PRESERVE ROWS;
This operation is not defined for system tables.
unsuccessful metadata update.
global temporary table "GTT_PRESERVE" of type on commit
preserve rows cannot depend on persistent table "PERSISTENT".
```

```
- create temporary table with scope of connection
CREATE GLOBAL TEMPORARY TABLE GTT_PRESERVE (
  ID INT
) ON COMMIT PRESERVE ROWS;
- Lets look how rows in different transactions and connection are handled
INSERT INTO GTT_DELETE VALUES (CURRENT_TRANSACTION);
INSERT INTO GTT_PRESERVE VALUES (CURRENT_TRANSACTION);
SELECT * FROM GTT_DELETE;
```

```
      ID
=====
      5
```

```
SELECT * FROM GTT_PRESERVE;
```

```
      ID
=====
      5
```

```
COMMIT;
SELECT * FROM GTT_DELETE;
- no rows
```

```
SELECT * FROM GTT_PRESERVE;
```

```
      ID
=====
      5
```

```
- our one row still here
start another isql session:
SELECT * FROM GTT_PRESERVE;
- no rows, as expected
```



IB Undelete

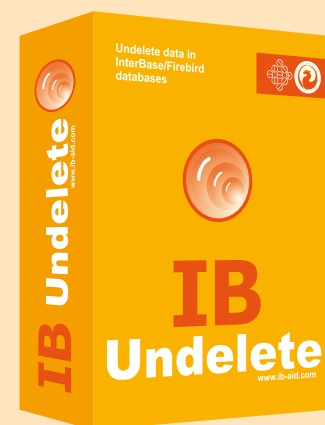
Undelete data in InterBase/Firebird databases

IBUndelete is a tool which can undelete occasionally deleted records in InterBase or Firebird databases.

It uses unique IBSurgeon core engine for direct work with data inside database.

Supports InterBase 5.x7.x and Firebird 1.x2.x

Buy Paper Versions



www.ib-aid.com

TPC-C based tests results

Well, it's time to say who the winner of our comparison test is. Usually tests are similar to beauty contests – someone should be the winner. But I have to disappoint you – it wasn't the case for our tests and there is no absolute winner.

While the test results indicated certain trends that were fairly dependent on settings and hardware configurations, it did not bring a single overlord into the light. For the impatient amongst the readers I can say that InterBase 7.5.1 showed the best results in most but not all of the high-end configurations, with some stability issues.

Let's proceed to the test results.

Test team

First of all, I need to express my sincere gratitude to Alexey Karyakin and Vlad Horsun who created and adapted the TPC-C based test toolkit for public usage.

Then I'd like to thank people who helped to perform tests on different hardware and software configurations: Vadim Dokukin (www.niklaus.ru), Anton Glazunov, Sergey Chernyak, Eugeny Putilin and Sergey Mereutsa.

Special thanks to Thomas Pfister (www.nevrona.com) and Daniel Magin (www.better-office.com) for great help in carrying out tests on special configurations.

Test technique

I anticipate a lot of questions related to the execution of the tests, so please read this part carefully to avoid a lot of them.

We used a TPC-C based toolkit to carry out our tests. In essence this test simulates a database warehousing system with W warehouses and T terminals (i.e. users) working simultaneously with these warehouses, mostly inserting and updating records.

All tests were carried out on Windows platform. I am first to agree that it is a major omission but we've had insufficient time and resources so far to adapt the tests for Linux.

The test toolkit contains 3 parts – 1) SQL scripts to create the database, 2) the load_ib.exe module to populate the database with sample data, and 3) the tpcc.exe module to run user processes simultaneously in order to simulate the OLTP workload.

All source code for these tools is open and can be downloaded from <http://ibdeveloper.com/tests/tpc-c/>, so you can compile and check its workings.

How the test is conducted

1) Database creation. The test database is always created from scratch using SQL scripts. We used page size 4096 for all databases. 4096 was chosen because our previous tests¹ did not show significant difference between a 4 Kb and an 8 Kb page size, and because 4096 is the default page size for the modern server versions of InterBase and Firebird.

2) Populating the database. The module load_ib.exe populates the test database with sample values. Look below in the "How to configure tests" section to see how to set the database size.

3) Index creation. All indices are created after data are loaded into the database, not just to speed up data loading but also to avoid spoiling the selectivity of indices and thus forestall optimizer mistakes.

4) Testing. Testing is performed by tpcc.exe module which runs N simultaneous threads with user queries to simulate real-world OLTP- activity. Please notice that all actions are done by stored procedures, so the client (tpcc.exe) consumes very little CPU and RAM resources. Tpcc.exe is run on the same host as the database server. It can be run remotely, but it requires very wide bandwidth: during the course of testing our attempt with a regular 100Mbps network produced a lot network errors (10054), so I suppose a gigabit network is called for.

5) Database checking. After the test is complete, the "gfix -v -full" command line tool is run to check database state after intensive work and ensure it is correct.

6) Database statistics. Then we run "gstat -r" to gather database statistics with record versions.

Test duration (testing only, without loading, checking and statistics time) is usually three hours – 0.5 hours for startup and a 2.5-hour period of measurement. Where we increased times we note it specifically.

All test results are written to appropriate logs into "Log" folder.

Raw results

Test logs can be downloaded from <http://ibdeveloper.com/tests/tpc-c/>

How to run tests

It might seem a difficult thing to get these tests set up to run but that's not the case at all. Thanks to a useful toolkit, all you need to do to run the test is change the location of the appropriate server version in the setup_VERNN files, set the desired database size and user count in prepare.cmd and then launch RUN.CMD.

How to configure the test

Let's consider a simple example of configuring the test. Assume that you have InterBase 7.5 installed in folder "C:\InterBase751", so you need to change the first line in configuration file "setup_ib751.cmd"

¹ Probably we will publish their results in the next issue

By Alexey Kovyazin,
editor@ibdeveloper.com



```

SET IB=C:\interbase751\bin
SET NAME=IB751
SET DATABASE=%~dp0%NAME%_tpcc.fdb
SET ISC_USER=SYSDBA
SET ISC_PASSWORD=masterkey
SET PATH_SAVE=%PATH%
SET PATH=%IB%;%PATH%
SET BUFFERS = 99999
%IB%\instreg install %IB%\.. instance gds_db
%IB%\instsvc install %IB%\.. instance gds_db
net start IBS_gds_db

```

Please note that database buffers are set in setup_VERNN too.

Then you need to select the desired database size and user count. It involves changing two lines in the prepare.cmd file.

Database size is set by -W variable (W means warehouses) at the database creation step. Each warehouse adds ~30Mb to database, so -W50 means to create 4.5Gb database:

```
bin\load_ib -W50 -D\\.\%DATABASE% > log\%NAME%_load.log
```

User count and number of actually used terminals are set up at the testing step:

```
bin\tpcc -W50 -T50 -Dlocalhost:%DATABASE% -r30 -R180 -i30 >
log\%name%_tpcc.log
```

Please note that we always set W= T. It's done to avoid annoying lock conflict errors in the logs. In fact, lock conflicts are perfectly normal in real-world OLTP systems, but we're trying to avoid exceptions for the testing because they pollute the test logs and make them difficult to analyse.

Once all the locations in the setup_VERNN.cmd files are set up, the test – RUN.CMD – is ready to launch.

Test participants

Which InterBase and Firebird versions were tested? We tested the following server versions:

Server version	Notes
InterBase 7.5.1	All configurations
Firebird 1.5 SuperServer	Omitted in some configurations due to the lack of time
Firebird 1.5 Classic	All configurations
Firebird 2 Release Candidate 1 SuperServer	Omitted in some configurations due to the lack of time
Firebird 2 Release Candidate 1 Classic	All configurations
Yaffil Classic	Omitted in some configurations due to the lack of time
Yaffil SuperServer	Omitted in some configurations due to the lack of time

Because a lot of people are interested in the Vulcan results, we tried to run the public pre-alpha Vulcan of Vulcan. Unfortunately it was too unstable to run and crashed every time, so we had to omit it. We still hope there will be an alpha version before too long that will be stable enough to sustain our test.

Test results

We ran the tests many times in different software and hardware configurations and received a lot of raw results that are not yet fully processed. Space in this issue of magazine is limited too, so we decided to publish a special issue of "The InterBase and Firebird Developer Magazine" devoted solely to test results analysis and

Expert's note

Yaffil

Yaffil is a Russian clone of Firebird 1.0. It was made in the middle of 2001 to check where optimizations can be done, and how it will affect server performance. Since Yaffil was made only for Windows, there were a lot of windows-related optimizations. The whole list of optimizations is very long – new memory manager, faster inserts, disk i/o, additional configuration parameters, optimizer fixes, query scheduler, garbage collection, compatibility with previous InterBase versions (5.x and lower), and more. Also some little parts of Firebird 1.0 code were rewritten in pure assembler.

Thus, some users that moved from InterBase 4.x and 5.x to Yaffil, got not only better compatibility with legacy databases than InterBase 6.0 and Firebird had, but from 2 to 6 times speedup of common server performance. Optimization of the internal server scheduler still allows Yaffil SuperServer to have less processor load (from 5 to 40 percent) than Firebird SuperServer on some applications (with short and long-running queries being executed simultaneously).

Yaffil was the pioneer in reconstructing Classic for Windows functionality, because Borland nearly dumped Classic architecture before version 6.0, and the published source code for InterBase Classic for Windows was broken. Yaffil successfully revived Classic architecture for Windows (with services api support), having high stability on ~40 gigabytes databases nearly year before Firebird Classic for Windows was released.

Our expert

Dmitri Kuzmenko

Dmitri Kuzmenko is the chief expert of "The InterBase and Firebird Developer Magazine" with 20 years of InterBase and Firebird experience

e-mail: kdv@ib-aid.com

performance optimization.

Here we will highlight the main trends and commonalities in the results to illustrate the general picture.

Configuration 1

Hardware configuration:

Hardware:

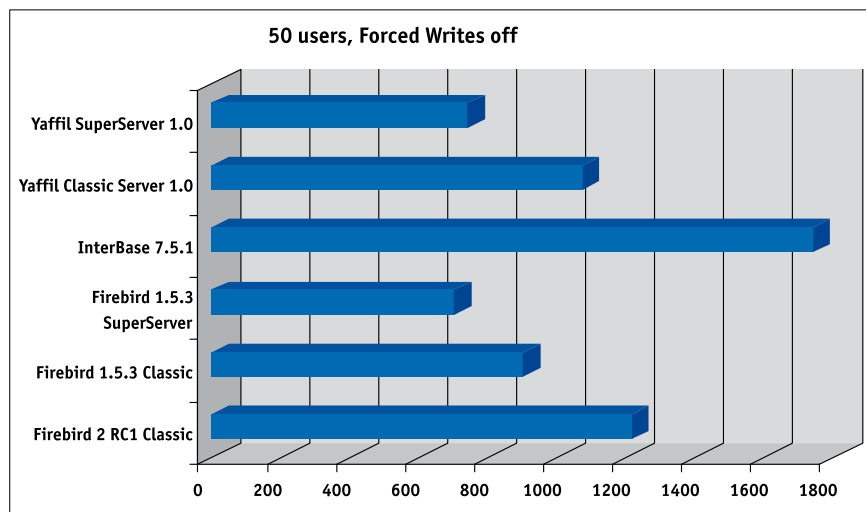
2xCPU 3.0Ghz, 2GB RAM,

2x36Gb Ultra SCSI 320 in RAID1+0 with 128 RAM,

Windows 2003.

50 users, Forced Writes off

Server	Performance, Tpmc	Buffers
Firebird 2 RC1 Classic	1218.29	2048
Firebird 1.5.3 Classic	905.14	2048
Firebird 1.5.3 SuperServer	704.02	9900
InterBase 7.5.1	1746.16	99999
Yaffil Classic Server 1.0	1075.08	2048
Yaffil SuperServer 1.0	743.86	9900



This was the most balanced hardware configuration – a very good disk system is combined with good CPUs and a lot of RAM. The database size for 50 users is 4.5Gb would not fit into RAM, so disk performance was high in importance.

We set Forced writes off to get maximum performance. It's the default mode in InterBase 7.5.1. For Firebird, in addition to setting Forced Writes off, we set the parameters MaxUnflushedWrites and MaxUnflushedWriteTime were to -1 in firebird.conf to disable flushing.

So, InterBase 7.5.1 is winner here, with Firebird 2 Classic Release Candidate 1 showing its enhanced capability, compared to 1.5.3, to work with disks.

Firebird 1.5.3 and Yaffil 1.0 SuperServer versions showed lower performance because, in this test, dual CPUs played an important role – fast disks and a lot of RAM being able to keep the CPUs busy.

100 users, Forced Writes On

After that, simulated a heavily loaded system. We increased the workload and ran the test with 100 users and 100 terminals, so the database became 9.5Gb. We also set Forced Writes on and increased test time to 6 hours.

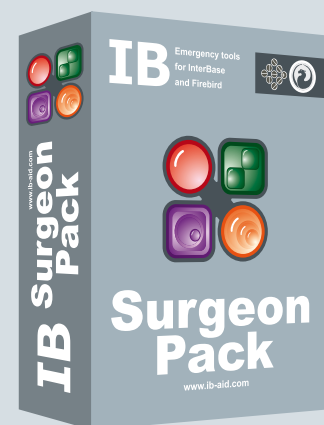


Surgeon Pack

Emergency tools for InterBase and Firebird

IBSurgeon Pack is a comprehensive set of tools to repair InterBase and Firebird databases and backups, undelete occasionally deleted records and foresee database problems.

Supports InterBase 5.x7.x and Firebird 1.x2.x databases.

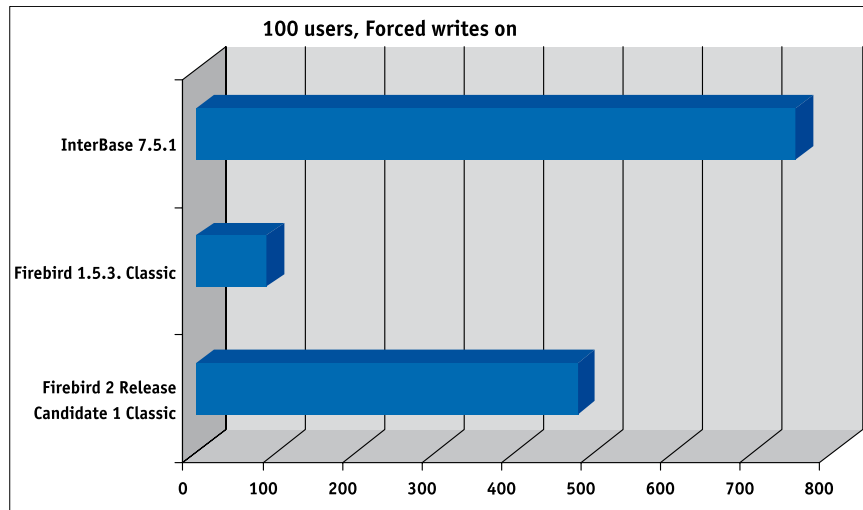


www.ib-aid.com

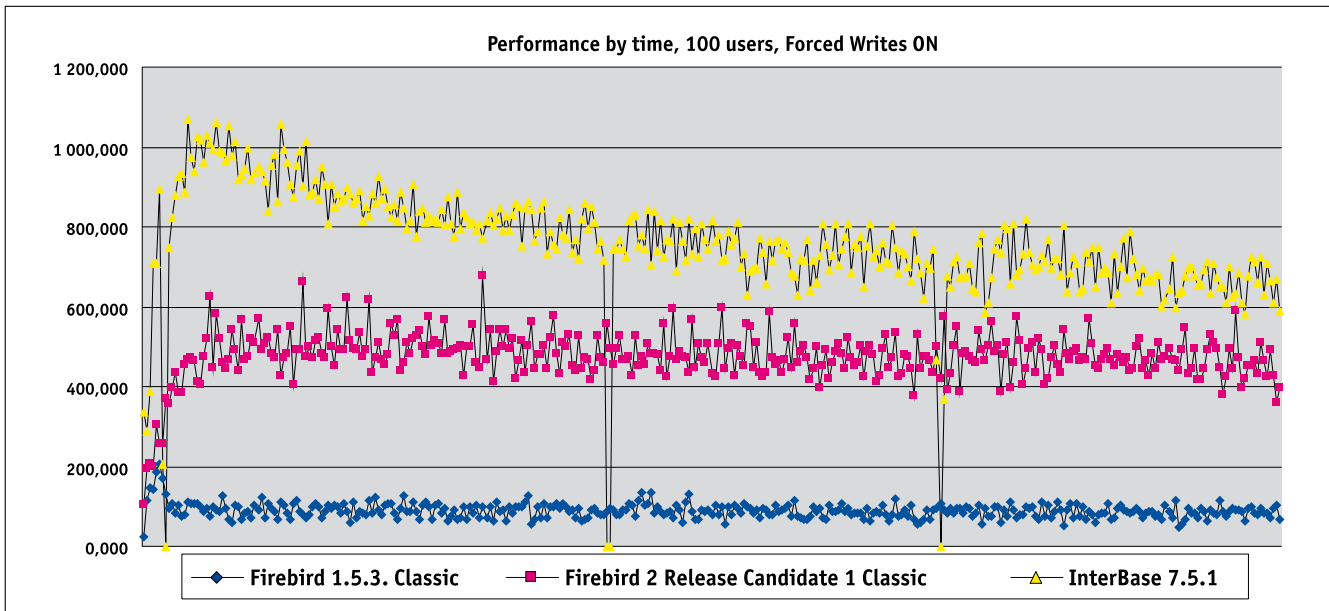
We launched only 3 servers to compare the maximum workload of InterBase SMP and the Firebird 1.5.3 and Firebird 2 Release Candidate 1 Classic architectures. Firebird 1.5.3 SuperServer, Firebird 2 Release Candidate 1 SuperServer and also Yaffil were omitted because of time constraints. Below are results:

Server	tpmC	Buffers
FFirebird 2 Release Candidate 1 Classic	479.95	2048
Firebird 1.5.3. Classic	88.43	2048
InterBase 7.5.1	754.76	1000000

Below is the results graph. As you can see, Firebird 1.5.3 showed the lowest performance, and InterBase 7.5.1 is the winner here.



But there are some circumstances which are rather important to mention. The activity of InterBase 7.5.1 was rather unstable and performance tended to degrade – please see graph below.



First, InterBase 7.5.1 had a lot of deadlocks. As I commented earlier, it's rather normal, but InterBase didn't exhibit them in the test with the lower user count. More importantly, Firebird 2 Release Candidate 1 and Firebird 1.5.3 exhibited no deadlocks at all, even with 100 users.

The second point is InterBase's tendency to decrease in performance, with no obvious explanation. This test cycle (all three servers) required more than 24 hours to complete, which is why we repeated it only twice. We got the same results (+/- 1%) on the repeat runs. It would be interesting to run InterBase alone for 2-3 days and see what happens.

Tips

Do you know how many transactions your application starts every day?

You can check it by running `gstat -h db.gdb` every day, but IBAnalyst will show daily average transactions count on the fly. System with 50-70 users usually starts about 50-500 thousands transactions per day.

Is it much, or not?

It depends on how you have written your application, i.e. how you manage transactions within application.

There may be not many transactions per day, but if these transactions are long, they can cause lot of record versions in database, and performance may degrade (as usual).

On the other side, even 1-5 million short transactions per day may not load server much.



FIBPlus components

Direct access to all InterBase and Firebird features in Delphi, C++ Builder and Kylix applications!



- No middleware
- Easy porting from IBX
- Improved performance and optimized network traffic

MultiProfile tool

Launch Delphi, C++ Builder and C# Builder with different IDE settings for separate projects. With MultiProfile you can install different versions of the same components simultaneously without any conflicts!

See online flash-demo!

Special offer for IBDeveloper readers:

Two products for one price!

Visit ibd.devrace.com and buy FIBPlus and MultiProfile for 235 Euro!

The third instability we saw with InterBase was that InterBase, three times, went to zero performance for 30-60 seconds – just stopped to answer to client requests. The yellow vertical lines on the graph were where these trouble spots occurred.

I'm guessing that, as a result of these stops, database checking (gfix -v -full) indicated 1 record error:

Summary of validation errors

Number of record level errors : 1

Configuration 2

by Daniel Magin, daniel@pauer-magin.de

Hardware: Pentium 3.0, HT ON, 2Gb/1Gb, SATA 300Gb

Many of our customers run InterBase on standard workgroup servers so I was interested to analyse the results of testing the different server versions on a host machine that was specifically not a high-speed system with multiple CPUs. My base test platform was a simple machine with a Pentium 4 3.0 GHz processor with Hyperthreading and 2 Gb RAM on an Asus P4P800-E mainboard. The hard drive was a 300 Gb Maxtor SATA2 V300FO with a 16 Mb Cache but it was running on the mainboard's SATA1 interface.

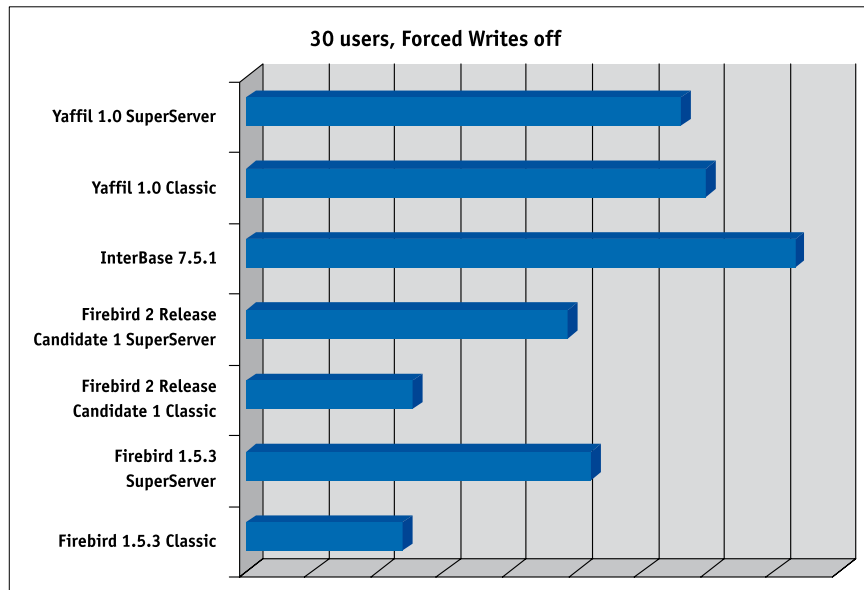
On the only existing partition C (90GB) I installed a new Windows 2003 Server Enterprise Edition. All tests ran on this single partition. Though there's no question that results could be sped up by spending more money on SCSI-RAID and more RAM, the focus of my interest for this test was how this fairly ordinary configuration would respond.

I used the latest official version of each of the different database engines. I started out by playing around quite intensively with some settings to find the best configuration for each one for speed. I settled on the following buffer settings:

Server version	Buffers
Firebird 1.5.3 Classic	2048
Firebird 1.5.3 SuperServer	9000
Firebird 2 Release Candidate 1 Classic	2048
Firebird 2 Release Candidate 1 SuperServer	100000
InterBase 7.5.1	100000
Yaffil Classic	2048e
Yaffil SuperServer	9000

I ran Alexey's complete TPC-C Test for many hours. The test system was a warehousing system with 30 terminals, with a starting database size of around 2.8 Gb. The 30 terminals would represent, for example, a web application with between 500 and 1000 users on-line.

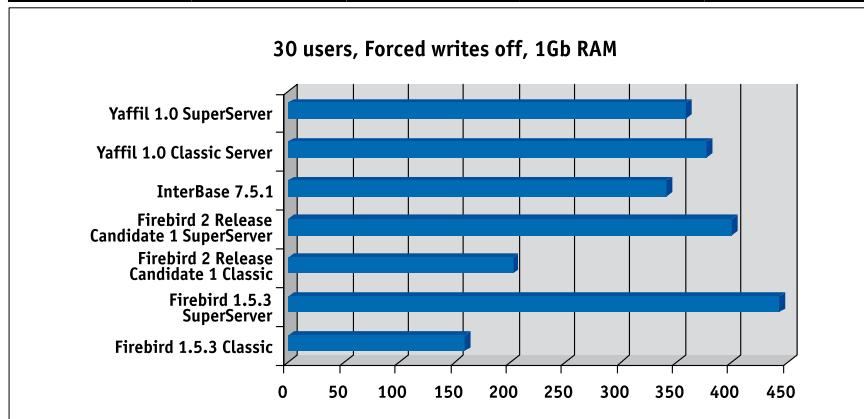
Server	tpmC	Loadingtime	RunningTime	Transactions	Deadlocks
Firebird 1.5.3 Classic	238,83	872,59	10800	82422	0
Firebird 1.5.3 SuperServer	522,55	815,67	10800	180251	3
Firebird2 Classic	252,85	951,7	10800	87234	0
Firebird 2 SuperServer	488,77	944,891	10800	168615	1
InterBase 7.5.1	832,64	1010,57	10800	287260	1
Yaffil Classic	697,77	768,531	10800	234516	0
Yaffil SuperServer	659,77	765,53	10800	227630	0



InterBase, current Version 7.5.1, was the winner in my TPC-C based test, except it came last in loading the datascrip. I don't know why. Most significant is the extremely good speed InterBase delivers if it can find 2 Gb of RAM.

With only 1Gb of RAM, InterBase 7.5.1 and Firebird 2 Release Candidate 1 Classic delivered similar speeds, and Firebird 1.5.3 SuperServer became the leader:

Server	tpmC	Loadingtime	Transactions	Set Buffers
Firebird 1.5.3 Classic	159.18	1475.36	54900	2048
Firebird 1.5.3 SuperServer	443.11	825.65	152889	9900
Firebird 2 Classic	202.88	951.53	69977	2048
Firebird 2 SuperServer	400.46	940.68	138213	6400
InterBase 7.5.1	341.47	1151.56	117820	6400
Yaffil Classic	377.3	776.18	130212	2048
Yaffil SuperServer	359.09	783.26	121807	9900



So if you use InterBase, having the full 2 Gb of RAM available will be rewarding. Yaffil—which is based on Firebird 1.0 but has a better memory manager—also showed itself very good at handling the bigger memory. If you use open source versions and you have at least 2 Gb RAM, Yaffil would be worth a try if you need more speed.

News line

Devrace has acquired Metadataforge

Devrace has recently acquired Metadataforge company (SQLHammer project) and will immediately start selling the product.

The CEO of Devrace Serg Vostrikov, says: "We are very happy to have signed the contract with Dmitry Kovalenko, the author of SQLHammer. SQL Hammer is a simple, fast and elegant IDE for database development and invaluable tool for real administrators. When developing SQL Hammer Dmitry was concentrated on improving the performance of access to databases, and so far judging by recent tests these attempts are very successful. I am sure soon we will prove InterBase/Firebird database developers that SQL Hammer is worth buying!"

Sophisticated internal SQLHammer architecture is based on plug-in modules which extend IDE functionality. For example centralized SQL Editor is used in most database objects and editors, whereas it is physically present only in a single module. Such a module architecture provides developers with additional advantages:

- saving computer resources,
- an ability to use only necessary functions and deactivate the other tools,
- full support of all Borland InterBase/Firebird versions, as we use different access means for each server,
- better support of client development tools, in particular, SQLHammer has SQLMonitor for FIBPlus and SQLMonitor for IBX,
- more convenient updates: developers get an update of a certain module or of the product core instead of full package at once.

www.devrace.com



IB FirstAID

Repairing InterBase/Firebird databases

IBFirstAID is a tool that can be used for automatically diagnosing and repairing corrupted Firebird or InterBase databases.

It can fix up to 80% of often corruptions.

Supports InterBase 5.x7.x and Firebird 1.x2.x databases.



www.ib-aid.com

Hyperthreading

The next point for analysis was hyperthreading. Does HT really speed up the system? Which versions actually work with HT?

I decided to increase the work for the server by changing the terminal/warehouse setting to 50.

Bigger DataBase + More Thread Connections:=more work for the server :-)

The following results are just for comparing the differences between having HT enabled and having it disabled. With a high-speed hard disk system such as SCSI, you could vastly expand the Delta because the CPUs would not be kept waiting for read/write operations.

Server	HT ON	HT OFF	Delta
Firebird 1.5	311,62	301,23	103,45%
Firebird 2	323,08	353,25	91,46%
InterBase 7.5	442,82	414,52	106,83%

The Firebird 2 results are interesting: enabling HT actually causes the TPC-C values to decrease, apparently with a negative impact that is higher than the gains shown by using HT with InterBase, Firebird 1.5 and Yaffil.

Daniel Magin has more than 18 years of experience in a variety of hosting, multi-tier, and client/server projects. He is a frequent tutorial speaker in Germany and is a certified trainer for Borland Delphi, InterBase and Microsoft.net. He is also employed by Better-Office. Daniel has spoken at multiple international conferences in the USA, Europe and the Arab Emirates on object-oriented programming, Java, Delphi, InterBase, workflow management, application servers, and database and application design.

Customer satisfaction is our top priority! Integrating the Enterprise!



better office is a German company specialized in Borland and Microsoft development environments

With offices in Oldenburg (head office), Frankfurt, Berlin, Pennsylvania (USA) and a Team of appr. 25 associates.

better office provide development tools, custom made software development, software consultancy,

Project management, training and support.

They have extensive experience in developing client-server and web based database systems.

Benefits:

Borland Delphi WIN32 and .Net
Microsoft .Net Framework
(C# Visual Studio)
JAVA JBuilder and Eclipse
Borland InterBase
Microsoft SQL-Server
IBM DB/2/40 (iSeries)

Head Office: Oldenburg
Stau 19 – 6.0G -
26122 Oldenburg

Tel.: +49 (0)441 926740

Fax: +49 (0)441 2488675

E-Mail: info@better-office.com

Configuration 3

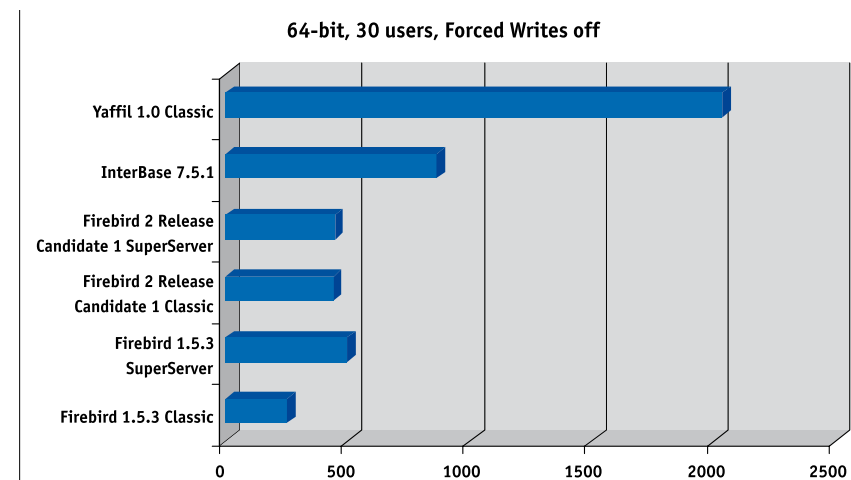
Hardware: Fujitsu-Siemens 200TX S2, 1xXeon 3.2Ghz 64EM-T, HT OFF, 2Gb RAM, 6x36Gb UltraSCSI320, RAID 1+0, Windows 2003 64 bit Edition

30 users, Forced Writes off

This test was interesting because of the abnormal speed of Yaffil Classic Server – it was more than twice as fast as InterBase 7.5. We believe the reason for it is Windows-optimized code in Yaffil (as you probably know, Yaffil is for Windows platform only) and its spinlock-based lock manager. Critical low-level operations are implemented in Yaffil in Assembler. Yaffil's memory manager also differs from both the InterBase and Firebird memory managers.

64-bit based CPUs and OS are coming soon, so it may be useful for InterBase and Firebird engineers to take a look at the Yaffil implementation.

Server	Performance, Tpmc	Buffers
Firebird 1.5.3 Classic	252.170	2048
Firebird 1.5.3 SuperServer	497.010	9000
Firebird 2 Release Candidate 1 SuperServer	448.33	9000
Firebird 2 Release Candidate 1 Classic	440.400	2048
InterBase 7.5.1	863.760	100000
Yaffil Server 1.0 Classic	2,038.160	2048



Yaffil 1.0 SuperServer was omitted due to a service startup problem that we didn't notice until it was too late to fix it. The hardware was no longer available to repeat the test.

Servers results overview

InterBase 7.5.1

Currently it's the most powerful server for the most powerful SMP hardware configurations. It has some problems with stability and is sensitive to the amount of installed RAM. And we should not forget about TPC-R results which showed that InterBase 7.5.1 required a lot of query optimizations in real-world applications. But, if SQL developers are optimizing queries effectively, the two tests indicate there is a performance advantage for InterBase from investing in multi-CPU computers with fast disk systems.

Firebird 1.5.3

It's an old horse, but it's not too bad! For single CPU computers SuperServer does

Expert's note

64-bit

64-bit processors now are nearly everywhere, from desktop to server. What benefit could be gained from these processors?

The initial win is for gamers, especially those who already have AMD 64 processors in their systems. The next comes when 64-bit operating systems come into currency. Windows XP Professional and Windows 2003 Server 64-bit editions are available already but the big problem for both is drivers and software.

Before installing these operating systems you need to be sure that stable drivers exist for your hardware. If you will use only 32-bit programs, performance will be the same as for the 32-bit operating system (except strange Yaffil results in tpc-c test), and could be worse (especially in games, up to -40%).

For now, it doesn't make a lot of sense to install 64-bit operating system unless you have a specific need to do so and the software and driver support to make it work. Nevertheless, be prepared to install it in the not-too-far-distant future.



Our expert



Dmitri Kuzmenko is the chief expert of "The InterBase and Firebird Developer Magazine" with 20 years of InterBase and Firebird experience

e-mail: kdv@ib-aid.com

quite well. On a 64-bit computer with Windows 2003 it does even better than Firebird 2 Release Candidate 1. Firebird 1.5.3 looks good for small to medium size databases and user bases. And we should not overlook its very high stability.

Firebird 2 Release Candidate 1

It's not good to test beta versions, because they change so quickly. For example, Firebird 2 Release Candidate 2 is available already and we'll certainly test it and publish the results in the next issue. But we have to admit that, for a beta, Release Candidate 1 is doing very well. Its performance is only 30-40% lower than InterBase 7.5.1 running on powerful SMP configurations with a lot of RAM installed (2Gb+). On more modest hardware configurations it is even better, and also demonstrated very good stability. The forthcoming release of Firebird 2 looks likely to be more than a match for InterBase. We intend to pay a lot of attention to its evolution.

Yaffil 1.0

And about our dark horse. I suppose many of our readers first got to know of the Yaffil project in this article. Firebird 2 incorporated quite a lot of Yaffil's functionality (mostly System Defined Functions and some improvements), but there are still many interesting things, especially in speed issues and heavy-load user management. Yaffil 1.0 is rather common in Russia – it is used for quick replacement for old InterBase 4.x and InterBase 5.x databases due its special backward capabilities and high speed. With single backup and restore and without any changes in application code users get faster and more stable databases. Of course, Yaffil isn't in line for new development, but there are a lot of ideas inside it to explore.

Summary

I think many readers (especially fans of Firebird or InterBase) will be dissatisfied with this article.... It's not possible to determine an absolute leader and too many things depend on hardware, operating system and configuration parameters that we haven't specified here.

It was a real discovery for us that performance of modern InterBase and Firebird versions can vary so much. As we discovered, performance is not a simple thing.

It's hard to describe all the testing we did in a single article. We'll be having a special issue of our magazine devoted exclusively to the results and their interpretation. It will cover configuration aspects like playing with firebird.conf and IBCONFIG parameters, the impact of new settings like GROUP COMMIT on performance, comparing tables of results for different settings, detailed explanation of the interdependencies between elements of hardware configuration.

We hope that some readers will be able to repeat the tests using our toolkit and send us the results for publication.



INTERBASE/FIREBIRD DEVELOPMENT STUDIO

is a power-packed, fully-loaded solution that most database programmers and administrators can only dream about - at a price you can afford.

All stages of your database development, from design to deployment to maintenance, are easy, flexible, and powerful.

With Interbase/Firebird Development Studio, you get the most powerful tools for the first time ever - dynamic errors highlighting and refactoring of SQL code.

Refactoring allows you to quickly and easily change the project structure to accommodate new features, requirements, or external systems.

And, at last, you can now develop error-free code quickly and easily - when code errors highlight dynamically.

Our exclusive Database Designer presents the next step in the evolution of database development.



IBADMIN 4 FOR LINUX

SuSE LINUX Inc.
THE LINUX EXPERTS

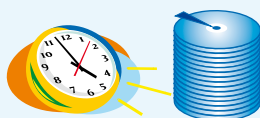


Simple but well packaged solution for managing Interbase/Firebird databases under Linux.

Aimed with deployment support features like Database Comparer, SQL Debugger and more.

Full support for Server Management is also available.

TIME TO BACKUP

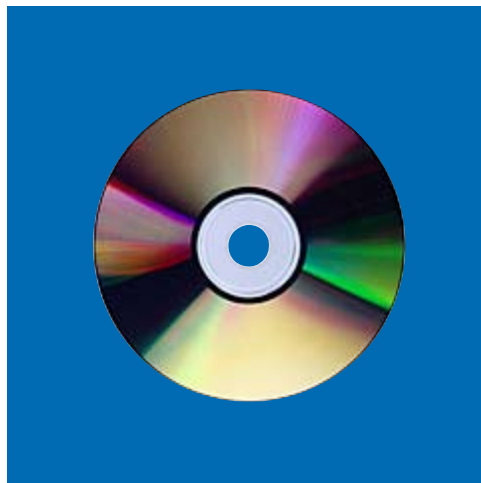


"Time to backup" is a suite of programs that allow to set up a scheduled backup of Interbase or Firebird databases on multiple hosts.

Scheduler services can be managed remotely from Windows or Linux machine.

WWW.SOLLY.COM

Buy Paper Versions



**Buy electronic version now
and get instant access
to full PDF!**