Why Not MySQL?

by Ben Adida, part of the OpenACS Project.

NOTE: This Document was written in May 2000. Thus, it is outdated and does not represent the latest data concerning MySQL. I will attempt to find time to rewrite this with more current information soon (August 10th, 2001)

It happens almost every week, and sometimes more often: someone wants to know why we're not using MySQL as the RDBMS behind OpenACS. The ACS Classic team (ArsDigita) gets the same question over and over again on their Discussion Forums. If it's good enough for Slashdot, it must be good enough for OpenACS, right?

Wrong. This quick paper attempts to explain why MySQL is not just the wrong choice for OpenACS, but how it should never be used by any system that handles critical data.

The Purpose of an RDBMS

An RDBMS exists for the purpose of providing a reliable permanent storage mechanism with very strict properties embodied in the ACID test. I will quote directly from Philip Greenspun's great explanation

Atomicity

Results of a transaction's execution are either all committed or all rolled back. All changes take effect, or none do. Suppose that a user is editing a comment. A Web script tells the database to "copy the old comment value to an audit table and update the live table with the new text". If the hard drive fills up after the copy but before the update, the audit table insertion will be rolled back.

Consistency

The database is transformed from one valid state to another valid state. A transaction is legal only if it obeys user-defined integrity constraints. Illegal transactions aren't allowed and, if an integrity constraint can't be satisfied the transaction is rolled back. For example, suppose that you define a rule that postings in a discussion forum table must be tied to a valid user ID. Then you hire Joe Novice to write some admin pages. Joe writes a delete-user page that doesn't bother to check whether or not the deletion will result in an orphaned discussion forum posting. Oracle will check, though, and abort any transaction that would result in you having a discussion forum posting by a deleted user.

Isolation

The results of a transaction are invisible to other transactions until the transaction is complete. For example, suppose you have a page to show new users and their photographs. This page is coded in reliance on the publisher's directive that there will be a mugshot for every user and will present a broken image if there is not. Jane Newuser is registering at your site at the same time that Bill Olduser is viewing the new user page. The script processing Jane's registration does inserts into several tables: users, mugshots, users_demographics. This may take some time if Jane's mugshot is large. If Bill's query starts before Jane's transaction commits, Bill won't see Jane at all on his new-users page, even if Jane's insertion into some of the tables is complete.

Durability

Once committed (completed), the results of a transaction are permanent and survive future system and media failures. Suppose your ecommerce system inserts an order from a customer into a database table and then instructs CyberCash to bill...
the customer $500. A millisecond later, before your server has heard back from CyberCash, someone trips over the machine's power cord. Oracle will not have forgotten about the new order. Furthermore, if a programmer spills coffee into a disk drive, it will be possible to install a new disk and recover the transactions up to the coffee spill, showing that you tried to bill someone for $500 and still aren't sure what happened over at CyberCash.

If what you want is raw, fast storage, use a filesystem. If you want to share it among multiple boxes, use NFS. If you want simple reliability against simplistic failure, use mirroring. Want a SQL interface to it all? Use MySQL.

Now, if what you want is data storage that guarantees a certain number of invariants in your data set, that allows for complex operations on this data without ever violating those constraints, that isolates simultaneous users from each other's partial work, and that recovers smoothly from just about any kind of failure, then get your self a real RDBMS. Yes, it will be slower than the MySQL file system. Just like TCP is slower than UDP, while providing better service guarantees.

The State and Future of MySQL

Building a true RDBMS is a tough problem, probably tougher than any other systems issue. Most products on the market (Oracle, Sybase, PostgreSQL, Interbase) have been in development for years, sometimes more than 10 or 15.

MySQL claims that they have compromised on certain features to guarantee better performance. While this may be an interesting way to track non-critical data such as clickthrough tracking, compromising on perfect data integrity is not acceptable, even for speed, when dealing with critical data.

The OpenACS team is happy to take a closer look at MySQL as it matures. However, it doesn't seem that the MySQL team understands the concepts and importance of true ACID capabilities: The MySQL Todo mentions "transactions" in a long list that includes questions such as "do sleeping threads take CPU." Furthermore, the MySQL manual claims that MySQL will soon implement "atomic operations" through the use of table locks, but without rollback. This is a blatant misuse of the term "atomic," which implies that either none or all operations will complete. A hardware or power failure in the middle of a set of statements will break the atomicity of the block if there is no rollback capability.

Rollback is not just a convenient feature, it is a critical basis for solid data storage.

There are very good reasons for using MySQL. A need for a reliable, ACID-compliant datastore isn't one of them.

A Few More Details

- **MySQL has no subqueries.**
  Instead of performing one complex query that is entirely processed on the database end, MySQL users have to perform 2 or more serial queries that each must go over inter-process or network communication between the app and the database. This significantly reduces the speed advantages of MySQL.

- **MySQL has no stored procedures.**
  If a series of DB actions need to be performed in a block, MySQL requires each SQL statement to be sent from the app, again in a serial manner, again over IPC or network.

- **MySQL has no triggers or foreign key constraints.**
  Data invariants must be maintained by application-level code, which requires building

carefully-planned abstractions to guarantee integrity (for every means of accessing your DB),
and even more unnecessary back-and-forth communication between the app and the database.
• **MySQL only has table-level locking.**
Only one user can write to a table at the same time. For web usage, that falls under the
category of "pathetic."

**The Bottom Line**

An enterprise-level system will never compromise certain features for speed. The ACID properties of
an RDBMS are an absolute necessity for any critical data. Critical web sites that run on non-ACID-
compliant systems are asking for trouble.

The OpenACS project refuses to break with the important principles of the ACID test. We are out to
build an enterprise-level Open Source web toolkit. PostgreSQL and soon InterBase are appropriate
RDBMS candidates for this project. MySQL is just a glorified filesystem with a SQL interface.

---

**Reader's Comments**

Richard,

Note that my "diatribe" never mentioned that MySQL is inherently a bad piece of
software. Instead, I claim that:

• MySQL calls itself a database and compares itself with PostgreSQL, Oracle, etc..
in one of the most misleading comparison charts in the industry. That is
tremendously irresponsible, and shows either MySQL's dishonesty, or complete
misunderstanding of the importance of real transactional systems.
• Transactions are critically important! OpenACS, like any serious web toolkit,
needs them. I strongly disagree with your claim that 99% of web sites would
function happily without. This is an aberration brought about by the excessive use
of MySQL, and the logically-unsound conclusion that MySQL must therefore be
adequate for the job. Read Philip Greenspun's stuff to find out exactly why.
• My subselect, trigger, and stored procedure discussion about the number of
queries performed is extremely relevant. Network and IPC communication are
slow. Doing 4 or 5 such data transfers instead of 1 is tremendous overhead.
• At the end of the day, simply reading the MySQL manual and their discussion of
transactions, foreign key constraints, and other RDBMS features shows a deep
lack of understanding of these necessities.

What's truly appalling is that the Open Source community is usually very critical of
certain things, like OS's that crash. Somehow, MySQL gets a free pass (and it's not even
truly Open Source!). The purpose of the document is to set the record straight:

MySQL claims to be an RDBMS when it is far from deserving that title.

-- Ben Adida, May 4, 2000

You've missed the point about MySQL. Go read the documentation more thoroughly. I
had a long discussion with David Axmark last year after Monty's presentation at the
O'Reilly open source conference about exactly this issue. Transactions (i.e., SQL
COMMIT/ROLLBACK) are not necessary in probably 95% of all production databases.
And after all, before COMMIT/ROLLBACK appeared in commercial products starting around 1993 we seemed to survive OK without it somehow.

On the other hand, in 5% of applications SQL transaction capability is *essential*. I would certainly stipulate to that.

MySQL is not designed for "enterprise-level" situations and nobody is claiming that turf for it. Instead, we have a situation where Oracle is downselling into markets it has no business being in. Since when do you really have to spend $20,000 for licensing and hardware to run a basic level of Oracle when MySQL or a number of other competitors are equally capable for the job?

The real issue here, I think, is Larry Ellison's greed. Pure and simple.

I would certainly use Oracle, DB/2 and maybe SQL Server in enterprise-critical applications, or when critical financial or security data requiring the additional reliability of COMMIT/ROLLBACK is needed.

But why put money in Larry's pocket in cases where Oracle adds no incremental value?

Then there is the issue of what COMMIT/ROLLBACK does to a database server internally. An old friend of mine who's a longtime Oracle engineer described that part of their code to me as "hairy."

Finally, ACID isn't everything. The Ars Digita people ought to know this. Database design is just as important. As could be seen by the mess that was made with the EDF Action Network design.

The chest-beating by the Oracle crowd is a little bit embarrassing. It's a good product, I have no problem with that. It is not the right product for many people who are being stampeded into using it by Oracle's FUD marketing, much of which focuses on the "transactions" issue.

-- Fred Heutte, May 4, 2000

It seems there are going to be a lot of comments by people who believe that I "just don't get it." I urge you to read my document and follow-on post more carefully. I very clearly state that MySQL overstates what it can do, that it's a fine file-system, just not an RDBMS. So yes, it has its uses, just not the ones it advertises.

The one addtional point I will make concerns your repeated mention of Oracle. Click around the OpenACS.org site a bit, you'll see that we do *not* use Oracle. In fact, we use PostgreSQL. It's truly Open Source (which MySQL is not), and it's ACID compliant (which MySQL is not).

-- Ben Adida, May 4, 2000

Fred, you seem to miss the point. We're not suggesting people abandon MySQL and spend $20,000 on an Oracle license. We're the people offering the ACS on PostgreSQL, after all. We're also considering porting the toolkit to InterBase, now available for no cost and soon to be Open Source. That means we're offering one, and in the future perhaps two, Open Source versions of the ACS that run on DBs that pass the ACID test (or "nearly pass" in the case of PG today, absolute ACIDity will come later this year).
In other words, we're NOT the "Oracle crowd". We're busting our butts to provide a good, Open Source web toolkit based on Open Source RDBMS systems. Don't confuse us with ArsDigita (though they are providing us with a server).

We're no more interested in buying Oracle than you are. We're just not willing to give up ACIDity. It is crucial for mission-critical data, and damned handy for the rest.

The MySQL folks DO claim the "enterprise turf" for their product, at least their written documents freely available on their web site do.

My major problem with the MySQL folks are that either they don't understand what ACID is about, in which case they should keep their ignorance private; or they do, and are lying to people, in which case they're simply unethical.

So, Fred, the issue isn't only Larry Ellison's greed but the MySQL folks dishonesty and/or ignorance.

You can pass the ACID test and keep your money in your pocket rather than Larry Ellison's, so Oracle pricing isn't really an issue unless you really need the incredible scalability of the product.

Of course database design is important. And, if you want to know a dirty little secret, parts of the ACS break atomicity due to a gross and pervasive programming error, making Oracle an expensive version of MySQL, as I put it when I reported the problem (it's being fixed, and if it doesn't get fixed right by aD we'll fix it ourselves).

So, a true RDMS can't guard against all possible programming errors.

On the other hand, with MySQL you can't get it right no matter what. The "solutions" they offer in their documentation simply don't work.

Transaction processing in financial institutions have been the norm for over twenty years. Commercial databases in use in this environment have long supported atomicity, despite your comment suggesting that the concepts have only been around since 1993. You're confusing the adoption of SQL with the deployment of transaction-based database systems.

-- Don Baccus, May 4, 2000

I think statements like "transactions are not necessary in 95% of production databases" is either based on a very narrow set of experiences, or a basic misunderstanding of the reasons for transactions in the first place.

Transaction (or rather, atomicity) should be a crucial component of any database activity. Certainly in the financial industry (where I work) databases have had to use atomic transactions since long before 1993. Of what use is a database in which an action can leave the data in an inconsistent state? I think we would all agree that we would prefer our banks to use ACID-compliant databases.

Certainly there may be cases in which transactions are not important, but as Ben notes then you are really talking about a filesystem and not a true database. Many of these sites could probably be built using flat text files, in fact.

But luckily, we are not forced into Oracle's high-priced family to get an ACID-
compliant RDBMS. PostgreSQL is an excellent product, and Interbase will probably be
great, too.

-- Brent Fulgham, May 4, 2000

While it's true that MySQL does not support transactions, and is not suitable for a
mission-critical system that requires transactions, I don't think these comments are
wholly warranted:

I don't think it's fair to claim that the MySQL team doesn't think transactions are
important simply because they are in an undifferentiated ToDo list with other items:
Everybody I know has ToDo lists with radically different levels of importance all
jumbled together.

Nor do I think they are being stupid to work towards transaction support one step at a
time by adding "atomicity without rollback" (which is an admitted oxymoron).

Finally, the discussion of subselects is a bit misleading: If MySQL's two queries are
faster than the single query with sub-select in a bloated (for the given need) database,
which tool should one use?

MySQL has carved out a very interesting niche: Raw speed and simple to
setup/use/maintain database backend for online applications. 99% of what most people
use (or should be using) an online database for would find MySQL perfectly acceptable.

All that said: PostgreSQL is an extremely viable almost-enterprise-class OpenSource
alternative anyway, as you note. It's a pleasure to see it grouped with Oracle, MS-SQL
and Sybase (which cost thousands of dollars just to play with, and 10s of thousands to
use in the Real World <tm>). Of course, PostgreSQL still lacks some of the more
esoteric features of those expensive, bloated, old beasts...

In the final analysis, one has to pick the right tool for the job; as is always the case.

If any data loss at all, no matter how small, is mission-critical, MySQL is the wrong
tool.

If raw speed at the expense of a tiny bit of insecurity about Pepsi Syndrome, people
tripping over server power cords, and nuclear war is acceptable MySQL is an extremely
attractive option.

I know almost nothing about what OpenACS is doing (I was referred from a forum on
http://www.zend.com), so can't say which tool is right. But if you're not using
transactions for anything important, this diatribe against MySQL is rather misplaced,
imnsho.

Disclaimer: I'm a PostgreSQL user, mostly, because my ISP needs transactions, so that's
what I have handy. I also work for Zend, so have inherent biases for, err, well, both
actually :-)

Oh yeah: Why not PostgreSQL? :-) :-) :-)

Whoops: I conflated the eCommerce package allegedly requiring AOLServer and Oracle
(and even that is by heresay) with the site using PostgreSQL. SORRY!!!
Richard, in regard to this comment:

"Nor do I think they are being stupid to work towards transaction support one step at a time by adding "atomicity without rollback" (which is an admitted oxymoron). "

You can't have atomicity without rollback, whether or not you call it that. You agree it's an oxymoron, why not take the next (small) step and admit they're being stupid? I spent part of yesterday poring over their docs, and they clearly don't understand that such statements are oxymoronic. I wasted far more time reading the docs than I should've because I kept looking for signs that they couldn't possibly be as clueless as they apparently are.

The strange thing about reading their docs is that though they don't get it, they're obviously are under pressure from some of their user base to implement ACIDity. A lot of the flavor of their commentary seems to be "we don't do this because it's hard", followed by "and besides, it's bad stuff". Read their section on why foreign key constraint checking is bad, for instance. It's a joke.

-- Don Baccus, May 4, 2000

Not to be a curmudgeon, but it seems to me that CICS has offered COMMIT/ROLLBACK since the 60s. Now, granted, it's not a RDBMS, but it's a viable system that is still used in much of the world (particularly financial systems). Saying that 'COMMIT/ROLLBACK was added in 1993' isn't telling the whole story--it may be that the RDBMS world took longer to figure this out...admittedly, they haven't been around _nearly_ as long as CICS, but they still should have had this ability from the start.

From the discussion of the weaknesses in mySQL, I think I'd rather use CICS/6000 than mySQL, and believe you me, I don't make that statement lightly. CICS/6000 is antiquated, aggravating, a pain-in-the-ass to configure, and not many people want to work with it. But at least it passes the ACID test...

-- Jeremy Anderson, May 5, 2000

In response to Don Baccus' comment:

MySQL has **NEVER** claimed to be for enterprise class applications. Read through the comments on this page and you'll find people talking about the need for transactions in financial applications. This is something even TcX agrees that MySQL is not appropriate for.

Much of the comments on this page are of an inflammatory nature. This puzzles me, especially for people with an academic interest like ArsDigita and the OpenACS. MySQL is a nice system for people who want a simple database to help them manage their data quickly. Nobody ever said it was for complex financial applications.

Comments like the following are the most offensive:

"You can't have atomicity without rollback, whether or not you call it that. You agree it's an oxymoron, why not take the next (small) step and admit they're being stupid? I spent part of yesterday poring over their docs, and they clearly don't understand that such
statements are oxymoronic. I wasted far more time reading the docs than I should've because I kept looking for signs that they couldn't possibly be as clueless as they apparently are."

Whoa, hold on there. Name calling? Come on, thats just as unprofessional as the "misleading table" you cited on MySQL's web site. Calling the developers clueless for building an in-house tool to solve problems they needed, and now adapting it to a more general market is just not fair. MySQL was created because TcX needed a FAST system to handle their large databases. Now they have recognized that large numbers of other users have the same needs; a FAST system. Do these users need ACID (which, if you re-read each of the four points with a cute buzzword-compliant name, is just saying the same thing about Transactions four times)? The answer is obviously no. Do web sites looking to provide a database interface to their news page need a slow, complicated transaction to add or delete a news story? To add or delete comments? The answer is again obviously no.

The following is a particularly egregious comment:

"The strange thing about reading their docs is that though they don't get it, they're obviously are under pressure from some of their user base to implement ACIDity. A lot of the flavor of their commentary seems to be "we don't do this because it's hard", followed by "and besides, it's bad stuff". Read their section on why foreign key constraint checking is bad, for instance. It's a joke."

Once again the harsh bitterness is puzzling. Saying that they are "obviously under pressure from their user base to implement ACIDity" is a little absurd. Try, they are slowly grafting transactions onto their system in a way that doesn't break their design goal: SPEED. They also have legitimate concerns about foreign keys; making it difficult (if not impossible) to move data between database servers is a problem, and one that I encountered several times before. MySQL does make this simple task simple.

So what's the real story? MySQL is a wonderful database for people who don't need all the complexity of the big commercial (or the big OpenSource) RDBMSes. It's wonderful because it's fast, it's simple, and it's inexpensive. Cut through the name-calling and the hype to realize that there is a good product underneath, despite Don's rhetoric.

-- Jimmy Sieben, May 5, 2000

When people talk about databases, they combine several things at once:

1. Interface language : SQL -- This feature is not necessary in a database. However, most relational databases have an SQL interface. However, due to brain-dead nature of SQL, one needs a host language. There lies the madness, which Java hopes to rise above.

Here MySQL is adequatre. If all you need to a good way of querying over a datastore, MYSQL is great.

2. Transactions: ACS, and OpenACS depend of transactions in the db. They sensibly avoid trying to implement in the application. Remember, it took years for these vendors to get transactions right! Trying to do transactions is worse than trying to memory allocation in programs by yourself! (The same reason why you would not want to program in C++)!
However there are a class of applications for which transactions may not mean much. Consider OLAP applications, where you do not have updates. Having or not having transaction properties mean little here. It is possible that MYSQL is a good choice here.

3. Data Modelling: Object/Relational etc. Of course, interface language and the choice of datamodel are interlinked. Still, there are ways to extend the beaten up old relational model. It is sad that the world has not moved beyond the relational model after all these years.

Here, Postgresql has some interesting features. So does Oracle 8i. And, MySQL is strictly relational (with some web friendly features thrown in).

If I have lot of money, I will support Postgresql monetarily and add full SQL92 compliance, Oracle compatibility module, and online backup and recovery.

-- rama

-- Ramarao Kanneganti, May 5, 2000

I'd like to make two small notes:

1. Almost the entire article focused on "transactions".
   I have done database programming for 5 different companies, all of them used either Oracle or Sybase (MSSQL counts as Sybase), and not /ONE/ of them was making use of transactions. Given the fact that many companies do not use the features that are so "critical", according to your article, what other arguments can you make /AGAINST/ mysql?

2. Up until less than a year ago, Sybase only had table level locking!

-- Joey Smith, May 5, 2000

The company that I work for incorporates a database in an' appliance' which is used to manage enterprise networks. We were originally using Postgres, but we were forced to switch because of huge memory leaks. We literally had to create a job that would kill and restart the database after 5,000 transactions, or 5 minutes (whichever came first). The load Postgres placed on the system was astronomical. It was not uncommon for load averages to reach 10.x during heavy database load. We switched to MySQL because at the time, it was one of very few alternatives for Linux. Surprisingly, those same boxes rarely hit load averages above 2.0, although they are performing exactly the same functions as before. Only the database was replaced. Have the memory leaks and overhead problems been addressed yet?

-- Michael Avery, May 5, 2000

As the primary developer of the www.DearDiary.net project I've been rewriting our older (cough cough) flat file (cough) code over into an RDBMS, I don't feel I need to say why :) The biggest hurdle we've had has been choosing the 'right' RDBMS to use.

We started out with PostgreSQL because we know the product and we like it. But after doing some test imports and then testing the data to make sure it had imported correctly
we ran across a few problems, the biggest of which was the maximum size of a returned record - I might have this wrong since I didn't find this out it was my colleague, but I believe i'm on the right track. It's limited to 8k and given that 99% of the data that we deal with on Dear Diary are diary entries the 8K limitation basically renders it useless to us as the solution we were given from the PostgreSQL people was to take the larger data out of the Database and use flat files....

We eventually chose MySQL because it supports much larger return result sizes, I believe as high as a megabyte?

I concur with your opinions that MySQL is by no means a top quality RDBMS - it doesn't have the featureset that one should by any stretch, but if you code carefully and you make sure that none of the limitations are going to cause you any problem (remember, we're coming from using Flat Files, which is race condition and file/data corruption city!) then MySQL will do the job nicely and extremely quickly.

I should note that at a future date, when the system is more heavily accessed, we do plan on upgrading to either Postgres if it functions with our data, or to Oracle if necessary. I should also note that we arent a big company, this is all coming out of our pockets, so I don't bandy the word Oracle around lightly at all.

-- Matthew Peddlesden, May 5, 2000

While it's true that MySQL doesn't offer transactions, it has non-ANSI SQL feature, table key autoincrement, which is atomic. With simple (flat) table structures which most WWW-pages use and/or careful database design, this is enough. For e-commerce and other more complex areas where *writing* to several different tables at the same time is required, MySQL is a bit, how would I say, inadequate :-).

Using autoincrement & no transactions will later make moving to a full ANSI RDBMs a bit harder though. It's you that decides between features and speed...

-- Eero Tamminen, May 5, 2000

I want to correct Joey Smith's comments.

1. *Everything* you do inside an RDBMS like Oracle or Sybase uses transactions. If you don't explicitly use them, then they are still being using implicitly. Otherwise, an update statement that updates half the table before the UPS fails and the power goes out could still leave the database in an inconsistent state. On an application level, explicit transactions are only needed when protecting state across multiple SQL statements.

1a. Depending on which versions you are referring to, MSSQL most certainly does not count as Sybase. Both Sybase System 10 and MSSQL 7 (neither of which is a new product) were completely rewritten.

2. Sybase has had page-level locking for many years. Table-level locking would have made it (and any other enterprise-class RDBMS) unusable.

-- Curt Hagenlocher, May 5, 2000

I've used both MySQL and Oracle for different projects.

Given MySQL's vast userbase, and very good track record, I think it is empirically clear
that MySQL is an acceptable choice for many people's needs. (Unlike MS products, which have a vast userbase and poor track record). In many ways, MySQL's lack of cost, overall complexity, ease of installation and reinstallation, and simple database-checking utilities make it more "reliable" in real terms than Oracle's theoretical reliability, which is more expensive in software and human costs than most organizations can afford, and even then they tend to screw it up.

(Oh, and compare sqlplus to mysql's shell -- maybe you can do more with sqlplus, but mysql is far more user-friendly.)

The criticisms voiced herein seem to be about semantics. Maybe it isn't a "real" RDBMS. So what? The real world doesn't always map to academic concepts or product categories. You can't dismiss the experience of thousands of developers, that MySQL is pretty darn reliable and useful, with a wave of the hand saying that it is not ACID, therefore unsuitable for more than little toys.

(Sometimes the disaster scenarios dreamed up to justify ACID are a little bit far-fetched. On the other hand the lack of record locking is a valid criticism, even on MySQL's stated goal of speed.)

This isn't to say that Oracle and certain other databases aren't greatly superior to MySQL in all areas except speed and cost, and possibly usability in certain areas. And nobody is suggesting MySQL is the right solution for arsDigita.

MySQL is a good solution, certainly not perfect, and certainly not without room for improvement. If PostgreSQL has overcome the reputation for slowness it had when I last evaluated it, maybe I'll switch.

-- Neil Kandalgaonkar, May 5, 2000

To me it comes down to a straight choice. Do I want a bombproof database which will ensure my data integrity at all times? Or am I happy to sacrifice a little of that resilience, perhaps at the cost of having to restore old data from backup every so often, in order to get a huge increase in speed? PostgreSQL and MySQL serve different purposes, and I use them both for different things.

For most of our web usage, we really honestly don't give a hoot if a machine dies in the middle of a transaction. The data on a typical web database changes quite infrequently, so you're not losing much by going back to an older version, if you really have to.

It's the same as the argument over journalling filesystems. You take a performance hit by having one. Sure, you're guaranteed no data loss. So what? Most people simply don't need that level of resilience, so why should they have to put up with the speed drop?

It's possible that a few people are labouring under misapprehensions about MySQL, perhaps believing it to be more than it is. But most of us know and understand its limitations, realise that they are actually design decisions, not shortcomings, and that's why we use MySQL in the first place.

-- Jamm'n Wheeler, May 5, 2000

I use MySQL on my website...why? Because on the previous "collaborative" version, I was storing user submitted articles in flat-text files. Occasional manglings of these text files were common due to contention (which I knew would happen when I built the
thing; I was just experimenting). MySQL solved that problem (assuming the power stays on). It also gave me a nice query interface that I was already familiar with. The data model started out simple, a couple of tables, and the drawbacks of MySQL didn't seem to be a show-stopper for me. However, as my data model has grown, and I work on the newest version, I'm hitting those limitations.

The biggest of those is just the lack of MySQL to protect me from myself within the data model. I'm going to have a couple of people working on the code this time around, so I'd like to make reasonably sure the data model does not get corrupted. With no referential integrity supported, and no transactions, my nicely designed data model could easily be corrupted by some errant code written by someone who maybe doesn't understand the full scope of things. The right place for those integrity checks in almost certainly in the database. When working with MySQL, what happens if one of the programmers forgets to touch a table on something requiring more than one update? The database happily accepts this bad data, without complaint or warning. Lack of stored procedures for abstracting the interface is another blow to MySQL. Of course you could emulate this using the host language, but it would very hard to ENFORCE those rules.

So MySQL for me served me very well for quite some time. It's just that I've outgrown it now, and not having umpteen-thousand dollars for an Oracle license for my fun little website, I've moved on to PostgreSQL. Which has it's own little issues...I'm going to miss outer joins for one....The bottom line ends up being that full blown RDBMS' are hard to build, and no one has it completely right yet. Oracle probably leads the pack, and it's about all I'd consider for a commercial production environment. For cheaper, less critical installations though, PostgreSQL can be lived with, and I hope to have a long and happy relationship with it. :-)

-- Rob Meyer, May 5, 2000

Transaction support comes in a number of forms, often not usable in a particular situation. In one application using Sybase a few years ago, several simultaneous processes needed to update 2000 records each and then either commit or rollback the whole set at the end. The problem with Sybase was that it used page level locks meaning that even if each process updated rows that were unique between transactions, it could still deadlock because an unrelated row would eventually be on the same page as one it wanted to update.

My solution turned out to be generally applicable to situations where you can't or don't want to use the native transaction support: essentially you add a generation key to each row that refers to a row in a very small generation table. Any selects must add a where clause that ignores rows with non-completed generations and selects only the newest resulting version. When an update is completed, the generation row is added, making any number of rows in other tables atomically visible.

This is my solution to adding transactions to MySQL, and it some cases to Sybase and Oracle. It is a form of long-term transaction that is more controllable by the application, or better by a database interface library. It's not a perfect solution of course.

Another technique that I have used with MySQL is one that is suggested in their documentation: adding a lock column to a row so that multiple competing processes know which one has received the lock and can proceed with an update. It's clunky, but it works and is relatively fast.

My view, for many reasons, is that the appropriate solution is a middleware layer
that acts as a transaction monitor, similar to CICS (I think), Tuxedo, and other systems. This layer would not only manage locking, transactions, rollbacks, logging, etc. but would also handle cross-database partitioning and replication.

I've started work on this layer, but am also looking for collaborators. The first version will probably be specialized to handle XML requests and responses rather than SQL which will allow handling of fully structured data.

-- Stephen Williams, May 5, 2000

I agree with what the article says, however, I think you are downplaying the importance of the 'little things' MySQL achieves very well ... and for which it has very little competition. Before listing them, I would also like to add that it can be both very easy and very sensible to use MySQL along with a 'real' RDMS, each handling it's own job -- depending on the specs, it might or not be though.

MySQL is the best tool to:

- Store user comments in a web site -- why would there be any need for ACID here?
- Storing user preferences that need to be queried at each page view
- Storing cookies
- Logging activities that you want to query/aggregate elegantly later
- Store temporary data (example: in a wizard type web app, there was an upload file widget, and I stored the data in MySQL to keep the data persistent accross pages, and when the user clicked 'finish' it was sent to an oracle DB).
- Store mailing list datas (email addresses, etc ...)
- Store access control information that need to be queried at each requests
- Implement a special purpose search engine.

That being said, that leaves lot of applications it can't fulfill, and I'm guilty myself of having reached its limits and, I have to admit, it's a pain in the ass.

Also, people here tend to claim that MySQL is just a glorified storing system. It has one important feature that makes it much more than that: a powerful client server architecture, which makes quite a good use of multiple processor architectures.

-- Nicolas MONNET, May 5, 2000

MySQL is a great read-only database where speed is critical. Once database updating occurs, it is not an appropriate tool. Using MySQL in ANY database where updating is done is basically the same as deciding since computers are becoming more reliable you don't need to perform regular backups. There may be a lot of people who don't back up their hard drives, but the minute they fail they wish they had. MySQL would be a more honest tool if it only supported bulk loading and selects. Many developers are not database administrators and really don't understand the basic principles of why the ACID test is REQUIRED for updated databases.

-- Larry Charlton, May 5, 2000

This document is entitled "Why Not MySQL?" and it is on the OpenACS.org site, not MySQLSucks.com.
IMHO, the three main points from Ben's quick article are:

- "This quick paper attempts to explain why MySQL ... should never be used by any system that handles critical data."
- "There are very good reasons for using MySQL. A need for a reliable, ACID-compliant datastore isn't one of them."
- "We are out to build an enterprise-level Open Source web toolkit."

Considering the goals: 1) an enterprise-level web toolkit, 2) that is 100% end-to-end open source, 3) where the data is considered critical by the implementors/users.

Is there anyone who would argue that MySQL is appropriate 1) for enterprise level use AND 2) meets the definition of 100% open-source AND 3) is the right choice when the integrity of the data is considered to be absolutely critical?

-- Michael Cleverly, May 5, 2000

Let me interject something here. Transactions at the core DB server level is required. I have a very old UNIFY 3.x database that when the OS crashes during the day. We are pretty much required to restore from backup and replay all completed transactions.

Now, before I started three years ago. They would just bring the system back on and continue with their work. Which as a nice side affect would cause corruption within the database that would appea on reports or would imped the users work later on down the road (and when you have almost 100 tables in almost a 1/2 a gig database, finding a bad (set of) record is not the funnest thing in the world.

I don't see how in the world one could (in good faith) roll out a MySQL database (or mSQL for that matter) as anything but a read-only database. Because if the system crashes during a peak time of the day. What procedures do you take to ensure you have a clean database?

Reload the last version from tape and replay the transaction logs? Oh, =) Forgot no transaction logs. So you either throw away all the work done since the last backup, or you just bring the database up and pray the corruption will not cause application problems.. I guess if one could (and the DB was small in enough) they could write a 'link walking' script to verify all database entries and tag questionable entries for the DBA to verify, but that is a lot of work.

If mysql would at least do SQL logging to disk (or tape) since the last archives. One could at least come up pretty clean. Roll back is not always required, but a file showing: 

<TRANSACTION> SQL STATEMENT(S) </TRANSACTION> Would improve things very much. It would at least provide the means to safely recover up to a reasonable point. And last I checked it would be a very simple hack and performance degrade would be slight. (And for those with Read only DBs. It could be turned off.)

<Shrug> But, hey..What do I know.=) I'm just a lowly SA/DBA/programmer. <grin>

-- Ben Lindstrom, May 5, 2000

I think a lot of people have missed the point. I don't believe the author is out to slam the functionality of MySQL, as it is certainly useful for the majority of non-critical situations. I believe the point is this: looking at the comparison pages provided by MySQL, an outsider might say "look - MySQL does lots of stuff Oracle and Sybase do
not; why would I ever choose anything but MySQL?"

It seems that MySQL does a fair job of misrepresenting itself. It is not a hardened industrial solution. It does, however, do a nice job of fitting in to the common case scenario. My parents don't need an Alpha server or a Stratus Continuum to browse the web and receive email - a simple PC or Mac will do nicely. Similarly, most applications do not need an ACIDic RDBMS - it's overkill, and MySQL quite nicely fills this rather large niche [sic].

For the industrial applications which do require strict ACIDity, MySQL appears to misrepresent its capabilities. In these situations, the RDBMS generally runs in fault tolerant clusters or on fully fault tolerant machines; failure, in this case, is not an option. In this separate world, a package such as MySQL is almost never an option, and this should be apparent from their comparison pages.

I don't think the issue is one of functionality, rather it is one of misrepresentation.

-- Jon L, May 5, 2000

This last post implies that crashing is common. It just ain't so with Linux!

I have my systems set to backup, automatically, with a crond entry, every night at 4:00 AM. (when system load is sure to be light, if at all)

The backups are versioned, daily for 1 week, monthly for 3 months, and are stored on a second hard drive in the server. A CD of the backup is made periodically.

If the server crashes (?) then restore from the backup and away you go!

Yeah, you might have to re-enter a few hours worth of transactions, but if you use decent hardware, (TI memory, Abit or DFI Motherboards, etc instead of "Taiwan ROC") you shouldn't see a crash more than once every few years. Maybe take the system down every year or so to blow out the dust and replace the fans? =)

I've only seen Linux crash when the hardware was questionable or mediocre.

In this kind of environment, how important is journalling, etc? I won't pretend that I'm running enterprise-level databasing, (small biz is more like it) but if I were running an enterprise-level system, the relatively incremental cost of Oracle on a Sun server is well worth it.

Don't use a shovel to replace a backhoe, eh? Shovels are great - for their intended use. As are backhoes.

(Just try to prepare your back-yard garden with a back-hoe, or mine coal with a shovel!)

So call a spade a spade and a back-hoe a back-hoe, and get over it!

MySQL does what it's intended to do quite well. It's inexpensive, fast, easy to set up, and isn't overly complicated. It runs rapidly on entry-level hardware, (I've searched an UN-indexed database with 150,000 entries in under 45 seconds - on a Pentium-75 with 64 MB RAM! Indexed, that search time dropped to < 15 seconds!) supports enough SQL features to be useful, and integrates nicely with PHP and Perl.

http://openacs.org/philosophy/why-not-mysql.html
So, it's not a back-hoe. So what?

-Benjamin Smith

-- Benjamin Smith, May 5, 2000

I believe that MySQL does now have a transaction log. Take a look at their documentation page here:


Here is a bit of a quote:

"When started with the --log-update=[=file_name] option, mysqld writes a log file containing all SQL commands that update data."

-- Ian Ragsdale, May 5, 2000

Folks here have mentioned they have been (1) "writing database apps for years and never once needed transactions" or they have a (2) "database that is strictly for querying, so transactions aren't necessary". Finally people are saying (3) "99% of database interaction doesn't require transactions, so transactions don't matter".

I have been writing database applications for about 5 years (programming for about 18). About these to points I have this to say:

(1) Look at your database code AGAIN. If you have more than one INSERT or UPDATE statement that is dependent on another (should logically be atomic) you NEED to put this in a transaction. Just because it works today doesn't mean you won't hit some failure point in the future and your data integrity will be lost and HARD to recover (you probably won't know it was failing until your database is a mess).

(2) The data got in there SOMEHOW, didn't it? Go check THAT code.

(3) Sure, 99% of database interaction may be simple inserts where you can catch the failure or selects where transactions aren't needed, BUT, almost every app I have written has 1 or 2 (or more) inserts or updates which are logically atomic and THEREFORE REQUIRE transaction code, in my opinion. SO, even if your CURRENT app requires NO transactions, why not learn a database that will support transactions for when you DO NEED the ability to do transactions? No need to support lots of database, one well chosen database should serve you well.

-- Kevin Dorff, May 5, 2000

After having just moved off a MySQL installation that grew far too large, I'd like to offer some support for Ben's opinion of MySQL. However, I also see why MySQL is fine for many, despite it's problems.

-------------

If you need to modify data, and it's not okay to lose data, then you just shouldn't use MySQL.

I agree with others that most applications don't need transactions. However, I assert that the real problem with using MySQL is that it does not handle failure cases. Most notable, MySQL does not have roll-forward recovery in the case of some downtime event. You need to perform a full-table check to repair the tables before bringing them back online, and even then there are no guarantees about what data will survive.

However, I believe most websites out there are willing to use MySQL primarily because they care more about the database being free and easy to administer than about it losing data. This sounds like a strange proposition, but when your website is free, it often does not seem that bad to possibly lose some small set of data from time to time. The administration ease of MySQL ends up meaning more than ACID to these sites.

MySQL also tends to perform "better than you'd expect" in real world situations. It's fairly bug free and will stay running without problems and without corrupting data most of the time. I've not heard the same confidence out of PostgreSQL (yet) in high-load configurations (they are working on it).

-- David Jeske, May 5, 2000

I'd like to add a few comments here about mysql. The first is that stored procedures are *bad*. They improve performance a non-zero amount, but, they make maintenance a nightmare for website-based apps. If you plan on not shooting yourself in the head while making changes to the site, put all of your application logic in one layer, period. I've heard people who prefer this to be in the database, but with websites there is presentation logic that you *have* to put at the webserver level, so my preference is to have everything there. The performance hit is just not worth having two different languages/sets of code to maintain.

That being said, I'm facing a choice soon between oracle and mysql, and I'm going to use oracle. MySQL doesn't support a critical feature I need - transparent database mirroring. Once you get beyond 1 database server (and you have to if you're even vaguely serious about uptime), you have to have transparent database mirroring, period. I need to have 4 database servers, and at any point, I can unplug one of them, swap in a fresh box, and the data will get replicated (slowly) over to the new box. I'll take a performance hit, but the data will be there, and the site will be up. Until I can get this from mysql, I can't consider it.

-- David Fallon, May 5, 2000

The framework for this conversation should be RECOVERY TIME. How much can you afford? Most commercial sites cannot afford much. If the data gets really hosed, recovery time goes to infinity (manual re-key?).

I think MySql may be a great solution for READ-ONLY applications where the database can be rebuilt from scratch within an acceptable recovery time. Should you decide to use it for anything else, understand that you may need to walk away from the collected data.

The arguments regarding referential integrity and the need for transaction are par for the course - if you use MySql and are happy with it, good for you. Just don't delude yourself into thinking that you're gaining database experience.
Jan

-- Jan Zawadzki, May 5, 2000

Although many people state that MySQL is good, fits most needs and are wondering why this criticism is coming out, I feel there is one issue not being addressed here.

The credibility of open source software is at stake, especially if MySQL is misrepresenting their own works or if people are saying "sure, this works, I've used it for years", when the product being discussed doesn't stand up if bad things happen.

It might not be that that the software isn't good, it just happens to break in certain places, like ACID. If the (R)DBMS can shut down in the middle of a transaction without being able to cancel out the effects, bad things happen.

Especially with claims like "at TcX we've been using it without problems since....". I know many people who have used windows without problems since 96 and never had a single problem with it. Does that mean that Windows is stable?

They might have a very good product, but they need to market it for what it can do, not what it can do in the best case. Misrepresentation can hurt MySQL first and then Open Source (or open source like stuff like MySQL) later.

So, for some users even Oracle might be the better solution, peace of mind is also worth $$$ to people, especially if they don't really want to check every second if their data is correct...

Geir

-- Geir Bjune, May 5, 2000

Just a quick comment concerning Sybase products:

I feel that referring to "Sybase's database" is a little vague. Sybase offers several different products, each of which are focussed on different sections of the market. Two of Sybase's database servers are Adaptive Server Enterprise (ASE) and Adaptive Server Anywhere (ASA). ASE, the enterprise-level database, is designed to support the demanding requirements of Internet and traditional, mission-critical OLTP and DSS applications. ASA, on the other hand, is a high-performance, self-tuning database server that requires just 2MB of RAM, and runs on Windows 9x, Windows NT, Windows 2000, Windows CE, NetWare, Solaris, HP-UX, AIX, and Linux. Recently, ASA was awarded an "Analyst's Choice" award by PC-Week.

If you're interested, check out the PC-Week Article

In addition, ASA is extremely affordable, easy to use, and quite reliable -- Sybase's products ensure stability in mission critical environments, like financial services.

-- Kris Vorwerk, May 5, 2000

If you are knowledgable about enterprise RDBMs, and you've chosen MySQL as the right tool for your application despite its limitations, chances are you've made a good choice. There are many applications that its appropriate for and with a little extra effort, you can make it as robust as you want. Note, that I'm not saying that it scales to large...
distributed international enterprise level solutions.

In house, we use MySQL on Solaris on Sun Enterprise Servers and have found it to be very reliable. The system configuration is well power protected and backed up, so its fairly safe for us to assume that our foundation isn't going to fail us. So if you assume away the "kicking the power cord out" scenario:

Commit and Rollback statements allow programmers to be lazy. They are a convienence for something that programmers could do themselves. The same thing goes for referential integrity. If you are unwilling to check the columns that you know to be foreign keys before doing inserts or deletes, then you've gotten lazy. When using MySQL, its your responsibility to do this.

Same thing goes for commit and rollback. If you are using MySQL and you need this functionality, then you've chosen to handle this kind of thing yourself. For every SQL statement you write, write the inverse statment. As transactions are commited, add the inverse statement to your rollback script. Consider it a homebrew transaction log. There is no reason you can't do it yourself.

Of course thing kind of thing gets old and it doesn't scale well. But its a perfectly acceptable way to make the best out of a very fast but feature limited database such as MySQL. To say that MySQL shouldn't be used in any application is just enforcing a lazy attitude to a simple problem.

-- Rafael Rubio, May 5, 2000

MySQL can log statements that update the database with the --log-update=file_name option when you start the database. You can use this for database recovery and/or a primitive form of database replication/mirroring. MySQL offloads maintaining database integrity to the developer in order to achieve better performance. I think they spell this out reasonably well in their documentation. If you have a complex schema or a slew of developers all working on the same database; you probably ought to look elsewhere. If your needs are more modest and you are willing to take on the task of ensuring database integrity in your application instead of relying on the database to catch everything; then, MySQL seems to work well. In regards to safety... in three years I have never lost any data using MySQL. That include a few episodes of the plug being kicked out of the wall.

-- John Brundige, May 5, 2000

There seems to be a persistent assumption thread in this discussion: That the alternative to MySQL is to use a commercial RBDMS. We have used Postgres for years with great success. It seems to have all the advantages except for speed of MySQL with none of the disadvantages.

Real database programmers have been working with the trade-off of database speed and reliability for years. We know how to right efficient code that minimizes database use while maximizing performance.

-- Fletcher Kittredge, May 5, 2000

It's obvious that everyone has strong opinions on this subject, but most, if not all (OK, most.. 8-) of these rants could be avoided by keeping in mind these simple points:

http://openacs.org/philosophy/why-not-mysql.html
1) Know your requirements. This includes the balance of performance vs scalability vs speed vs cost (monetary/otherwise) vs the money cost of time (All you econ and business majors should know this - if this concept is new to you, LOOK IT UP!).

2) Know the tools available to do the job. This includes the RDBMS (or whatever term you choose to use), the language that will be used to address it, the skill sets of the people involved, etc.

3) Know how to balance the features of each tool to successfully meet the requirements. This includes proper coding to maximize the features of the tools - for example, using scripting to limit database access, in addition to whatever security features are available in the RDBMS. It also includes not 'putting all your eggs in one basket'.

4) Know how to manage the process.

The clear answer (to me, anyhow) is to use the appropriate tool(s) for the task at hand, NOT to cling to whatever is a personal favorite. Also, why not mix and match?? For example, within one program I point to a MySQL server for access permissions and general information while also pointing to an MSSQL server for other details. That way, users get the benefits of BOTH environments.

Enough said. It's nice to see so much discussion, though!!

-- Tom Fillmore, May 5, 2000

I'll have to chime in and say that postgresql, at least version 6.5.3, is a bit too buggy to be "enterprise-class".

Its indexing only works when your data doesn't span the entire 32-bit range. Fortunately, there's a 5-line patch. Unfortunately, we can't seem to turn it into an RPM.

We are seeing several "NOTICE: AbortTransaction and not in in-progress state " every day from our numerous servers, and we're not using explicit transactions.

Routinely we get rogue files with no table, or rogue table/index with no file.

We get inexplicable disconnects.

We recently started seeing inexplicable crashes.

I look forward to the day when postgresql quits chapping my ass.

-- Robert Forsman, May 5, 2000

There is no "issue of what COMMIT/ROLLBACK does to a database server internally". Take an Oracle dba class. This stuff is actually designed very intelligently. Oracle DOES NOT obfuscate how it works (outside of not letting you look at the source :) because it's complex and you need to understand it to be an effective DBA.

-- Tim Keating, May 5, 2000

Let me add one more statement, now that I've read a bunch of comments, based on my experiences:
A lot of people in this forum are saying they are willing to take "a little" risk of data loss in exchange for speed.

I promise you, you will stop saying that after a data loss happens.

-- Tim Keating, May 5, 2000

I'm from the camp of use the tools that are easiest to do the job. Currently I am running data that does need transactions in a MS-SQL DB. This is all production data, it has to be consistent. We also have to run reports against it. The reports hit the server harder than the production use does, so we moved them over to an MySQL server that is read-only. It just gets data from the MS-SQL server every few minutes. We get the best of both worlds. I also mostly use Apache for the web server (With PHP to get data), but some of the stuff can only be done easily on NT (ASP calling DCOM stuff) so I use IIS as well.

-- Adam Wilkinson, May 5, 2000

Commit and Rollback statements allow programmers to be lazy. They are a convienence for something that programmers could do themselves.

This is very wrong and a shows lack of understanding and experience. If a transaction is updating several records and you have, say, 300 users doing this simultaneously, you will cause huge problems with the above rollback method. The stored information in the programmer's own rollback functionality can very easily be made out of date by another user doing a simple table update midway through the "transaction". The programmer then hits a situation where a rollback must be performed and overwrites the latest value (s) with his/her out of date values.

The db should always force integrity as I don't believe any site has ever had 100% bug free software and zero failure. Best intentions are all well and good as may be the original development team, but in the real world this approach invites database integrity errors.

-- tony lambley, May 5, 2000

I felt compelled to comment on Rafael's message above. My eyes got really wide as I read what was being said. I don't think I can make a case for our programmers to write inverse logic (and the associated checks) to cover the lack of transactions, or do their own checks for integrity before and after updates, and after the system comes up after a powerfail.

It might be construed as lazy to rely on the database to do these things, sure. But programmers make mistakes. Okay, you might make a mistake with the schema design (incorrect constraint or trigger or whatever) but you can take more time over that, get the team to validate it before implementation.

If you write inverse logic and checking for every atomic operation you want to perform, you've just put a bunch of extra load on the programmer (and/or time on the schedule, and/or cost to the customer).

Once you start getting above a small number of tables, it's going to get really old checking the schema to make sure you've not violated a constraint - and there will be mistakes. Mistakes which, with a good schema and constraints that work, will show up
immediately. You only need to miss one single related table and your data isn't integral any more.

I, as DBA and programmer, would trust MSSQL or Oracle or Sybase's transaction rollback or rollforward replay over a bunch of home-grown logging and/or inverse logic any day.

-- John Hawksley, May 5, 2000

First, I'd like to state that the language of the article was inflammatory and unprofessional. It infers that MySQL is immature, which it is not (unless "maturity" implicitly requires transaction support and subqueries). Sentences like "MySQL is just a glorified filesystem with a SQL interface" are derogatory and opinionated. And most importantly, the MySQL dev team is called dishonest, misleading, and they are accused of having a "deep lack of understanding".

I fail to see anything misleading or dishonest about the comparison charts. Perhaps the article authors would be happier if the charts all had bold, H1-sized headings at the top that read "MySQL DOES NOT SUPPORT TRANSACTIONS--REPEAT, REPEAT, MySQL IS NOT ACID COMPLIANT. AND NO SUBQUERIES, EITHER." And I'm sorry, but differing design goals != deep lack of understanding.

Second, many people here have posted personal experiences of PostgreSQL being too slow, having memory leaks, crashing often, or severe limitations--it seems to me (based on these postings) that PostgreSQL is far less mature than MySQL, and furthermore that it is not "enterprise-level", as the authors suggest. (FYI, I have no personal experience with PostgreSQL).

Third, in my experience, MySQL serves adequately for web development, even in "enterprise level" situations. Unlike financial transactions, things like individual email addresses, cookies, news articles, comment postings, and user preferences/logins are NOT mission-critical. The nature of the web is such that email addresses and cookies will go bad much sooner than a MySQL database failure (people's hard drives crash, they forget their passwords, they change ISPs, they never re-visit the site, they unknowingly create several user accounts, etc.).

Next, I'd like to say that transactions are not always necessary, even when dealing with e-commerce. Imagine two e-commerce sites, where Widgets are being sold at $100 each:

1. Company Foo is using PostgreSQL. It needs to be restarted once per day to deal with stability/memory loss issues. During the restart (while the purchasing web page is unavailable), Company Foo looses 10 web-based sales, for a guaranteed (minimum) net loss of $365,000 in sales per year.

2. Company Bar is using MySQL. It does not need a nightly restart. But, some random disaster occurs (which Postgres, because of its transaction handling, could theoretically better recover from). The MySQL database looses half a day's worth of sales, say, $150,000 worth.

Here's another scenario, where MySQL might be preferable for e-commerce:
1. A disaster occurs. Company Foo is using Oracle. Because of its ACID support, all transactions up to the disaster are recovered—however, because Oracle is difficult to administer, it takes the inexperienced DBA 2 hours to get back online.

2. A disaster occurs. Company Bar is using MySQL. Without its ACID support, 10 transactions (that should have succeeded) are lost—however, because MySQL is so easy to administer, it takes the inexperienced DBA 1 hour to get back online.

I guess my point is that, if you're doing many thousands of transactions per month, you won't give a hoot if one or two get lost due to a lack of transaction support in your database—as long as your web page is up and running, is stable, and is easy to administer. (Of course, with banks and financial institutions, it's a different story)

Finally, Stephen Williams posted a work around that might give MySQL the transaction security you need. Given all this, I think the article author's conclusion, that MySQL "should never be used by any system that handles critical data", is false.

--Derek Simkowiak

-- Derek Simkowiak, May 5, 2000

I see the problem of accepting MYSQL as being based from a limited perspective on the part of a large segment of DBA's and developers.

In truth, no matter how it is said, most applications do not need transactions, and the level of protection that is presented by inserting data as a group. There is always a granularity of inserting data at one record at a time, whether this is accomplished with a software hook that is called a transaction - or not. What is really more important for the data integrity is roll backs, and this is accomplished through the transaction logs.

MYSQL suffers more from the lack of Foreign keys and by the lack of constraints. But for most applications, this is not critical either. Most applications are being done with MS Access - not Oracle, Sybase or Informix. And increasingly these projects are getting more complex.

So to say most database jobs need transactions is just untrue, but is an understandable statement from DBA's who have spent their lives working with million dollar budgets in finance.

Crashes, are flat out rare. I've never, in 4 years, had a MYSQL/Linux crash. The powerful client/server capability, with ODBC capability and DBI interface, high level of support, SQL compliance and stability makes MYSQL far better to use than Fox Pro, Access and Paradox. I've seen Citibank use these apps for Funds Transfer analysis.... a task which would have been much better done on MYSQL and Linux.

In comparing MYSQL to Oracle, Monty is looking at the problem from the perspective of the current set of PC base tools. If you want the stability of Unix and the large subset of expanded capability of Oracle, MYSQL can look good, and do the job.

If you need something that is going to automatically roll back and have a parallel server, where you need complete rock solid perfection of the mirrors, and zero chance of data loss no matter the cost.....(And if your PC can't tell that the power has died, the use of
transactions becomes limited)

..then you need to look at something like DB2.

But to call MYSQL a file system, IMO, is not justified.

Ruben

-- ruben safir, May 5, 2000

Has anyone tested postgres7's speed against mysql as well as postgres6? I'd be quite interested in the results of that...

-- Andres Salomon, May 5, 2000

It is simply overreaching to say that because MySQL does not support transactions, therefore it does not support ACID (or CRUD), and therefore it's not relational, and therefore it can't be an RDBMS.

Hogwash.

The standard for relational databases was set by Dr. E. F. Codd's two foundational papers written 30 years ago. "Relational" simply means based on and completely adhering to the requirements of predicate logic. This is hardly news; it's been discussed widely for three decades now. Codd wrote a paper in 1985 that discussed why many of the vendors claiming to be "relational" at that time were not, in fact, according to a dozen criteria which then became famous as the "12 Rules." The industry mostly cleaned up its act thereafter in regard to claims of relational implementation.

Dr. Codd would probably quarrel with the notion that SQL is "relational" according to his original framework. In a variety of areas, including implementation of NULL (the most famous example) as well as referential integrity and set operations, SQL diverges from his approach in order to address practical issues.

On balance, most observers would say that even adherence to SQL89 would qualify a database as being "relational." Certainly, any database that mostly if not completely follows the SQL92 standard is most certainly relational. And MySQL fits the description just fine.

What's so funny about all this is that the large commercial databases like Oracle have often been slow to pick up on the ANSI SQL standards as they come out. My own favorite database, R:Base, had numerous features of SQL92 long before the major-heavyweight ones did.

The one missing feature of MySQL that slightly undermines its claim to being fully relational is lack of support for foreign keys. But again, as with transactions (COMMIT/ROLLBACK), this wasn't a factor in SQL89 and is easily coded around or has no impact in many applications.

The one missing feature of MySQL that makes it a hard call for me to use, personally, is the lack of subselects. This is a matter of implementation and has no bearing on the otherwise relational nature of MySQL. But having it avoids a great many situations involving unwieldy joins and possibly the use of temporary tables for performance reasons.
The other thing missing from the discussion so far is any sense of the actual implications of the theoretical positions being advanced. If ACID and transactions matter so much, how often do databases that fall short of Oracle's advanced position have actual problems? Where is the empirical research?

Curiously, there is none (or very little). There is plenty of work on throughput, there's TPC, there's vendor "benchmarks", all of it only partially convincing. But there is no research I've ever seen on system robustness. Therefore there is a tendency to fall back on faith and "facts," except facts are interpreted within the context each of us is familiar with.

Data corruption is actually quite rare in the database world, in my experience. For example, I had one single instance in nearly 15 years of using R:Base in both single and multi-user situations, and that problem was fixed in a subsequent release. I simply don't hear complaints from users of any serious database, whether Oracle, MySQL, or anything else, of actual data corruption.

I do hear about performance problems and feature limitations. That's basically what the argument here gets down to anyway, assuming you have a reasonably SQL-compliant database.

And frankly, from what I have seen, suboptimal and even incorrect data design is the biggest issue out there, well beyond whether a database has "transactions" or other features. Even relatively experienced database managers make basic mistakes (although we tend to find them and fix them :), but as database technology has become more widely available, the proliferation of really bad database design has rapidly increased.

A final note, although this is really tangential. Even beyond bad design, the real looming crisis is data quality. I would say at least 50% of my work is repairing and improving the data quality of the data I work with.

Frankly, these issues of robustness, design and data quality are more important than advocating for one or another product as being the best available for all uses. Henry Ford tried that with the Model T, and the Model T was a fine machine, but not everyone needed just that.

The right tool for the job is the way to think about these issues. There are things I would only use Oracle (or DB/2) for, and there are things I would never use them for. And the same goes even for my beloved R:Base.

So my advice is: leave the windy opinionations about the absolute superiority of any given product to the salesmen. Give him credit, Larry Ellison is the king of that hill. The rest of us need to get work done, and no product or approach is absolutely perfect. The only constant is continuous improvement in the tools and methods we use.

-- Fred Heutte, May 5, 2000

I think that like RAID drives, we should create more levels of compliance. Instead of being either ACID compliant or guano, mySql should qualify for some wonderful acronym + number so that you can get right to the heart of the matter. The argument started when the ACS folks said that mySQL isn't ACID compliant as a minimum requirement at this time. We should provide a tiered set of qualifications so that you don't have to spend time covering the same bases over and over again when talking about what mySql is and isn't capable of doing. For example, we could say that mySQL
version XY is "MISER" level 3 compliant, and then you could hop right over to www.MISER.ozg to see what the heck that means. There are thousands of copies of very ACID non-compliant databases out there, most of those users have never heard of ACID, nor do they think they care.

-- Michael Brian Bentley, May 5, 2000

I whole heartedly (sp?) agree with the original explanation of why MySQL is a (non-transactional) file system and not a database. What really bugs me is the use of the acronym "RDBMS". For those who may not know, the "R" in "RDBMS" stands for "Relational". This may be only semantic, but it is clear from the definition of RDBMS that MySQL is not relational, because you can't define any relationships...

To MySQL's credit, they don't seem to use RDBMS any more in their docs(although, they used to, for example in the now unavailable "BENCHMARKING RELATIONAL DATABASE SYSTEMS" page).

Mohammad

-- Mohammad Rezaei, May 5, 2000

>Commit and Rollback statements allow programmers to be lazy. They are a convienence for something that programmers could do themselves.

This is very wrong and a shows lack of understanding and experience. If a transaction is updating several records and you have, say, 300 users doing this simultaneously, you will cause huge problems with the above rollback method. The

Actually, it shows a lack of keeping things in context. If you have 300 users updating/deleting/inserting the same records as the norm, then the demand on the database is beyone the scope of what MySQL is meant to handle.

On the other hand, if you have a "read and append mostly" database, the mentioned scheme works just fine. As I said earlier, MySQL doesn't scale to large distributed international enterprises... which in my "lack of understanding and experience"... I wouldn't have considered it when tasked with integrating Japanese Oracle 8, IBM DB2/400, and MS SQL 6.5 to a single RDBMS for a well known 24/7 semiconductor manufacturer, it wouldn't have been the right choice.

To each his own.

-- Rafael Rubio, May 5, 2000

There is a persistent problem here concerning the attribute "relational" in the domain "database management systems."

Whether you agree with it or not, the attribute was definitively described in E. F. Codd's initial papers -- it's even obvious from the name of very first ACM paper in 1970: "A Relational Model of Data for Large Shared Data Banks."

Relational in this sense does not mean what we might think it does. Relational has a precise and mathematically valid meaning. That meaning, as defined in Codd's original
research, involves the implementation of data storage according to the principles of predicate logic. That is all.

It does not mean, by the way, that SQL is "relational" in completely conformity with Codd's model. In fact, that is an issue which has developed a great deal of heat and perhaps a little light over the last 15 years.

But for practical purposes, SQL is relational, despite its shortcomings. And the products such as Postgresql, Oracle and MySQL that implement some reasonably large portion of the SQL standard (and none does so completely) must also be considered relational.

After that, it's fine to argue the merits. And I don't see a sharp delineation between the proper use domains of these different databases. An organization with a $20 million budget may well put Oracle to work where one with one-hundredth of that would prefer MySQL. And in both cases, given a proper design and attention to maintenance and upkeep, they would be correct in doing so, since the cumulative risk involved would be about the same.

On other projects, it would make no sense to use anything less than a full-power, industrial-strength SQL RDBMS with all of the associated features.

Just because R:Base has SQL transactions, row-level locking (which it had years before many of the big-iron competitors), triggers, subselects, and all the trimmings -- on top of which it's quite fast -- does not mean I would necessarily recommend it over Oracle for many applications.

Postgresql, with a roughly equivalent feature set, may well be a better choice especially where Solaris or one of the free-Xes is the preferred platform (I'm a big FreeBSD fan, for example).

I think in the end what struck me as off-kilter about the initial posting and some of the discussion both here and on Slashdot is the immediate leap from, "here are some superior features" to "you are stupid to run anything less."

Let's agree that is not the appropriate direction to go in. One can uphold the ACID principles (or even more to the point, the original mathematical basis for relational databases), and still manage to find a way to get work done in the real world, even if our tools are never as perfect as our theory.

-- Fred Heutte, May 5, 2000

I did not see, in scrooling all the way to the bottom of the comments, that the actual problem at hand was addressed, and it is this: I did not see that the OpenACS people's problem was with *what MySQL is*; it appears to me that their problem is with *what TcX represents MySQL to be*, which is another matter entirely.

I have not read the TcX material, so I cannot make an informed judgement as to whether they are, in fact, representing it to be more than it is capable of being, but the items which were quibbled with rang true in the description, at least to me.

And truly it is written: INSERT and SELECT do not an RDBMS make.

But then, maybe it's just me.
So many things are just me.

-- Jay R. Ashworth, May 6, 2000

for some jobs one don’t need ACID Transactions at all. Look at a web-based forum, at access logs. These are applications where a full featured RDBMS is far to large. MySQL is no RDBMS, it’s a small ISAM Library with fast access, locks but no ACID Transactions -- some wouldn’t call this an RDBMS either, despite the SQL-like Query Language.

But there are applications where this is sufficient. We use MySQL for these and Oracle for the other... This seems to do the job pretty well, although Informix is a nice cheap alternative...

-- Paul Witta, May 6, 2000

This is really amazing:

Second, many people here have posted personal experiences of PostgreSQL being too slow, having memory leaks, crashing often, or severe limitations--it seems to me (based on these postings) that PostgreSQL is far less mature than MySQL, and furthermore that it is.

I've been using PostgreSQL for quite some time now and never had any problems of memory leaks, server crashing or anything in the likes. PostgreSQL keeps improving at incredible speed at every release.

Third, in my experience, MySQL serves adequately for web development, even in "enterprise level" situations. Unlike financial transactions, things like individual email addresses, cookies, news articles, comment postings, and user preferences/logins are NOT mission-critical.

Oh really ? So if I have an e-commerce site (say Amazon.com), than the e-mail addresses, comment postings and user preferences of my clients are not important ??? What is then ? Storing this data safely is crucial to any e-commerce site (since you said MySQL is adequate for enterprise level situations, e-commerce is an example of that).

Next, I'd like to say that transactions are not always necessary, even when dealing with e-commerce. Imagine two e-commerce sites, where Widgets are being sold at $100 each:

1. Company Foo is using PostgreSQL. It needs to be restarted once per day to deal with stability/memory loss issues. During the restart (while the purchasing web page is unavailable), Company Foo looses 10 web-based sales, for a guaranteed (minimum) net loss of $365,000 in sales per year.

2. Company Bar is using MySQL. It does not need a nightly restart. But, some random distaster occurs (which Postgres, because of its transaction handling, could theoretically better recover from). The MySQL database looses half a day's worth of sales, say, $150,000 worth.

I laugh at your stupidity. PostgreSQL has never behaved in that manner with me. I don't know about versions prior to 6 though. Don't you get it ? A kid in grade school book would know more about databases after readinh a pamphlet about transactions and atomicity than you do.
Picture this then: You have an e-commerce web site that has 10 clients logging in, browsing prices, shopping, or posting comments. One of them is shopping, so MySQL locks the table (it doesn't have Multi Variant Concurrency Control as PostgreSQL and Oracle. You probably don't what that is but oh well). The other 9 can't neither browse, shop or post comments. They all go away. You just lost 9 clients that will probably never go back.

Here's another gem of weirdness in thought:

1. A disaster occurs. **Company Foo** is using **Oracle**. Because of its ACID support, all transactions up to the disaster are recovered--however, because Oracle is difficult to administer, it takes the inexperienced DBA 2 hours to get back online.
2. A disaster occurs. **Company Bar** is using **MySQL**. Without its ACID support, 10 transactions (that should have succeeded) are lost--however, because MySQL is so easy to administer, it takes the inexperienced DBA 1 hour to get back online.

Oracle is as hard to administer as your needs go. If you need a farm of clustered/db servers running for maximum reliability (say, IRS), then Oracle is really hard to administer. If you just want a plain vanilla Oracle installation, it's not too bad. MySQL in turn, **has absolutely no way** of getting complicated, because... it doesn't even try to solve the problems that thousands of big companies around the world are relying (successfully) on Oracle to solve.

Oracle is not perfect, but comparing MySQL to Oracle is, to paraphrase Philip Greenspun "like comparing Emacs with Oranges !".

*I guess my point is that, if you're doing many thousands of transactions per month, you won't give a hoot if one or two get lost due to a lack of transaction support in your database--as long as your web page is up and running, is stable, and is easy to administer. (Of course, with banks and financial institutions, it's a different story)*

Wrong ! If you don't give a hoot about one or two clients lost, you are an irresponsible businessperson that will be wiped out of the market when others offer better services than you do. What if these two transactions lost are the best clients in your start-up business ?

Every e-commerce site does financial transactions. You have to debit a credit card account and credit the user's account on your site. What if MySQL loses it in the middle ? Either your client lost money or you did. Both are unacceptable !

Besides all this, OpenACS is a project to deliver an awesome web toolkit for anyone to build a butt-kicking web site with **TOTALLY FREE SOFTWARE**. MySQL is **NOT FREE SOFTWARE** ! You need to pay for commercial use. If I had money, I would happily give money to the PostgreSQL team, because they are doing a great job and trying to do it the right way, AND freeing their software, unlike MySQL.

And do me a favor, please post here a list of all websites you built or manage so that we know where we should **never** buy !

-- **Roberto Mello**, May 6, 2000

It's seems to me, that you have no clue about what MySQL really is. Noboddy is claiming it to be the most secure RDBMS in the world, but YOU on the other almost claim that whats they say.
Then i belive it's really strange that all the people who use MySQL haven't got that impression. The people who use Mysql, normally knows what Mysql is good for, and what it is not good for.

Alos the things you talk about are pretty fare aout. You talk like that the RDBMS should be able to do ALOT of things which is not the job of a RDBMS. For example, you talk about Oracle, which has the feature that if a disk dies, it will still be able to remeber, what it was doing. Well my firend. That is not the job of a RDMBS, it's the job of the actually disksystem on the server. And for that, i only have one word. RAID. The RDBMS, isn't ment to handle disklevel security, thats what you use RAID for. Now with a double mirroed RAID 1 disksystem, i can trash several disks, and still be able to read and write to Mysql.

Transactions. while i agree with you, that this should be in Mysql, it isn't needed for 98% of the jobs you use a RDMBS for, and even then, there's alsways the possibility, of using a dedicated Transaction server program, which would often prove to be a better solution, than to use the builtin transaction system.

Also you talk about openacs, like it has to more secure, than a system that runs a country. Thats pathetic. You have no real terms of understanding what is needed for the job at hand, and what tools to use, to solve it. I sugest that you lay of, all you theoreticcaly understanding, which you have read about in a book, and then get some real handson experience, with Mysql, Oracle (NOT interbase...The levl of interbase, is pathetic) and Sybase...THEN you can talk about what to use for different jobs. Also, you a clearly biassed...Brushing of MS SQL Server, just by saying it dosn't have any track records. Now thats just about the most pathetic thing i have heard in a long time. But then again...You are just talking from thin air, and theoretically knowledge, than from actually experience. And also, please understand what the job of a RDBMS is. Because it is certanly not to give you DISK level security.

-- Mihcael Lykke, May 6, 2000

Two things: One rather interesting you chose CyberCash as an example. While I would agree MySQL falls short of having full transacation abilities and the like, considering your take on stability I am suprised you mention CyberCash. Use CyberSource or someone else similarly more respectable. I would say CyberCash is far worse for the enterprise when compared to a real credit card processor than MySQL to Oracle. (which quick disclaimer: I don't work for either company, just have been a client on both... )

Two, one very big thing you fail to really cover is how poor a "enterprise" database like Oracle is on its on for reliability. without third party products I don't think it is possible (and our Oracle DBA and Oracle's own docs) to create a 24/7 Database environemnt (clustered servers and the like).

My second point is the big one. you seem heavily concerned about stability and critical data, but you only cover the SQL aspects. now MySQL doesn't have the clustering ability either, but neither does the "big boys". Hopefully food for thought.

-- Steven Roberts, May 6, 2000

This appears to have become this decade's equivalent of the PC v Mainframe debate...

MySQL's preferred playground is as a back end data store, preserving inter-session state, and context, and acting as a data repository for web enabled applications.
In such applications, arguing about the presence or lack of multi-user access capabilities is somewhat of a red herring; multiple contexts, yes, multiple concurrent sessions, certainly, but multiple users, no; the only *user* present is that of the web server owner.

In MySQL's preferred playground, not only are we generally dealing with a single user, we are also frequently dealing with a single application author; MySQL wins time and again precisely because it is currently the fastest and cheapest way to jump start an application which requires a centrally managed data store.

It's possible for a single application author to ensure that all write requests against a single table are serialised. It's even desirable to do so in some instances; exposing the pipeline allows an author to decide how best to bundle updates for optimal performance.

Rigorously enforcing a single update pipeline is a good way to eliminate needless, performance sapping cruft from any application - whether or not it involves a RDBMS; Linux's VFS layer has taught us that this approach wins in both the initial time to market *and* performance stakes, once the design matures.

Hiding application complexity on the RDBMS side of the fence doesn't strike me as being a particularly clever thing to do; if there are performance or durability gains to be made by doing so, I'd wager that there are artificial constraints being imposed.

Is anyone else being reminded of the whole "silicon is expensive so lets microcode it" debate?...

-- Simon Mackinlay, May 6, 2000

It's actually not possible to correctly implement the SQL standard without some sort of rollback facility. For instance:

Update product set price=price * 1.05

(i.e. Raise the price of all products by 5%)

The SQL spec says that each statement is atomic (either the whole thing is committed or nothing is committed). Although there is little chance that that this will fail, there are other cases that are more troublesome. For instance:

insert into table1(value1, value2) select v1, v2 from table2

If table1.value1 doesn't allow NULLs and table2.v1 contains a single NULL column in a single row, then *ALL* the effects of the statement must be rolled back.

You can handle individual cases like this by pre-checking the data before making changes but when you throw in complex referential integrity rule (unsupported by MySQL), then rollback becomes essential.

BTW, Good transaction processing isn't theoretically hard but it requires that you are very careful about making sure that all data modifications are properly logged.

One note in all this: Everybody is saying that MySQL is good for read only applications because it's faster. In theory it's possible to make a fully transactional database just as fast for read only operation. The logging/rollback mechanism should only really be a factor in database writes. In the write case the logging activity is the limiting factor.
That's why Oracle implements "boxcar" transactions.

Bottom line: Would I run a mission critical application on MySQL? Nope. Are there commercial transactional database engines that I wouldn't run my business on? Yep. There are about four that I know of that I wouldn't touch with a 10 foot pole. For me, the ACID test is truly that. No database that doesn't do proper transaction processing would ever be used in my shop for anything mission critical.

-- Ted Carroll, May 6, 2000

Well, we've triggered an interesting set of comments, that's for sure.

It is pleasing to see that at least 5% of you understand that our problem isn't with the fact that MySQL isn't ACID-compliant and therefore not suitable for for mission-critical data, but with the fact that the MySQL folk MISREPRESENT THEIR PRODUCT.

I also have no problem with those who've posted their cautious use of MySQL for certain tasks, and the use of an ACID-compliant RDBMS for critical data. This shows you understand the niche that MySQL fills far better than the MySQL folk themselves.

I love the argument that using an ACID-compliant database and transaction semantics is a sign of laziness, and that real programmers will do it all themselves in MySQL. Hmmm...of course, that negates much of the speed benefit of MySQL, and exaggerates the evils of table-level locking. In other words, you sacrifice the supposed benefits of using MySQL and are left with complex, fragile, and hard to maintain code to boot.

Reminds me of the arguments used to support coding in assembly rather than a system programming language back in the late 70s. (For the record, in my youth I was one of the top half-dozen PDP-8 assembly programmers in the world. I got smart and turned compiler-writer when the PDP-11 was introduced). I thought we'd learned a few things in the past three decades, but apparently not.

Regarding Postgres, we're not out to tell people to use Postgres rather than MySQL. As we've pointed out, there are two no-cost commercial RDBMS options available for Linux, Interbase and the original Sybase port to Linux. That gives three no-cost ACID-compliant RDBMS systems to choose from.

Those who have given up on Postgres might want to check it out again. I'm on record - many times - as saying that 6.5, released about a year ago, was the first version that was really worth considering for 24/7 web use. Many of the problems alluded to above, including memory leaks, were rooted out for that version.

PG 7.0, which will officially roll out Monday, May 8th, fixed a very long list of bugs and also offers improved performance, particularly in situations where you have heavy concurrent access, a situation I should think all web site owners hope to see develop on their site's server.

The person who's having problems with PG 6.5 has, I trust, submitted your problems to the development group? There's one guy in particular who's a regular pit-bull when it comes to bug fixing. As a compiler-writer for nearly thirty years, I can confidently state that the worst user is one who runs across bugs but doesn't bother reporting them. Unreported bugs never get fixed.

Again, my main problem is with misrepresentation by the MySQL folks, which I've read

http://openacs.org/philosophy/why-not-mysql.html
with my own eyes. Ben's article was pointed to on zend, and the second or third respondant posted "what's this about ACID and transactions and all this? You mean, MySQL might lose data? I never knew that?" THESE are the people I'm concerned about, because the MySQL folks do nothing to educate them, but rather tell them that ACIDity isn't, in essence, important.

Those folks who make an intellegent, informed opinion aren't the problem (by definition, those who choose MySQL for e-commerce are either unethical or uninformed).

Oh, for the guy who said "just back up the files every night in a cron job", this doesn't work unless you knock down mysqld first or otherwise ensure nothings going on. If the database is busy doing inserts or updates, backing up the files leaves you with the database in an inconsistent state. Real RDBMS systems, including Postgres as of 6.5, offer consistent online backups.

This is a relatively minor issue. Most websites can certainly afford to go down nightly for the length of time necessary to back-up your database files. I just have a suspicion that the person who made this point doesn't understand the need to do so. Hopefully he does now.

-- Don Baccus, May 6, 2000

Hmm....You still don't have a clue. Just your comment about doing backup, is far off. I backup Mysql 8 times a day(not because i really need to, but because doing it 8 times, instead of one, makes it more secure...and again...thats not for the RDBMS to do, but for the backup program), and i don't need to stop mysql to do that.

For 6 months now, i have been making online backups, without taking down mysql.

Get your facts straight, before wasting our time, by posting theese posts, without backing up your statements with some solid evidence.

just going around screaming "mysql people sucks...they don't understand nothing"...well thats DAMN pathetic!

-- Mihcael Lykke, May 6, 2000

If you're simply backing up the files, and the database engine (not only MySQL, but any database engine) is in the midst of updating a series of tables that reference each other, you'll end up with inconsistent data. The filesystem backup program has no knowledge of either implicit or explicit write locks on the tables being changed.

If you can't see how that will frequently lead to inconsistent backups, I don't really care.

This is why RDBMS systems have backup programs in the first place. If it weren't a problem, they wouldn't bother. Until version 6.5, Postgres couldn't make consistent backups, either, and folks had to shutdown the database or otherwise guarantee there'd be no writes before backing up their database.

It's not a problem specific to MySQL. It's not a problem specific to database systems, for that matter.

Making the backups isn't the problem. It's restoring from those backups and
reconstructing a stable state that's the problem.

-- Don Baccus, May 6, 2000

I have used PostgreSQL since version 6.1.1. It was not their best release. :-) However, I have noticed consistent improvement over the years, and a great bunch of developers to work with.

I have been bitten by the pseudo-online backup type before -- copying the database binary files. I got burnt once -- and once was enough. Thankfully, PG 6.5 gave us true online backups as a 'side effect' of MVCC.

MySQL, while good enough for many things, is no full-fledged RDBMS -- and I'm running mission critical stuff on PostgreSQL. It is _critical_ that transactions be fully supported for some of what I am doing and developing. MySQL simply wouldn't cut it.

And MySQL simply won't cut it for OpenACS -- after all, one of the nicer modules is e-commerce -- and, frankly, you're an idiot to run e-commerce on a non-transactional DBMS. Some might say that I'm an idiot for thinking about running e-commerce on PG -- but, you know, PG has been good to me.

As to the argument that business logic doesn't belong in the backend, I pose the counterargument -- business logic doesn't belong in the client. The client should be oblivious to the integrity enforced by the DBMS, unless there is an error. The client should be oblivious to the errors that will inevitably occur -- until one occurs, and then the client should not be asked to undo all its work -- that IS the DBMS's job.

Why is MySQL not an RDBMS? Well, MySQL certainly implements relational behavior -- so, yes, it meets the R and the DB and the S of RDBMS -- but, it does not _manage_ the data if the application has to do the locking and the rollbacks. It may be a Relational DataBase System, but it is not a Relational Database _Management_ System -- the management implies the the backend can take care of itself, which, from the messages I have seen (as well as the MySQL docs) MySQL cannot do for subqueries, better then table-level locking, or transactions.

Hey, if someone wants to do so, you are certainly welcome to port OpenACS to MySQL. Good Luck!

-- Lamar Owen, May 6, 2000

As a director of a small company I recently managed the construction of a search site for a particular industry group. We were forced to use MySQL because the customer's chosen web hosting company only offers MySQL (this is common). While we were aware of the limitations of MySQL and tried to design our solution around them we hit one huge hurdle.

For some reason an append or update would take a long time to finish. During this period all other operations on that table are forced to wait. Since we wanted to store session information in a table and return it as users navigated throughout the site this was a hopeless situation.

It it also a clear example of using MySQL where it is not suitable. Concurrency is a huge issue for many web applications (since many users can easily interact with the system). InterBase or Postges would not have suffered this problem due to their Multi-
Generational transaction capability. This is how it should be.

Cheers

-- Adam Clarke, May 6, 2000

If you need transaction support then use a DB that provides it, but don't assume that an "ACID" sticker on box means you are totally safe!

Some issues to think about: a) ACID requires the ability to force the order in which writes occur to the hard disk. I heard that the linux kernel ignores the O_SYNC flag on opens. This can break the database when a server under load crashes. Smart / caching hard disk controllers also break the assumption. b) Another common false assumption is that failures will corrupt only recently changed data, and never touch older data / directory entries etc. I have experienced corruptions in files that hadn't been modified for a year (even under SCO opensever, which has a transaction based filesystem). c) ACID assumes that it's bug free, the file system code is bug free, ... d) Unless you have correctly matched up transaction start / end to what the user expects it to be, then you have again broken the ACID qualities. This especially breaks on multiple statement updates where autocommit is left on.

So real life tramples over the theory sometimes, it doesn't mean that theory can't be helpful in its place!

-- Ian Heggie, May 7, 2000

I still think that anyone who says that mySQL isn't good enough for large scale projects hasn't worked with databases long enough to be making that decision. Having worked in the Enterprise where Oracle was running, as well as in an environment where MS-SQL Server was running, plus vast experience in other RDBMS such as mySQL, etc. I can safely say that in at least 80% of all the projects I have worked on we have considered using mySQL over an Enterprise based system. In fact, currently I am working on a project which will use mySQL to tie together several AS/400's, a NT machine running SQL Server 7 and a website running a mySQL backend. This is in an Enterprise environment and I see no problems with using the mySQL server for this application. In the future we plan to move the databases on the AS/400's over to lower cost Linux systems with mySQL databases because it can be done. For those of you who are upset that transactions are not natively supported, maybe you should read the documentation, the FAQ, and the user posts on such places as deja.com, you will quickly find that transactions can be easily reproduced via code. I think people are focusing too much on the capabilities of the database system itself and not enough on their (obviously) lesser programming skills.

-- Brian Mueller, May 7, 2000

Ian, you're right in some ways. I don't know if you're right about O_SYNCH under linux. Postgres uses an fsynch() call explicitly on files at specific times, and this is supposed to be guaranteed to work on all variants of Unix, including Linux AFAIK. I know that Oracle had a problem with O_SYNCH on Linux when they first issued their release, but that was a bug in the Oracle file handling code which they quickly fixed.

Which of course brings us up to the fact that bugs in ACID-compliant RDBMS systems can break ACID-ity. Some folks (not you!) try to build some sort of inverse statement from this fact, implying that this means MySQL is actually safer than ACID-compliant
RDBMS systems for this reason. This is a strange argument, akin to saying that jumping out of an airplane without a parachute is safer than jumping with one, because parachutes occasionally fail.

Disk buffering’s a problem - fancy RAID exclosures include their own battery backup to ensure that cached data gets written on power failure. A UPS that warns the operating system will do the same, if you shutdown immediately (my little server will run for many, many minutes off its little UPS). Then, again, the UPS can fail.

Nothing can guarantee data safety in all cases - nuke your datacenter, and unless you have remote backups you're hosed no matter what!

An ACID-compliant RDBMS provides the foundation for safe data storage. MySQL provides nothing in this regard...

-- Don Baccus, May 7, 2000

How about using mysqldump for backups?

Sure it expands everything to ASCII, but I suspect it avoids some of the issues with binary copies of the files.

Does this avoid issues with getting an inconsistent backup of mysql?

-- Eric A. Raymond, May 7, 2000

You know... when I read the words "you will quickly find that transactions can be easily reproduced via code", I get a full-body shiver.

It it just me?

-- Jay R. Ashworth, May 7, 2000

If mysqldump does make consistent backups, then it most certainly should be used. It would also show that the MySQL crew understand why filesystem backups of an active database doesn't work, even if Michael Lykke doesn't.

I would check closely, though, to make certain that it does indeed make consistent backups rather than blindly trust it (RTFM, in other words).

As far as comments like this by Brian Mueller:

"I think people are focusing too much on the capabilities of the database system itself and not enough on their (obviously) lesser programming skills"

personal insults don't excuse your ignorance. The "workarounds" for transactions given in the MySQL manual don't safeguard data, and you don't have to be a rocket scientist, or even a particularly good programmer, to see why.

-- Don Baccus, May 7, 2000
My biggest problem with MySQL is the way that it is portrayed. It seems that MySQL is often mentioned the same way Linux is mentioned--as if it were the only free 'open source' product available. When a product gets so much press and alternatives aren't mentioned, people will make uninformed decisions. Part of the problem seems to be MySQL's representation of itself. The press that MySQL receives is another part.

Someone choosing software over using software ought to be important. Instead of talking about MySQL as the only solution, people should talk about why it suited their needs, and what strengths it has. Saying that it is the easiest and cheapest thing around simply isn't true--ease of use is a matter of perspective (as someone who used PostgreSQL before MySQL I found MySQL maddening to use), and if you have several pieces of software going for the same price (free), one of them isn't any cheaper than the others. Give accurate information ("MySQL is better at X") so that every MySQL user is one by informed choice.

-- Ann Barcomb, May 8, 2000

Reading all of this, I find myself in a quandry. I am relatively new to the database world, having only done it for about 3 years now. During those three years, I have used only MySQL. The reasons behind this were many and varied, but suffice it to say that some large corporations want to charge their sub-divisions upwards of $70,000 JUST TO TIE-IN to the existing Oracle server.

I never knew about Foreign Keys. I only recently (about a year ago) learned about sub-queries, etc. After buying some more books recently, I begin, as the wise man once said, to understand the term "The more I learn, the more I realize how little I know".

However, throughout all this ranting and raving I read here, many have lost site of one thing: MySQL, when combined with Perl/DBI, has allowed many a simple Perl programmer such as myself to create things for his/her web site which never could have been done by such a beginner with one of the large commercial db packages.

All the talk of rollback, etc. has me concerned. In the few scripts where I need control of execution of updates, based upon a pass/fail test or successful $sth->prepare() of a second or third statement, I simply set autocommit to off. Only when all statements can successfully be implemented do I issue a commit. I also keep a log of all update/insert transactions, using the built-in ability of mysql to do so. A poor man's rollback?

So far, locking hasn't been a concern as I always use autoincremented fields for ID, etc., and using the DBI.pm also helps out there considerably.

Maybe some day I will "graduate". Until then, I do things that I could never have done before without MySQL. And yes, I do them with mission-critical data. I just make damned sure that I do a dump every night (via cron) to a dated dump file. Getting only around 500 queries a day allows me to only do it once a day. I could do more if needed.

MySQL gave me an ability no other DB could do without extensive learning. It has it's place, and I for one intend to stick with it and grow with it.

-- Jonathon Robison, May 8, 2000

Could everyone please listen up?
The topic here is not "MySQL sucks". It's not. Go back up top and look. The topic is "Why *we* use PGSQL instead of MySQL, and why you should be *very careful* choosing MySQL for your project, and know exactly what you might be getting yourself into".

It isn't necessary to tell everyone how fantastico MySQL is, or how well it's been working for you; we got that. You'll note that none of the postings above that are favorable to PGSQL have been *lauding* it; merely noting that "these are the requirements of this problem space, and it fulfills those requirements".

So let us, shall we, shift down a couple gears, think about what we read before responding to it, and try to keep this a technical discussion rather than a personal one?

"Perhaps another choice would have been {safer,more productive}" is *not* the same thing as "You ee-diots! How could you make such a stupid choice?!" :-)

-- Jay R. Ashworth, May 8, 2000

I think its great the OpenACS team have elected to choose an Open Source database such as PGSQL. One of the things that has kept the ACS at bay for me has been the cost of implementing Oracle. I thoroughly enjoyed Philip Greepspun's book and highly recommend it. It will be great to now deploy the ACS without having to be concerned with licensing fees. Oh... AOLServer was not Open Source either until recently, but now we all have a great toolkint to work with:- Linux (*BSD with Linux emulation - AOLServer is still being ported to *BSD), AOLServer, PGSQL, OpenACS.

Thanks to all for helping in giving us a great free toolkit to work with...

-- Philip Jensen, May 9, 2000

This thread has provoked quite a bit of useful discussion. However, I have to take Brian to task for his inflamatory rhetoric.

Of *COURSE* MySQL is useful for certain applications. No one is suggesting otherwise. The sole purpose of this thread was to explain why MySQL was rejected as a choice for the OpenACS's backend. Aside from its technical deficiencies, it is also under a restrictive license that has prevented it from being included in Debian's main distribution. PostgreSQL has no such limitations. The jury is still out on Interbase.

I am pleased that in your "vast" experience you have never had a problem with MySQL in the enterprise. I sincerely hope you never do. However, to brush off the very real technical issues involved with correctly and ubiquitously providing rollback and data integrity code at every level of your application results in a flimsy architecture that will surely be a nightmare for your replacement to maintain.

I'm not sure why you take criticism of MySQL's technical features so personally, but I can only assume you have a lot dependent on its successful operation in the enterprise. To be faced with the realization that you have probably made the wrong choice must be a terrifying place to be. Caveot Emptor! :-)

-- Brent Fulgham, May 9, 2000

Slashdot is often offered as an example as to why real RDBMS features like transaction semantics with rollback, etc., aren't needed.
Here's a quote from an article by the Slashdot crew posted on April 28th. It's primarily describing their new hardware setup, but they talk briefly about increasing database reliability:

"As for fault tolerance, we're working on two fronts:

First we're funding development efforts with the MySQL team to add database replication and rollback capabilities to MySQL and that's coming along on schedule (and yes, these improvements will be rolled into the normal MySQL release as well)...(I know what you're saying 'Why not use Oracle...', well just because... you know... the zealousy thing...)

Note the fact that the Slashdot crew think that transaction semantics are so important that they're FUNDING DEVELOPMENT. The MySQL crew might not get it, but one of their most visible users do.

One source of our complaints regarding MySQL will disappear once they've finished this development effort...

-- Don Baccus, May 9, 2000

Several people use arguments about a "$20.000" Oracle license... I would suggest those people to take a look at the Free Sybase 11.0.3.3 for Linux. This version is indeed totally free. Both for use in development and production. It is a full featured database with all the needed features like transactions, rollback, stored procedures, triggers foreign keys, different ANSI isolation levels, on-line backups and much much more. This version also runs very stable on the FreeBSD platform if you prefer to use that as your server platform. I've used this version of Sybase (on another Unix platform) for 2 years in a banking environment without ever loosing 1 bit of information. If it's good enough for that, it should be good enough for backing of your website. I've looked shortly into MySQL, but I found the mentioned arguments enough to stay away from it.

-- Reinoud van Leeuwen, May 10, 2000

This comment is idiotic: "MySQL has carved out a very interesting niche: Raw speed and simple to setup/use/maintain database backend for online applications."

I'll tell you what, if I was able to strip out 30% of the functionality of enterprise-level RDBMS, they would be pretty damned fast too.

I have read many comments that basically stated that transactions are useless in most cases. That's insane. How can you ensure data integrity without transactions? Ok... most schemas center around one or a small group of table(s). All or most other tables directly or indirectly relate to that center. How do you intend on safely deleting a record in one of the center tables without transaction/rollback functionality? I think it's careless to write a series of DELETE SQL statements and assume that every one fires off okay.

And that's not even the end of it. Since MySQL doesn't support foreign keys, you would never know if you have orphan data from the half-executed batch deletion process above.

And I'm quite suspicious about MySQL's speed claims. How many concurrent connections were conducted during these benchmarking tests? Let's say we have 10 clients accessing the same table at the same time. EACH CLIENT HAS TO WAIT...
UNTIL NO OTHER CLIENT IS ACCESSING THAT TABLE! That means that the first client’s command might get executed quickly, but the 10th client doesn’t even get attention until clients 1-9 have completely finished their commands. This is pitiful.

How can you create a database system that doesn't support stored procedures? People who have commented about network traffic as being a moot point are crazy. When a call to a database is made from the Web, the majority of the processing time is the traveling of that command over the network, and then the traveling of the result (if any). So, let's take that batch deletion process again. I can make 10 DELETE SQL statements to get what I want done, or I can make ONE call to a stored procedure that executes that batch deletion and returns one "everything is okay" result. Which do you think is faster?

And this is what kills me. MySQL's development team hangs on the speed factor. But there are a HUGE number of standard features that could greatly increase speed. There's a different between "SELECT VERSION();" and a 15-command deletion. They don't seem to realize that.

Speed isn't everything.

-- Matt Warden, May 10, 2000

Just wanted to point out the erroneous statement by Ben Adida:

"In fact, we use PostgreSQL. It's truly Open Source (which MySQL is not), and it's ACID compliant (which MySQL is not)."

Philip Greenspun says in his article:

"As of November 1999, however, there were a few flies in the PostgreSQL ointment, most glaringly lack of OUTER JOIN and no enforcement of referential integrity constraints (i.e., PostgreSQL can't be said to do the "C" in ACID)."

So Ben should really be saying that PostgreSQL is AID compliant.

I don't know if it's still at this level or not, but since PostgreSQL is being analyzed with a late 1999 mySQL we should stick to the same time period for comparative purposes.

-- Christopher Lindsey, May 11, 2000

1. You can use triggers to enforce referential integrity even if the foreign key syntactic sugar is missing. PG has supported triggers for a couple of years now, MySQL doesn't because "they're too expensive" (to paraphrase their manual).

2. Ben's comparing the latest RELEASE VERSIONS of the two databases. PG 7.0, which is the latest release, supports the foreign key syntactic sugar, too. Is there any particular reason we should compare a pre-release version of MySQL with a release version of Postgres? The current Postgres sources have code for infinite-size types with no need to use nasty words like CLOB or BLOB, yet we don't crow about that...should we?

-- Don Baccus, May 11, 2000
At the time that this page was written (May 4), the latest release version of PostgreSQL was 6.5.3 (released November 5, 1999 -- the same time that Philip Greenspun wrote his article referencing PostgreSQL as non-"ACID" compliant.

Version 7.0 was released on May 9th, so at the time that this article was originally published on the Web, 7.0 was still a pre-release candidate.

I'm not arguing that one is better than another; frankly, I could really care less. :) I'll use whatever works for my needs, whether it's a flat text file, DBM file, or PostgreSQL. I just think that if Philip Greenspun's article is going to be used as the crux of the argument, then his words should be represented truthfully.

-- Christopher Lindsey, May 11, 2000

Ben quotes Philip's paraphrase of the definition of ACIDity. Take a very close look and you'll see it doesn't mention foreign key constraints at all:

"A transaction is legal only if it obeys user-defined integrity constraints. Illegal transactions aren't allowed and, if an integrity constraint can't be satisfied the transaction is rolled back."

6.5.3 meets that definition. User definable constraints which aren't met result in transaction rollback, and that's the truth. There is nothing in Philip's definition (nor the standard definition which he's paraphrasing) that says user-defined constraints have to be of a certain form. A constraint defined via a trigger is a user-defined constraint, and can rollback the current transaction.

Foreign key constraints are an SQL-ism. ACID-compliance isn't dependent on SQL-compliance no matter what you, or for that matter Philip, might think. In fact it is relatively recently that SQL has come to dominate the RDBMS world.

Did SQL89 even have foreign keys? I'm pretty sure the answer's "no".

Indeed, Oracle doesn't implement all of the SQL92-defined foreign key semantics. Postgres V7.0 implements more of the standard than does Oracle. If this were the criteria for ACID-compliance, neither Postgres nor Oracle meet the test, though Postgres is closer. Fortunately, this is not the test, and the fact that foreign key support in Oracle is anemic compared to that provided by Postgres V7.0 doesn't mean that Oracle is ACID-compliant. Note that Oracle's anemic implementation of foreign key constraints means that the user must resort to the use of triggers to enforce consistency for complex multi-column foreign keys.

I'm not quite certain why you're bothering with your foolish attempt at hair-splitting. The fact is that MySQL doesn't implement transaction semantics at all, while Postgres (and Oracle) do along with a rich set of user-definable constraints and a trigger mechanism that lets the user extend the pre-defined constraint system to their heart's content. Both pre-defined and user-defined (via trigger) consistency checks can roll back the transaction if conditions aren't met.

Perhaps you're of the mind that unless Philip says something is true, it's not. Sorry, that presumption is false. The fact is, Philip was wrong in his stating that Postgres 6.5.3 isn't ACID-compliant because it doesn't provide foreign key constraints. Philip's often right, but he's not always right.
Now that certainly was an interesting discussion...

I'm sorry, but the tone of the initial statement was clearly unfavorable to MySQL. Calling it a glorified file system with an SQL interface doesn't accomplish anything except annoy the people for whom the thing works, and works extremely well.

A system that handles critical data is of no use at all to me if it can't handle the kind of load I subject it to.

If monty@mysql.org is to be believed, they did ask the postgresQL people to please send in additions to the benchmark which would show off the strengths of their server, but the only feedback they got was continued whining about MySQL's unfair benchmark and crashme tables.

Sure they're unfair! It's their product and they don't get any money (or ego-boost, for that matter) from writing tests that capitalize on stuff their database can't do yet.

If you want a more balanced comparison, send them a few new test cases. They promised to put them in.

A final side note to the postgresQL people -- add a row for MySQL to your comparison table. The argument that it's no true RDBMS doesn't hold water. Neither is mSQL, and it's been in that table for ages. If _your_ argument demonstrably doesn't hold water, accusing others of warping the facts is just childish.

Thus endeth this message.

-- Matthias Urlichs, May 16, 2000

"Calling it a glorified file system with an SQL interface doesn't accomplish anything except annoy the people for whom the thing works..."

It's an accurate description. If accuracy annoys people who use it, the hell with them. We're supposed to avoid being accurate in order to avoid hurting people's feelings? That's silly. Ben's description of mySQL is far more accurate than the technically flawed arguments the mySQL folk give for transactions being useless, foreign keys being harmful, etc etc. If I were a mySQL user, I'd be offended by knowing that the creators if mySQL lie to their users, rather than find offense in accurate comments made by an outsider.

Your mileage apparently does vary, though. If you prefer the lies and misrepresentations of the mySQL crowd to an honest technical assessment, it ain't no skin off my back.

This misrepresentation (or accurate reflection of technical ignorance, take your pick) by the mySQL folk is inexcusable, regardless of whether or not mySQL works for you.

The Postgres folks HAVE sent in corrections to the mySQL folks, along with requests that they test with versions newer than two years old, and the mySQL folk have ignored the requests. Don't presume facts not in evidence...
I find it interesting that you leapt to the conclusion that the chart is in error because the Postgres folk haven't asked the mySQL folk to update it, rather than due to the fact that the mySQL folk apparently can't be bothered to make the chart accurate. Given the misrepresentations of technical issues that permeate the rest of their documentation, why would one expect the comparison chart to be accurate?

-- Don Baccus, May 21, 2000

I'm curious about Don's comment earlier this month:

"...if you want to know a dirty little secret, parts of the ACS break atomicity due to a gross and pervasive programming error, making Oracle an expensive version of MySQL, as I put it when I reported the problem (it's being fixed, and if it doesn't get fixed right by aD we'll fix it ourselves)."

What's the story here? Any further information about this error?

-- Helen Sonner, May 29, 2000

I raised this issue in web/db a couple of months ago. They supposedly will weed it out for version 4.0. If they don't, we (OpenACS) will. You can find an explanation of the problem over there (at http://photo.net/wtr).

-- Don Baccus, May 30, 2000

-->

-- asdf asdf, June 5, 2000

As mentioned in the above comments that we can use mysqldump for backups, but it should be noted that whenever you will use mysqldump you are not guaranteed that a "transaction" has been completed as there is no transaction-control.

While is PG, the backup utility make it 100% sure that the backup is in consistent state, but this cannot be achieved using mysqldump utility.

-- Asad Yaqoob, June 8, 2000

After investigating the matter it has become clear to me that MySQL is not an ordinary relational database but a key element in a plot being hatched by a global conspiracy of Communists, Homosexuals and Ethnic Minorites.

Thank you.

-- Michael Ellis, June 13, 2000

The article says:

"This quick paper attempts to explain why MySQL is not just the wrong choice for OpenACS, but how it should never be used by any system that handles critical data."

and

"If what you want is raw, fast storage, use a filesystem. If you want to share it among
multiple boxes, use NFS. If you want simple reliability against simplistic failure, use mirroring. Want a SQL interface to it all? Use MySQL."

Brent Fulgham wrote: "Certainly there may be cases in which transactions are not important, but as Ben notes then you are really talking about a filesystem and not a true database. Many of these sites could probably be built using flat text files, in fact."

So someone *can* use critical data with MySQL as he could also use flat text files. After reading the article, the main point I remembered was the "Do not use MySQL with critical data" part! That should be cleared.

-- Thomas Michel Wanka, June 16, 2000

Contrary to Ben Adida's protestations, this discussion is not just about MySQL's deficiencies or T.c.X's attempt to oversell MySQL's capabilities. It's also about Ben Adida's hyperbole.

I encountered a paragraph in an article on REBOL in DJJ that made me think of MySQL's documentation:

"Communication is the exchange and interpretation of information. To communicate, we need to share a common language. However, in many cases we are dealing with a specific domain, where a dialect can provide greater productivity Â…"

The technical language used by programmers contains well-defined, publicly-known terms. However, specific groups of programmers and even specific shops tend to develop their own dialects. Distinctions that aren't useful for particular groups or organizations tend to disappear, and some terms change their definitions in ways that ease communication within a given organization. Witness the way that VB programmers use "object" to mean both class and object.

In the case of T.c.X., reading the MySQL documentation makes it clear that they have their own dialect of the technical language used in the database field. This isn't surprising; their requirements are sufficiently unique that they felt compelled to develop their own technology (MySQL), and their workforce seems to have little turnover. (The chief architect of MySQL has been with the company for over two decades.) A group of people working together for a long period of time and doing a unique job will inevitably develop their own terminology for describing what they do.

As a case in point, look at the following phrase from the MySQL documentation: "Note that foreign keys in SQL are not used to join tables Â…". Of course this is wrong. SQL statements use foreign keys to join tables. According to my references, that's part of the definition of the term "foreign key". But T.c.X. has narrowed the definition to cover only foreign keys declared as part of the database structure.

With the phrase "atomic operation", it's not quite as clear what the MySQL documentation means. But it is clear that they need only a subset of the requirements that atomicity covers, and they appear to have adopted the word to suite their needs. (This is speculation on my part. I have no contact with anyone at T.c.X. and don't know how they came to use the word in this sense.) There's no evidence that this is an attempt at deception. T.c.X. has offered MySQL for free, and their free documentation is good enough that most people won't need the for-pay support services T.c.X. offers. Anyone who needs more that MySQL can deliver and who is qualified to do the necessary development can read the manual and get a good idea of what MySQL won't do. It's not
likely that very many people have lost data because they didn't understand MySQL's limitations. I'm sure it has happened, but I would imagine that it has happened to people who didn't understand databases. MySQL is not for beginners. (But the combination of inexperience and critical data always dangerous, regardless of the DBMS.)

If you want to see an example of deliberate deception, look that the marketing material Microsoft has distributed comparing SQL Server with Oracle. Unlike Microsoft, T.c.X. is very straightforward about what their product offers.

In fairness to Ben Adida, anyone attempting to communicate with the world at large has a responsibility to avoid dialect and use words with their standardized meanings. Why T.c.X. didn't do this I can't say. It's possible that they've been sufficiently isolated by their unusual requirements for a sufficiently long time that they forgot that they were speaking a non-standard dialect.

At any rate, calling MySQL a file system with an SQL interface is absurd. If you look up "relational database", the definition is independent of computer technology. All that matters is how the information is organized. Whether the DBMS helps enforce relational integrity or not is irrelevant. As long as the DBMS depends on relational integrity, it's a relational DBMS. Adida seems to have siezed upon the fact that MySQL uses the OS's file system to create its databases and tables, and used that in an unwarrented way to belittle MySQL. Any reasonable person recognizes that the term "file system" is not intended to cover information systems that include C libraries of data access and indexing routines and an SQL interface. Unless, of course, Adida is using a non-standard dialect.

As Adida says, MySQL lacks subqueries and stored procedures. However, many people have found that their websites give better response times with MySQL. The "significant" reduction is speed apparently isn't all that significant for many applications.

MySQL lacks triggers and foreign key constraints. This is made clear in the documentation, and people who need triggers and foreign key constraints don't use MySQL. This leaves a very large number of people who don't need them and use MySQL.

MySQL has only table-level locking. Of course this increases contention under high concurrency and with write-intensive applications. If this is "pathetic", why do so many web developers find that their projects run faster on MySQL?

Adida writes " An enterprise-level system will never compromise certain features for speed." I've read that most Oracle databases are set at READ COMMITTED rather than SERIALIZABLE. This means that many enterprise-level systems are sacrificing some ACID compliancey for speed. On a busy web site, slow response times will drive away customers. While the data may be critical, keeping customers may be more critical. Businesses that use slower, fully ACID compliant systems may be asking for trouble.

MySQL is suitable for the sorts of applications T.c.X. uses it for, which includes critical data maintained by skilled administrators, and it is very good at blasting out web pages at a high rate of speed. If that's what you need, MySQL has proven itself. If you need more, consider Interbase. Once version 6 gets past beta, it will be free and be backed by sixteen years of development and a strong track record. PostgreSQL still has kinks and a very short history as a production tool. The development team has done an outstanding job of cleaning it up and making it useful, but the job isn't quite finished. Unless you can do enough testing to assure yourself that it does what you need, I'd adopt a wait-and-see
attitude towards it. I'd adopt the same attitude towards anything that uses PostgreSQL as a back end.

-- Bob Hall, June 23, 2000

There are enough misleading comments on this page to warrant a follow-up article, which I will put up shortly. However, I'd like to immediately respond to the above comment, which I find rather surprising.

Whether or not the MySQL folks are *purposely* misinforming the world is irrelevant to me, and should be irrelevant to this discussion. What matters is that their material *is* incorrect, regardless of how much good-will there is behind their effort. At the end of the day, technology isn't about warm, fuzzy feelings. It's about building solid tools through solid engineering principles. It's about using commonly accepted definitions and standards, and not making up one's definition for "atomicity." I'm reminded of Microsoft's claim that Windows 98 has "memory protection" or that IE is "standards-compliant."

Using incorrect definitions is bad engineering. The MySQL people are surely great, fun folks, and I'd probably enjoy going out for a drink to chat with them. But they are misinforming the public, and that is not acceptable.

-- Ben Adida, June 23, 2000

Ben missed my point, and it's probably my fault for being too discursive.

First of all, someone (not Ben) accused T.c.X. of lying. I pointed out that we all misuse technical terms occasionally, and sometimes consistently, and for very good reasons. We all have our dialects, and these dialects help us do our jobs.

In the case of T.c.X., they should have been more careful, but it's hard to believe that they've benefited from what appears to be unintentional misuse. They really don't benefit by referring to "atomic operations", since the documentation makes it clear that MySQL can't support atomic operations and they're really not trying to make a lot of money from MySQL anyway. They've also used their own definition of "foreign key", which can't possibly give them any sort of advantage. The evidence is that they're using their in-house dialect, rather than lying. Their sin is carelessness, and we're all occasionally guilty of that.

In Ben's case, he also seems to be speaking a non-standard dialect. In his clarification to his original comments, he wrote "MySQL claims to be an RDBMS when it is far from deserving that title."

An RDBMS is a DBMS that can combine data from two tables to create a third, result table. That's not my personal definition; I looked it up in a dictionary of computing terms and a database book. This definition seems to be the standard definition. It is publicly known and is used by the people who specialize in database science. As near as I can tell, it is based on Codd's original work, although I'm not enough of a scholar to confirm that independently. MySQL fits this definition, and is therefore an RDBMS. Even if you expand on this definition by requiring that the result table be created through joins between referenced keys and foreign keys (as opposed to the pointers used in older DBMSs), MySQL still fits the definition. Foreign keys are part of the definition, but foreign key constraints are not.
Ben and someone else referred to MySQL as a file system with an SQL interface. File systems store unstructured data and SQL won't work with unstructured data. (Again, that's not my personal definition of "file system". I looked it up.) Furthermore, the data has to be structured a particular way before SQL will be able to function. A data store that can be created, maintained, and accessed with SQL statements is a relational database, and a DBMS that uses SQL to create, maintain, and access databases is an RDBMS. This follows from the publicly acknowledged definition of an RDBMS.

If Ben is arguing that a general purpose RDBMS *should* include features like foreign key constraints, I agree completely. I don't agree with the statement that businesses never sacrifice integrity protection for speed. Businesses don't maximize, they optimize. Some degree of trade off is common. Businesses don't have the resources to insure themselves against all forms of risk, and they are forced to manage risk rather than eliminate it. Sometimes the optimum level of protection is not the maximum level.

MySQL is obviously not a general purpose RDBMS. It is wonderful tool for some purposes and an entirely inadequate tool for others. PostgreSQL is a general purpose RDBMS, and I think the team that developed it did an incredible job. I don't believe it's been used in enough applications over a long enough period of time to be trusted with critical data. I expect that I'll have reason to change my mind in a relatively short period of time, i.e. I expect PostgreSQL to prove itself a reliable production tool after the last kinks have been worked out and PostgreSQL undergoes a brief shakeout period. My statement that I was taking a wait-and-see attitude wasn't intended to be the least bit disparaging, merely cautious. If Ben amends his comments to say only that one should also be cautious in using MySQL, then we will have nothing to disagree about.

At any rate, I'd like to thank Ben Adida for sparking a really interesting discussion.

-- Bob Hall, June 26, 2000

They've also used their own definition of "foreign key", which can't possibly give them any sort of advantage. The evidence is that they're using their in-house dialect, rather than lying. Their sin is carelessness, and we're all occasionally guilty of that.

I suppose I should give this whole topic a rest, but the above comment is far too charitable.

The mySQL people know and use the standard definition of "foreign key" and the semantics and intent behind foreign key constraints. I know this because they've written, and I've read their documentation claiming that such constraints are not only unnecessary, but bad.

In regard to whether or not they lie on other issues, well, one can only guess. There's a pattern of either genuine ignorance - which is hard to believe, since they seem to be fairly good hackers - or genuine dishonesty in their writings. If they're ignorant, should you feel better about using their product? Personally, I'd almost rather they be dishonest - if true at least it means they're technically competent and actually understand the issues.

But perhaps they're really honest and so ignorant that they shouldn't be trusted with the writing of "Hello, world!". If so, why should you trust them with the implementation of a database engine?
Regarding the term "RDBMS", Codd invented the ACID nomenclature as well as the relational model for organizing tables of data. An RDBMS is understood to be ACID compliant.

More to the point, ACIDity is not simply assumed by the SQL standard, but is the foundation upon which the standard is built. Whatever mySQL is, it is not an SQL RDBMS. If you don't believe me, I've got the SQL92 and SQL3 drafts available for downloading via ftp. E-mail me if you want a copy.

The mySQL people would like you to believe that atomicity, transactions, etc are all a minor bit of the standard implemented only by overblown commercial database systems, but it ain't the truth and I think they know damned well it's not the truth.

-- Don Baccus, July 9, 2000

The mySQL people know and use the standard definition of "foreign key" and the semantics and intent behind foreign key constraints. I know this because they've written, and I've read their documentation claiming that such constraints are not only unnecessary, but bad.

I made the point that the MySQL people consistently use the term "foreign key" when they mean foreign key constraint, and they do it under circumstances where the confusion offers them no advantage. I did not say that they didn't know the standard definition of "foreign key". I have no way of being sure whether they know it or not, although I assume that, as database professionals, they have learned it. It is obvious from the MySQL documentation that they understand the concept and can join tables using foreign keys. What I did say was that they were using the term "foreign key" when they meant foreign key constraint, and I stated that they did this so consistently that it appeared to be part of their in-house dialect.

Like Mr. Baccus, I have also read their documentation, and I reread it after reading his response. The term "foreign key constraint" never appears in the manual. They always use the term "foreign key" to refer to the constraint. They have not said that foreign key constraints were bad, but they have said that they were unnecessary. They describe some of the databases that they've worked with, and these are research databases with sociological data. They don't need to update records very often, and there's none of the high volume transaction processing characteristic of many business databases. They do need to access the records quickly. The DBMS they've produced perfectly matches their experiences and needs, as they've described them. On rereading the documentation, I found no evidence to support the claim that they are either lying or incompetent. If Mr. Baccus believes that such evidence exists, he should produce it.

Regarding the term "RDBMS", Codd invented the ACID nomenclature as well as the relational model for organizing tables of data.

I said that the generally accepted definition of an RDBMS was based on Codd's early work. In his early papers there is no mention of ACID or anything resembling ACID. I haven't read his early papers, but I've read enough secondary material analyzing his early papers to be confident about that claim.

An RDBMS is understood to be ACID compliant.

I'm prepared to produce citations showing that this is not the case. I don't have them handy at this moment, but I can post them if Mr. Baccus thinks I'm lying or incompetent.
Three sources that I have used recently and can cite are: a dictionary of computer terms, a recent textbook on relational databases, and C. J. Date's summary of how Codd's early papers describe relational databases. I can look up the titles and authors, if that's important.

_More to the point, ACIDity is not simply assumed by the SQL standard, but is the foundation upon which the standard is built._

The following is taken from the introduction to the SQL92 standard:

_The first SQL standard, in 1986, provided basic language constructs for defining and manipulating tables of data; a revision in 1989 added language extensions for referential integrity and generalized integrity constraints …_

The committee that produced the first standard didn't consider ACID a fundamental, or even necessary part of the standard.

The foundation of SQL is relational algebra, which doesn't include integrity constraints or ACID. The first language based on relational algebra was Codd's sublanguage ALPHA, which also did not include integrity constraints or ACID. (It was never implemented as a computer language, but it included the features that Codd considered important at the time.) ACID is neither the foundation of SQL nor a necessary part of a language based on relational algebra.

_Whatever mySQL is, it is not an SQL RDBMS. If you don't believe me, I've got the SQL92 and SQL3 drafts available for downloading via ftp. E-mail me if you want a copy._

SQL RDBMSs were first produced in 1979, before there was an SQL standard. The SQL standards are not intended to provide either an invariant definition of SQL or a definition of an RDBMS. They are a cooperative and constantly evolving effort by DBMS vendors to create a basis for further development. Whether or not MySQL meets one or more levels of compliancy for either standard is irrelevant. The only practical way to set up, maintain, and access a MySQL database is through the MySQL implementation of SQL. However deficient it is, it is still SQL.

_The mySQL people would like you to believe that atomicity, transactions, etc are all a minor bit of the standard implemented only by overblown commercial database systems, but it ain't the truth and I think they know damned well it's not the truth._

TcX used the C libraries underlying MySQL for two decades without recourse to an SQL overlayer. Apparently their needs are fairly rudimentary. There's no evidence that they're lying, and there is evidence that they haven't needed atomicity, transactions, etc. Of course, other database professionals have had other experiences, but that doesn't make the MySQL gang liars. They are enthusiastic about their product, and sometimes make unsupportable statements. An excess of enthusiasm sometimes leads Ben Adida and Don Baccus to make unsupportable statements. I confess to having done the same thing myself, on occasion. I have no reason to consider any of these people liars or incompetents.

-- **Bob Hall**, July 13, 2000
Well. After spending a good hour pouring through the all these comments, I'm pretty blasted - but thought I'd add my views. To be honest, I think that from about a quarter of the way down the argument seems to have become more personal than constructive.

Whether or not MySQL is an RDBMS is subjective, quoting papers from times gone by is of limited use - in fact the general argument whether or not it is an RDBMS seems of limited use.

At the end of the day, MySQL can serve a purpose, just like text pad does despite the availability of Word 2000. Why do I use text pad (sometimes :-}) - because its quick and does the job. I don't use it when I'm writing a letter to a client, my requirements are obviously different, however, when I'm jotting down a few ideas its perfect.

This argument can be applied to MySQL and, say, MSSQL (of which I'm just playing with 2000 beta and I'm quite impressed after long being a Oracle fan). For simple read-only/small web sites, where speed is the primary concern, and data integrity is not an issue its fine. For an advanced e-business site - well of course not.

Incidentally, I read a report recently that suggested that in a dbase test, postgres was 10x slower than MySQL. However, more interesting, was that Interbase 6 was only 4x slower. Um.

-- Ian Renfree, July 20, 2000

Well, well Mr. Hall I don't feel like quoting the entire SQL standard to you, but suffice it to say that MySQL doesn't meet ANY level of compliance with SQL92. And without ACIDity, it can't. You don't feel that this is important, well, you're entitled to your opinion. In my opinion, if an engine can't meet the lowest level of SQL92 compliance, then it ain't SQL. Any other point of view makes the standard, and the whole process of developing standards, meaningless.

Hmmm...maybe MySQL is really C, after all, it doesn't even remotely comply with the standard. Why not, if compliance isn't an issue in the naming of a product?

You make the following comment:

Like Mr. Baccus, I have also read their documentation, and I reread it after reading his response. The term "foreign key constraint" never appears in the manual. They always use the term "foreign key" to refer to the constraint. They have not said that foreign key constraints were bad...

You are correct that they use the term "foreign key" frequently when they mean "foreign key constraint" (just one more sign of ignorance on their part). Have they never said that such contraints are bad? Perhaps you read the following sentence, pulled from their documentation, differently than I do:

There are so many problems with FOREIGN KEYs that we don't know where to start.

They mean foreign key constraints, of course. You are right that the sentence doesn't contain the word "bad", but damned if I don't read it as a statement on their part that foreign keys are bad. The list of items following that comment make this reading of the sentence even more reasonable.
Incidentally, I read a report recently that suggested that in a dbase test, postgres was 10x slower than MySQL. However, more interesting, was that Interbase 6 was only 4x slower. Um.

Database benchmarking's tricky. The best I've seen recently, done by an independent evaluator on Postgres 6.5.3, showed that for a single user, Interbase was about twice as fast as Postgres on a large-scale industry standard benchmark. However, Postgres throughput was about twice as fast as Interbase under heavy load. Postgres reached maximum throughput with fifty jobs running concurrently on the testbed platform (256MB dual-P450 Linux box with UW2 disks).

MySQL, of course, couldn't run the tests, so there are no figures for it.

Now ... throughput under load is what most folks with busy websites are interested in, and the above demonstrates that Postgres scales very well under load. Postgres 7.0 also tends to be noticeably faster than version 6.5.3, at least for my queries on my tables (I've got about a million rows in one of the table supporting one of my personal websites).

Of course, OpenACS is also interested in supporting InterBase, now that it's Open Source - it's very good.

As has been mentioned more than once above, the table-level granularity of locking implemented by MySQL is bad news for scalability for websites, which is why users using it for high-volume sites jump through implementation hoops to get around it, using additional tables, copies of tables, etc in order to avoid readers from blocking. The need to do this kind of thing manually will wipe out much of the apparent speed advantage enjoyed by MySQL on simple queries.

Oracle, Postgres, and Interbase all offer row-level locking granularity, and writers never block readers, a big win for systems with many simultaneous users.

I'm probably throwing some fuel on the fire here, but GreatBridge, which is helping out with PG (Postgres) development issued a press release today about the performance of PG 7 compared to major databases, MySQL, and Interbase.


In summary, PG did really well, and scaled well past mySQL. All testing was performed on a Compaq ML350, which is a dual PIII box. With 100 concurrent users, PG 7 achieved 1070.4 TPS, and mySQL achieved only 117.87 TPS.

Enjoy,
Mark Radulovich
eBusiness Architect

I just came out of a .COM where we used MySQL for some intranet stuff and Oracle for the company jewels. MySQL was okay for batch applications, or straight forward
applications with limited concurrent users. But if you have 4 users waiting for 1 to update the member table, the net gain in MySQL performance is lost in gross wait! And if you regularly use subqueries, you quickly bang your head against the wall.

For any organization where the database IS the main asset, the more robust functionality found in Postgres is needed for insurance. If Postgres engine speed is really bothersome, consider more/faster CPUs and memory.

There's an anecdote for the debate from the commercial RDBMS market. 5 years ago, the data warehousing RDBMS Red Brick (now Informix) used many of the same rationale in competitive situations against Oracle - As it was optimized for data warehousing, much of Red Brick's speed came from a combination of star and bitmap indexing, and also, from no transaction logging overhead. Without Rollback or Commit, it screamed for both queries and loading.

But it's optimization for read-only environments limited its market appeal where DBA's wanted data recovery capabilities (logging) and versatility for use in DW AND OLTP apps, e.g. row-level locking was required. Oracle introduced knock-offs to Red Brick's innovation by 97 with the result that Red Brick never emerged as more than a niche player. MySQL would do well to make transactions a priority.

By the way, login says cookies are required for session management. This not true as a general statement. Maybe it's a limitation of Tcl, but with Php, you're able to pass sessionid as name-value pair in URL.

Chris Straka

-- CB Straka, August 14, 2000

Speed vs ACID vs Automatic Recovery?

Much of this debate has focused on the perceived tradeoff between speed and simplicity (MySQL) vs transactional integrity (Postgres).

The GreatBridge benchmarks mentioned above (which claim that Postgres scales better than MySQL even for read intensive applications)raise the possibility that with the release of version 7 Postgres may win on both counts.

Do these benchmarks compare with people's practical experience with Postgres in the field? If Postgres 7 is really faster as well as safer, is there any case for MySQL for any web application?

But now that Interbase is Open Source, I would appreciate hearing the views of the group on another mission critical issue. While the benchmarks claim that Postgres also scales better than Interbase under load, Interbase claims to offer much more robust self-recovery after a system crash or failure. With our servers located 6000 miles from our office, this is an attractive feature. Where 24/7 availability is an issue, what are the pros and cons of Postgres vs Interbase?

Geoff Caplan

-- Geoff Caplan, August 17, 2000

It is clear that the writer was really very angry at someone when he wrote it - certainly
not objective. Obsessive about this concept he calls ACID, which is just another word for "Like my Database".

Atomicity and isolation, etc are not irrelevant concepts. However, they can be solved in different ways. They can be solved in the database, yes, but also in the application (which is what we did before the databases delivered it). Many times this is easier.

Is this guy afraid of loosing his job? Take a step back and look at it objectively you will see that MySQL is the right tool if you use it right. Oracle and other databases have a different approach. That's all. If you take into account what it can't and cannot do, I can make an application that works perfectly safe with financial transactions on MySQL.

Koos Bezemer

-- Jacobus Bezemer, October 3, 2000

Oh, and by the way,

PG is only available on Unix?

MySQL (as well as DB/2, Oracle, etc) are also available on NT, more and more the platform of choice now that it's growing up.

Unix is becoming less and less open and more proprietary and, compared to NT and platforms like the AS400, less and less attractive. Why limit yourself to this dying concept?

-- Jacobus Bezemer, October 3, 2000

PostgreSQL runs on NT.

As to E. F. Codd and ACIDity, I now quote the book "The Practical SQL Handbook", by Judith S. Bowman et al:

"Dr. E F Codd, the inventor of the relational model, has developed a detailed list of criteria that implementations of the relational model must meet. A comprehensive explanation of this list, often called "Codd's Rules" would introduce terminology and theoretical issues not really within the scope of this book. ... Here, we summarize the features of Codd's twelve-rule test for relational systems and use these, in combination with Date's more basic definition, to come up with a more general definition. To be considered fully relational, a relational database management system must

- Represent all information in the database as tables
- Keep the logical representation of the data independent from its physical storage characteristics
- Use one high-level language for structuring, querying, and changing the information in the database...
- Support the main relational operations (selection, projection, join) and set operations such as union, intersection, difference, and division
- Support views, which allow the user to specify alternative ways of looking at data in tables
- Provide a method for differentiating between unknown values (nulls) and zero or blank
Support mechanisms for integrity, authorization, transactions, and recovery

Seems simple enough to say that MySQL does not meet those criteria, but PostgreSQL does meet those criteria, more or less. Some of the set operations are not terribly complete, nor do we have outer joins in 7.0, but the basic stuff is there. MySQL fails on two of those points: No views, and no transactions or integrity (referential integrity, BTW).

-- Lamar Owen, October 11, 2000

Unix is becoming less and less open and more proprietary and, compared to NT and platforms like the AS400, less and less attractive. Why limit yourself to this dying concept?

You know, that's not bad, as trolls go.

I don't think you'll find any sympathy for that position, in light of the expansion of Linux (among others) to run on everything from Palm Pilots to IBM 390s. Can *your* OS do that? :-)

Keep trying...

Cheers,

-- Jay R. Ashworth, November 7, 2000

I think, yeah right, MySql doesn't probably pass the exams as a full, 99% reliable RDBMS system.

Then again, as mentioned by somebody earlier, Postgres would get the utilisation to unacceptable levels on our linux box (512MB RAM/Pentium III 750MHz)

MySql did (and does) the job for our few web sites. I do have to run a backup every night and I do have to check the data integrity. I also had to buy a UPS (I could possibly buy 100 of them for the money I would pay for Oracle.)

Finally, who of you are running any RDBMS's on a machine not connected to a UPS and with no daily (or nightly I should say) backups?

Having said all this, I do agree that MySql doesn't provide what Oracle or Postgres do. One needs to assess the features depending on the nature of the site. I have suggested both Oracle and Postgres to various customers for reasons that have been explained elsewhere in this forum.

But much of the critique seems a bit over the top, particularly because MySql does now support update logs and master/slave replication and because being file based a backup is fairly easy.

-- Petros Diveris, November 9, 2000
Well ... what about MS SQL 7.0?

It is an ACID RDBMS? Locking of rows for read.write operations? Can anyone tell me?

Aurelian POPA

-- Aurelian POPA, November 19, 2000

Well, those of you convinced that PG will drive your system resources through the roof on busy systems ought to read Tim Perdue's article (referenced below). As has been pointed out endlessly here, just because a simple system like MySQL does a single, simple query quickly is not necessarily evidence that it will scale well. Which is exactly what Tim's discovering as he tests with the SourceForge sources and real data dumped from the active site. Yes, the bookkeeping required to track transactions does indeed add overhead to simple queries. It also allows for PG to implement an extremely high level of concurrency, i.e. row-level locking (exposed to the programmer via "select for update", BTW, as well as used internally during transaciton processing). It is the high-level of concurrency that leads to scalability.

As far as MS SQL goes, yes, it is a true RDBMS (based on an older version of Sybase, which some folks swear is the easiest of the commercial DBs to administer) complete with transactions, etc. All of the commercial ones are, it is only MySQL crowd which considers transactions unnecessary and foreign key constraints harmful.

-- Don Baccus, November 19, 2000

One thing for the MySQL people saying PostgreSQL needs to be restarted every 5min and that kind of comments: It may be I never used PostgreSQL 6.5.3, but I've got PostgreSQL 7.x running in my Unix (not Linux, though) workstation working flawlessly for about 18mo. Though I know some people have reported huge memory leaks, I have not seen that occur in my workstation yet.

In any case, though I am forced to use MySQL somewhere out of my own desktop at home, I would not recommend it for anything but a address/phone lookup service, and I'd certainly not recommend it for anything that would need more than 1 single user inserting data at a time -- It's just not safe. Even if MySQL compiles just fine on my system, and there are recent versions around already compiled for my machine, and I had to hack a couple of assembly code lines and a couple of makefiles to get PostgreSQL working on my system.

If you want a ready-to-use RPM or pkg or tardist file, then just go use what someone else compiled and packaged for your system before, but if you can afford compile your open source database package at home, then you don't need to use MySQL. And if you care about data integrity, then you simply should not.

-- Marcelo Paredes, January 4, 2001

Well, the memory leak issue was real enough until PG 6.5.x, which was released 18-20 months ago. Some huge leaks were found and fixed. Other, comparatively minor, leaks continue to be fixed. Bad? Sure, but leakage is extremely minor and fixes over the past several months have been for leaks triggered by errors. Is MySQL 100% leak-free? If you believe that, you probably believe that OJ was innocent...
This is just another example of how PostgreSQL's horrible reputation from past days - a well-deserved reputation - continues to dog it today, when it has matured into a very reliable and useful RDBMS.

-- Don Baccus, January 9, 2001

I have just read this entire serial diatribe to see if anyone addresses my observation. They don´t. I sure hope someone with greater wisdom than myself still checks back here now and then to fill any gaps.

It has been my observation that, based on merely a half dozen eCommerce databases, _ALL_ eCommerce database designers use SQL as a glorified ISAM, or simply a spreadsheet. Joe Celko wrote, "A normalized database will tend to have a lot of tables with a small number of columns per table. ... People who have worked first with file systems ... tend to design one monster file for an application and do all the work against those records." "Monster file" describes _EXACTLY_ those half dozen DBs. Only my own DB designs feature many tables with few columns. Based on these observations, five out of six database designers don´t understand normalization! (Naturally, I get some pretty long-winded SELECT and CREATE VIEW statements!)

Updating such a database can be tricky. You can update a view so long as only one or two (I forget which) of the underlying tables are affected. More complex updates must be done in serial UPDATE statements, wrapped in BEGIN/COMMIT statements, for concurrency control, and integrity maintenance.

Based on what I´ve read here, I submit that MySQL users are caught in a deadly embrace. These developers embrace MySQL because it does ISAM faster than anything else, and MySQL embraces these developers because all they want (or understand) is ISAM. And for all you "non-lazy" programmers out there who apparently want to abandon 4th and 3rd generation languages, there´s always Berkeley DB.

I would liken MySQL applied to eCommerce to building a race car without a roll cage - you may get a faster car, but you need to find a driver who has no idea what the roll cage is for. Fortunately, most drivers do know, whereas too many entrepreneurs don´t know about maintaining data integrity.

Thank you for your comments.

-- Andy Nymous, January 16, 2001

A short comment regarding the Andy's remark: in my opinion normalization should not be treated religiously (and even the normalized database need not contain huge amount of the tables). Just two points:

- efficiency;
- the need for an easy modification.

As far as we are talking about efficiency: sometimes you really need to create 'wide' indexes...

Going to modifications: I really like and often use something I call 'data-driven database structures'. Instead of creating the 'user' table with name, surname, city, street, ... I can
create 'user' table with almost nothing more than key data and 'user_attribute' table with records like ('gender', 'M'), ('city', 'Warsaw'), etc. Such a structure happens to be far easier to maintain - especially in case of rapidly changing applications (frequent case in internet business). My example exploded one table into two (or three - if we use table which define all possible attributes) but using such a structure for a greater domain allows to limit table count (for instance we can reuse the same tables for information about organizations).

Yes, I know, it has nothing to do with MySqL. So just a short vote: I use Oracle (when something expensive is required) and PostgreSQL when not. I dislike mysterious data inconsistencies, so I feel I need commit/rollback semantics in any database application I write.

-- Marcin Kasperski, January 17, 2001

Late update: the situation may be changing somewhat; the commercial company who is to MySQL as GreatBridge is to PGSQL reportedly {has released, is releasing, will release} mods to MySQL that will provide it with RLL, Transactions, and some other pertinent items.

-- Jay R. Ashworth, February 4, 2001

but what are the database applications mysql is aiming at? they are competing all the times with every database available.

do they think they need to pursue people with some windy benchmarks they developed? the crashme program doesn’t even show up some features that postgresql has, and it’s IMHO clearly in favor of postgresql as the some of the tests even showed that some feature in MySQL didn’t show up the expected result, while others return the correct things. i have now chosen postgresql for the things i need. mysql seems to have go performance ratings, but in a multi-user/processes workspace it seems to lack some important feature like the transactions. for the normal 08/15 webhost MySQL might be the right choice, because it don’t put so heavy workload on the system and there might be a lot of databases, but they are often only accessed by one user.

-- Christoph Nelles, February 9, 2001

Good news:

Only very recently SAP AG (yes, that company) has taken the big step of taking their database 'sapdb' open source. The binaries are freely available and the source becomes available in April. There are over 700 SAP implementations using this DB. It is a derivitive of ADABAS. Yes, it is ACIDified. Look up www.sapdb.org for details (docs and binaries).

As an SAP Basis Consultant, I cannot imagine a DB without ACIDity.

-- Jay Kelkar, February 16, 2001

Yes, SAPDB's going open source. The more the merrier, as far as I'm concerned.

As far as MySQL gaining row-level locking, transactions, and maybe even referential
integrity checking - good for them!

Hopefully they'll stop arguing that table-level locking with user hacks to get around the resulting performance problems is better than row-level locking.

Hopefully they'll stop arguing that transactions are useless and that users should code their own solutions.

Hopefully they'll stop arguing that referential integrity is harmful because it only catches errors and errors are rare.

-- Don Baccus, February 17, 2001

With regards to Marcin's "data driven" model, with a data_type column and a data_value column, this is going to suck big time once you get to even only 10s of 1000s of rows. (I know because I have had to rewrite data models that were written this way by contractors.) Not only does it suck to have to write queries like "... where data_type = 'last_name' and data_value like '%foo%'' every time you mean "... where last_name like '%foo%''; an index on data_type will be MUCH less useful than individual indexes on the specific columns you need. Finally, it makes foreign keys a nightmare -- keys referenced from this table as well as referencing it.

-- Jonathan Ellis, March 8, 2001

I for one am glad the article was written, caustic or not. I had no idea from the MySql site or any of the press about MySql that it was not ACID compliant.

Kind of reminded me of the "good old" days of being a foxpro/dbase developer.

As a person who has spent many an hour trying to "uncorrupt" a corrupted database, I'll never willingly go back. AND I'll never go back to hand coding referential integrity. $20k for Oracle? A client would save more than that in my hourly rate in not having me spend time coding what a modern RDMS handles natively (and they'll not have to sweat it out that the junior programmer that will be supporting the product years from now will fail to remember and keep up to date the relational integrity code as the database project matures).

MySql sounds like a winner for a read only database or in situations where "free" is not only the price of the database but the value of the data. (ok, that was a caustic remark, sorry)

Lee

-- Lee Harrington, March 22, 2001

I'm one of those people that do believe that ACID testing most web apps is gross overkill.

Looking at the Holy Grail of ACS apps, Photo.net -

○ Will it really matter to people if thier message insert gets nuked?
○ Will they never come back again if they have to enter thier free classified add
twice?
  ○ If thier photo upload dies, will they runaway forever?

Until recently, given how many AOL server errors, and how many server too busy errors, and the overall unusably slow search engine, I would come to the conclusion that it makes no difference. And I would argue that most people would prefer to enter some data twice every once in a blue moon as a trade off for better performance the other 99.9% of the time they spend on the website. But hey, thats just me.

I think it is very revealing that a dedicated server with the specs the likes of photo.net turns into a big pile of molases with only 20K visitors a day. Now that is pathetic, truly pathetic. But as a user gets a AOL server error, or a server too busy error, I'm sure they are saying to themselves, thank god this site has row level locking and passes the ACID test, eh Marge?

But anyway, that doesn't matter anymore. MySQL now supports transactions with commit/rollback. And with two types of tables, Gemini, or Innobase, with non locking selects and row locking inserts.

The inclusion of these table types, along with MyISAM and the new HEAP type allow you to build a site with an optimized DB structure -

  ○ HEAP tables in main memory for lightning fast session management on multiple load balanced web servers.
  ○ MyISAM for lookup tables of infrequently changing data, or low priority data.
  ○ Innobase tables for the few site operations (ecommerce, etc) that really do require transactions.

The flexibility of MySQL allows you to choose what speed/integrity trade-offs you feel are appropriate for your data and your audience.

I'm certainly not arguing against the level of data integrity that is offered by PostgreSQL and Oracle with foreign keys and stored procedures, I just think there are plenty of apps that just don't need it. After all, in the end, if you have a commercial application that requires the type of data integrity outlined by many in the above posts, management just isn't going to let you chose a free-ware solution. As one of my friends says, 'No one gets fired for choosing Oracle'.

But hey, this was a great topic, but it seems to have lost some steam. I'd like to suggest these scintillating topics -

  • Why not Apache
  • Why not PHP

-- E.J. Sadler, March 23, 2001

Let's see...EJ begins by parroting old MySQL "truisms":

1. ACIDity isn't important

2. ACID dbs are too slow
He "proves" #1 by making the point that most folks who post to photo.net aren't really dependent on the site maintaining integrity.

Hmmm...photo.net does include credit card processing of contributions to one of Philip's pet charities. Does EJ also claim that integrity is unimportant to photo.net users who use their credit cards on the site?

Because, after all, the MySQL folks claimed for a long time that ACIDity was unimportant in *all* contexts, as it was better for the application programmer to guard data integrity "by hand" rather than have the DB do it for you.

If the MySQL folks had restricted themselves to pointing out that many web sites don't need the assurance given by ACIDity while also pointing out that many do, and that in those cases ACIDity is far from useless, you'd see no bitching by this crew.

Ditto their attitude towards referential integrity and the like.

Does photo.net prove that ACID-compliant dbs are too slow?

Given that sourceforge moved recently to PG 7.1 because they found MySQL to perform very, very poorly under load - a move not taken lightly and only after heavy-duty testing with simulated traffic similar to what is actually seen at sourceforge.net - I'd humbly suggest that whatever problems plague photo.net aren't due to the fact that it uses an ACID-compliant database.

BTW, Sourceforge runs a single x86 dual-processor database server, i.e. much more modest hardware (costwise, at least) than that which underlies photo.net.

And of course the problems you see with Intermedia aren't in the least bit relevant to the discussion regarding ACIDity. Intermedia sucks but it sucks on its own (de)merits. It might lead one to think that Oracle sucks, and in many ways Oracle does suck, but again it isn't related to ACIDity (the basic Oracle engine doesn't suck, though seemingly just about everything else Oracle offers does).

Slashdot, running MySQL, was unbearably slow until they got enough cash to afford a server farm - and they were the force behind MySQL adopting the Berkeley db as a backend because apparently they don't share EJ's beliefs that ACIDity is unimportant for a mere chat and news service. For every slow Oracle site EJ names that "proves" that ACID-compliant dbs are too slow for web work I can name a MySQL-based service that "proves" that only ACID-compliant dbs are fast enough for web work...

In other words the speed argument's bullshit. Much more important is the concurrency argument, something Innobase is solving for MySQL.

EJ doesn't stop by taking poorly-chosen swipes at ACID-compliant databases. He swipes at AOLserver - ignoring the fact that AOLserver runs a couple of the busiest sites in the world (and on relatively modest hardware, as of 1999 Digital City ran on a single db server and six AOLserver front-ends).

Then of course we have point #3:

MySQL now offers two backends, including Innobase, that offer ACIDity and in the case of Innobase row-level locking and an overwriting storage manager similar to Oracle's.
It seems that MySQL has been listening to the criticism over their attitudes regarding ACIDity, table-level locking (remember when this wasn’t considered a problem but rather a feature by the MySQL crew?), referential integrity etc. Unlike EJ, they are responding positively to criticism and working hard to turn MySQL into a real RDBMS product.

Kudos to the MySQL crew. As soon as I heard about Innobase I e-mailed Ben and said, "hey, maybe we *will* support MySQL some day in the future after all". Innobase has some problems to work out (BLOB data limited to 8KB, for instance - sound familiar, PG users?) but the trajectory looks good. It needs solid use in the field before one's likely to feel overly comfortable committing vital data to it, but that real-world experience will be coming along soon.

So ... OpenACS may well support MySQL with the Innobase backend in the future. This will be due to enlightenment on the part of the MySQL folks, though, not due to ignorance on the part of EJ...

-- Don Baccus, March 30, 2001

There is a lot of people on this list bashing Mysql, I'd just like to add that Mysql is one of the most kick-ass databases I have ever used. Granted it is very limited in functionality, and sometimes i'd love for it to have simple features like sub-selects or views, but no other database can touch it's speed.

I am using Mysql on my personal web site that recieves roughly 1.5 million hits a month. Pretty much the whole site it built from the database, and it performs great!! I couldn't' imagine attempting to use another bloated database like Oracle for this same job.

-- Mike , March 31, 2001

Well, then, perhaps you too should read Tim Perdue's discussion as to why Sourceforge switched from MySQL to PostgreSQL

You might also want to think about why the MySQL folks themselves are now offering two new backends, one of which (InnoBase) offers row-level locking in order to avoid the problems with high levels of concurrency run into by sites like Sourceforge (ironically, if the InnoBase backend had been available earlier Sourceforge probably wouldn't've switched).

Oracle would run your site just fine, BTW. So would PostgreSQL.

-- Don Baccus, April 2, 2001

Here's a direct, if somewhat salty, quote from Mike's site discussing kick-ass MySQL:

FUCK FUCK FUCK FUCK FUCK!!! My FUCKING database screwed up and I lost all my FUCKING posts.. I had to revert back to a old copy .. from like Jan 29th!... I should do backups for often!!!.. shit.. oh well..

I am NOT making this up! Yeah, it's a cheap shot...

For the record, Oracle corrupted the first block of its data file on my laptop a couple of days ago. Take heart, Mike!
Ironically, I'm currently reloading a backup for a customer, whose entire site is based on MySQL/PHP. Over the course of the past few months, I've had to setup a backup servers and use some of the replication services in the lastest version of MySQL, which for the most part works.

Concerned about transactions, the customers actually attempts some protection in the code itself, which I see as a tremendous waste of resources, why re-invented (or at least try to) what is already there. Looking further into the code, I see a bit of the old "fetch a query and loop through it for the stuff I really want". Now I know it is not the case for all people that choose MySQL, but it seems from my personal experience that people that are more familiar with SQL, choose other products of MySQL or they choose it because they think it will meet their needs and by the time they find out differently changing is "too much of a hassle".

Personally, I see MySQL in the same light as I see Perl. Great for "quick and dirty things" or some rapid development for proof of concept since they move easily to other products (in early development stage). MySQL has a place, but for sites that are database dependant (such as OpenACS, ACS, etc.) MySQL seems to not fit the bill.

Of course, the important part is understanding what the capabilities of each product is and making your choice based on that. Even /. has admitted (I believe, the minds are a bit cloudy and lost right now) that they used MySQL because it was there and by the time they seem the problems, the code was already too MySQL centric to change.

Just my thoughts :)

-- Mat Kovach, April 8, 2001

If the probable savings of using MySql exceed the probable cost of not having functionality that other databases do, then MySql is the DB for the job. If the cost of ACID failure is high, then one should not be using MySql for that application.

The answer to the question is not global. In one instance, I've chosen to use Oracle over MySql. My conclusion was that it was worth the expense. On the other hand, the decision had nothing to do with transactions. Had other considerations been equal, I don't think that (lack of) transactions would have been a show-stopper for that application.

I can't stress the "for that application" enough. I can definitely see applications in which the lack of transactions should be a complete show stopper. In such instances, MySql would not be considered seriously.

Let me put it a different way: For situations where a partial transaction would be simply annoying (e.g. tracking web page customers), MySql may be just fine. For applications where it would be costly/dangerous (e.g. hospital medication tracking), it would be a complete no-go. For the situations you used as examples in your article, I think that many people would consider the availability of a partial transaction annoying, but not fatal. On the other hand, if an annoyed customer could cost you a $2M sale.. it would be a big deal.

-- Stephen Samuel, April 8, 2001
I found the reading on this page very interesting. I am a long time (relatively speaking) MySQL user. I have used it on several projects and have had not a single issue with the software stability wise. I would like very much a solid transaction service before I will feel 100% confident in using the system. Would I ever use Oracle? Probably not. Oracle rubs me the wrong way. It strikes me as an over priced, over marketed piece of software. I can't see the value in a $10,000 piece of software when similar performance can be obtained from packages costing far less or nothing at all. We can debate all day the "what if's". What if someone trips over a power cord a millisecond after a request is sent. What if the building burns down. I can't see spending $10,000 for the one in a billion chance that a problem of this nature occurs. I would say the biggest risk to data storage would be actual hardware failure or poor scripting by a programmer. I am going to explore postgresql and see how it performs but until the people at Oracle get a clue on pricing they are never an option. Just my opinion.

-- Dan Hammer, April 16, 2001

I've used Oracle for years, MySQL for months. What I can say is that Oracle have a lot of features, suitable for enterprise applications, for big time companies. MySQL is quite light and fast, suitable for smaller, lighter applications. Enterprise applications needs those features like transaction, store proc, triggers, etc.

The bottom line is... It all depends on what kind of application you will do, thus what kind if DB you need......

-- Willy C, April 19, 2001

Aloot of people use MySQL for speed, but MySQL is only faster in situations where flat files would be the fastist. There are tools available much faster than MySQL for simple login/password based web sites, (type MAN DB from any UNIX prompt). If thats all you are doing, you should not be using an entire server process consuming tons of memory and resoruces for that need.

If your data is complex, it has been proven time and time again, PostgreSQL is much much faster with complex joins and other things. Consider that I can get more done in each SQL statement also.

Speed is reletive also, Its rather inexpensive to upgrade, OR buy another computer these days. I'd much rather spend an extra $100 on a faster CPU than have to tell my clients I lost all their information in a database crash.

In summary, MySQL excells at nothing and serves no one.

-- Derek Young, May 4, 2001

This article is rather dated. I suggest that you go take a look at http://www.mysql.com and take a look at current feature lists. In particular, transactions are now supported, albiet at a performance cost.

-- Greg Fortune, May 6, 2001

Actually, InnoBase is touted as being just about as fast as the original backend, which lessens the credibility of the inital MySQL implementor's point of view regarding the so-called unacceptable expense of supporting ACID characteristics (which require transactions among other things).
If you read the more recent comments preceding yours, you'll see that we do
understand that MySQL now supports two backends that pass the ACID test. I think this
is a great thing. Now if they'll see the light in regard to things like referential integrity
and support of a much richer SQL dialect they might actually produce a product I might
use.

This document is out-of-date, but it and the debate following its appearance had wide
exposure. I suspect it played a small role, at least, in awakening the MySQL community
to the fact that perhaps things like transaction support aren't unnecessary foolishness
after all. Rewriting the article would be a bit like rewriting newspapers to reflect events
that hadn't happened at the time the newspaper was published.

-- Don Baccus, May 13, 2001

I have not used MySQL so far. But my experience of 15 years of database programming
and designing dissuades me from using it. I do not see how can a RDBMS be designed
without Transaction Tracking System (TTS). Why not use flat files systems (like xBase,
Paradox, Access) instead if you do not need TTS, triggers and stored procedures?

-- Prakash Bajracharya, June 9, 2001

TO THOSE OF YOU WHO USE MySQL BECAUSE OF SPEED: Why don't you use
LDAP? Its faster for lookups.....the same as read-only SELECTs that MySQL seems to
be so good at. For purposes where you require a database, make sure you use a good,
ACID database. And for those of you who mention what difference it makes for a
normal user to lose data on a POST operation like an email address, you should please
provide a list of the sites you have contributed to so that I know where not to go and
waste my time. If the data you ask of me (the User) is of not much use to you, then
either you should not be there or I should not be visiting your site. Data is important and
needs to be protected. Thats why so many years have been invested into research and
development of RDBMSs. Database are not meant for those of you who don't beleive in
the sanctity of data.

-- Gerard Fernandes, June 28, 2001

Maybe it's not clear to most people reading this but: I estimate that 90% of the worlds
financial applications are written in languages like COBOL and RPG on platforms like
370's, 390's, AS400's that use databases that (until 7/8 years ago) weren't ACID. Most of
these programs are the same as they ever were, even though the databases now have the
additional capabilities. And they work just fine. Bottom line: ACID is a concept that
focuses on the database as the only possible application to ensure integrity. However,
real life shows most applications do it outside of the database. Even now, on the Oracle
platforms, how many people take the trouble to build in a rollback in their transaction
processing?

-- Jacobus Bezemer, July 1, 2001

Greetings Open Source Database fans,

This discussion has so much valuable insight for anyone doing research on Open Source
databases. Even more important, the comments are dated so it's easy to see which are
likely to be obsolete. Might it be possible to order like a mail archive, newest first? That
way folks with an historic interest could still track the evolution, while others (most)
looking for current knowledge would have the latest scoop by default.

http://openacs.org/philosophy/why-not-mysql.html
BTW, my only comments are to:

First, identify the immediate needs, as well as long range requirements of a project, and select the tools to match. There may be a conscious decision to start with one tool, with a plan to migrate when the time comes. It will always be easier to execute that migration if you have continually made design choices with that contingency in mind from the start. If ACID is critical from day one, you very well may need a tool with ACID support. Otherwise, maybe not.

While many comments pointed out limitations or plain old bugs in one system or another, they may very well be outdated. It's most important to understand your true CURRENT and FUTURE needs, and the CURRENT capabilities of the systems you are considering. It would be nice to be able to plan based on anticipated future system capabilities, but resist that temptation. Your needs will change, and systems will improve, but DO NOT count on both evolving in perfect synchronization.

Thanks for hosting a great discussion!

Jerry S.

-- Jerry Spicklemire, July 28, 2001

Explanation of durability is just wrong... Drives aren't like tape... besides to provide durability, you'd rather rely on a raid array that's perfectly suited for such tasks with stripping and mirroring and ECC (pick your favorite RAID setup) and part of the responsibility also falls onto the filesystem used on the system... since if the filesystem gets corrupted by power failure or whatever may cause such an accident no matter how good your database was (though doesn't oracle have like it's own filesystem if you set it up?)but these days realtime data replication would be enough.JUst my $0.02...
IANADBA

-- Dee Ceek1, August 2, 2001

I am surprised and somewhat disheartened by the level of diatribe and arrogance in some of the postings. Although there is much technical truth in what is said, the OpenACS folks appear to come out swinging and then flame on all other points of view.

I use MySQL in one extensive site and am totally pleased with it because I carefully matched my technical needs with what it offers. I was not misled in the slightest by their site; it was obvious to me that it wasn't ACID and that it would be very problematic in any situation involving updates. The site is 100% bulk loaded and 100% read only. It rips, administration is easy, the MySQL documentation was clear, and it was trivial to tune.

I will probably not use MySQL in another extensive site because of vastly different technical needs. I am likely to choose PostgresSQL because of its technical merits. (btw, I have 15 years of experience with a variety of other databases, including Oracle, Sybase, MSSQL, Sybase, and DB2. I don't consider myself a ninny about this stuff).

I am troubled, however, by a persistent sense of arrogance and defensiveness I see in the parts of the PG user community I've explored. That sense troubled me 18 months ago when I selected MySQL for the other site, and it troubles me again now as I face another choice. I rely very strongly on my peers' experiences and insight to help me solve my problems faster; the user community is an important consideration for me.
I would *like* to like PG. It looks great from a technical perspective, and the progress from 6.x to 7.1 looks terrific. I would really prefer to find out that my perception (from this and other PG-centric postings) of PG heavy weights treating people with different opinions as raw flamebait is misplaced.

Can someone point me to other sites that have balanced, friendly and technically astute discussions about PG?

-- B. Hardwell, September 4, 2001

Hmmm....

I have to say that I'm somewhat surprised at the anti-MySQL bile-spewing that some posters have indulged in.

The simple truth is that for a lot of people, MySQL is by far the best available solution. As an example, consider presence providers who have bazillions of customers on shared machines wanting 5 SQL databases each for their chickenshit cookies and pagecounters. Eek! I miss a page count every month-of-Sundays because my database isn't ACID. Frankly, my dear...

What about a web-site where the data is stored in a SQL database but is almost never updated? It's just stuffed full early on, has bits and bits added over time, and a few amendments now and then. But it gets the holy hell hammered out of it all day, every day, on reads/searches. In some situations, MySQL is a no-brainer.

Why am I using PostgreSQL now? Because MySQL simply can't cut the mustard for my current application. I need the stuff that MySQL hasn't got. Why didn't I use PostgreSQL before? Because I didn't need to. Will I use MySQL again? Damn sure, when it's right.

As an observation, I've come across more annoyances in PostgreSQL in a month than in several years of using MySQL and a couple of dalliances with *cough* MS-SQL Server and Sybase. Try deleting a view that you've renamed --- now it thinks it's a table. A relational database with arrays? Curious. Why does it mangle table and column names? Case-insensitive, fine, but not even MS-SQL Server 7 clobbers your capitalisation. How about trying to create a view on the system tables that gives you useful metadata? I gave up and scripted it.

As to the alleged "misrepresentation", I'd think of it as not advertising your shortcomings. I don't see the lack of updatable views, the atrocious hack that masquerades as a SERIAL (SEQUENCE/IDENTITY/AUTO_INCREMENT) implementation, the lack of outer joins, a reputation for sloth and memory leaks, or anything other PostgreSQL foibles plastered all over the PostgreSQL home-page. Nor does any reasonable person expect such.

But then, the vitriolic MySQL bashing reads more like "I've got a major corncob up my ass about MySQL" rather than the words of a reasonable person. MySQL has carved out a nice little niche for itself -- it's small, fast, easy-to-use, provides an adequate subset of RDBMS functionality for many applications and lots of people use it and like it.

What's so wrong with acknowledging the both MySQL and PostgreSQL have their place? Bitching about it will only give ya' heartburn!
I think you misunderstand the historical problem with the MySQL implementors. You have experience with a variety of databases and choose MySQL for an application space for which it is well suited.

Goody for you.

This doesn't make the fact that the MySQL implementors for years claimed that standard RDMBS features like ACIDity and transaction control ARE USELESS IN ALL CONTEXTS and indeed DANGEROUS. They touted the lack of ACIDity and transaction control in MySQL as POSTITIVE FEATURES in all application environments.

This was (I use the past tense because with the advent of InnoDB and BDB table types the MySQL implementors have changed their tune) grossly irresponsible. Rather than educate users who were new to SQL - which describes the vast majority of web developers in the mid to late 1990s - they worked to teach them that traditional RDBMS safety features (notably ACIDity and transaction control) were not only useless but actually bad. They claimed the MySQL was superior to, say, Oracle precisely because MySQL lacked these "bad" features.

Now ... MySQL has grown up considerably in the past two years. Actually this article and the stream of comments you find so objectionable somewhat coincide with the MySQL implementation crowd changing some of their public positions regarding traditional RDBMS features. Cause and effect or sheer coincidence? Hard to say.

But certainly many of the historical comments you find in this thread, while accurate and fair while posted, are no longer accurate. As one of MySQL's harshest critics, I say "Bravo!" I'm happy the the MySQL implementation crowd is starting to see the light. InnoDB in particular looks like a real winner and once mature will remove the biggest objections we've had to MySQL - the lack of transaction control, ACIDity, and brain-damaged table-level locking.

As far as the PG crowd goes, actually you'll find heated discussions on the importance of outer joins, subselects, and the like in the archives of the various PG discussion groups. Outer joins, in particular, were under-appreciated by some in the PG development community and attempts at implementation were allowed to drift along without much emphasis for far too long. This point was made emphatically by me, among others. I'm a fan of PostgreSQL but that doesn't mean I worship its flaws. I'm more of a thorn in the side than a gushingly worshipful user - ask around.

However the attitude within the PG development group is largely night-and-day different than the attitude displayed for years by the MySQL development crowd. The PG crowd understood that their mission was to work towards making PG a full-featured RDBMS, not to defend deficiencies as virtues as was the habit with the MySQL development crowd.

Typical was the view towards referential integrity displayed by the two projects some years back. The MySQL response was to write documentation claiming that referential integrity checking is harmful. The PG development group's attitude was "it's important, but it's a bit down our priority list and we haven't quite decided the best way to
implement it. We know we need it and we'll have it in a couple of releases, wanna help?" (I did wanna help and did, too, though only in a minor way).

This pretty much sums things up for me.

However I'm lightening up my attitude towards MySQL. As I mentioned above, the InnoDB back end has great potential. They're announcing subselects and stored procedures later this fall. Referential integrity is still lacking and it's not clear that they "get it", but at least the "get it" in regard to row-level locking, ACIDity, and transaction support.

I might add that the fact that they're adding such crucial features certainly does little to convince me that our criticisms were unwarranted. If we're wrong, why are they adding the very things we've complained were missing?

-- Don Baccus, October 10, 2001

Some previous posters have asserted that they have never used transactions because they never needed to handle rollback or explicitly commit a transaction in their app/SQL code, and besides, ACID isn't really that important. Several folks have even said that RAID and a UPS eliminate the need for Durability. Finally, lots and lots of folks say that MySQL is the right tool for read-only databases (or almost read-only databases), and that it's so much faster than "bloated" ACID compliant RDBMSes.

If you have used one of the following: Sybase, Informix, DB2, Oracle, MS SQL Server (of course there are many others), you have been using parts of ACID transparently. It's important to realize (and I think some of the folks who have posted regarding the need for ACID haven't done a great job so far) that there are TONS of ways for data to become hosed in a concurrent application, and only some of them are due to hardware failure. Every application is different, but there are definitely cases where no matter what the data model and no matter how clever you are about doing stuff in a certain order, once you put that application in a concurrent context (many simultaneous users), all that cleverness breaks down because a race condition defeats your carefully written code. This is why multithreaded application programming is so tricky as well. Code that's airtight when executed serially does ugly stuff when executed in the "wrong" order by multiple users who are at different places in the same code, operating on shared data. It's not just a matter of catastrophic system failure dropping an insert once a year.

If you want to increase the size and complexity of your code to implement functionality that's already in some databases to keep your data consistent, that's your call, but then you don't get to point fingers at database products and say that their code is bloated. Bloat is in the eye of the beholder; you just moved required functionality from the database into your own code. If you don't care if your application creates data inconsistencies from time to time, then I only hope no one is paying you for your work.

RAID and a UPS will solve the momentary power failure scenario; I think we need a more realistic scenario. For example, an update to a 10,000,000 row table, on a system under load. This may take quite a while to execute, depending on the table, disk performance, etc. Let's just say that there will be more than a millisecond delay before all the data is written to disk. RAID and a UPS will not save you if someone does "kill -9" to your database server. RAID and a UPS will not save you if your SQL code was halfway through doing something, there was an error, and there was no rollback so the database remains halfway changed. What you'll end up with is a mirrored corrupt database. Backing it up eight times a day means that you have a mirrored, backed up...
corrupt database. You have to prevent it from becoming inconsistent in the first place.

If your application is truly 100% read-only, then I suppose that MySQL would be suitable. However, in 100% read-only situations, ACID is irrelevant, so I doubt that MySQL can really be that much faster than an ACID-compliant database. After all, in a read situation, you're really only relying on indexes and buffering to make it faster than reading the whole table from disk, or from a text file for that matter. If performance were to be equivalent (and I'm not saying it is; I haven't benchmarked MySQL) I think the only reasonable argument in favor of MySQL might be the simplicity angle - that is, it's the fastest way to get an indexed, SQL-like query language on top of your static data. I've seen cases where Berkeley DB was fast enough but wasn't as flexible for querying as SQL. (I say SQL-like since it obviously isn't SQL compliant, just mostly SQL compliant, and adds lots of extensions.)

If someone else out there has a good reference link for specific icky situations that ACID handles safely, please post it... it seems that a lot of folks reading this page need some education about it, and I don't have time to elaborate further.

-- Jamie Flournoy, January 14, 2002

At the time of the SourceForce switch to PostgreSQL, Tim Perdue had made available the benchmarking he had done showing that PostgreSQL had the advantage.

Now that InnoDB is here with many of the advantages that PostgreSQL has, did anyone do such similar testing?

-- Pierre Phaneuf, March 16, 2002

eWeek's benchmark (February 25, 2002) said "Oracle9i and MySQL had the best performance and scalability" - Does that make any sense to you? I can see a couple of loopholes in the article:

- They were measuring the response times. A file-based application may do the job with a response time in the same order, which does NOT imply that a file-based application is comparable to Oracle 9i.
- There were parts of the test used the non-transactional (less expensive processing) engine of MySQL!
- The benchmark generated 200,000 orders (assuming that was a number of records), I do not think this is an enough number for benchmarking a RDBMS.

I think the domain of testing was quite narrow to benchmark DB2 and Oracle.

-- Ahmed Moustafa, March 22, 2002

One other related point to the eWeek article is that MS SQL's problem came not from the DB Software but from MS's beta JDBC driver. When the benchmark was done in a .NET platform, MS SQL 2000 had better Throughput and Response times then any other DB.

-- Jay Austin, March 26, 2002

Okay, I see a comment dated March 2002 saying InnoDB (which appears to be a MySQL internal table type) is here, which offers ACID (rollback/commit etc etc) features.
So it's June now. Does that mean MySQL has only had transactions for three months?

I'm trying to get an honest idea of how mature MySQL is as a fully-functional RDBMS compared against Sybase, Oracle, DB/2, Informix, and Postgres.

All my research so far says Postgres is 99% of the way there, and MySQL is about 80% of the way there. Discussions on the PG-ADMIN mailing list talking about internals has convinced me that Postgres' transaction system is up-to-snuff, but any discussions I've found about MySQL seemed to remain soundly in the "MySQL transaction handling is immature at best."

Is this fair to say at June 2002?

I'm trying to set up a database that will be a drop-in replacement for Oracle. We will use it as a datastore plus stored procedures & triggers, but won't really do anything with PL/SQL or PL/pgSQL or any complex database logic. Mostly just ACID and stored procedures and triggers is what we're looking for.

I know Postgres will be one candidate. Someone asked if MySQL would be another candidate and I would like to give an intelligent answer.

-- philo vivero, June 11, 2002

Whatever the perspective on mySQL, there will always be several camps for, several camps against, and those sitting on the fence.

If mySQL solves your problem who gives a rats ass if its not fully ACID/RDBMS compliant. Your given a problem and one to solve. You have many choices. For someone who is a purist to say "this is what you should use" will forever be lost in theoretical mumbo jumbo. You use what works irregardless. And if someday that purist says "See, I told you so" then goody for them. But during that time, I made money. And I will say "so what".

Its about whether your solution made you money.

Tell that to Yahoo! finance that mySQL sucks. Frankly, it works perfectly fine that they are using mySQL to solve their problem. Not every tool needs a Mercedes to get the job done.

-- Mr Buzz, September 22, 2002

If you are seeking truth, you are wasting your time trying to find it.

-- Mr Buzz, September 22, 2002

Mr. Buzz writes:

If mySQL solves your problem who gives a rats ass if its not fully ACID/RDBMS compliant. Your given a problem and one to solve.

I'd agree with your point of view on this (use the right tool for the job, thus use MySQL if it's the best fit for the job) but for one thing: experience shows that the nature of the job changes over time and that it's usually a great deal of work to change your fundamental toolset when the problem changes. Now, except for replication, the
capabilities of PostgreSQL are a superset of those of MySQL, so my natural inclination would be to choose PostgreSQL (or some other more capable database) unless it couldn't handle the job, because it's more likely to be able to handle future variations of the job than a less capable database like MySQL.

That is why you should care if your database is ACID/RDBMS compliant. You might not need it now, but my experience shows you'll probably need it later -- when it's too late or too expensive to do anything about it.

-- Kevin Brown, September 23, 2002

It seems to me as a newbie in the world of open source databases that there are similar problems between the open source and commercial database products. I see some of you complain both ways about transactional (commit/rollback) integrity. In reality I believe that it is a requirement of the RDBMS to handle transactions, it's just that developers are very poor sources of what a transaction is and we consequently rely on the RDBMS way too much. How hard is it inside one's code to add a 'commit' or 'rollback' call every once in a while when a logical set of statements have been issued? Not too hard, but we all don't do it.

Data loss is something that cannot be tolerated, no matter what the cause. Yes we have RAID systems, if you can tolerate the overhead the array places on things. And UPS's fail, or shutdown commands go into the black hole. All of these cause data corruption, and then there's the diamond of them all, someone wrting over or deleting the file "by accident". One such accident can cause each of us our jobs! Therefore none of us can tolerate it and having data corruption due to a poorly defined backup strategy is the worst of them all. Imagine yourself looking the boss in the face & telling him that you can't recover the database at all. If one is going to make backups, one HAS to verify them every once in a while. BTDT and it was well worth the effort. Also, over the last 17 years of working around Oracle the only database recovery problem that took too long was the one where the SA was an idiot. Otherwise recovery is easy, figuring out what happen is the hard part.

Relational integrity and business logic I happen to really like being in the database, why because it's a database thing that's why. Yes the developer may have the ability to do that in his/her application, but what about those who modify data from somewhere else? They may not know, understand, or care about the integrity of the data, particularly if their a hacker.

Also, before I end, I agree with those who say we should judge a database product on it's merits and the requirements of the project. I may well recommend MySql for our next e-commerce project, but not to store customer data or credit card information, or e-mail addresses, but for product information or shipment information that I can safely restore from other datasources, sure Sounds like a good application. But then it might be PostGreSql too! Because what I need is a low or nocost database to hold that type of data. The real important stuff will be in Oracle.

-- Richard Goulet, October 16, 2002

Add a comment

Related Links

- make a new page for these comments: For "Why Not MySQL", maybe we can make a
discussion forum for this specific topic. It's very difficult for me (and maybe other people, too) to read through these comments. These messages would look more organized, and more readable if we can somehow thread them.  (contributed by John Lin)

- **Open Source Databases: As The Tables Turn** - Tim Perdue compares MySQL and PostgreSQL in some real-world benchmarks (November 12 2000).  (contributed by Eric Baquiran)

- "**Databases and PHP**" by Zeev Surask - Evaluates and tests current database options.  (contributed by Sam Snow)

Add a link