

# The Eternal Donut of the Soul

## 3. Being Locked into a Database Vendor

[podcast@xor.com.au](mailto:podcast@xor.com.au)

The mantra in the IT community is that open is good. There are different forms of being open, from open source to just supporting standards allowing a common format for access. When it comes to open standards and following them, it's hard to argue against it. The argument goes something like "we should make sure we are not locked into any database vendor, as they are now all the same, we should position ourselves to be able to quickly move from one to another".

In this podcast we tackle this issue head on and question it. In fact we are going to look at why being locked into the database vendor is not such a bad position to be in. So the topic for the podcast today is why it is fine to be locked into Oracle.

The argument goes against common sense, as it can be seen that there are so many good reasons for being able to build an application that can be picked up and moved with minimal to no changes between different databases. This is actually a very dangerous position to take. Incredibly dangerous. It's one of the biggest causes of performance problems I see when it comes to tuning a database environment. My view is that it costs more to take this viewpoint and implement it and enforce it, than it would be to take the other view.

The starting reason why this viewpoint is not right, is that it enforces in the mind that it is a good thing to separate the application code from the data, in a way that the two are very independent. In these podcasts I try and show that for performance reasons it's best to keep the application code as close as possible to the data. By all means keep it separate, but keep it separate within the database. When using Oracle the only methods to achieve this are by using PL/SQL, Java or writing your own cartridge. Out of these three, PL/SQL is the most ideal as it has been designed to work closely with the data.

Physically separating the application code from the data means adding distance between the two, which adds cost. Adds cost on the network, on the CPU to pass the data, and it has scalability issues. It also enforces in the developers mind that the code should drive the application and not the data. In today's multimedia, web rich environment, the rules have changed, and it's now the data that drives the application. So the thinking might have been fine five years ago, when it was good to put the application code on one box and the data on the other, and then say "phew, life is easy now". But not now. Different times, different rules. The hardware architecture of today, and even the software architecture that Oracle pushes, is better geared towards having the data and code in the database. Why?

1. Backup and recovery. Code and data is backed up together. No issues on recovery and syncing the code with the right data version. Maintenance is easier.
2. Easy to activate Oracle features. It's a lot easier to migrate a PL/SQL, HTML app to RAC than one using Forms, C or other GUI type tools.

3. Hardware today supports large memory, multiple cores and CPUs. It makes more sense to centralize the code and data. When capacity is reached, multi-tier at the database level, and not application level.
4. Tuning. Its easier to tune an application when you have removed the interactivity between the application server and database server. If its all in the server you can diagnose problems and identify them quicker than in a multi-tier platform.
5. Security. You don't have a middle tier that can be compromised. You don't have a middle tier that has to be managed, tuned and understood. You don't have to worry about multiple versions of hardware and software to manage and co-ordinate.

So lets now look at the other issues why its best to be locked into Oracle. Not all databases are the same, they differ in a number of crucial areas. Oracle being Object/Relational is a big difference. Its support for multimedia inside the database and the fact its block size is independent of the row size means that once you program for Oracle, it is very hard to port to another vendor. For example, if you build a table that can store in it, potentially more than 8k of data per row, then it will not port to most databases. Bit of an issue this, especially now when its so easy to do this. A couple of varchar2(4000) fields and you are in trouble. You have to start doing tricks with the other databases to get it converted and eventually sacrificing capabilities.

The view should be : Lets live with the fact we are locked into Oracle, and optimize our environment for it". Lets take advantage of all the good features and use them. If you try and make your code portable/open to multiple database platforms, all you do is program for the lowest common denominator, then wonder why it runs so slow. And there is nothing you can do about it, except complain about how bad Oracle is, for running so slow when other databases run better. When what you have done is crippled Oracle, limited its use and then whinged about it (whinged is an Australian term for what one might say is a developer who complains about donuts not being healthy as the reason for not eating them).

On to the next point. There is a view that SQL is the same across all platforms. This is also wrong. Maybe its because all vendors say they comply with some ANSI or ISO standard. Its like saying all cars are the same because they all have engines in them. Its true, but hopelessly inaccurate. Its this binary thinking which causes so much grief. This is another point and another podcast that I will tackle, which is on how to use fuzzy logic and thinking for performance tuning.

OK, lets look at some examples that highlight how backward this argument is:

***Select myproc('abc') from mytable;***

Might work on Oracle, but doesn't port to anywhere else (why? – well being an arrogant DBA, all I can say is that if you can't see why, get a programming life. I have better things to do with my time than pander to some wannabe techo who believes they can understand the core concepts behind database

just by reading a manual. If you haven't ever programmed then go out, buy a dozen donuts and at least act like you are a DBA. And yes, I am not a very nice person - I am a DBA, being nice is not in the job description, in fact the job description says I can only ever be nice when being fed by the users).

***Select to\_char(sysdate) from dual;***

Will not port on most platforms because

1. the table dual does not exist
2. sysdate is not portable between all databases
3. to\_char of a date is not only database dependent, it can even differ in Oracle depending on your language settings.

And that was a very simple query. So if you adopt the view that I want to be able to take my SQL statements and move them between any database vendor, what you are doing is writing SQL statements that will work across all vendors, which is spending a lot of time to find the Rosetta stone of SQL statement translation and then cobbling your statements to work consistently. It would be easier to find a fat free, guilt free donut that also tastes good.

And when it comes to scalability and performance, there is no guarantee that the query s will perform the same across different databases."Select column from tablea, tableb, tablec where.. "

will perform the same and use the same indexes and join techniques as if it was written

"select column from tablec, tablea, tablev where..."

The behavior of optimizers varies considerably between vendors. Most use cost based optimizers, which means the performance of them is at the whim of vendor and their ability to handle the complexity of SQL and their own database environment. Not others. Their own. Their own features, capabilities and strengths.

Of course the attitude is that the cost based optimizer works flawlessly and developers don't have to worry about tuning SQL. Lets just write bad SQL and let the database work it out. At this point the DBA comes in with their trusty baseball bat, called "Mr Whamo" and subtly explains to the developer why this is the second stupidest idea they ever thought of. The stupidest being why are they talking to DBAs when they obviously have no clue about anything, and why aren't they feeding them donuts?

Cost based optimizers work well, but have issues. That's why all the vendors either support or are moving to support the concept of hints. Sometimes us lowly humans have a better grasp of the situation to tune. Also, the checker grammar in word works better also rewriting my sentence at.

Anyhow, its better to write statements efficiently and optimally. Makes them easier to tune, debug and understand. Which is a novel concept for some.

Lets review some of the features in the Oracle database that we take for granted, which do not behave the same in other databases or do not even exist in other databases:

- Row level locking. I could spend weeks talking about this. In particular, just because a vendor has row level locking doesn't mean they want you to use it. Case in point is SQLServer. They have it, you can use it, but if you want real scalability, don't use it.
- Field types and column length. With Oracle10g a column can be any length. But not all vendors have this, meaning you have to max out the lengths to some obscure number like 256 or 1024. Does varchar2, char, number behave the same between vendors? Not always. On porting you might get some interesting results.
- Maximum row size. Already covered this. But the nice feature Oracle has, is that the logical design maps to the physical design.
- Multiple blobs per table. Supported by Oracle. Supported by some vendors but not all. Most have maximum lengths. Not an issue really, as its pretty hard to get 4Gb into a blob. So if there was a limit its unlikely in the next couple of years of reaching it.
- Dealing with blobs. Not all blobs are the same. How do you get them in and out of the database. How do you manipulate them or update them? Recovery is the big one. Some vendors don't support blob recovery. Its just too hard in their architecture. So you can load them in, but if you lose the database you lose the blobs.
- SQL Queries. If you look at the actual SQL statement, each vendor offers their own extensions. In Oracle there are new features like. model clause, regular expressions, SQL3 standard command (join syntax), OLAP and warehouse commands (cube, rollup). There are different ways to sort and of course you have the issue of embedding functions in a SQL statement that you have written. Lets not forget that in Oracle10g to improve performance sorts on group by statements are not done, meaning the ordering of results is changed. A lot of application do not do additional order by's if a group by, because they have been told by their tuning guides not to do this (case in point, older versions of Oracle)
- Object support. Allowing XML, Spatial and Text documents in the database. And then queries to retrieve them. Each vendor is different, and only Oracle allows the one SQL statement to access all three. Which means you can do a query that accesses Oracle spatial and context at the same time. In the line of work I do, this is a nice feature. (Side note: Context – note I am a die hard here and will always call it Context cause eventually Oracle will do sufficient name changes and come back to this one)
- User defined functions. Its now so easy to build your own PL/SQL function and embed them in the SQL statement. Which means different ways of passing parameters, behavior and tuning to contend with. Nice to have and to use, so hard to port.

- Data Dictionary. Different data dictionary tables between different vendors. Reference a table and its hard to port it. But referencing dictionary tables is great to have. They make it easy to do a number of advanced features. They are great to have, so we just can't ignore having it.
- Rowid. Which is unique to Oracle. If another vendor has an equivalent concept it can be a pain to try and use it. Which means don't use it. But they are one of the fastest methods used to access a row, so you are crazy if you don't. Did someone say crazy?
- Constraints. Yep. They behave differently between different vendors. Locking issues for starters.
- Performance. Already talked about it, but index usage, transparent performance features (like materialized views) and join techniques differ between vendors. Not easy to port.
- NLS (language support). This is a big issue outside of the US. The world speaks different languages, has currencies other than a \$ and live in different time zones. Can make it a challenge to work with the little idiosyncrasies that different vendors have in this area.
- Security. A user in Oracle is not the same as a user in SQLServer. A database is not the same. Security is different. There might be roles, but connecting to the database? Profiles? Password control? Lets just gloss over this one quickly and hope that no one realizes how many fundamental core issues there are. Because if they did, security on its own could be a showstopper for porting.
- PL/SQL and Java. Both live inside Oracle. Make use of any of them and sorry, no cigar. They don't port. Yet, and this is big, and yet they are both so important to Oracle and performance. Ignore them and you are missing the big picture. You might as well store your data in a flat file system.

So even entertaining the idea that you can build an application that can port without change or even minimal change between vendors, and yet get optimal performance from that application is as ludicrous an idea as it is to make a whole grain donut.

And yet this insane concept is bandied around by pointy haired managers who have read an article in a magazine and think it's the best way to go. As if they would have a clue. My contempt for management and the performance issues they continually cause will be raised in another podcast.

## **Conclusion**

Worried about being locked in? Take a deep breath, relax, think happy thoughts. Feel better? Good, now turn the organization around and set it up to take advantage of the features, capabilities and strengths of the database. Make the application work well.

My view is this, educate management. Tell them that it its a fact of life you will be locked in to the database. It is not such a bad thing, it means you can then build applications that work really well for that database. They are still portable across hardware platforms.

And what makes this even better is that a large number of tuning issues will disappear. Life will be so much easier for the Database Administrator and developer. We can all get back to doing more important things with our time like reading the latest "donut lovers guide magazine".