Live with being Locked into Oracle

Marcel Kratochvil
XOR

A Passionate Viewpoint

Nothing angers me more than having to listen to some project manager start evangelising about why the whole design of a project is built around being open and not locked into any one vendor. “Lets use cold-fusion or better still, lets use Visual basic with ODBC”.

Well I have news for them, and lately I am not so humble, meek and mild in delivering this news. I am short and too the point: You are an idiot. You have just doomed your project to almost certain failure. Sometimes I am not so nice as the above delivery, sometimes I get really angry.

Lets face some reality here. Its not healthy to admit being locked into a database vendor, especially one that has made a lot of aggressive sales. If you admit being locked in, you are vulnerable to licence price increases. That’s a separate issue to deal with and there are ways to get good deals out of Oracle, but I am not covering them in this article. Also, vendor loyalty isn’t the issue here, its really about tuning, scalability and management of the database. The important issues. I am beyond loathe for having to tune databases written to be ported to other databases.

Live with it, when you have Oracle, write your applications to work their best on Oracle. If you do this, then you will get best value for your licence dollar. If you don’t do it, then you hobble your database and what’s more, you are still locked in to it. Lets look at why its hard to build an application that is portable across other databases.

Row Locking
Sure other database offer row level locking to the same isolation level that Oracle offers by default, but for a number of vendors, it’s a trade off. Turn on row level locking with read consistency and performance as well and scalability suffer.

Field and Column Length
Not all databases offer the same datatypes. Some offer maximum field lengths of only 256 bytes, some goes as far as 8000 bytes. With Oracle a Varchar2 field is 4000 bytes, but as of Oracle9i you can start to treat Clobs as Varchars. As an example try porting a field that is over 100k in size and experience the pain.

Maximum Row Size
A table in Oracle can have 1000 columns. Some databases support no more than 256 columns. Its rare though to have a table with more than 100 columns it, but the real issue is the row size. A number of databases still have an enforced limit of 8k per row, which matches the block size of the database. Even though the total number of columns in size could exceed 8k, there is a maximum limit of 8k per row (in the old days it was 2k). That is, a row cannot exceed the block it resides in. A large number of Oracle tables I work on, routinely exceed 16k in size per row. Port the table and who knows what errors will occur.

Multiple Blobs per table
In Oracle7 there was a limit of one long field per table. In Oracle8 you could have as many blob fields as columns. Not all vendors offer the same luxury as this and treat blobs as the forgotten cousin. It can be challenging to port a table with more than one blob (or clob) in it.

Dealing with Blobs
When it comes to working with lobs different vendors use different techniques. Oracle provides a package called dbms_lob to allow developers to manipulate them. Lob’s are notoriously hard to work with, but this package makes it fairly painless. Once you start to use it, you will find it nearly impossible to port it.

SQL Queries
A SQL Query follows a standard but not every vendor can keep pace with the standard. In Oracle9i Oracle started supporting the new join syntax, and yes so did some other vendors, but how many? Write a query using this syntax and there is no guarantee it will run on other databases.

Look at Oracle10g and we see the new model clause. How many vendors can or will even be able to support this clause?

Extend this further and look at all the statistical functions Oracle offers support for. This includes support for multi-dimensional analysis. Look at the CUBE and ROLLUP clauses as well as GROUPING SETS and transparent portability disappears.

So the choice is there, make it portable and don’t use these wonderful features or use them, and get the performance, get the simplicity in application development and truly use the power of the database.

Look at Scalar queries, order by statements in views, and nested queries in the from clause and it's a case of some do but most don't support it.

But it doesn't stop there.

Objects
Abstract data types, varrays, nested tables and REFs. Start using these features and as wonderful and powerful as they are, and ones which I strongly recommend developers use (by strong I mean, you are a novice donut muncher if you don't), there are hardly any databases in the marketplace that can make basic sense of them.

And Oracle internally makes use of Objects. Look at Intermedia, Spatial and Text and immediately you are using objects.

User Defined Functions
Build your own function in PL/SQL and embed it in a SQL statement and voila it will not be able to run on another database. And it gets more challenging, look at all the supplied functions Oracle offers, how many of those are portable? The best example is the *decode* function. Though superseded by the *case* statement, it does not port easily, yet it is frequently used in a large number of applications.

Data Dictionary
If your application references a data dictionary object then it will be a challenge to find the equivalent (if there is one) in another database. Next to impossible to port, but really useful to make reference to in an application.

Rowid
Most developers cannot live without utilising the rowid, yet each database has their own equivalent construct. To be honest, some databases don't even support a rowid equivalent, they need a defined primary key. How much code would need to be rewritten to remove a rowid at your site and how could you live without it?

Constraints
Just because a database offers support for constraints, it doesn't mean they behave the same way or work. Oracle6 allowed you to define constraints but didn't enforce them. There are some databases out in the market place in the same category.

Try doing a cascade update prior to Oracle 8i and yep its very hard to do. The constraint needs to be broken during the update. Only since Oracle 8i did Oracle allow a constraint to be broken during a transaction and only enforced at commit. It's a key feature which most databases do not and most likely, will never be able to support. Very hard to program around.

Performance
All database vendors support the concept of index creation, but have the limitations and behaviour of those indexes been looked at? Limitations with indexes are directly linked to the key length and the Oracle block size. What this means is that an index created in Oracle might not be able to be created in another database.

SQL queries are tuned at run time by the cost based optimizer (CBO). Each database vendor writes their own optimizer geared to work with their database. Start tuning, tweaking or rewriting statements to make use of the Oracle optimizer and there is a good chance they will have to be completely rewritten for other databases. Not all vendors even use CBO and rely on syntax parsing to optimize the statement.

And all those other features
Lets not forget NLS (National Language Support) and dealing with dates, sort behaviour and time zones. And then there is using materialized views, defining batch jobs, sending mail and handling other data structures like XML.

Security
Security between databases can be very different. Not all database vendors have the equivalent of profiles, roles, grants or even integration with the operating system security. Most in the end require the developer to build their own security model on top of the database. Build an application that extensively uses Oracle security and then try and port it. It will not be easy.

PL/SQL and Java
Start building stored procedures and triggers and you are likely to do this in PL/SQL or Java. How many other vendors support PL/SQL and Java? Each has their own programming language if they have one, and none are easily portable. So you either write triggers in PL/SQL and get great performance or you build them into the application using C,

Perl or VB or something else and suffer the consequences.

What is comes down to is that it is possible to write an application that is portable across most databases, but all that will happen is that it will perform to the worst behaving database. It will be clunky, clumsy and ultimately a catastrophe.

So if you are in a project where your managers insist on building an application that can be easily ported between databases then remind them they are incompetent. Use Dilbert cartoons to back this up, nearly any one will do.

If they come back and say, “Well SAP can run across multiple databases”, ask them how much effort each database vendor has to put in to tune and port SAP to run properly with each vendor, and even then there are still many issues.

Find an application vendor that has written an application that runs across many database, and you will find one that does not use the database at all except as a simple table repository. No features of any note are used and the database might as well be an Oracle Version 6 one. All this means is that the complexity is moved into the application rather than being in the database.

My view is this, educate management, tell them that it’s a fact of life you will be locked in to the database. Its 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.