Oracle Facts 1

Not all databases are equal.

What is my criterion for judging the superiority of a database?

Good math. Two principles of good math: 1) Correct implementation of the laws of math. 2) Minimalism. The closer the architecture of the software adheres to the principles of math, the superior the software.

A software system should correctly abide by the laws of math. One would think this is a given, but it is not. Oracle is the only database which screws up basic definitions of set theory (see #23). Would you buy software from a company that believes that 2 + 2 = 5?

Superior software should display minimalism. Minimalism, simply put, is the shortess path to the solution. Minimalism means the fewest number of functions to perform an operation. Minimalism means the fewest number of arguments to a function. Minimalism means the fewest number of steps to calculate a value. Minimalism means the smallest datatypes to define the attributes of an object. Minimalism means consistency. Consistency means that one part of the system does not have to translate information coming from a different part of the same system. Consistency means no cut-and-paste. Consistency means seemless boundaries. Consistency means that the grammar of a language does not change rules within itself.

By these standards, Oracle is a terrible database. It is the least mathematically correct, the least consistent, and the least minimal. I never say that Oracle lacks functionality. I only say that its functionality is so poorly implemented, that it takes a developer far more time to figure it out than Oracle's competitors.

The Oracle zealot is convinced that his volumuous manuals describe magnitudes more features than the competitors. Hence the zealot believes Oracle is better than SQL Server. He simply does not realize that Oracle duped him to believe that more functions mean more functionality. For illustration, what does it take for a developer to do a real-time print from a proc? The T-SQL user writes one line: print 'Hello World.' But in Oracle, the PL/SQL user first has to realize that dbms_output.putline() is not real-time. So the PL/SQL user must create a special table, send information to be table within the proc using insert statements, then write a totally separate program to poll the table for new rows. After all that, he is left to maintain the special table.

The PL/SQL user says, "See my real-time output." And I tell you the truth, he will be proud of it. After all, it took him 8 hours to devise it. Indeed a noble effort. But then if you know other databases, you know that should have taken him about 10 seconds. And then you are thoroughly disgusted with Oracle. You will tell the Oracle zealot that you could have written that in T-SQL in 10 seconds. The zealot will say, "T-SQL is a simple scripting language. " What? Ah-ha. The red herring. Oracle zealots incorrectly equate simplicity with lack of functionality. I guess the zealot would think a hammer lacks functionality.

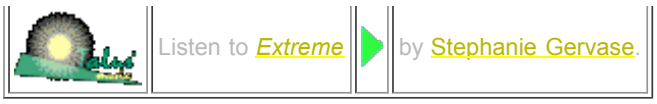
Much of Oracle's functionality is the like the real-time print statement. Oracle's concept of cursors, result sets, procedure language vs. data definition language, misdefinition of null set,, etc, incur astronomical amounts of developement overhead which you don't see in other databases. My complaints below are about such mathematical faults. These faults mean bad architecture.

ORACLE®

Download the Microsoft Word document containing these Oracle facts.

1	The Oracle database does not support basic IEEE integers. Oracle does not have a bit type, a 8-bit integer, a16-bit integer, a 32-bit integer, a 64-bit integer. IEEE formats are the number representations readily processed by the CPU. Without IEEE representations, Oracle must translate billions of numbers to IEEE format in order to do any calculations. There is no work-around for this. It is a feature. Starting with Oracle 10g, the database now has IEEE floating point numbers. Binary_float and binary_double are the IEEE single and double precision floating point formats. Prior to 10g, the Oracle database did not support any IEEE formats. There is still a problem with

	Oracle's binary_float and binary_double. Each type requires any extra length byte. Why? IEEE float and double representations are fixed length. So there is still a needless performance hit. again.
2	Oracle page chains when a column's size increases. That means that a single row splits into uncontiguous fragments each time a column increases its size. Change a null to a string and the row has a new page chain. Change a column from "hello" to "hello world" and the row page chains again. Null, varchar and all number columns are subject to page chaining. Yes . . . even number columns--because Oracle stores all numbers as variable length columns (verifiable with *vsize()*). The larger the number, the more bytes required. Update a column with a smaller number--you waste space. Update a column with a larger number--you now have a page chain. Page chaining is a tremendous performance hit. If you have a 40,000 row table and one of its columns is null, then you update the columns to "hello world", your table now has 40,000 page chains. Oracle DBAs suggest that you make such columns *char*. You may add terabytes of blanks to your table, but you won't page chain. As opposed to Oracle, Sybase never page chains (except for image or text datatypes). A single row is always contiguous. And all numbers are fixed length. Therefore, Sybase does not have to search and put together all the pieces of a single row. They are already together.
3	Oracle;s only bit operator is BITAND. Since Oracle does not store IEEE integer formats, every BITAND consumes tens of thousands of clock cycles instead of a single cycle. Yet another performance hit. If your application needs the results back from frequent bit comparisons, it is imperative that you go with Sybase or Microsoft SQL Server. Sybase and Microsoft store standard IEEE datatypes and use standard bit operators like "*update table set bitmask = bitmask	66536*" which under the covers, Sybase and Microsoft executes with a single CPU instruction. I have written a set of PL/SQL functions which do bit operations for Oracle. They are in DBPowerSuite under the ./dbaccess/sql/oracle directory. However since they are functions, the Oracle optimizer will be inefficient with them.
4	Oracle has at least three different languages: Server SQL, SQL*Plus and PL/SQL. The three languages imply functionality and flexibility. The implication is a facade. The three languages , though, originiate from the fact that the first language Oracle invented failed to do what it needed. So Oracle added a second. The second failed, and so they added a third. Oracle never got it right. Not one of the three languages covers the basics. A user of Oracle must learn all three. It is like learning a subset of T-SQL, a subset of pg-SQL and a subset of mySQL. Only the three combined allow you to do work. Your problem is not absence of functionality, but rather difficulty of use. The languages are not the same. Though all are SQL, they have different syntax. They do separate things. Each language has truly confining limitations--stupid limitations that do not make sense. Also, the languages do not communicate well with each other. They require interfaces. You must also learn the interfaces. Several Oracle fans e-mailed me about this saying, "There is only one SQL language in Oracle. They are all the same." Another person, on the other hand, admitted the disparity. He defended, "SQL*Plus and PL/SQL are a wonderful example of the dichotomy between I/O and the procedural . . . unlike the Sybase popourri." He is right! SQL*Plus and PL.SQL is a wonderful example of dichnotomy. So complete is the split, that one cannot speak and walk at the same time. Below are a few examples of the dichotomy.
5	PL/SQL blocks within a SQL*Plus script ignore SQL*Plus's 'set autocommit on'. Committing transactions within PL/SQL is independent from the SQL*Plus in which the PL/SQL is embedded. The commit syntax is also different.
6	You cannot write any DDL with PL/SQL. 'create table' is invalid with PL/SQL. To the Sybase user, that is like saying that you cannot create a table within a stored procedure. If you want to create tables in an Oracle stored procedure, you have to use dynamic SQL. The table definition may be *static*, but you have to use dynamic SQL because PL/SQL cannot create tables to begin with. This is an up-front misuse of *dynamic*. Oracle's dynamic SQL has a different and obtuse syntax. You will have to learn it. Also, any future DML statements in your PL/SQL stored procedure that manage that table will also have to be in Oracle dynamic syntax. After all, you cannot write insert statements into a table that Oracle doesn't think exists.
7	You must have special permission in Oracle to create a table in a PL/SQL dynamic statement. You must have CREATE ANY TABLE permission. You may already have the ability to "CREATE TABLE X", but if you use dynamic SQL to do the exact same thing, you don't have the permission.
8	There is no *if* statment in SQL*Plus. "if ..., then ...." is impossible with Oracle. It is a standard in Oracle for a programmer to issue SQL*Plus DDL commands knowing that they will fail. You just let SQL*Plus generate an error. That is normal for Oracle. The problem is that "error" means error. One's attention is drawn to it. Each one requires examination. DBAs hate this stuff. Customers hate this stuff. Tech support hates this stuff. The typical tech support's response is "Yeah, that error is okay but that is normal for *Oracle*." To attempt to handle the error, you can introduce the WHENEVER SQLERROR clause in SQL*Plus, but that is only a Band-Aid. That does not prevent the error to begin with. You can also spool out a SQL command generated from a prior select from USER_TABLES, and then execute this spooled file later. The latter method is the best choice to prevent errors. But the method is just another kludge. It introduces a timing problem for the transaction, and you will also find yourself executing empty commands . . . but that is normal for Oracle.
9	Even though you can do a "select col1, col2 . . . from table" in Server SQL, you can not do a standard *select* statement in PL/SQL. This tells you that *Oracle does not support result sets*. Oracle users will rebuke me, "Oracle does support result sets!" But when you read the Oracle manual, Oracle's definition of a result set is not the same. Oracle purposely introduced the term *result set* only so that Oracle fans can say, "We support result sets." It was a marketing ploy only. A result set implies nothing in Oracle. Oracle has no concept of parameter result sets, row result sets, cursor result sets, compute result sets, etc., nor the concept of result sets arriving in a stream to the client.
10	You cannot print anything to your screen from within a PL/SQL block in *real time*. If you want to print out the current iteration of a cursor, just forget it. If you want to print the rows of a cursor as they are being selected, just forget it. Oracle is incapable of doing it. Oracle buffers all the output from a PL/SQL block for printing until *after* the PL/SQL block completes. Even at that, Oracle limits you to 1 MB of buffer after which SQL*Plus blows up. So, when you have a long running loop, there is no way to tell your user about the progress of your loop. To the user, it looks like your script doesn't work. But that is *normal* for Oracle. Oracle programmers will spend a day of programming just to simulate a simple Sybase *print* command.

.	
11	In Oracle, you can only execute one SQL command at a time. Oracle has no concept of a *batch*. This is true in SQL*Plus as well as through ODBC. So, I assume this is a basic Oracle design limitation. In Sybase, you can send 100s of commands in a batch-- an operation which takes only a single network I/O. In Oracle, you must execute 100s of commands to accomplish the same thing. In Sybase, you load up 100 commands in your car and go to the shop and have them all serviced at one time. In Oracle, you must load you car 100 times, go to the shop 100 times, and have each command serviced one at a time.
12	Oracle does not support client-side timeouts. In Sybase, you can program your client to timeout if the server doesn't respond to a login request or to a query in a settable time. This mechanism allows your client to recover gracefully when the server or network has a problem. Oracle offers no such mechanism. In Oracle, you have to kill -9 your client. If you are a capable programmer and have an extra three man-months to spare, you can program threads into your own client which handle timeouts in lieu of Oracle's inability to do so for itself.
13	When creating a stored procedure using SQL*Plus, you may have errors in your procedure. The best error message Oracle can give you is "Procedure created with warnings or errors." One would think Oracle would be more specific. You actually have to go out of your way to request more information. You must additionally type SHOW ERRORS. Note that the error message implies that Oracle will create a procedure with errors in it.
14	Ever try to change your line width in SQL*Plus on Solaris? Do it and then select from the dictionary. Watch SQL*Plus core dump. SQL*Plus has been core dumping for years.
.	
15	Oracle is inconsistent with the quoted identifier feature. Your company may literally waste man months of work over each instance of this problem. As you know, in Oracle you can use a reserved word for a table name if you put quotes around it. The problem is, is that Oracle will let you get by with a reserved word without quotes in one instance but blow up months later when you try to use it another. For example: *create table CLUSTERS*. This command works even though CLUSTERS is a reserved word. You can also *select * from CLUSTERS*. But try: *. . .where CLUSTID not in (select CLUSTID from CLUSTERS)* and watch Oracle blow up. While I can create a CLUSTERS table and select from it, I cannot refer to CLUSTERS in a subquery. The grammar rules of Oracle SQL change within Oracle SQL. A true linguistic nightmare.
16	The Oracle data dictionary stores all its object names in upper case. This is a throwback from the 7-bit ASCII days of the 1960s. Oracle allows you to access the object in upper case, lower case and mixed case. That is fine. But that concept doesn't work for object names where you have used the quoted identifier. Given *create table TEST*, you can *select * from test*, but you cannot *select * from "test"*. Oracle regards this as a *feature*. Any object-name within quotes will be copied exactly to the data dictionary; yet that is not so for *normal* unquoted objects. This is another major inconsistency. Because of the reserved word problem, it is tempting to just put quotes around everything. If you do that, however, then all the object names in your SQL scripts have to match exactly those in the data dictionary Sybase is refreshing after this particular Oracle nightmare. Sybase is simply consistent. What you create is what you get.
17	A blank line within a SQL command in SQL*Plus is a syntax error. Actually, it is a *feature*. A blank line tells SQL*Plus to erase your last command. Any leftover SQL clause following your blank line therefore blows up. To the Oracle user this is normal; to the Sybase user this is stupid. Sybase ignores blank lines and *reset* is the command to erase your last command. The problem is this. You have to make sure all your SQL scripts do not have mid-command blank lines, because SQL*Plus will treat each blank line as an error. Some third party code generation programs are lax with Oracle's wonderful feature, and so those code generation programs will not work with Oracle. In Sybase this problem never happens because Sybase has an architecture. You can change this SQL*Plus default behavior by *set sqlblanklines on*.
18	The problem of not being able to drop the childmost table without the "cascade constraints" clause has been fixed in 9.2.
19	Oracle is incapable of truncating a parent table; that is, the table has incoming referential integrity constraints. It doesn't matter if the table is already empty. It doesn't matter if all its child tables are empty. You have to disable its RI constraints. The extra coding to disable and then reenable RI constraints before and after a truncation, is substantial and a waste of time. The operation is logically unnecessary.
20	The system manager (Sybase equivalent of "sa") is not allowed to grant permissions on a user's objects to other users. In other words, the system manager does not have the permissions to manage the system. In Oracle, only the user himself can grant others permissions to his objects. Oracle calls this a feature.
21	You cannot create a *read-only* view in your *own* schema which reads from tables in another user's schema. You may have select permissions on the other user's tables, but that doesn't matter to Oracle.
22	Avoid creating views based on large tables. Avoid creating views based on views. Oracle's optimizer is extremely lousy at finding the fastest way to the data. Whereas Sybase can return results of a third generation view based on a 20 million row table in 45 minutes, Oracle can never figure it out, even after days of processing, for a table that is 1/20th the size. This Oracle behavior will force you to create work tables, which is what you tried to avoid in the first place.
23	Oracle has screwed up the definition of the null set. NULL has a specific meaning in mathematics, yet Oracle gets it wrong. In math, the null set is the set containing nothing--the empty set. In Oracle, the null set also includes the set containing 0-length strings. The ramifications are severe: 1) An application can no longer discern if a varchar2 column has been touched or not. (An empty string usually means that a user entered data even though the data has no length. A NULL means that the user never touched the column in the first place) and 2)

23	You can longer use a varchar2 as one of the columns in a primary key if it is possible that the varchar2 value is "". The latter limitation will force the DB designer to use a heap table instead of the more desirable index-organized table. That will increase the size of the table significantly and thus will slow down access to it. All this slowness and excess bulk because Oracle has adopted a math convention akin to 0 + 1 = 0.
24	Oracle is single-threaded. Look at your process list (Solaris). Every connection to Oracle has its own process. The listener is its own process. The writer has its own process. The monitor has its own process. Each connection has its own process. In Sybase, all connections and listeners are threads inside the dataserver.
25	Oracle has actually planned a core dump directory for itself. That should tell you something. Note that is it full all the time.
26	Oracle only supports one database per server. Sybase supports 32,767 databases per server.
27	Oracle uses the temporary tablespace to build the indexes of *create index* commands. If you are creating a large index, your temporary tablespace better be huge too. Hint: make sure your init.ora variable SORT_AREA_SIZE is about 20 MB and that you have turned off logging in the temporary tablespace. If you don't do these two things, Oracle may never complete your create index command.
28	Oracle will use rollback segments to create the indexes. The problem is, is that whether the index gets created or not, there is no need to *log* any of the rows of an index to the rollback segment or to the redo logs. Either you can create the index or not. So there is no reason to log individual rows of the index. Nonetheless, Oracle will consume gobs of time and resources to log them.
29	Oracle will use the redo logs to create an index.
30	Oracle will step over its own shared memory bounds set in the init.ora file when creating a large enough index. Once it steps over its bounds, no one can log in anymore because Oracle not only had overstepped its bounds, but also it had leaked all the shared memory.
31	To the person who knows only Oracle, core dump directories, index logging, single-threaded and one-database servers are the trappings of home. To the Sybase user, they are museum relics from Jurassic Park. Sybase does not log the rows of an index. Sybase does not use the temporary database to form permanent indexes. Sybase does not have a rollback segment to also log temporary transactions, as if you would want to do that in the first place. And so, the Sybase user is surprised when it takes Oracle 16 hours to build a unique index on a large table whereas to took 40 minutes to build the same index on the same table in Sybase. I became aware that all this was happening when I tried to build an index and saw Oracle run out of temporary tablespace, run out of rollback segment space (2.5 GB), while busily writing to the redo logs. There is a way to ease the pain of this logging. Shut off logging in the temporary tablespaces and in the rollback segments tablespaces.
.	 Listen to *The Search* by Fields of White.

Home | Buy Software | Privacy