Death to the Health Check… Long Live the Health Check

I have a long history with conventional database system "health checks." I don't like them. I haven't liked them for a really long time, since about 1997, when I realized that conventional health checks are an inferior way to help customers. There is a better way.

by Cary Millsap

In a conventional health check, a consultant gathers system statistics and then creates a long, mostly boilerplate report.

A conventional health check can identify important issues, like whether your software versions are up to date, but it can completely overlook problems that are obvious to anyone actually using the system.

A health check should also include a detailed look at how your top priority business functions spend their time.

When you know how programs spend time, you upgrade your situation from a long list of recommended remedies that might help if you try them, to a short list of opportunities where you know in advance what specific improvement to expect from a proposed investment.

What Is a "Conventional Health Check"?

A "conventional health check" is a consulting service wherein basically a consultant proposes, "Here, run this tool. Send me the output. I'll be back in a few weeks with a report." The report is usually long. Longer reports tend to make everybody happy. Longer reports mean more findings and more recommendations, which in turn means more value.

Except it doesn't.

What's Wrong with Conventional Health Checks?

Here's one thing that's wrong. You get a long list of recommendations. It's almost always long, because, as I mentioned, both customers and consultants prefer longer lists. Because value.

But what are you going to do with a list of fifty-eight recommendations? You got five there marked high priority, fifteen marked medium, and thirty-eight marked low. What are you going to do with them?

Actually, I know the answer to this one, because I visit places all the time that have had conventional health checks.

You will probably try only a few things on the list.

It probably won't be the high-priority ones, because those are hard—maybe politically, maybe just technically. You'll probably try some of the easy ones, no matter what their priorities are. You'll do the "Method C" thing (see Millsap & Holt's *Optimizing Oracle Performance*, page 6), where you try something, and if it doesn't make your system noticeably worse, you'll keep the change; otherwise, you'll roll it back. You'll fiddle. Probably nothing great will happen.

But that's not the worst thing.

The Worst Thing about Conventional Health Checks

I've seen a bunch of health checks over the years. I've participated in some. Here's what happens behind the scenes. If you're lucky:

```
INT. OFFICE BREAK ROOM

K.T., L.V., and the rest of the health
check team consider their situation
after having constructed a 60-page
report using output from hundreds of
scripts. But they know that the scripts
aren't sufficient to create a real rela-
tionship with the system.
```

```
K.T.: So, what if the customer has some
specific, really horrible response time
problem, and our findings didn't even
mention it?

L.V. (exhausted): Oh [redacted]. …That could
_totally_ happen.
```

You could do a whole health check, and... Miss. Everything.

Even More Things Wrong with Conventional Health Checks

Doing your own conventional health checks (or building a business to do it) is really difficult. It's mainly two things:

1. It takes a genius to do a good health check. Such people exist. They're monstrously talented, and they're awesome. But they're rare.
2. Creating a health check is a nondeterministic process. If you take two geniuses, you'll get two different health checks, and you won't be able to prove which one is better.

But these two things are symptoms of a third, more profound thing. You can't know what's important in a system by measuring how it spends its time. You have to measure how the users of the system spend *their* time.

Death to the Health Check

Instead of playing "Wait Wait… Don't Tell Me!" with the business, you need—first!—to learn how the business uses the system. What are the half dozen processes that, if they don't run smoothly, the business doesn't run smoothly? Those are the things you should check the health of.

How do you do it? You trace those processes. Maybe you've heard of tracing and think it's too hard. It's probably not as hard as you think. It's certainly not as hard as hiring a genius to spend weeks looking for weird patterns inside your system, and then following up on dozens of recommended changes.

Tracing is the best way there is to see how your top priority functions use every tier of your system architecture. It's the only way to know if your system is serving your business efficiently. And it's the only way to know if your business is using your system efficiently.

Long Live the Health Check

Checking the health of top priority business functions is the kind of health check that I can stand behind. The report from this kind of health check quantifies opportunities like this:

[Business function F] will execute in P% less time if you [perform remedy action A].

Now you can discuss proposed remedy action *A* as an investment decision, instead of some item in a long list (which may not even have been on your list, because of the break-room realization I described earlier). Now, remedy action *A* isn't a coz-we-said-so recommendation; it's a business opportunity that has a cost and a quantified payoff.

(Consultants are usually good at estimating for you what the cost of a proposed remedy is, but if they're only looking at the interior of your system, then they're probably not going to be able to tell you how much execution duration you'll save.)

Once an opportunity is identified, then the recommendation part is all about the business. Is the payoff—net of the cost, and considering risk—worth the effort? That is a business decision.

What's Right with the Conventional Health Check?

There is value in the conventional health check. Ironically, it's not even the genius stuff that's usually the most important. Here are some items you should definitely be checking:

1. If your database support ends soon, then plan your upgrade now.
2. If you're behind on patching, catch up.
3. Set `db_block_checksum=full`.
4. Run `orachk` (and `exachk` if you have an Oracle Exadata machine). If you've messed with Oracle's default parameters, then you should probably change them back.
5. Test your recovery process. No, not your backup process, your *recovery* process. Actually recover your production database. Make sure you know how to do it, and make sure it works. Practice so you can do it fast.
6. Look at your top CPU- and disk I/O-consuming SQL statements. Even if they don't participate in your highest priority business functions. If they're wasting resources, then they're costing you time and money. Especially if you're in The Cloud.

There are absolutely systemwide things you need to check periodically. But systemwide checks alone are not enough.

You Deserve More

You deserve to know if your business's most important programs are running efficiently, or if they're poised to jeopardize your business. You deserve to know exactly how your business will benefit from some "recommendation" you're considering. You deserve a health check that checks how your business uses your system.

With Method R and our service delivery partners at Cintra Software and Services, that's what you get.

Technology

Method R Workbench is easy-to-use, high-precision *Oracle time measurement software* for health checks, software development, code reviews, performance tests, concept proofs, hardware and software evaluations, upgrades, troubleshooting, and more—for Oracle developers, DBAs, and decision-makers in every phase of the software life cycle. The block ID report described in this monograph is a standard report that is shipped with the product.

◇ Method R™
method-r.com
info@method-r.com