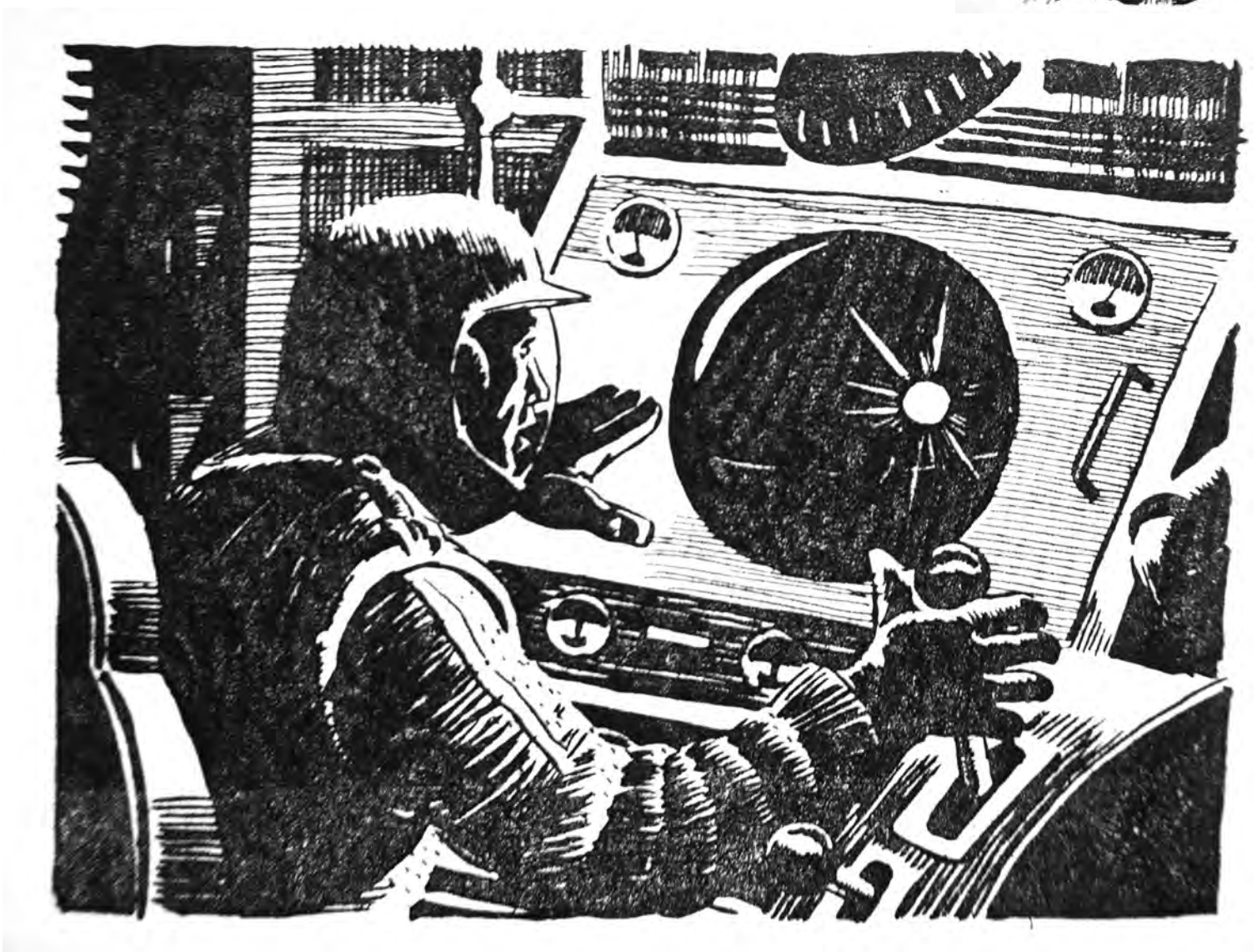
simple talk

The future of database administration edition - Fall 2014



Does NoSQL = NoDBA?

Hekaton in 1,000 words

Win a sombrero

Win a kindle
Installed with all of Red Gate's books

page 28

Welcome to our magazine

Around the Simple-Talk Team, the idea of a magazine to complement the site has been bounced around for a while, and now, happily, we've put that thought into action for some 2014 fall events. This magazine is a trial run, so if you'd like another edition, you're going to need to tell us. This may take the form of a new sign-up to our bi-weekly newsletter, a chat with a member of the Red Gate events team, or leaving a comment on our blog post about the magazine.

In the magazine we have focused on the role of the DBA, including popular articles with useful tips, light-hearted favorites from over the years, and brand new articles on future technologies and trends. This magazine is just a snapshot of some of the content that Simple-Talk produces. On the site you'll also find articles for SQL and .NET developers, free books, and blogs, which have a comments area for our readers. We hope you enjoy this magazine, this event, and this fall.

Happy reading,

Melanie Townsend
Part of the Simple-Talk Team

www.simple-talk.com

Contents

Hekaton IN 1,000 WORDS

By Kalen Delaney

When the SQL Server team set out to design and build a specialized database engine specifically for in-memory workloads, their vision was of a new relational database engine that would be 100 times faster than the existing SQL Server engine, hence the codename Hekaton, from the Greek word (ἑκατόν) meaning 100.

SQL Server 2014 delivers this new engine component, now called SQL Server In-Memory OLTP, and so allows us to work with memory-optimized tables and indexes, in addition to the disk-based tables and indexes which SQL Server has always provided. These new data structures are designed from the ground up to exploit terabytes of available memory and high numbers of processing cores. It potentially marks the beginning of the end of the days of "I/O-bound" SQL Server.

In-Memory OLTP is integrated with the SQL Server relational engine, allowing us to access in-memory data using standard interfaces such as T-SQL and SSMS transparently. However, its internal behavior and capabilities are very different from those of the standard relational engine. Everything you knew about how your SQL Server stores and accesses data is different in Hekaton. Everything you understood about how multiple concurrent processes are handled needs to be reconsidered.

What is a memory-optimized table?
A memory-optimized table is one where SQL Server will always store in memory the whole table and its indexes. Therefore, when accessing in-memory data structures, user processes will always find the required data in-memory.

It sounds like a glorified DBCC PINTABLE…
SQL Server In-Memory OLTP bears no relation whatsoever to **`DBCC PINTABLE`**, a feature available in older versions, whereby if we "pinned" a table, SQL Server would not remove from memory any of its data pages. These pinned tables were no different from any other disk-based tables. They required the same amount of locking, latching, and logging and they used the same index structures, which also required locking, latching, and logging.

By contrast, table and index structures in Hekaton are completely different from their disk-based counterparts. SQL Server can guarantee the ACID properties of every transaction, without taking any locks or latches on memory-optimized tables and indexes, during reading or writing. Readers don't block writers, and writers don't block readers, or other writers. Also, logging changes to memory-optimized tables is usually much more efficient than logging changes to disk-based tables.

What's so different about how data is stored in Hekaton?
The standard relational engine is architected to assume that the data resides on disk. It reads and writes 8 KB data pages, as a unit, to and from disk, and stores them in extents. Each page "belongs" to a particular object. This process can generate a lot of random I/O and incurs a high latency cost.

For memory-optimized tables, there are no data pages and no extents. There is no notion of data rows being written to a particular location that "belongs" to a specified object. There are just "data rows", written to memory, in the order the transactions occurred, with each row containing an index "pointer" to the next row. All "I/O" is then in-memory scanning of these structures.

Many versions of the same row can coexist at any given time. This allows concurrent access of the same row during data modifications, with SQL Server making available the row version relevant to each transaction, according to the time the transaction started relative to the timestamp values stored in the header of each row version.

So there really is no blocking in Hekaton, ever?
Hekaton removes the standard assumption that the data resides on disk. For memory-optimized tables, SQL Server will never have to acquire latches nor perform I/O, in order to retrieve data from disk. Furthermore, SQL Server can guarantee to preserve the ACID properties of all transactions without acquiring any locks. Therefore no transaction will ever be blocked, waiting to acquire a lock.

The row structure described above underpins a new, truly optimistic concurrency model using a Multi-Version Concurrency Control (MVCC) system, where SQL Server maintains multiple row versions, and determines the correct row version that each concurrent transaction should access.

In the MVCC model, writers don't 'block' writers. If transaction Tx2 tries to modify a row currently being modified by Tx1, SQL Server optimistically assumes that Tx1 will commit, raises an update conflict, and aborts Tx2 immediately (rather than block it, awaiting the final outcome of Tx1).

Also, readers don't block writers, nor vice versa. SQL Server optimistically assumes that concurrent readers and writers won't 'interfere', but performs validation checks, post-commit, to detect any potential violations of the properties specified by the transaction isolation level.

However, this does not mean there is never any waiting when working with memory-optimized tables in a multi-user system. A complex or long running transaction on an OLTP system can still cause problems, even with Hekaton. For example, it may cause waits while dependencies resolve during the validation phase. Transactions might also need to wait for log writes to complete. However, in either case the waits should be of very short duration.

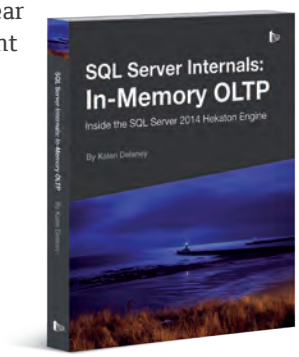
Is Hekaton really 100x faster?

You may not find that Hekaton is 100 times faster, but could be many times faster, depending on the workload. A key factor is that memory-optimized tables are natively compiled. SQL Server holds in memory not only the table and index structures, but also a set of DLLs for accessing and modifying these data structures. The table metadata encodes into each DLL a set of native language algorithms that describe precisely the row format for the table and how to traverse its indexes. Hekaton also offers natively compiled stored procedures, which generate far fewer CPU instructions for the engine to execute than the equivalent interpreted T-SQL stored procedure, and can be executed directly by the CPU, without the need for further compilation or interpretation.

When we access memory-optimized tables from natively compiled stored procedures, we have a highly efficient data access path.

What's the catch?

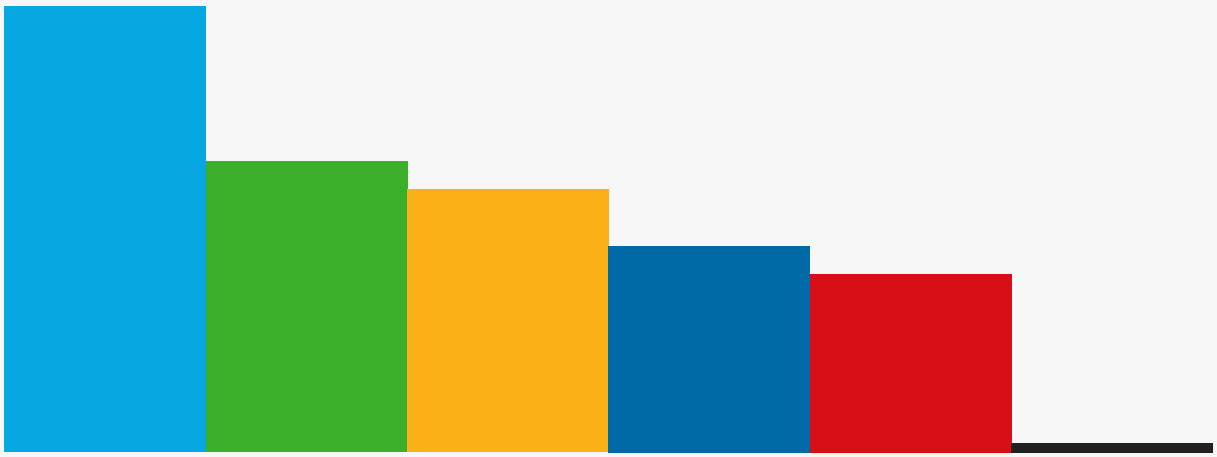
Currently, there are quite a few limitations on the range of data types supported for memory-optimized tables, restrictions around use of indexes and constraints (no FOREIGN KEY or CHECK constraints), and limitations on the T-SQL language constructs that are allowed inside a natively compiled stored procedure. In addition, there are other requirements, such as the need to specify a binary collation for any indexed character fields on in-memory tables, which could make migrating to Hekaton difficult, in some cases. Some of these 'barriers' may disappear in subsequent releases.



I'm intrigued...

Download Kalen Delaney's new book for free at: **www.simple-talk.com/books**

What is going to cause the biggest change to the way people work with databases in the next 5 years?



31% Cloud adoption (public, private, or hybrid)

21% Non-RDBMS alternatives

19% Proliferation of mobile devices/ apps

15% Increased virtualization

13% Internet of Things

1% Other

The biggest change we'll see is in the area of "intelligence in the data" – things like Cortana, Watson and the like – requiring higher levels of meta-data around a specific item. We're moving to a more AI-based world, something that has been predicted since the 1980s, but this time it's real, it's rapid, and it's a new way of interacting with data.

Buck Woody

If you're a DBA, you won't see big changes in five years. Memory and SSD costs will continue to plummet, which will continue to make on-premise database servers blazing fast and ever cheaper, thereby negating some of the cloud's advantages. Indie developers have already seen the change, moving their work toward the cloud, but when they want to get serious and move into the enterprise, they'll be working hands-on with databases again.

Brent Ozar

The next five years will find many DBAs needing to learn how to work with data and not just plan on administering systems. Our platforms are so mature, that if you can't provide more value from data analysis or reporting to a company, I'm not sure they'd pay you instead of letting a developer, Windows Admin, or other Accidental DBA to manage databases.

Steve Jones

Version Control as a Cloud Service

By Robert Sheldon

This is an excerpt from Rob Sheldon's article. In the full article he discusses centralized vs distributed version control, public and private development strategies, and the costs involved. Find the full article on Simple-Talk.com.

Why bother with hosted version control?

A version control system protects code, maintains a history of changes, and enables collaboration. These days, few developers would consider building an application without it. Some teams see hosted version control in the same light, believing a hosted cloud service to be as essential as the version control system itself. And they have good reason for feeling this way.

As the number and size of the version control repositories grow, so too do the implementation and maintenance costs associated with housing an on-premises system. Even with distributed version control, a team will still often designate a single repository as the one source of truth that can integrate with other development tools and serve as the primary source for deployments and backups. Having a "central" server also ensures a credible copy of the code is stored offsite, should disaster hit a development center.

A cloud-based version control service can eliminate many of the implementation and maintenance costs associated with in-house hosting. Because service providers have access to massive infrastructures and data centers, such as those available through Amazon EC2, they can also better support geographically-dispersed development and high-availability requirements. In addition, these services ensure that regular maintenance tasks are performed, such as backing up the repositories at regular intervals and implementing tested recovery strategies. Cloud services also have the advantage of on-demand scalability beyond the capabilities of most in-house environments. Organizations can expand and contract their operations as necessary, without costly long-term investments in equipment, software, and other resources.

Cloud-based version control services also make it easier to set up a new project and get started. Plus, most providers offer additional services that are integrated with the source control system, helping to streamline the entire development process. For example, source control providers usually offer integrated issue tracking so that repository commit operations can be linked to specific tasks or issues. They might also offer project management services that are integrated with the version control system. Most vendors provide much more than just a "repository in the cloud", and it's all those extras that can make hosted version control shine.

That's not to say a hosted cloud service doesn't have its downside. If you're building an open source solution, public exposure is more a benefit than a risk. However, many organizations using hosted services want to keep their development efforts private, and uploading proprietary code via the internet and storing that code in data centers in the cloud carry its own set of risks. What happened at Code Spaces (the target of a DDoS attack that resulted in the deletion of most of the company's data) is just one example. For those subscribers whose data was wiped out, the degree to which they'll be able to recover that data will depend on whether any of their users have an up-to-date workspace (though chances are, someone will). But other tracking and project management data could be lost.

Yet it's not just deleted data that's a concern. If a provider's security protections are compromised, the integrity of the code could be compromised. In addition, code files and other sensitive data that end up in the wrong hands could make a company even more vulnerable. What happens to that data if the provider goes out of business? Which of the provider's employees have access to your data? What safeguards are in place to protect against malware?

In addition, the very nature of a cloud-based service makes it vulnerable to disruptions in internet access and system-wide outages. Disruptions can range from overloaded ISPs, to regional power failures, to internal system failures, to cyber attacks. True, an on-premises system is also vulnerable to an assortment of risks, but at least with an in-house solution, you have some level of control over data access and what you can do if your systems are compromised.

On the other hand, if you do go with a cloud service, you can take steps to help protect your code, such as digitally signing your files or implementing automated vulnerability tests as a part of the build process. In some cases, you might even find that the service provider has put into place a more robust security model than what you have available in-house. And for those development projects open to the public, it might be especially beneficial to give the hosted service a try, especially if that service is free.

Mistake number 5:
One table to hold all domain values

By Louis Davidson

"One Ring to rule them all and in the darkness bind them"

This is all well and good for fantasy lore, but it's not so good when applied to database design, in the form of a "ruling" domain table. Relational databases are based on the fundamental idea that every object represents one and only one thing. There should never be any doubt as to what a piece of data refers to. By tracing through the relationships, from column name, to table name, to primary key, it should be easy to examine the relationships and know exactly what a piece of data means.

The big myth perpetrated by architects who don't really understand relational database architecture (me included early in my career) is that the more tables there are, the more complex the design will be. So, conversely, shouldn't condensing multiple tables into a single "catch-all" table simplify the design? It does sound like a good idea, but at one time giving Pauly Shore the lead in a movie sounded like a good idea too.

For example, consider the following model snippet where I needed domain values for:

- Customer CreditStatus
- Customer Type
- Invoice Status
- Invoice Line Item BackOrderStatus
- Invoice Line Item ShipViaCarrier

On the face of it that would be five domain tables...but why not just use one generic domain table, like figure 1?

This may seem a very clean and natural way to design a table, but the problem is that it is just not very natural to work with in SQL. Say we just want the domain values for the Customer table:

```
SELECT *
FROM Customer
  JOIN GenericDomain as
CustomerType
    ON Customer.CustomerTypeId
= CustomerType.GenericDomainId
      and CustomerType.
RelatedToTable = 'Customer'
      and  CustomerType.
RelatedToColumn =
'CustomerTypeId'
  JOIN GenericDomain as
CreditStatus
    ON  Customer.CreditStatusId
= CreditStatus.GenericDomainId
      and CreditStatus.
RelatedToTable = 'Customer'
      and CreditStatus.
RelatedToColumn =
'CreditStatusId'
```

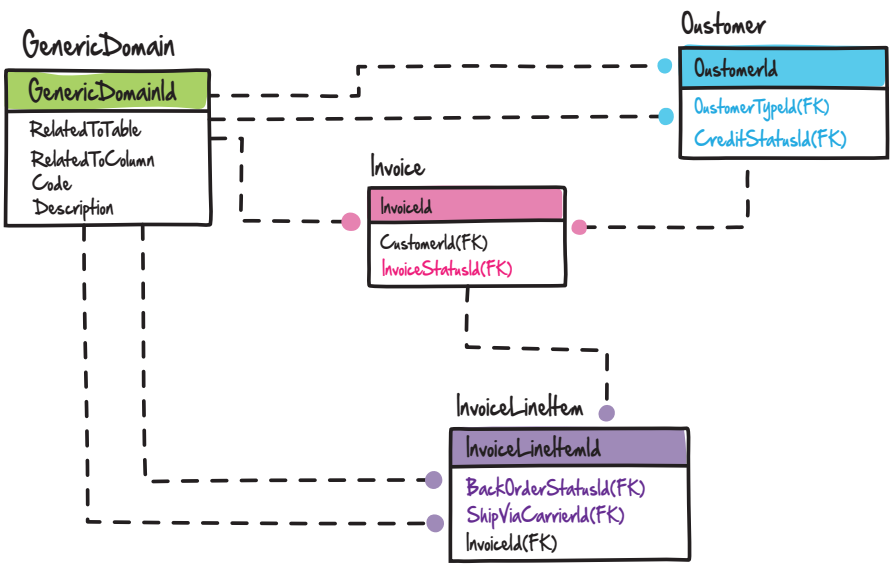


figure 1

As you can see, this is far from being a natural join. It comes down to the problem of mixing apples with oranges. At first glance, domain tables are just an abstract concept of a container that holds text. And from an implementation-centric standpoint, this is quite true, but it is not the correct way to build a database. In a database, the process of normalization, as a means of breaking down and isolating data, takes every table to the point where one row represents one thing. And each domain of values is a distinctly different thing from all of the other domains (unless it is not, in which case the one table will suffice). So what you do, in essence, is normalize the data on each usage, spreading the work out over time, rather than doing the task once and getting it over with.

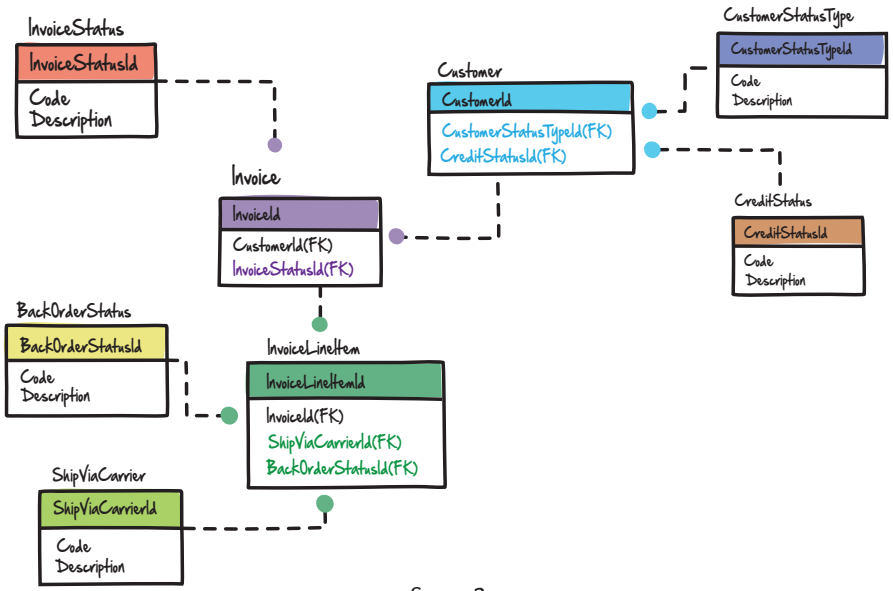


figure 2

So instead of the single table for all domains, you might model it as figure 2.

Looks harder to do, right? Well, it is initially. Frankly, it took me longer to flesh out the example tables. But, there are quite a few tremendous gains to be had:

- Using the data in a query is much easier:

```
SELECT *
FROM Customer
  JOIN CustomerType
    ON Customer.CustomerTypeId
= CustomerType.CustomerTypeId
  JOIN CreditStatus
    ON Customer.CreditStatusId
= CreditStatus.CreditStatusId
```

- Data can be validated using foreign key constraints very naturally, something not feasible for the other solution unless you implement ranges of keys for every table – a terrible mess to maintain.

- If it turns out that you need to keep more information about a ShipViaCarrier than just the code, 'UPS', and description 'United Parcel Service', then it is as simple as adding a column or two. You could even expand the table to be a full blown representation of the businesses that are carriers for the item.

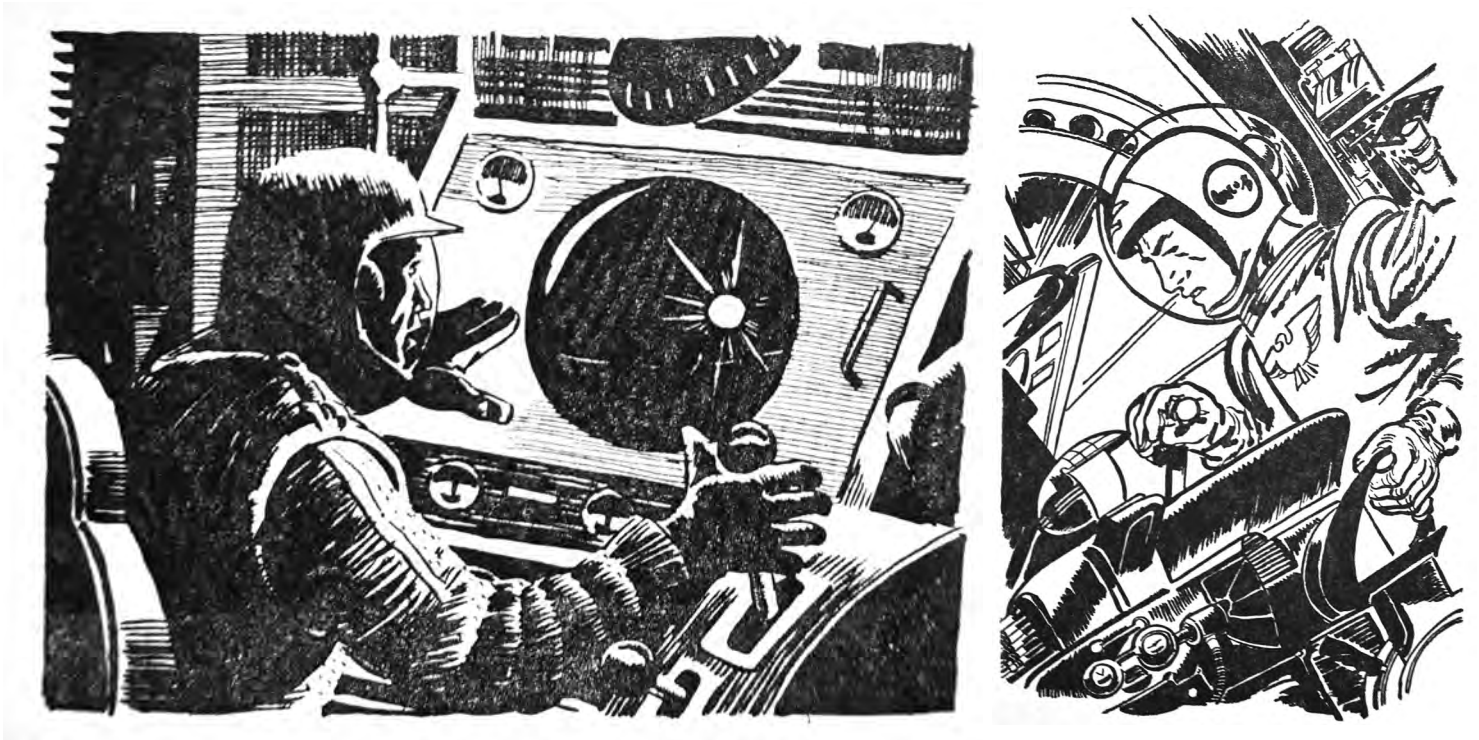
- All of the smaller domain tables will fit on a single page of disk. This ensures a single read (and likely a single page in cache). In the other case, you might have your domain table spread across many pages, unless you cluster on the referring table name, which then could cause it to be more costly to use a non-clustered index if you have many values.

- You can still have one editor for all rows, as most domain tables will likely have the same base structure/usage. And while you would lose the ability to query all domain values in one query easily, why would you want to? (A union query could easily be created if needed, but this seems unlikely.)

I should probably rebut the thought that might be in your mind, "What if I need to add a new column to all domain tables?" For example, you forgot that the customer wants to be able to do custom sorting on domain values and didn't put anything in the tables to allow this. This is a fair question, especially if you have 1,000 of these tables in a very large database. First, this rarely happens, and when it does it is going to be a major change to your database either way.

Second, even if this task was required, SQL has a complete set of commands that you can use to add columns to tables. Using the system tables, it is pretty straightforward to build a script to add the same column to hundreds of tables all at once. That will not be as easy a change, but it will not be so much more difficult to outweigh the large benefits.

The point of this tip is simply that it is better to do the work upfront, making structures solid and maintainable, rather than trying to do the least amount of work to start out a project. Keeping tables down to representing one "thing" means that most changes will only affect one table, after which it follows that there will be less rework for you down the road.



TO BOLDLY ASK... ...IT FOR DEVELOPMENT

BY PHIL FACTOR

Phil has always been mystified by the way that, in science fiction films, the crew of spaceships are able to reprogram their ships' computers in order to respond to emergencies, needing no more than a brief klip...klop... klip on the keyboard to effect a huge software change. A life in IT has seemed so different, and so he wonders if there is a more realistic way that one might imagine IT's contribution to space adventures.

When it happened, chill fear swept the control room of the spaceship. The thunderous booms against the hull were the sound of the shields being subjected to an attack of unparalleled ferocity. The crew were thrown from side to side in their chairs. Briefly the lights went out, and then flickered back on as the emergency power supply sprung into life.

The viewing screen burst into a frenzy of weird static, suddenly replaced by the sight of a ghastly alien creature, resembling a giant cockroach. Briefly, there were whirs and ticks from the loudspeaker until the automated language translators kicked in.

"ldstr-ldloc-stloc-ldarg-cctor-ctor...(Click) ...vile and remorseless. So, pitiful specimens of a degenerate species, you dare to approach our galaxy. Your intrusion is beneath contempt, and you will be ruthlessly eliminated."

"We come in peace! We seek only to promote understanding and harmony across the galaxy," intoned Admiral Clarke.

"Silence! Your impertinence will lead to your immediate destruction, you lamentable products of a degenerate species. We of the Xunil civilization will prevail, and crush all resistance! Our space-fleet approaches!"

The vile alien creature flexed his mandibles in rage.

"The ruthless swine: he's armed with a thesaurus and prepared to use it," muttered Dr Strabismus.

"You know, it may be a good idea to get the hell out of here." Admiral Clarke leaned over to talk to the chief engineer. "Scotty, how much power can we mobilize to put the main engines in full reverse-thrust?"

"Och noo, captn," came the crackly voice from the engine room, "I just canna doo it. Any more powwr, and the wee engines will bloo."

"We could, of course, reprogram the shield generators to give us the necessary reverse thrust," said Dr Strabismus.

"I was just wondering that myself," replied Admiral Clarke, blushing slightly. "But what do you mean by 'reprogram'?"

"We'll have to get the IT department involved."

For the second time in a day, a look of terror flickered across Admiral Clarke's face.

"It's either that or being eaten by a very angry giant insect."

"A surprisingly similar experience, in fact," sighed Admiral Clarke. "Ask Dan, the IT Man, to come to the control room."

-=+=-

A few minutes later, the door opened with the familiar 'Fssshhhwheffffft' sound. Dan the IT Man entered the control room looking important.

"Dan. We need you to reprogram the force...er...thrust things."

"The shield generators, Admiral," corrected the doctor.

"I don't think that's quite the request you intended to make, Admiral," replied Dan firmly. "If you can couch the request in business terms, justifying the required resources, we can come back to you and respond with a technical proposal to meet the business need. We can certainly discuss timescales and priorities, bearing in mind that we

are currently focused on our 50-day maintenance window. If you consider it an urgent change, you can add it into our change-control system and we can append it to our list of ongoing projects. However, it would help if you would indicate priorities so that next month we can plan the work for the developers."

Dr Strabismus made a 'Fssshhhwheffffft' sound, remarkably like the door. Dan gave an ingratiating smile before continuing. "Of course, we'll need a paper that outlines the business benefits so that we can then get management-signoff for the work to be done, and perform the appropriate benchmarks."

"Dan, we are about to be attacked by psychopathic creatures that look remarkably like giant cockroaches. Isn't there some quick hack you can make to the code that runs the shield generators that will allow us to apply a reverse thrust to supplement the power of Scotty's engines?"

"We could always form an Agile Scrum and rely on test-driven development techniques to reduce the lead-time, but we run great risks if we depart from our usual commitment of high standards of resilience and three-nines downtime. That," he drew himself up to his full height and looked severely at the Admiral, "is surely something that you would never wish to countenance."

"Dan... they are rumored to paralyze their victims, and lay eggs in them to provide fresh meat throughout the larval stage. The larvae then hatch messily and dramatically."

Dan was impassive. Then a placating look flashed across his face. "Look, the best I can do for you is to use the current maintenance window to provide an interim solution to your current 'issues'. As you are aware, there are already numerous 'overdue' bugs reported with regard to the generator units. I think the best way forward, vis-à-vis your ad-hoc request for extra development resources, is to add "lack of reverse thrust" to this list of bugs and then go through them and appraise and refine our understanding of their precedence, reprioritizing the most urgent as 'requested features'. Of course, we must also bear in mind all budgetary implications and ensure that all health and safety issues are appropriately considered..."

Admiral Clarke stared quizzically at Dr Strabismus. "I got a bit lost in this speech. Did he say no or yes?"

"I think he is managing our expectations, Admiral."

"This is going to be a long day."

-=+=-

'FSSSHHHWHEFFFFFT'

The vile space creature clanked awkwardly into the control room inside its space suit, from which wires and tubes protruded, dribbling a disgusting green gas. "Ah-ha!" it cried, seeing Admiral Clarke hiding behind his rather grand seat. "You have evaded me for the last time, Admiral!"

Behind him, a group of Xunil cockroaches trooped into the room.

Xunil Ruler: Xunils, annihilate these pathetic examples of an endoskeletal life-form! Exterminate! Conquer! Destroy!

All Xunils: [in unison] Xunils conquer and destroy! Xunils conquer and destroy! Xunils conquer and destroy! Xunils co……

Xunil Sidekick: Exterminate!

All Xunils: [in unison] Exterminate! Exterminate! Exterminate! Exterminate!

Xunil Ruler: (interrupting) OK, OK, enough of that. What are we? Daleks? Take up positions, ready to eradicate all human beings.

Xunil Sidekick: Obey, or you shall be extermin…eradicated!!

Admiral Clarke: What do we do now, Doctor?

Dr Strabismus: (quickly glancing up at the open door) This way!

Door: [the doctor, Admiral Clarke, and the rest of the cabin crew rush through the door. The Xunils watch intently as the humans leave] Fssshhhwhefffft.

Xunil Ruler: Ha! Their lamentable attempts at evading capture are utterly derisory! We have these pathetic creatures at our mercy. Their fate will be swift. Xunils, commence the eradication procedure!

Xunil Sidekick: Erm….what does that entail, exactly?

Xunil Ruler: Fool! We will merely need to reprogram the crude and ridiculous space vessel to use a chlorine-based atmosphere so we can discard our space suits. The pathetic carbon-based life-forms will die immediately. Send for the effete spokeshuman of their decadent IT department!

All Xunils: [in unison] Send for Dan! Send for Dan! Exterminate! Exterminate!

-=+=-

'Fssshhhwhefffft'. Dan walked briskly through the door, into the control room holding a file and looking important.

"So, impudent spokeshuman of a…"

"Despicable?" prompted the Xunil sidekick.

"...Yes, yes, a despicable and impudent IT department. I need your discreditable and impotent skills in reprogramming the space ship to use a chlorine atmosphere."

Dan opened his file and glanced at the schedule. "Of course, we are sensitive to the imperatives of the current business re-engineering pressures, and will give a measured and sustainable response just as soon as the current backlog of work can be cleared."

"Impertinent specimen of a degraded life-form! I just need a quick change to the current air-recycling system to make it use chlorine instead of oxygen! It is something that any of my Xunil IT team could do in minutes, if only they could read your piffling documentation and understand your pathetic IT architecture."

"Yes, but unfortunately, our dedicated air-recycling unit response team are all currently working on some reliability issues with our transporters. I'm afraid there is little hope of any sort of

reassignment of priorities."

"So you refuse to do my bidding, impudent life-form?" A stream of hot chlorine gas burst from a tube on his space suit.

"We have a number of policies that have been put in place with clearly defined targets. The definition of policies and targets represent a key element in our management approach to 'customer' focus. We are responsive to your changing business requirements, within the context of these policies and the constraints of our existing workload," replied Dan with only a momentary loss of composure.

"Enough! Silence. Tahter, take this impudent human back to our ship and perform the customary business with the anal probes."

"Sure, Boss. I've forgotten though, stupendous master, why we do all that stuff with anal probes whenever we come across a human?"

"The great God Sdlavrot created all sentient things with a button that you press to revert the system software to its factory settings. It is usually down a hole somewhere. Now, enough idle chat, we shall get our own heroic and accomplished developers from our glorious mother-ship to spend their weekend making the necessary change to the software. It can't be too hard. Then we can destroy the crew of this ship."

Xunil Sidekick: Exterminate!

All Xunils: [in unison] Exterminate! Exterminate! Exterminate! Exterminate!

-=+=-

The two insect-like forms hunched over the terminal in their alien space suits. "Who, in the name of the great God Sdlavrot, wrote this software? How primitive their thought-processes are."

"I expect that they still peel their own potatoes, boil them for 20 of their minutes, then smash them to bits." The aliens collapsed into helpless laughter.

"Look, it is all table-driven. You just load the proportion of all the gases in the atmosphere that you want. You then execute that stored procedure and the software does the rest. Easy. What a fuss that those human-female blouses were making about such a simple change."

"Shouldn't we set up some sort of test-harness?"

"Test? Nah! Do creatures like us with our superior brainpower make simple mistakes? What could go wrong?"

"Well," answered the other Xunil software developer as he scanned the contents of the table, "there are some pretty toxic gasses here. If we accidentally used helium instead of chlorine, it would rot the plastic fibers of our space suit almost instantly, and leave us to fry in that corrosive oxygen-nitrogen mix that these humans seem to thrive on. Still, the chances of that happening appear to be slim."

"And when we're being threatened by Tahter with instant disemboweling if we don't meet the deadline, where better to make savings than in the Test Cycle? Who'll notice?"

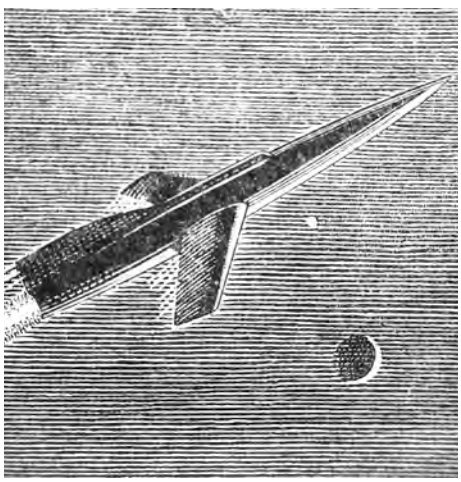
-=+=-

Deep in the darkness of a hiding place in the ship's hold, Admiral Clarke, Dr Strabismus, and the rest of the crew lay waiting. "I don't like it, there were some blood-curdling screams and then the whole ship went quiet. They're up to something dastardly, I fear."

There was an embarrassed silence as the crew stared at Admiral Clarke, whose voice had come out as a high-pitched squeak.

"That's a neat trick; where did you learn how to speak like Pinky and Perky? Or is it Donald Duck?"

"Helium," said Dr Strabismus tensely, "The cunning bastards have put helium into the ship's air supply."



Everything you ever wanted to know about The Internet of Things but were too afraid to ask

What exactly is this 'Internet of Things' people are talking about, what effect – if any – will it have on developers and DBAs, and why should we care about it? Good question. So here it is. Not the most definitive, but definitely the most interesting answers to questions you might have about everything IoT.

What exactly is this Internet of Things?

Imagine a world where lots and lots of things are connected together, talking to each other and using the computing power of the cloud to be as smart as a supercomputer. Those things could be central heating systems that turn on when your car is five miles from home. Garbage cans on public streets that send an alert when they need emptying. Heating and lighting systems in offices that automatically adjust, depending on how many people are in the building. That's the Internet of Things. Lots of things working away in the background, communicating with each other to make our lives simpler and easier.

What's driving the Internet of Things?

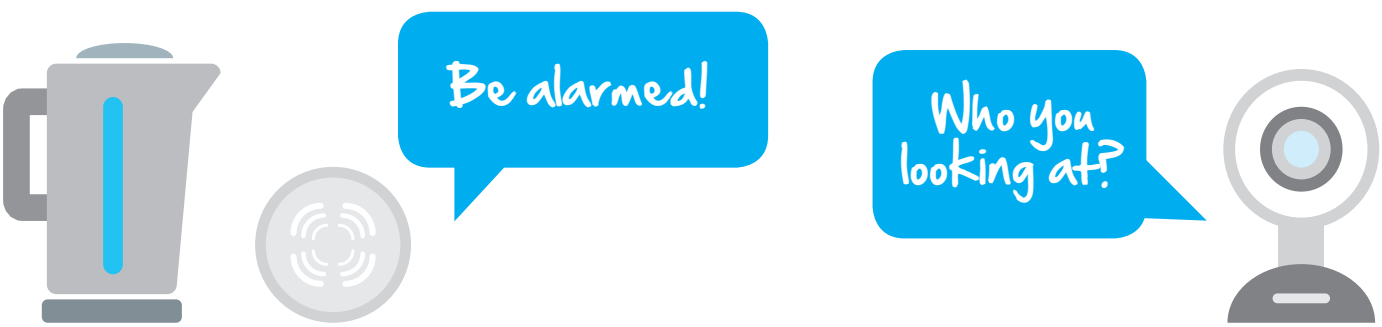
Two factors. The first is connectivity. The internet, Wi-Fi, and Bluetooth are now ubiquitous – so ubiquitous that it's becoming unusual not to be connected. The second is the miniaturization of technology. Engineers from Stanford and Berkeley universities recently created radio controllers the size of an ant. Costing just pennies to make, and able to power themselves from incoming electromagnetic signals, they're the blueprint for the controllers that will soon be embedded in all of the 'Things'.

So how big will this Internet of Things become?

Estimates vary – and they vary wildly. Gartner say there will be 26 billion internet connected devices by 2020. Morgan Stanley and Cisco forecast there will be over 50 billion. IDC raise the stakes to 'approximately 212 billion'. The truth is, it's going to start slowly and then scale very, very rapidly. Some people are already hinting that the Apple Watch is likely to be the device that kicks it off, with the power on your wrist to control everything from your TV to your central heating. Once that becomes the norm, expect a massive proliferation as more and more companies find niches, gadgets, and uses we never thought we needed but will find rather handy.

What's the biggest threat to the Internet of Things?

Security, security, and security. Let me say that one more time: SECURITY. Those billions of devices out there will also become billions of endpoints, with security varying from high to none. The point is that as the spider's web of the Internet of Things becomes ever more complicated, connecting more and more things to each other, one thing could be the gateway to compromising many things. The jury's out on the best way to address the issue, but there have been major presentations about it at conferences like Black Hat USA and DEF CON.

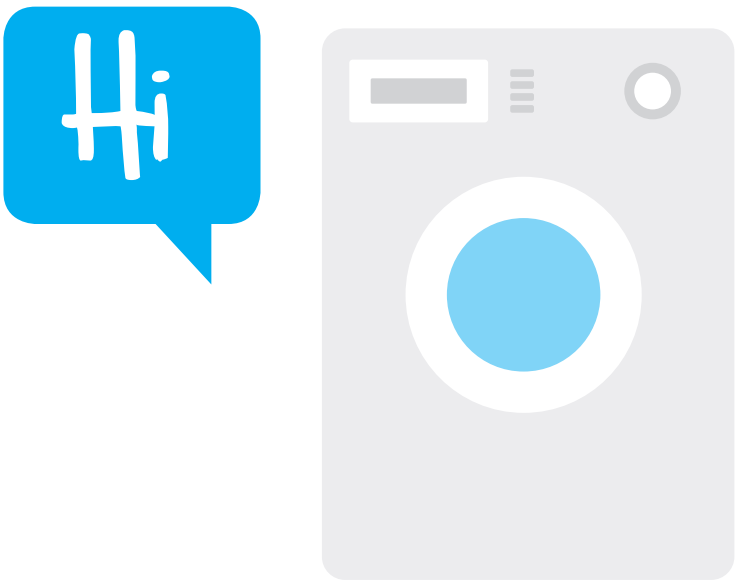


How much data will the Internet of Things produce?

A mountain of the stuff. In its April 2014 report, 'The Digital Universe of Opportunities', IDC predicted that the amount of data we create and copy annually will reach 44 trillion gigabytes by 2020. And of that, the Internet of Things will account for 10%. That's an awful lot of data to store, process, and analyze.

Who's going to handle all of this Internet of Things data?

Good question because, by its very nature, IoT data is Big Data. A mix of structured, semi-structured, and unstructured data from a plethora of sources. Just the kind of data that NoSQL databases are made for. That doesn't leave SQL developers and DBAs out in the cold, however. Far from it. The Internet of Things will actually be the enabler that brings together the SQL and NoSQL communities, because everyone will have to work together on different stages of the data lifecycle. NoSQL databases will process data on the fly. SQL databases will analyze much of the data later on.



Isn't the Internet of Things just a horribly complicated pipedream?

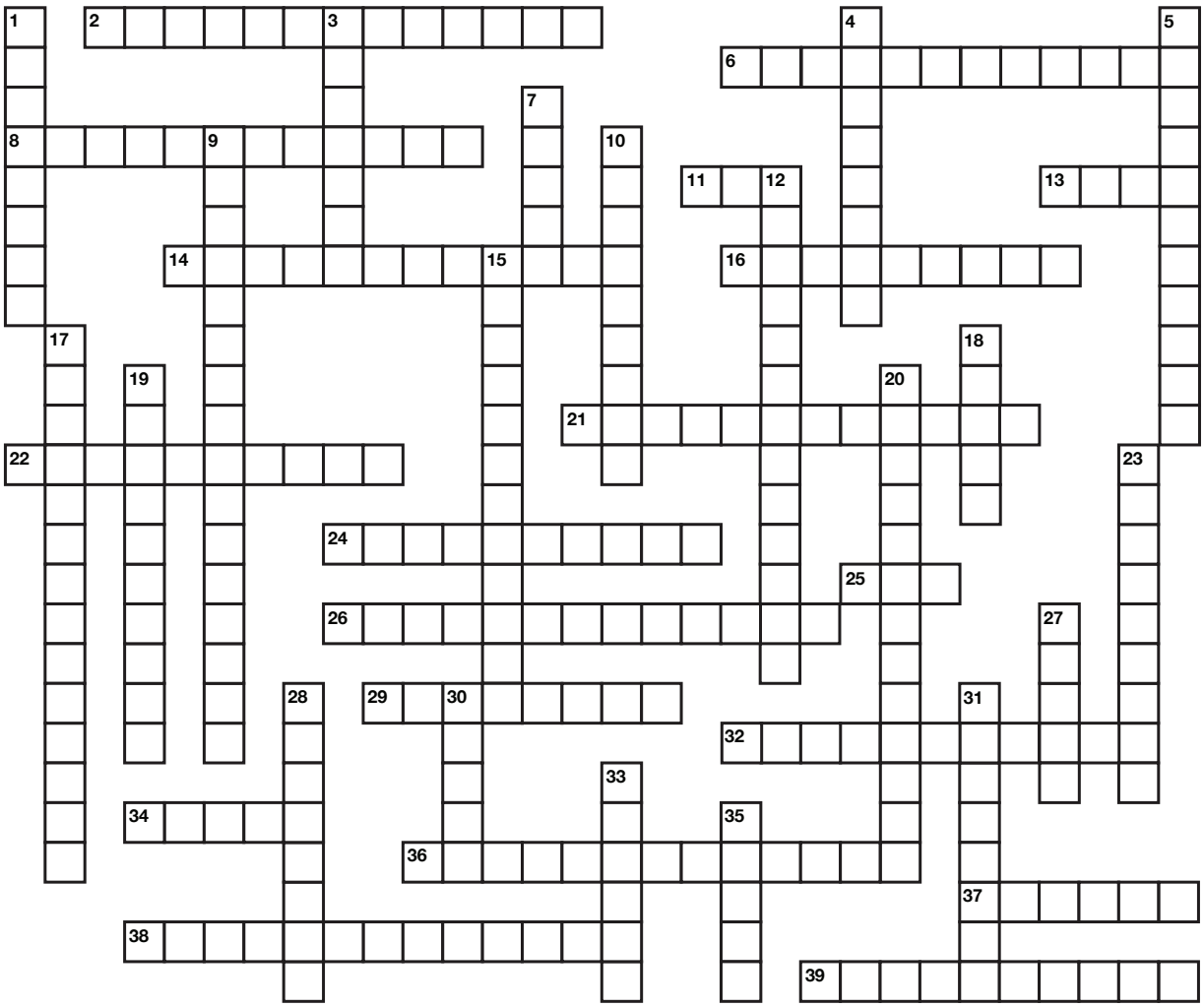
In its 2014 'Hype Cycle for Emerging Technologies', Gartner placed the Internet of Things right at the top of the peak of inflated expectations. In other words, there was a lot of talk, but it is still five to ten years away. Apple, Google, and a lot of other companies don't appear to be listening to Gartner.

There are already home security systems out there that you can access remotely with a mobile phone. Apple is broadening the possibilities by promoting its HomeKit as the foundation for home automation, the first step towards the Internet of Things. Google, meanwhile, acquired Nest Labs for US$3.2 billion in January 2014 as the cornerstone for its own home automation push. A July 2014 survey from Evans Data revealed that 17% of software developers are already working on applications for connected IoT devices, with another 23% expecting to start work on IoT devices in the next six months. Pipedream? No. The potential for changing the way we live our lives? Absolutely.

But hang on – why should I care about this Internet of Things?

Two reasons. One, it's going to change the way we live our lives. The internet has been the biggest revolution in technology to date. The Internet of Things will be the next revolution. And two, it's going to change the working lives of developers, DBAs, and any of us connected with databases. There's a tidal wave of data approaching on the far horizon and a lot of it will be heading your way.

Crossword



Across

2 A Showplan Operator through and through (7,6)
6 Got Responsive Web Design? (4,8)
8 He writes for PoSh DBAs (6,6)
11 A Piece of 8 (Sounds like the second in a series) (3)
13 (FN) Wrote the book on Exchange 2010 (4)
14 Author of a brand new book on Hekaton (5,7)
16 Microsoft man with a Data Science Lab (4,5)
21 .NET writer, likes to go Back to Basics (4,8)
22 The legendary confessor, manager, instructor, and mystery (4,6)
24 How Simple-Talk reaches 250k bi-weekly (10)
25 A Piece of 8 (Pretend there's a dot) (3)
26 Got a Question You Were Too Shy to Ask? (6,7)
29 Cloud author, also the editor of Just Azure (4,4)
32 Ninja Immutable Databases author (6,5)
34 With 28D (FN) He's pretty frightening (5)
36 The interviewer of all the weekly geeks (7,6)
37 (LN) Simple-Talk editor, author, and .NET lover (6)
38 This man can keep a SQL Beat (6,7)
39 Known for project management, TSQL, and fishing (5,5)

Down

1 She codes SQL in the wild (4,4)
3 A Piece of 8 (everyone's got one) (7)
4 A DBA Team baddie who writes SQL books (3,5)
5 He's got the Group Policy Management knowledge (6,5)
7 Simple-Talk stars Jonathan and Annette _____ (5)
9 An all-around web developer who covers LINQ, MVC, and CSS (6, 10)
10 Editor who has written many an editorial over the years (4,5)
12 What Counts for a DBA? Reading this guy (5,8)
15 Plagiarists flee this Simple-Talk Editor (6,6)
17 Have the Mindset of an Enterprise DBA? (6,8)
18 A Piece of 8 (A place of caption competitions, editorials…) (5)
19 Can you back up with PowerShell? (5,5)
20 Author of the TDD series, PowerShell One-liner Series, the list goes on (7,6)
23 A guest star in the DBA team series, also known for her Workbenches and Cribsheets (5,4)
27 A Piece of 8 (Things are looking up) (5)
28 With 34A (LN) And a DBA (8)
30 ___ Kellenberger (5)
31 A Piece of 8 (Starts sounding like a sibling) (8)
33 A Piece of 8 (A funny thing happened on the way to them) (6)
35 A Piece of 8 (Full of downloadable PDFs) (5)

Answers on page 28

4 PREVENTABLE BACKUP ERRORS

The Scenario: There's a system outage. Someone has dropped a table or deleted massive amounts of critical data. It's the first time it's happened on any of the systems that you're responsible for. You know that the systems team set up some sort of backup for all the files on your production servers. That's a relief. So, you go to get the .mdf and .ldf files that have been backed up... Only they're not there, or, they are there, but when you try to restore the database they don't work.

What Happened: This frequently happens in systems where you have no one responsible for the database server at all, or a really new data professional. It's not common knowledge out in the real world that SQL Server requires you to run a particular type of backup. A file is a file, right? Yes, but, SQL Server takes out locks on all the database files. This can cause a file backup system to skip the database files. It will usually raise an error, but after two or three weeks of looking at the same error, people frequently just turn it off, skip those files, whatever - but it means you don't have a backup of your database files, at all.

Some file backup systems can even figure out that a file is locked, but back it up anyway. Now, you'll have your database files, but you still won't be able to use them. First, you can't run a **RESTORE** operation from database files; you can only **ATTACH** them to the server. So, you think, fine, I'll just **ATTACH** them. But then you can't. The reason for this is that the SQL Server engine manages transactions, and transactions control how data gets written to the disk. It's entirely possible for your backup to capture the files with uncommitted, half-complete transactions, and you can't roll forward or roll back in an **ATTACH** unless you did a **DETACH** first, so that SQL Server can clean up the open transactions. These file backups are useless.

How You Prevent It: SQL Server has a method for creating backups, built right into the product: **BACKUP DATABASE**. Not only will it create a file (or files) that you can **RESTORE** the database from, but it also does it in a way that figures out all the transactions, so you don't have to worry about that either.

The Scenario: Suddenly, all the backups are failing. You've planned ahead and you have an alert that fires from the SQL Agent job that runs your backups, but everything is failing. Looking at the error log, or in the error message you see "Insufficient disk space for backup" or some similar message.

What Happened: You've run out of space on the drive where you have your backups. It happens a lot. It's extremely common. The causes are sometimes more difficult to nail down. The simplest cause is the one you have the least amount of control over. The database grew. Because of this growth, it needed more space for the backup. Other causes you can do things about. Some people like to create a new backup file for every backup, putting a date and time into the file name along with the database name, so that it's easy to tell when the backup was taken. But they don't always take as much care with creating or maintaining a clean-up script that removes backups after a few days.

How You Prevent It: You have to monitor your drive space. You need to also monitor your database sizes. If they're growing, so will your backups. You also need to be sure of your backup cleanup process. I've seen these fail so often, it's worth putting another check in place that validates the age of backups and sends you an alert, or sends you an alert after it deletes the old backup files. Just test these processes so that you're not deleting newer backups that you might need immediately, and be sure that you have older backups available from some off-site storage system.

Other than that, the best prevention here is to keep an eye on your disk space. Not only should you worry about the amount of free space, but you need to be cognizant of the rate of growth.

The Scenario: You've got backups in place, but they're running extremely slowly. They're running so slowly that sometimes you get timeout errors and the logs fail. Other times they're running so slowly that SQL Server is affected and you have to kill the backup process in order to speed up the system. In this same situation you may also be seeing excessive load on your network.

BY GRANT FRITCHEY

What Happened: There can be many causes for this, but one of the most frequently encountered is running the backups across the network. Backing up to a local disk or to a SAN through a dedicated fiber channel is usually just as fast as the disk can handle. But when you backup across the network, you run into all sorts of contention and bottlenecks, competing with Twitter, Facebook, and everyone's Words With Friends games. That contention causes backups to run slowly.

How You Prevent It: Don't backup across the network. I realize it can be more difficult than that, especially if you're in a crunch for disk space and there's no budget for more. But there's really little you can do about it. Where possible, backup to local disks. If space is really an issue, backup locally, then copy the file across the network. This two-step process might make for a longer overall backup routine, but the backups themselves are more likely to complete successfully, which is the important part.

The Scenario: You've been happily running a differential backup each day. You've never hit a failure and everything is good. Now you've had the inevitable outage and you go to your differential backup and run a restore. Only thing is, you get an error: the log or differential backup cannot be restored because no files are ready to roll forward.

What Happened: Differential backups work only in combination with a full backup. Sometimes the full backup is lost and people attempt to restore the differential without. Sometimes there's just a misunderstanding of how the differential backups work. It's also possible that a full backup was done prior to running the differential, but the database was not left in the recovering state necessary to apply differential and log backups. Any one of these issues could have been the cause.

How You Prevent It: You must take full backups and use them as the basis for a restore process in combination with differential backups. Since you can't simply restore a differential, this is the only way to make this work. Further, when you restore the full backup, it must be left in a recovering state in order to apply the differential. The script would look like this:

```
RESTORE DATABASE
MovieManagement
FROM DISK = 'g:\bu\
MovieManagement.bak'
WITH REPLACE, NORECOVERY;
```

This will restore the database in question, but leave it in a recovering state. In that state a differential backup can be applied. If logs must be restored as well, then the differential restore must also leave the database in a recovering state by using the **NORECOVERY** option.

Because differential backups are completely dependent on the full backup, it's very important that you have a tested full backup available in order to work with differential backups. There is no shortcut around this requirement.

BIG DATA

What's the big idea?

BUSINESS CASE

Big Data helps businesses uncover and take advantage of new opportunities. By analyzing all the data that comes in, businesses can gain insight into new products and services that will appeal to their customer base.

IT CASE

Big Data uses structured, semi-structured, and unstructured data sources, opening up a whole new way of modelling data. Instead of the ETL up-front approach, data specialists can extract and load the data, transforming it on an as needed basis.

LET'S TALK ABOUT MONEY

HP has partnered with Hortonworks, which includes a **$50 million** investment.
MapR have closed a **$110 million** financing round (including **$80 millio**n from Google Capital).
Intel have invested **$740 million** in Cloudera.
Market beam forecasts the Big Data market to grow to a **$50 billion** industry by 2020.

BARRIERS TO ENTRY

Steep learning curve

Recognize that all the skills you need for your team don't need to live in a single person. Red Gate makes beginner tools like HDFS Explorer to help people get started.

Security concerns

Look closely at what each vendor offers. They will each have built their own security layer on top of Hadoop. Healthcare companies and law-enforcement agencies have adopted Big Data solutions, so the answers are out there.

Investment

While big claims are bandied about on the cost savings when processing terabytes of data, there's no denying this will be an IT project and will require a budget. Get a quote, make a plan, and make sure the IT department and a C-level sponsor are on board.

Should IT Managers Code?

By Tony Davis

In one of his first ever Simple-Talk articles, Phil Factor tells the story of a freelance Sybase programmer who created a reporting system using exquisitely complex dynamically compiled stored procedures, and then promptly departed when he failed to secure a doubling of his contract rate. As Phil struggled to make sense of the code, with the business baying for bug fixes and impossible improvements, he learned a valuable lesson for any IT Manager:

"the way to do it is for IT managers to spend at minimum 30% of their time programming"

"Never let a programmer do anything that you yourself cannot understand" – Sir Tony Hoare (Computer Language expert, inventor of Algol and the Quicksort algorithm).

At first glance, it would seem an odd sentiment, but Tony Hoare was referring to code that couldn't be understood even after being explained to a technically-savvy manager. He was warning of the dangers of complexity.

Certainly, there needs to a restraining force that prevents the typical development team's tendency towards complexity. There is generally a yawning gap between the technologies that an ambitious programmer wants to use and the technologies that best suit a project. A manager needs enough programming nous to help keep the technology as simple and reliable as possible. In the face of their countless managerial, strategic, and company political duties, the struggle is to maintain their programming skills to the point where they can spot if a programmer is adopting an unnecessarily complex or unwieldy approach, or championing "unproven" technology, when something duller but more reliable would suffice.

"A manager needs enough programming nous to help keep the technology as simple and reliable as possible."

According to Eliot Horowitz, the CTO of MongoDB, the way to do it is for IT managers to spend at minimum 30% of their time programming. If they don't, he argues, they "encounter a significant degradation in their ability to execute their responsibilities", and their "connections to the concerns of developers atrophy...decision-making, planning, and leadership suffer".

Is Horowitz really correct that a failure to do this simply means one becomes an ineffective manager, as well as programmer, over time?

It is only recently that professional people, doctors, lawyers, engineers, or scientists have felt it to be an option to slide into a generic managerial role, to neglect or abandon the professional skills that first enabled them to rise in the profession. A part of their job was to keep up with changes in their profession. Until the 1980s, management skills were acquired as part of career development and it was unthinkable that a generic manager without professional skills could direct professional work.

Maybe technical leadership is undervalued. Recently, some of the most catastrophic IT failures have been precipitated by startling technical shortcomings, rather than general lax management. Would a more technically-savvy management have been able to spot the imminent disaster and prevent it?

Does NoSQL = NoDBA?

There's a joke doing the rounds at SQL conferences and seminars: 3 DBAs walk into a NoSQL bar and leave when they can't find a table. You may have heard it before, but it made us sit down and ponder at Simple-Talk. What's happening? Are DBAs and NoSQL mutually exclusive? Donning disguises and holding our collective breath, we bravely entered the shiny new world of NoSQL to investigate ...

What, no SQL?
NoSQL databases appear to be popping up all over the place. Google, Amazon, Facebook, LinkedIn, and Twitter all use them. New entrants in the NoSQL arena like MongoDB, CouchDB, Cassandra, and Riak are claiming converts everywhere. Articles about NoSQL (including this one) are being written, talked about, and discussed.

The figures behind it all sound impressive as well. In May 2014, the 'NoSQL Market Forecast 2015-2020' from Market Research Media forecast that the global NoSQL market would reach $3.4 Billion in 2020, representing a compound annual growth rate of 21%.

What's going on? The term NoSQL, which originally meant NotOnlySQL, was first used by Carlo Strozzi in 1998 to name the file-based database he was developing. It was, in fact, a relational database but it didn't use a SQL interface. The term re-emerged a decade later when a growing number of non-relational, distributed data stores started to appear.

Today, while there's still no formal definition of NoSQL, the boundaries have been clearly established. NoSQL databases still don't use SQL and the relational model has been completely abandoned. They're designed to run on clusters. They tend to be Open Source. And perhaps most importantly, they forego fixed schemas in favor of allowing any data to be stored in any record.

Where's the DBA?
Perhaps most interestingly, the rise of NoSQL seems to be outside the realm of DBAs. In September 2013, Nick Heudecker, a Gartner Analyst, conducted an informal survey of NoSQL adopters to find out who was using NoSQL and why.

'the rise of NoSQL seems to be outside the realm of DBAs.'

The result? Just 5.5% of respondents were DBAs. In Nick Heudecker's words: "DBAs simply aren't a part of the NoSQL conversation. This means DBAs, intentionally or not, are being eliminated from a rapidly growing area of information management."

So why are some companies turning to an unstructured model that leaves the four principles of ACID in the dust?

Let's talk Big Data
Relational databases are great for handling structured data, but Big Data isn't just structured data. Typically, it's large, complex data sets that are a mix of structured, unstructured, semi-structured, or multi-structured data. Think text-heavy social media posts at one end of the scale, and web log data with a combination of text and visual images at the other.

Take Facebook, for example. Over a billion users worldwide access it every second of every day, adding data, changing data, updating data, attaching videos and photographs, connecting to more and more people.

Relational databases can be created to handle this kind of data, but NoSQL databases work much faster because they don't have the ACID overhead.

And Big Data is going to get bigger. In its 'Worldwide Big Data Technology and Services 2013-2017 Forecast', IDC predicts that the Big Data technology and services market will be worth over $32 billion in 2017, and will have grown six times faster than the overall ICT market. That's a lot of data.

So what's the big idea behind NoSQL?
To find the answer, we need to pop over to Amazon. In 2007, Amazon discovered that every 100ms of latency on the Amazon website cost 1% in sales. At the time, their annual sales were around $14.7 billion. And 1% of $14.7 billion is a lot of sales to lose.

The problem was that, according to the CAP Theorem (see sidebar), it is impossible for a distributed computer system to simultaneously offer Consistency, Availability, and Partition Tolerance. Only two can ever be guaranteed at the same time and, in a standard relational database model, Consistency and Availability are always chosen over PartitionTolerance.

But it was the Partition Tolerance that was causing the latency problem.

So Amazon did a very clever thing. In their seminal 2007 white paper, 'Dynamo: Amazon's Highly Available Key-value Store', they outlined an approach for a new kind of database. One that guaranteed Availability and Partition tolerance at the expense of Consistency.

Rather than the Enforced Consistency of a traditional database where all clients always have the same view of the data, they looked closely at their data and analyzed the type of data and its requirements. The result? They opted for Eventual Consistency, where data would be consistent in the end. In effect, they transferred the latency issue from the time delivering the website content to users, to the data being delivered.

They figured that absolute Consistency wasn't absolutely necessary for all types of data. Even if a customer paid for a book and the last copy was sold five seconds before, they could simply order another book and the customer would wait another day or so.

They resolved their latency problem, they regained those lost sales – and at the same time, their white paper inspired the development of many other NoSQL databases including Cassandra, Voldemort, and Riak.

Saying yes to NoSQL

That development arose because it was realized that while relational databases are perfect for transactions and commerce, other types of data could be handled by a different kind of database at a lower cost.

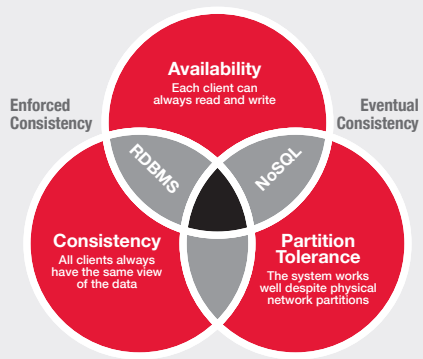
Why, for example, use a relational database for data from weather sensors, science research, machine logs, web click streams, social media, etc? Particularly, when the absence of fixed schema in NoSQL databases can make them a more suitable option.

The figures behind the facts

All this talk about NoSQL sounds uplifting for NoSQL followers but – and this is a big but – is NoSQL really grabbing the headlines as well as the sales?

The CAP Theorem

The CAP Theorem emerged from the University of California, Berkeley, in 1998, when computer scientist Eric Brewer posed that it was impossible for a distributed computer system to simultaneously guarantee Consistency, Availability, and Partition Tolerance.



In a relational database, Consistency and Availability are regarded as essential qualities at the expense of Partition Tolerance. As a consequence, if a network partitioning event occurs, causing a loss of connection, the database can't service all requests. This Enforced Consistency ensures that all clients always have the same view of data.

NoSQL databases take a different approach, by offering Availability and Partition Tolerance at the expense of Consistency. The idea is that network partitioning events can occur because the database is run on clusters. If one node fails, another one steps in, allowing read and write operations to continue. Updates are then propagated asynchronously so that Eventual Consistency is achieved.

For a bank where transactions have to be consistent, that just wouldn't work. For companies like Google, it's acceptable. It doesn't really matter if someone in Ohio has a slightly different search result from someone in Syracuse. There are still hundreds of thousands, sometimes millions, of results.

The Mexican Standoff

If SQL and NoQL were ever to meet in a bar, they would probably challenge each other to explain themselves. What do you think? Does SQL win? Does NoSQL come out on top? Or is it, indeed, a Mexican standoff?

SQL

Maturity

RDBMS have been around a long time, they're stable, richly functional, and companies are accustomed to them.

Support

RDBMS vendors go to great lengths to provide a high level of support to enterprises, ensuring their database are always available.

Analytics

Mining a database to get at the business intelligence it contains is simple using ad hoc SQL queries.

Administration

SQL databases need DBAs and developers to manage, administer, maintain, and enhance them.

Expertise

There are, literally, millions of SQL developers, all working to fairly common standards, and able to handle even complicated changes.

NoSQL

Future-ready

Data needs are growing exponentially, and only NoSQL can distribute the database across multiple hosts to accommodate them.

Economics

Rather than expensive, proprietary servers, NoSQL databases use clusters of cheaper commodity servers that need little support.

Big Data

The volumes of Big Data that can be handled by NoSQL systems like Hadoop can't be handled by the biggest RDBMS.

Simplicity

NoSQL databases require less management, with simpler data models leading to lower admin and tuning requirements.

No expertise required

Application changes and database schema changes are far easier with NoSQL (as long as it is remembered that schemaless structures still have an implicit schema).

Win a sombrero

We'd like your help to find out the result of the Mexican Standoff with our online survey. Visit www.simple-talk.com/sombrero and tell us who you think would win the debate – or if it would be a standoff. There are ten sombreros to give away in a prize draw. So visit the site, place your vote, and leave your email address so we can contact you if you win a sombrero. That's all it takes. And you'll see the survey results on simple-talk.com.

DB-Engines, for example, is an online initiative that ranks the popularity of database management systems, and updates the list monthly. In September 2014, only two NoSQL databases, MongoDB and Cassandra were in the top ten. The clever people at DB-Engines also provide deeper insights into the ranking, one of which is job offers. And do you know what? A lot more jobs are out there for Oracle, Microsoft SQL Server, and DB2 developers and DBAs than those in the NoSQL world.

Similarly, the '2014 State of Database Technology Survey' from Information Week also bucked the trend. It found that 75% of respondents were using Microsoft SQL Server and 47% were using Oracle. Compare that with 13% using Hadoop and 5% using MongoDB. Even Filemaker was ahead of new NoSQL companies like Cassandra and Riak.

As the survey points out: "While Riak and other newer databases may be making inroads into enterprises, they don't seem to be displacing existing RDBMSes as much as they are being used for greenfield applications."

So what's the deal, then?
The deal is that we need to use possibly the ugliest phrase in the technology dictionary: polyglot persistence.

Polyglot persistence simply means using different databases depending on the level and type of data persistence you require.

Many companies will keep their relational databases for applications like OLTP where the level of data persistence is, by default, very high.

At the same time, when new needs arise because of Big Data, new apps or cloud-based offerings, they might think non-relational.

And in some cases, both will be chosen. A relational database, for example, is an expensive way to store data. Depending on the requirements of the data, Hadoop might be used to store the raw data and the aggregated data might then be processed into a relational database for fast service and interactive queries.

So at the end, it really is a case of Not Only SQL rather than seeing NoSQL as No SQL. Instead of shying away from NoSQL, it's a good idea to start learning about it, because soon SQL and NoSQL will be co-existing alongside each other in the same company.

And we've saved the best news for last. The US Bureau of Labor statistics forecast a 15.1% employment growth for database administrators between 2012 and 2022.

The Bureau also advised an 'above average' stress level for DBAs, but you probably knew that anyway.

Staying ahead of the game

Change is afoot. A lot of change that will transform the way DBAs work. But what exactly is going on – and what can you do to stay ahead?

It's an interesting time, right now, to be a DBA. 44 years after E F Codd introduced the term 'relational database' in his seminal paper, 'A Relational Model of Data for Large Shared Data Banks', a revolution is looming on the horizon once again.

Nine years after Codd's paper, the first commercial relational database appeared and streamlined the way companies thought about data banks. If they became relational databases, information could be retrieved from them faster and easier. Rather than just being a place to store records, they could become the place to analyze data. Since then, DBAs have been administering the data. If there's a question, ask the DBA. If there's a problem, call the DBA. If there's a compliment due, praise the DBA.

And it's worked. Daily monitoring and maintenance by DBAs has supported the availability of databases. Performance tuning and recovery planning have ensured that availability continues with as few problems as possible.

Times have changed.

The world's moving faster

Deploying updates to databases is an essential part of an infrastructure DBA's job, but the frequency of those updates is increasing. Rather than now and again, updates are now being requested a lot faster. The faster changes can be shipped, the quicker new features can be released to ever more demanding customers. (Facebook typically ships updates twice a day.)

Alongside more demanding customers, Agile thinking has made developers more demanding. The Agile Manifesto emerged in 2001 and reversed common development thinking. Individuals and interactions became more important than processes and tools. Working software was preferred over comprehensive documentation. Responding to change was the calling cry, not rigidly following a plan. The consequence? A bottleneck at the point when database changes move from development into production, because DBAs are sometimes left out of the loop.

'Responding to change was the calling cry, not rigidly following a plan.'

Data is getting bigger

When Codd was busy writing his white paper, mainframes were still around and IT people talked in terms of kilobytes of data. When relational databases emerged at the end of the 1970s, megabytes of data were being used. The launch of the internet in the 1990s made us start thinking in gigabytes. Those gigabytes soon grew to terabytes. Now we're talking zettabytes.

We're all familiar with a gigabyte. A single zettabyte is 1,000,000,000,000 gigabytes. And a variety of forecasters predict that the exponential rise in data growth will result in the need to store 44 zettabytes of data worldwide by 2020. That's compared to around 4.4 zettabytes today. A ten-fold increase.

'Now we're talking zettabytes.'

It's not just the size of the data that is going to cause an issue. It's the form of the data. Alongside the structured data we currently work with, there will be Big Data – unstructured, semi-structured, and multi-structured data. All kinds of data from a multitude of sources that needs to be processed, stored, and analyzed.

The Internet of Things is arriving

One of the sources of unstructured and many-structured Big Data will be the Internet of Things. Lots and lots of things out there, collecting data and sharing data with lots of other things, using the internet as the communications channel.

The first building blocks of the Internet of Things are already out there. Apple is offering its HomeKit, a suite of tools for controlling devices in the home. Google has invested in the home automation system, Nest. And utilities are jumping in to claim the space for themselves, promising the ability to improve healthcare, communication and entertainment, and security.

And all of this is before business and industrial applications take off, adding sensing, monitoring, and remote management to everything from cars to streetlights to the heating systems within large commercial buildings.

'The first building blocks of the Internet of Things are already out there.'

Open Source is opening new doors

Relational databases still dominate areas like trading and finance, but new kinds of databases are emerging to handle the data types now emerging in fields like science, information-sensing, and internet search. They're mostly open source, and they've been developed to handle large-scale processing of data-sets across clusters of commodity hardware.

Hadoop, for example (named after the toy elephant of one of the founder's sons), is gaining more and more traction as the platform that makes Big Data easier to manage. It's not relational, it stores files and processes data in a completely different way to SQL Server, and it's not alone.

MongoDB promises to be more agile and scalable. Couchbase boasts that it provides the world's most complete, most scalable, and best performing NoSQL database. Alongside them, companies like Amazon, Google, Facebook, and LinkedIn run their websites on proprietary NoSQL databases.

Interestingly, a lot of this non-relational activity appears to be planned far away from the DBA. In its 2014 State of Database Technology Report, InformationWeek found that the NoSQL distributed database platform, Riak, was used by just 1% of respondents to its survey. Yet more than one third of the Fortune 100 employ the technology, supporting InformationWeek's conclusion that non-relational databases are not replacing relational databases. Instead, they're augmenting them to offer companies new capabilities that DBAs just aren't involved with.

The age of one company, one database, is over

For years – decades, even – companies and organizations have traditionally favored one kind of database, be it Microsoft SQL Server or Oracle, MySQL or IBM DB2. Virtualization helped, with its ability to run different databases – and many databases – on the same system, but the general truism remained.

'organizations have traditionally favored one kind of database'

That's going to change, and in many ways has to change, to accommodate the massive rise in data and types of data that's on the way.

Alongside the traditional relational databases, non-relational databases are going to become the norm in many companies and organizations. Each with an important role to play in the business.

Big Data is just ... data

Barely has the term Big Data become accepted and already analysts are out there saying that Big Data is just, well, data. A lot of it, but data all the same that has to be collected, processed and analyzed.

The thing is that much of the data isn't the structured data we're comfortable with in the relational database world. It will be multi-structured data from biological research, wireless sensor networks, social media, etc. Massive piles of the stuff that has to be processed as fast as possible because there's another massive pile that will arrive tomorrow morning.

Platforms like Hadoop are perfect for processing it, but guess what? Once it's processed, relational databases will be waiting in the wings to analyze it.

So make friends with Hadoop, MongoDB, CouchBase, Riak, etc. They're not the enemies waiting to steal your pay check. They're the allies that will make sure your pay check comes in on time.

It's time to think a little differently

In the past, applications have come along and they've been updated annually or every six months. There has been time to follow a natural process from testing through to staging through to production. Now, new applications are emerging that demand high availability, a high volume of writes, geographic distribution and lots of upgrades.

All of which means that the methodical release process of yesterday has to be replaced with a faster, more efficient process. One that borrows some of the Agile thinking, and starts to use methods like version control, continuous integration, and automated deployment. That way, rather than being the bottleneck that slows developers down, DBAs become the enablers that help developers work faster – and protect the data at the same time.

There's more than enough room for everyone

Remember that estimate we saw before, of the forecast need to process, analyze and store 44 zettabytes of data by 2020? A lot of that will be whisked away to non-relational databases which are better able to handle unstructured and multi-structured data. But a large part of it will be heading in your direction as well, because relational databases remain an important part of the equation.

The only difference is that in the near future – and it's not far away – you'll be sitting alongside the people doing the non-relational stuff. Make friends. It's going to be an interesting journey.

THE KRAMEK CODE CHALLENGE

By Andy Kramek

What is the result of executing the following code?


```
; WITH cte_customerstatus ( CustomerID
                          , CustomerName
                          , CustomerStatus
                          , AccountNum
                          , CurrentBalance
                          , LastCheck )
  AS (  SELECT CUS.customer_pk
             , CUS.customer_nme
             , XRF.Status_cde
             , ACT.Account_nbr
             , ISNULL(ACT.CurrentBal_amt, 0)
             , ISNULL(ACT.LastCheck_dte, GETDATE())
          FROM dbo.customer CUS
               INNER JOIN dbo.customer_xrf_account XRF
                  ON XRF.customer_fk = CUS.customer_pk
               INNER JOIN dbo.account ACT
                  ON ACT.account_pk = XRF.account_fk )


UPDATE cte_customerstatus
   SET CustomerStatus = CASE WHEN CurrentBalance > 0 THEN 'Good'
                             WHEN CurrentBalance = 0 THEN 'Neutral'
                             ELSE 'Delinquent'
                        END
```

1. Nothing at all, updating a CTE has no effect on anything and does not cause an error.

2. SQL will generate an error because although you can use a CTE in an update statement, you cannot actually update a CTE.

3. The column named "status_cde" in dbo.customer_xrf_account table is updated even though the update is to the CTE column named "CustomerStatus".

4. SQL generates an invalid column name error because although updating a column in CTE can update the underlying table, the column names must match.

To enter the prize draw simply mark your answer on the separate card included in this magazine and submit it at the Red Gate booth.

CROSSWORD ANSWERS

Down
1 Gail Shaw
3 Opinion
4 Joe Celko
5 Joseph Moody
7 Allen
9 Edward Charbeneau
10 Tony Davis
12 Louis Davidson
15 Andrew Clarke
17 Joshua Feierman
18 Blogs
19 Allen White
20 Michael Sorens
23 Robyn Page
27 Cloud
28 Fritchey
30 Kathi
31 Sysadmin
33 Forums
35 Books

Across
2 Fabiano Amorim
6 Dino Esposito
8 Laerte Junior
11 SQL
13 Jaap
14 Kalen Delaney
16 Buck Woody
21 Nick Harrison
22 Phil Factor
24 Newsletter
25 NET
26 Robert Sheldon
29 Mike Wood
32 Robert Young
34 Grant
36 Richard Morris
37 Massey
38 Rodney Landrum
39 Dwain Camps