

# Hadoop

## for Oracle Professionals

Arup Nanda 🏠

Oracle Technologist and Architect, Starwood Hotels and Resorts

Big Data is a term that keeps showing up more and more often; not just in technical literature but in casual conversations as well. Have you wondered what exactly it means, especially if you are already familiar with a relational database such as Oracle?

**Or do you get lost in the jargon jungle that is littered with terms like Hadoop, Map/Reduce, Hive and Sqoop? In this article, I will attempt to explain these terms from the perspective of a traditional database practitioner.**

The recent news of the American National Security Agency collecting and dissecting our call and email records is - to say the least - hard to ignore, especially on the aftermath of that disclosure. Political and ethical issues aside, the news does bring up the focus on the aspect of computation involving massive amounts of data. It's one thing to slice and dice a reasonable sized dataset such as sales data; but the sheer volume and variety of data in the case of the phone and email records, especially when correlated with other data, such as financial and travel records to identify pseudo-criminal activities becomes a

formidable challenge. The question is whether the traditional relational databases, such as Oracle and DB2, will scale well to meet that challenge. The answer is, sadly, probably not; at least not within typical monetary boundaries. But the challenge of mining that massive dataset is not limited to the realm of espionage and national security; it's very much a concern for corporations who - among other things - want to track customer behaviour to fine tune their service and product offerings.

Perhaps it's espionage of a different kind. The sources of their data may be different: website logs, Facebook and Twitter feeds instead of phone and email records; but the challenges are the same - to articulate meaningful intelligence from enormous amounts of completely unstructured and unpredictable data. This is the fundamental challenge of the "Big Data".

## Meet the Vs

About a decade ago, Yahoo! - the then internet titan - faced the same issue: how to present the selectivity of the individual web pages on its portal by analysing the pattern of clicks in order to attract advertisers. Google - years later - had a similar challenge of indexing the entire World Wide Web in its servers to present search results to the user very, very quickly. Both of these cases represent the issues others probably didn't face earlier - too much, ever changing and unpredictable data. Here are the relatively unique aspects of this challenge, which are known as the Three Vs of Big Data.

- **Volume** - the sheer mass of the data made it difficult, if not impossible, to sort through them;
- **Velocity** - the data was highly transient. Website logs are relevant only for that time period; for a different period it was different;
- **Variety** - the data was not pre-defined and not quite structured - at least not the way we think of structure when we consider relational databases.

Both these companies (or, in case of Google, before it became a company when it was a research project at Stanford) realised they are not going to address the challenges using the traditional relational databases, at least not in the scale they wanted. Necessity is the mother of invention. This challenge forced them to develop their own tools and technologies. They took a page from the super-computing paradigm of divide and conquer to solve a complex problem. Instead of operating on the dataset as a whole, they divided it into smaller chunks to be processed by hundreds, even thousands of small - not big - servers. This approach solved three basic, crippling problems:

1. There was no need to use large servers, which typically cost exponentially lot more than small servers
2. There was a built-in data redundancy since the data was replicated between these small servers
3. But the most important, it could scale well, very well, simply by adding more of those small servers

This is the fundamental concept that gave birth to Hadoop. But before we cover that, we need to learn about another important concept.

## Key-Value Pairs

A typical relational database works by logically arranging the data into rows and columns. Here is an example. We decide on a table design to hold your guests, named simply GUESTS. It has the columns GUEST\_ID, NAME, ADDRESS, PHONE. Later, the company decides to provide some incentives to the spouses as well and so we added another column - SPOUSE.

Everything was going well and then we discovered that customer 1 and spouse are no longer married and there is a new spouse. However, the company decides to keep the names of the ex-spouses as well for marketing analytics. Applying the right relational design, we decide to break the spouse away from the main table and create a new table named SPOUSES, which is a

child table of GUESTS, joined by GUEST\_ID. This change required massive code and database changes; but the company somehow survives. A few days later they had the same issue with addresses (people have different addresses - work, home, vacation) and phone numbers (cell phone, home phone, work phone, assistant's phone). So they decide to break them into different tables. Again, code and database changes invariably follow - more pain for the gains. But the changes did not stop there. They had to add various tables to record hobbies, associates, weights, dates of birth - the list is endless. Everything we record requires a database change and a code change. The bigger issue was that the code change could occur only after the database was ready with the new tables or the new columns. If the data came in when there was no place for it yet, the data was discarded. This led to data and application development challenges too hard to ignore. Further, in their quest to build a 360 degree view of the guest, they collected all the possible information; but there was no guarantee that all the data points would be gathered. They were left with sparse tables but with no guarantee that what they had would satisfy all the needed data points.

If you look at the scenario above, you will find that the fundamental issue is trying force a structure around a dataset that inherently unstructured - a square peg in a round hole. The lack of structure of the data is what makes it useful; but it's also the lack of structure that makes it difficult in a relational database which demands structure. This is the primary issue in capturing social media data - Twitter feeds, Facebook status updates, LinkedIn updates and Pinterest posts. It's impossible to predict in advance, at least accurately, the exact information you will expect to see in them. So, putting a structure around the data storage not only makes life difficult for everyone - the DBAs will constantly need to alter the structures and the developers/ designers will constantly wait for the structure to be in the form they want - slowing down capture and analysis of data and consequently the usefulness of the data.

So, what is the solution? If you think about it, think about how we - human beings - process information. Do we parse information in form of rows in some table? Probably not. We process and store information by associations. For instance, let's say I have a friend named Rex. I probably have nuggets of information like this:

Last Name = Smith  
Lives at = 123 Main St, Anytown, U.K.  
Age = 40  
Birth Day = June 7th  
Wife = Regina  
Child = Pamela  
Pamela goes to school = Top Notch Academy  
Pamela is in Grade = 3

... and so on. Suppose I meet another person - Layla - who tells me that her child also goes to Grade 3 in Top Notch Academy. My brain probably goes through a sequence like this:

&gt;&gt;



Search for “Top Notch Academy”  
 Found it. It’s Pamela  
 Search for Pamela.  
 Found it. She is child of Rex  
 Who is Rex’s wife?  
 Found it. It’s Regina.  
 Where do Rex and Regina live? ...

And finally, after this processing is all over, I say to Layla as a part of the conversation, “What a coincidence! Pamela, the daughter of my friends Rex and Regina Smith goes there as well. Do you know them?”, “Yes, Regina is a friend of mine,” replies Layla, “and Pamela is in the same class as my daughter, that of Mrs Ash”.

Immediately my brain processed this new piece of information and filed the data as:

Pamela’s Teacher = Mrs. Ash  
 Regina’s Friend = Layla  
 Layla’s Child Goes to = ...

Days later, I meet with Regina and mention to her that I met Layla whose child went to Mrs. Ash’s class, the same one as Pamela. “Glad you met Layla,” Regina says. “Oh by the way, Pamela is no longer in that class. Now she is in Mr. Anthony’s class.”

Aha! My brain probably stored that information as:

Pamela’s former teacher = Mrs. Ash

And it updated the already stored information:

Pamela’s Teacher = Mr. Anthony

This is called storing by a name=value pair. You see I stored the information as a pair of property and its value. As information goes on, I keep adding more and more pairs. When I need to retrieve information, I just get the proper property and by associations, I get all the data I need. But the storing of data by name=value pairs gives me enormous flexibility in storing all kinds of information without modifying any data structures I may currently have.

This is also how the variety in Big Data is tamed for processing. Since the data coming of Twitter, Facebook, LinkedIn, Pinterest, etc. is impossible to categorise in advance, it will be practically impossible to put it all in the relational format. Therefore, a name=value pair type storage is the logical step in compiling and collating the data. The name is also known as “key”; so the model is also called key-value pair. The value doesn’t have to have a specific data type. In fact it’s probably a binary large object (BLOB); so anything can go in there - booking amount, birth dates, comments, XML documents, pictures, audio and even movies. It provides an immense flexibility in capturing the information that is inherently unstructured.

### NoSQL Database

Now that you know about key-value pairs, the next logical question you may have is, how do we store these? Our thoughts about databases are typically coloured by our long-standing

association with relational databases, making them almost synonymous. Before relational databases were there, even as a concept, big machines called mainframes ruled the earth. The databases inside them were stored in hierarchical format. One such database from IBM was IMS/DB, which was hierarchical. Later, when relational databases were up and coming, another type of database concept, called a network database, was developed to compete against it. An example of that category was IDMS (now owned by Computer Associates), developed for mainframes. The point is, relational databases were not the answer to all the questions then; and it is clear that they are not now either.

This leads to the development of a different type of database technologies based on the key-value model. Relational database systems are queried by SQL language which, I am sure, is familiar to anyone reading this article. SQL is a set-oriented language - it operates on sets of data. In the key-value pair mode, however, that does not work anymore. Therefore these key-value databases are usually known as NoSQL, to separate them from the relational SQL-based counterparts. However, since their original introduction some NoSQL databases actually support SQL, which is why “NoSQL” is not a correct term anymore. So sometimes they are referred to as “Not only SQL” databases. But the point is that their structure is not dependent on a relational model. How the data is stored exactly is usually left to the implementer. Some examples of the NoSQL are MongoDB, Dynamo, Big Table (from Google) etc. Oracle has one called Oracle Enterprise NoSQL database.

I would stress here that almost any type of non-relational database can be classified as NoSQL; not just the name-value pair models. For instance, Object Store—an object database is also NoSQL. But for this paper, I am assuming only key-value pair database as the NoSQL one.

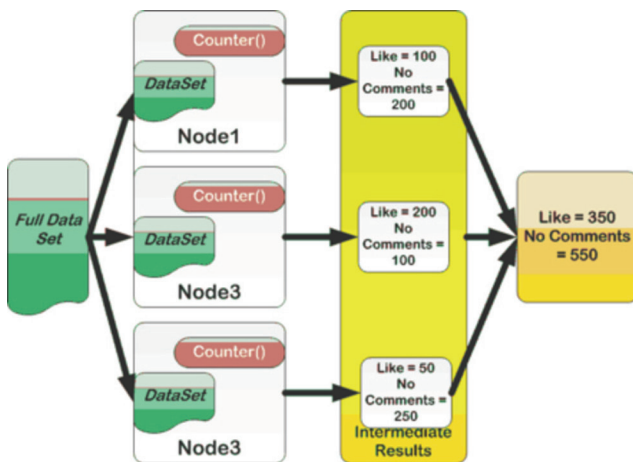
### Map/Reduce

Let’s summarise what we have learned so far:

1. The key-value pair model in databases offer flexibility in data storage without the need for a predefined table structure.
2. The data can be distributed across many machines where they are independently processed and then collated.
3. When the system gets a large chunk of data, e.g. a Facebook feed, the first task is to break it down to smaller chunks which are then fed to several machines simultaneously. This is how the machines perform parallel processing on the set of data. Note that all these machines get their independent chunk of data; they don’t all work on the same dataset. The act of dividing the work and assigning a sub-chunk to a specific machine is called Mapping. Later the output of these mapped results are collated to form summaries—called “Reducing”. These two activities are almost always performed together; hence the combined operation is known as Map/Reduce.

Here is a very rudimentary, but practical, example of Map/Reduce. Suppose you get Facebook feeds on your pages and you are expected to find out the total of likes for our company’s recent post. Facebook feed comes in the form of a massive dataset. The first task is to divide that among many servers, a principle described earlier to make the process scale well - or Mapping. Once the dataset is mapped, each machine runs some code to extract and compile the information and then presents the data to some central co-ordinator to collate for the final

time. Here is a pseudo-code for the process for each server doing the processing on a subset of data:



```

get post
while (there_are_remaining_posts) loop
  extract status of "like" for the specific post
  if status = "like" then
    like_count := like_count + 1
  else
    no_comment := no_comment + 1
  end if
end loop
end
  
```

Let's name this program counter(). Counter runs on all the servers, which are called Nodes. As shown in the figure, there are three nodes. The raw dataset is divided into three sub-datasets which are then fed to each of the three Nodes. A copy of the sub dataset is kept on another server as well. That takes care of redundancy of the data and the nodes. Each node performs its computation and sends its results to an intermediate result set where they are collated. If a node dies in the middle of the process, other nodes can be pulled into processing the sub-dataset from the copies they already have.

### Map/Reduce Processing

How does this help? It does in many ways. Let's see:

- First, since the data is stored in chunks and the copy of a chunk is stored in a different node, there is built-in redundancy. There is no need to protect the data being fed since there is a copy available elsewhere.
- Second, since the data is available elsewhere, if a node fails, all that needs to be done is that some other nodes will pick up the slack. There is no need to reshuffle or restart the job.
- Third, since the nodes all perform tasks independently, when the data size becomes larger, all you have to do is to add a new node. Now the data will be divided four ways instead of three and so will be processing load.

This is very similar to parallel query processes in Oracle Databases, with PQ servers being analogous to nodes.

There are two very important points to note here:

1. The subset of data each node gets is not needed to be viewed by all the nodes. Each node gets its own set of data to be processed. A copy of the subset is maintained in a different

node making simultaneous access to the data unnecessary. This means you can have the data in a locally attached storage; not in expensive SANs. This is not only brings cost significantly down but may perform better as well due to the local access. As cost of Solid State Devices and flash-based storage plummets, it could also mean that the storage cost for performance will become even better.

2. The nodes need not be superfast. A relatively simple commodity class server is enough for the processing as opposed to a large server. Typically servers are priced for their use, e.g. an enterprise class server with 32 CPUs is probably roughly equivalent in performance to eight 4-CPU blades. But the cost of the former is way more than eight times the cost of the blade server. This model takes advantage of the cheaper computers by scaling horizontally; not vertically.

### Hadoop

Now that you know how the processing data in parallel and using a concept called Map/Reduce allows you to shove in several compute intensive applications to dissect large amounts of data, you will often wonder - there are a lot of moving parts to be taken care of just to empower this process. In a monolithic server environment you just have to kick off multiple copies of the program. The operating system does the job of scheduling these programs on the available CPUs, taking them off the CPU (paging) to roll in another process, prevent processes from corrupting each other's memory, etc. Now that these processes are occurring on multiple computers, there has to be all these manual processes to make sure they work. For instance, in this model you have to ensure that the jobs are split between the nodes reasonably equally, the dataset is split equitably, the queue for feeding data and getting data back from the Map/Reduce jobs are properly maintained, the jobs fail over in case of node failure, and so on. In short, you need sort of an operating system of operating systems to manage all these nodes as a monolithic processor.

What would you do if these operating procedures were already defined for you? Well, that would make things really easy, wouldn't it? You can then focus on what you are good at - developing the procedures to slice and dice the data and derive intelligence from it. Well, the good news is that this "procedure" or the framework is already developed and available. It is known as Hadoop. It's an open source offering, similar to Mozilla and Linux; no single company has exclusive ownership of it. However, many companies have adopted it and evolved it into their offerings, similar to Linux distributions, such as Red Hat, SuSe and Oracle Enterprise Linux. Some of those companies are Cloudera, Hortonworks, IBM, etc. The Hadoop framework runs on all the nodes of the cluster. Just to be clear, the cluster is a Hadoop cluster; not an Oracle RAC cluster.

A very important point to note here is that Hadoop is just a framework; not the actual program that performs Map/Reduce. Compare that to the operating system analogy; an OS like Windows does not offer a spreadsheet. You will need to either develop or buy an off the shelf product such as Excel to have that functionality. Similarly, Hadoop offers a platform to run the Map/Reduce programs that you develop and you put that logic in the code what you "map" and how you "reduce".

Remember another important advantage you saw in this model earlier - the ability to replicate data between multiple nodes so that the failure of a single node does not cause the processing to >>

be abandoned. This is offered through a new type of file system called Hadoop Distributed File System (HDFS). HDFS, which is a distributed (and not a clustered) file system, by default has three copies of data on three different nodes - two on the same rack and the third on a different rack. The nodes communicate to each other using a HDFS-specific protocol that is built on TCP/IP. The nodes are aware of the data present on the other nodes, which is precisely what allows Hadoop job scheduler to divide the work among the nodes. Oh, by the way, HDFS is not absolutely required for Hadoop; but as you can see, HDFS is the only way for Hadoop to know which node has what data for smart job scheduling. Without it, the division of labour will not be as efficient.

### Pig

Let me once again reiterate one important property of Hadoop. It is a framework for dividing the work among nodes and making sure the assigned work gets executed. The actual work itself is not the responsibility of Hadoop. In the earlier example I used counter() as a program to count the likes in Facebook page. You must write counter(); Hadoop does not do that for you. So, the next set of questions is, how do you write this program? What languages can you use? etc.

Hadoop typically expects the programs to be in Java. This could pose as an issue for many. End users are not necessarily proficient in a language like Java. Additionally, Java is a third generation language that takes a lot of lines to perform complex calculation, making it inherently inefficient. Therefore, Hadoop provides another approach: a framework for dividing the work very easily. This is called Pig. The interaction with Pig is through a much less complex and user-friendly language called Pig Latin. Pig Latin, like SQL, is a fourth generation language that is designed to accomplish a lot of tasks with few lines of code. Let me show an example of Pig Latin. Suppose you want to find out the average page ranks of the categories of URLs where the click rate is at least 1 million and the minimum page rank is 0.2. You would write the following SQL if the data were in a relational database:

```
select category, avg(pagerank)
from urls
where pagerank > 0.2
group by category
having count(*) > 1000000
```

The same query in Pig Latin will look like:

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)>1000000;
output = FOREACH big_groups GENERATE category, AVG(good_urls.
pagerank);
```

Notice how similar they are. But the important point to note here is that Pig Latin is a typical programming language that has a sequence of steps to accomplish the objective, much like SQL. Likewise, similar to SQL, Pig Latin hides the exact details of the constructs such as FILTER, GROUP, etc. making the programmer more efficient.

### HBase

At the end of the day, you want to store data in some form and you are likely comfortable with the concept of a “database”.

HDFS is not a database; it's just a storage system, similar to a file system on a server. If you want to store data in a database, you can't just use the file system, you have to build a database which could be as simple as a text file with columns and rows or as sophisticated as a full blown database management system such as Oracle.

But wait, you may be wondering here, what is the business about “columns and rows”? Didn't we just establish that rows and columns are not good for the use case for which we designed a key-value pair system just to avoid that structure? If so, why are we again introducing the old column-row model?

The fact is that even if we store as key-value, we are comfortable in looking at most data using row-column model. It's just the way we are designed and perceive information, at least large chunks of similar information. Can't we have the best of both worlds - store them in a flexible key-value model; but build an abstraction layer of the familiar row-column model? The answer is yes. HBase is the tool that does it. HBase is database management layer that presents the data on HDFS to the user in a row-column format. It is built on the Bigtable concept introduced at Google. Designed to store massive amounts of data, HBase is optimised for data access in volumes, not row-by-row as in a traditional database. Transactions are available in HBase, but they work on one row at a time.

HBase is not the only database management system for Hadoop, though. There are others such as CouchDB, Cassandra, Hypertable and more.

### Hive

Now that you learned how Hadoop fills a major void for computations on a massive dataset, you can't help but see the significance for data warehouses where massive datasets are common. Also common are jobs that churn through this data. There is a little challenge, however. Remember the NoSQL databases mentioned earlier? That means they do not support SQL. To get the data you have to write a program using the APIs the vendor supplies making it inefficient and time consuming. Issues like this gave rise to fourth generation languages like SQL, which brought the power of quick and effective queries to users. In data warehouses, it was especially true since the power users issued queries after getting the result from the previous queries. It was like a conversation - ask a question, get the answer, formulate your next question - and so on. If the conversation were dependent on writing third generation language programs, it would have been impossible to be effective.

With that in mind, consider the implications of the lack of SQL in these databases highly suitable for data warehouses. This requirement to write a program to get the data every time would have rendered it ineffective. Well, not to worry, Hadoop has another product that allows an SQL-like language called HiveQL. Just as users could query relational databases with SQL very quickly, HiveQL allows users to get the data for analytical processing directly. It was initially developed at Facebook. Here is an example of a HiveQL query:

```
select count(*)
from store_sales ss
join household_demographics hd on (ss.ss_demo_sk = hd.hd_demo_sk)
join time_dim t on (ss.ss_sold_time_sk = t.t_time_sk)
join store s on (s.s_store_sk = ss.ss_store_sk)
```



```
where
  t.t_hour = 8
  t.t_minute >= 30
  hd.hd_dep_count = 2
order by cnt;
```

### Comparing to Oracle

When we talk about clustering in Hadoop, you may not help wonder - shouldn't the same functionality be provided by Oracle Real Application Cluster? Well, the short answer is, a big resounding NO. Oracle RAC combines the power of multiple nodes in the cluster which communicate with one another and transfer data (cache fusion), but that's where the similarity ends. The biggest difference is the way the datasets are accessed. In RAC, the datasets are common, i.e. they must be visible to all the nodes of the cluster. In Hadoop, the datasets are specific to the individual nodes, which allow them to be local. The file systems in RAC can't be local; they have to be clustered or available globally, either by a clustered file system, shared volumes, clustered volume managers (such as ASM) or by NFS mounting. In Hadoop the local files are replicated to other nodes, which means there is no reason to create a RAID level protection at the storage level. ASM does provide a software level mirroring, which may sound similar to Hadoop's replication, but remember, ASM's mirrors are not node specific. The mirrored copies of ASM must be visible to all the nodes. There is a preferred node concept in ASM, but that simply means that data is read by a specific node from one mirror copy. The mirror copies can't be local, they must be globally visible.

Besides, Oracle RAC is for a relational database. The Hadoop cluster, especially the HBase database is not one. There are traits such as transactional integrity, multiple concurrent writes that are innate features of any modern database system such as Oracle. The design of HBase does not necessarily lead itself towards those goals. The HBase database will eventually be consistent, but is not guaranteed to be consistent at a specific point in time or within the scope of a transaction—something you take for granted in an Oracle database. Therefore, HBase is not a great candidate for systems that require that consistency, such as stock trading systems, telephone call managers, point of sale systems, etc. So while both are technically called databases, a comparison may not be fair to either HBase (or Hadoop) or RAC. They are like apples and tomatoes. (Tomato is technically a fruit, just in case you are wondering about this analogy).

### Other Terms

Now that you understand the fundamental building blocks of the big data universe, let's jump to understand other important components as well. Due to space limitations I am not going to talk much on these terms individually but just enough to steer you in the direction of getting to know more.

**Sqoop:** Short for "SQL to Hadoop", it's a tool that is used to "ingest", i.e. pull data into a Hadoop system (or, more specifically an HDFS store) from any database that supports JDBC, including common ones like Oracle, Sybase, DB2, SQL Server, etc. Remember, the actual process of ingestion is always relegated to a Map/Reduce job which is spread over the nodes of the cluster. Sqoop makes it easier by producing snippets of code (called "writables", to differentiate itself from "executables") for the Map/Reduce jobs. Over the years, Sqoop has flourished and today we

see extremely high performance connectors for databases.

**Flume** Another ingestion tool.

**Zookeeper** A coordination engine that coordinates the components in the Hadoop ecosystem.

**Hue** A web interface to run Hive queries to get data from HBase. Think of this as SQL\*Plus in Oracle.

### The Players

So, who are the players for this new branch of data processing? The major players are shown below with small description. This list is by no means exhaustive. It simply is a collection of companies and products I have studied.

- **Cloudera**

They have their distribution called, what else, Cloudera Distribution for Hadoop (CDH). But perhaps the most impressive from them is Impala - a real-time SQL like interface to query data from the Hadoop cluster.

- **Hortonworks**

Many of the folks who founded this company came from Google and Yahoo! where they built or added to the building blocks of Hadoop.

- **IBM**

They have a suite called Big Insights which has their distribution of Hadoop. This is one of the very few companies who offer both the hardware and software. The most impressive feature from IBM is a product called Streams that can mine data from a non-structured stream like Facebook in real-time and send alerts and data feeds to other systems.

- **Dell**

Like IBM they also have a hardware/software combination running a licensed version of Cloudera, along with Pentaho Business Analytics.

- **Pivotal**

This is a Platform-as-a-Service offering from a joint venture between EMC and VMware, with investment from GE. This has the open source Hadoop. The most notable in this offering is the language called HAQ, which is an ANSI SQL compliant language.

- **MapR**

They also use open source Hadoop, but they use HDFS on the top of their own file system which is based on NFS. This makes the file system available even outside of HDFS, which may be useful in cases like massive data loads.

>>

## Summary

If the buzzing of the buzzwords surrounding any new technology annoys you and all you get is a ton of websites on the topic but not a small consolidated compilation of concepts, you are just like me. I was frustrated by the lack of information in a digestible form on these buzzwords that are too important to ignore but would take too much time to understand fully. Here is my small effort to bridge that gap and get you going on your quest for more information. If you have 100s or 1000s of questions after reading this, I would congratulate myself - that is precisely what my objective was. For instance, how HiveQL differs from SQL or how Map/Reduce jobs are written - these are questions that should be flying in your mind right now and will be the focus for the research later. ■



## ABOUT THE AUTHOR

### Arup Nanda 🌟

Oracle Technologist and Architect, Starwood Hotels and Resorts

Arup Nanda (arup@proligence.com) has been working as an Oracle DBA for last 20 years touching all aspects of the database from modeling to performance tuning to security and disaster recovery. He has presented 300 sessions, written 500 articles, co-authored 5 books and delivered trainings in 22 countries. He is an Oracle ACE Director, a member of Oak Table Network, a member of Board of Directors of Exadata SIG and an editor for SELECT journal. He was the recipient of two prestigious awards from Oracle: DBA of the Year in 2003 and Architect of the Year in 2012.

## PARTNER CASE STUDY

# BMS Partnership with UKOUG Delivers Direct ROI

Established in 1994, Beoley Mill Software (BMS) is a leading installation and support organisation for JD Edwards products, providing a complete range of services to help customers reach their strategic objectives.

**Ten years ago, BMS was a new organisation offering an exciting USP to the JD Edwards community. Generating brand awareness, exposure to the market and explaining their offering to potential clients was imperative to the business' success and the management team considered various ways that this could be achieved.**

In order to meet their challenging objectives, BMS experimented by taking a small stand and some sponsorship at UKOUG's JD Edwards conference in 1994. The results were fantastic. Exhibiting gave BMS a platform to talk face to face with genuinely engaged potential customers generating many leads to follow up on.

The work received from their first event turned into £240,000 of revenue, from only £3,000 expenditure.

BMS have returned to UKOUG's JD Edwards Conference every year since and now contribute to the conference sessions, as well as taking the two largest stands in the exhibition hall, filling them with a range of personnel to maximise the interaction with present and prospective customers. This has helped BMS to become the UK market leader in their field with a turnover of £12 million.

Commenting on this success, Stuart Rimmer, Chief Executive Officer of BMS commented "We would have eventually

reached our goal, but without partnering with UKOUG it would have taken far longer."

Stuart continues, "Looking ahead, partnering with UKOUG is an essential element of BMS's marketing mix, ensuring we retain our position in the market by continuing to engage with new customers and building on relationship with existing ones."

BMS employees volunteer with the UKOUG community which offers another method to meet the needs to the JD Edwards community, as well as providing valuable insight into customers' needs, enabling effective planning for the future.

UKOUG offers a number of bespoke marketing opportunities for Partner members. To find out how we can support the promotion of your organisation contact Kerry Stuart at [opportunities@ukoug.org](mailto:opportunities@ukoug.org)