Upgrade to the new generation of database software for the era of big data!

# DB2 10.5
## with BLU Acceleration

## New Dynamic In-Memory Analytics for the Era of Big Data

- Speed-of-thought analytics with new BLU Acceleration
- Always-available transactions with enhanced pureScale reliability
- Unprecedented affordability with optimization for SAP workloads
- Future-proof versatility and agility with NoSQL style services, such as a Mongo-styled JSON document store and RDF Jena graph store for great application flexibility

PAUL ZIKOPOULOS SAM LIGHTSTONE MATTHEW HURAS
AAMER SACHEDINA GEORGE BAKLARZ

# DB2 10.5 with BLU Acceleration

# DB2 10.5 with BLU Acceleration

**Paul Zikopoulos**
**Sam Lightstone**
**Matt Huras**
**Aamer Sachedina**
**George Baklarz**


New York Chicago San Francisco
Athens London Madrid Mexico City
Milan New Delhi Singapore Sydney Toronto

**DB2 10.5 with BLU Acceleration: New Dynamic In-Memory Analytics for the Era of Big Data**



1 2 3 4 5 6 7 8 9 0 DOC DOC 1 0 9 8 7 6 5 4 3

ISBN 978-0-07-182349-4
MHID 0-07-182349-2

**Sponsoring Editor**
Paul Carlstroem

**Editorial Supervisor**
Patty Mon

**Project Manager**
Hardik Popli, Cenveo® Publisher Services

**Acquisitions Coordinator**
Amanda Russell

**Technical Editor**
Roman B. Melnyk

**Copy Editor**
Lisa McCoy

**Proofreader**
Paul Tyler

**Production Supervisor**
George Anderson

**Composition**
Cenveo Publisher Services

**Illustration**
Cenveo Publisher Services

**Art Director, Cover**
Jeff Weeks

*This year I became a vice president at IBM, and when I look back at the journey I took to get here, it's obvious that I wasn't alone. I've had great mentors (Martin, Alyse, Bob, Frank, and a host of other names that won't fit here). I've had clients who partnered with me for joint success and who inspired me to learn and do more for them. And I've had the chance to work with some incredible practitioners who found a way to challenge me to "not let good enough be good enough." Perhaps the one relationship I've taken for granted during the decade since I left the group is my relationship with the development team. These folks are not only the best innovators in the business, but they've supported my quest for knowledge and my drive to communicate details about the great work that they've been doing for years, and I want to thank them for that. I* really *want to thank a recent IBM retiree and IBM Fellow, Curt Cotner. Although I didn't get as much time with Curt as I would have loved to, the time that I did get made me so much smarter. I remember taping calls with Curt (sounds creepy, doesn't it?) and putting them on my iPad to engrave in my brain the things he'd talk about—An incredible talent. I also want to thank my US HR partner, Debbie Burack, who works tirelessly for my division.*

*To Kelly: I didn't come this far alone, and couldn't have done it without a solid partner by my side. Finally, to Chloë: I often feel guilty about our time together because there's always a phone call coming in or someone is messaging me, but I do cherish the time that we spend together. I cherish your presence and find it calming and uplifting.*

—Paul Zikopoulos

*In loving memory of my parents, Rose and Reuben Lightstone.*

—Sam Lightstone

*This book is dedicated to the worldwide DB2 development team. Time and time again, release after release, no matter where they work around the globe, this team has proven that it's among the world's elite software development teams. It's easy to write a book when the subject matter is so rich! It's a true privilege working with this team.*

—Matt Huras

*I'd like to thank a very large number of family members, friends, and colleagues, without whose help I'd never have had the opportunities that I have. Notably, I'd like to thank my wife Jennifer, who is the most amazing woman ever: talented artist, mom, farmer, cattle rancher, and awesome wife who has supported me despite my many faults over the years. I also want to thank my two lovely boys, Hayden (nine) and Julian (six), the nicest kids a dad could ever ask for. Finally, thanks to a large number of colleagues—in particular, Matt Huras, who has been a mentor to me since my first day at IBM, and who has given me so many opportunities. I could not be more fortunate than to work with such a high caliber of colleagues who have helped to bring me so far; so thanks also to Christian Garcia-Arellano, Keri Romanufa, Ronen Grosman, and Kevin Rose, to name but a few.*

—Aamer Sachedina

*It isn't often that you get a chance to see some innovative technology that changes the entire nature of a product. The DB2 development team has created some great releases in the past, but BLU Acceleration is remarkable in its ease of use and outstanding performance. My thanks go to the entire development team for their hard work in creating DB2 10.5.*
*I'd also like to thank Katrina for her continued support and patience with me. The kids don't ask for cash advances any more (well, almost), so they don't care if I write another book. Tristan (our cat) unfortunately passed away this year, and Ellie (our dog) and Basil (our dog-in-law) have now given up on me and no longer sit around at night while I write these books. Perhaps I should take the hint!*

—George Baklarz

## About the Authors

**Paul C. Zikopoulos, B.A., M.B.A.,** is the vice president of Technical Professionals for IBM Software Group's Information Management division and additionally leads the World Wide Competitive Database and Big Data Technical Sales Acceleration teams. Paul is an award-winning writer and speaker with more than 20 years of experience in information management. Paul is seen as a global expert in Big Data and database. Independent groups often recognize Paul as a thought leader in Big Data, with nominations to SAP's "Top 50 Big Data Twitter Influencers," Big Data Republic's "Most Influential," and Onalytica's "Top 100" lists. Technopedia listed him as "A Big Data Expert to Follow" and he was consulted on the topic of Big Data by the popular TV show "60 Minutes." Paul has written more than 350 magazine articles and 18 books, some of which include *Hadoop for Dummies; Harness the Power of Big Data: The IBM Big Data Platform; Understanding Big Data; Warp Speed, Time Travel, Big Data, and More: DB2 10 for Linux, UNIX, and Windows New Features; DB2 pureScale: Risk Free Agile Scaling; DB2 Certification for Dummies;* and *DB2 for Dummies.* In his spare time, Paul enjoys all sorts of sporting activities, including running with his dog Chachi, avoiding punches during MMA training, eating Aamer's herd of Scottish Galloway cattle, and trying to figure out the world according to Chloë—his daughter. You can reach him at paulz_ibm@msn.com and follow him on Twitter @BigData_paulz.

**Sam Lightstone, B.Sc. Eng. (Electrical), M. Math (Comp. Sci.),** is an IBM Distinguished Engineer and DB2 business intelligence architect for next-generation data analytics. He helped to originate the BLU Acceleration technology in DB2 10.5. He is also the founder and past chair of the IEEE Data Engineering Workgroup on self-managing database systems. Sam is a member of the IBM Academy of Technology and an IBM Master Inventor with over 45 patents and patents pending. He is the author of several books, including the widely cited text *Database Modeling and Design* and the critically acclaimed career guide *Making It Big in Software.*

**Matt Huras, B.A. Sc. (Comp. Eng.), M. Eng.,** is an IBM Distinguished Engineer and senior architect for DB2 for Linux, UNIX, and Windows. Matt's areas of expertise include data management, index management, locking, concurrency, and other protocols. Matt has been with IBM for 29 years. He

holds a bachelor's degree in applied science from the University of Waterloo and a master's degree in engineering from the University of Toronto. Matt's most recent project was DB2 pureScale, a new feature that is focused on delivering new levels of scalability and availability.

**Aamer Sachedina, B. Eng, P. Eng.,** is an IBM Distinguished Engineer at the IBM Toronto Lab. Over the years, he's contributed to the design of the DB2 kernel, was the architect of DB2's Automatic Storage, and most recently headed the active-active availability effort in DB2 pureScale. Aamer is currently DB2's chief kernel architect. In addition, some of Aamer's other passionate interests include breeding and maintaining a herd of Scottish Galloway cattle at his hobby farm with his lovely wife Jen and their two boys, Hayden and Julian.

**George Baklarz, B. Math, M. Sc., Ph.D. Eng.,** has spent 29 years at IBM working on various aspects of database technology. From 1987 to 1991, he worked on SQL/DS as part of the product planning department, system test team, performance team, and application development group. In 1991, George was part of the team that helped move the OS/2 ES database to Toronto to become part of the DB2 distributed family of products. Since that time, he has worked on vendor enablement, competitive analysis, product marketing, product planning, and technical sales support. George has a doctorate in computer engineering, which took him only 13 years to get; the engineering faculty didn't like the fact that he worked at IBM, believing that part-time students would never finish their degree if they had to work for a living! George is currently the manager of the Worldwide DB2 Technical Sales team.

## About the Technical Editor

**Roman B. Melnyk, B.A., M.A., Ph.D.,** is a senior member of the DB2 Information Development team. Roman co-authored *Hadoop for Dummies; DB2 Version 8: The Official Guide; DB2: The Complete Reference; DB2 Fundamentals Certification for Dummies;* and *DB2 for Dummies.* Roman also edited *Harness the Power of Big Data: The IBM Big Data Platform; Warp Speed, Time Travel, Big Data, and More: DB2 10 for Linux, UNIX, and Windows New Features;* and *Apache Derby – Off to the Races.*

# CONTENTS AT A GLANCE

# CONTENTS

# FOREWORD

I remember the days when folks would ask me why I work on database software—a relatively boring, commodity product area. In today's era of computing, where greater business value is delivered through Big Data and analytics, that question has been well answered.

The term Big Data was first used in discussions about new technologies that enable the analysis of large volumes of unstructured data. Today, organizations increasingly look at Big Data as all data, and all paradigms for extracting value from it. All of the data that they own and all of the public data that they can access are largely untapped resources that hold the key to better service, increased operational efficiency, reduced cost, and reduced business risk.

The explosive growth of data is the result of billions of people using the Internet and mobile devices for commerce, entertainment, and social interactions, as well as the internet of things that constantly share machine-generated data. Along with the task of analyzing all of this data to extract its business value, we are challenged with capturing and managing this data in ways that remain uncomplicated, reliable, and affordable.

So once again, data system innovations have moved front and center in the pursuit of business innovation. And data professionals are emerging from behind the cloud of misperception that they are simply expensive "administrators" of the status quo. They are being increasingly recognized for the critical role that they play in translating technology innovation into business success.

At this pivotal time for our clients, I am thrilled to have the honor of leading the IBM team that delivered breakthrough innovations in both BLU Acceleration and an industry-unique set of capabilities in the latest release of DB2. Enhanced pureScale clustering technology for always-available transactions, new BLU Acceleration for speed-of-thought analytics, and NoSQL-style agility (a new Mongo-style JSON document store to go along with the work already done for graph stores and XML in previous releases)

are just some of the things that make DB2 an ideal choice for this era of Big Data and analytics.

If you are reading this book, you likely play an important role in tapping the growing global resource of data. I commend your efforts to remain at the forefront of delivering value to your organization. This book is an excellent way to accelerate your journey.

I would like to thank Paul, Sam, Matt, George, and Aamer for this book. This is an outstanding group with an unwavering dedication to the success of our clients and all data professionals who use our products. Behind this group is a large ecosystem of IBMers who are equally passionate about delivering the innovation and high quality that help our clients to succeed.

Thank you, and enjoy the book.

Bob Picciano,
General Manager, IBM Information Management

# ACKNOWLEDGMENTS

We want to collectively thank the following people who, in one way or another, contributed more to this book than the amount of space that we use to thank them: Jessica (Escott) Rockwood, Adam Storm, Bill Birely, Skyla Loomis, Darrin Woodard, Michael Kwok, Christina Lee, David Klamuk, Liping Zhang, Marko Milek, Bernard Spang, Kathy McKnight, Steve La, Michelle Arsenault, Jesse Lee, Jeff Goss, Sean McKeough, George Anderson, Christian Garcia-Arellano, Roger Zheng, Michael Springay, Mokhtar Kandil, Paul Tyler, Kelly Schlamb, Michelle Archenault, and Eric Alton.

We also want to thank our McGraw-Hill Education acquisitions and production teams, especially Paul Carlstroem, Patty Mon, Hardik Popli, Lisa McCoy, and the rest of the team.

Finally, there are also a number of IBMers that help us with each of our books—from getting it off the ground to a safe landing—and it's more work than one thinks. So folks like Susan Visser, Steven Miller, Anita Ching, among others, deserve their own paragraph of thanks.

# INTRODUCTION

## About This Book

The DB2 10.5 release delivers one of the most game-changing technologies ever produced by the IBM Information Management development teams, let alone our competitors. We wrote this book with the intention of presenting to you those features that we think will prove to be most noteworthy in the DB2 10.5 release. We hope that this book inspires thought and a thirst for first-hand experience with the features that we describe.

We also hope that you enjoy reading this book as much as we enjoyed writing it!

*Paul, Sam, Matt, Aamer, and George*

## How This Book Is Organized

We organized this book into six chapters that cover what we believe to be the highlights and key features found in the DB2 10.5 release.

Chapter 1 gives you the 101 on DB2 10.5. If you've got only 15 minutes, this is the chapter with the value proposition and an overview of what's so great about DB2 10.5. We even give you a quick, but thorough, recap of the DB2 10.1 release.

In Chapter 2, we detail the features that DB2 10.5 adds to the DB2 pure Scale technology, which has been in the marketplace for almost half a decade. DB2 pureScale gets a boost to deliver even more performance, scalability, continuous availability (even during planned maintenance, such as applying DB2 Fix Packs without taking the cluster down), some nice multi-tenancy features, integrated disaster recovery services, and more.

If each chapter were a painting in a gallery, Chapter 3 is the *Mona Lisa*—it simply can't be skipped. We cover the A–Z of BLU Acceleration, the inflection point technology that redefines in-memory columnar and takes it to a whole new level. We'll frame the need for this technology, the seven big ideas that inspired it, and more.

Chapter 4 is the miscellaneous performance and SQL enhancements chapter; some of these enhancements are "in-your-face" performance improvements, and some are improvements that you never hear about, but that you'll be glad to know are there.

What Chapter 4 does to performance, Chapter 5 does for high-availability and disaster recovery; specifically, this chapter covers the DB2 10.5 enhancements other than those associated with the DB2 pureScale features that we cover in Chapter 2.

Finally, Chapter 6 gives you a tour of what DB2 10.5 has done to extend its support for the NoSQL world: DB2 10.5 introduces a native JSON document store that can even run MongoDB applications without changing a single line of application code! (We assume that changing the connection string in your application is acceptable.) You'll also find highlights of other NoSQL DB2 technology in this chapter (like the triple graph store that was introduced in the DB2 10.1 release), reasons why agility rules the enterprise of today, and more.

Let's venture into the wild BLU yonder!

# 1

# All Aboard! 15-Minute DB2 10.5 Tour Starts Here

Being a CIO or CTO is tough these days. Think about it—these executives face challenges that are so wide and varied, it's dizzying and makes you sometimes wonder how anything gets done. From bring your own device (BYOD), to separation of duty (SoD) and concern (SoC), to high availability (HA) and disaster recovery (DR), to on-premise or off-premise cloud, columnar, and Big Data, there are more than enough buzzwords and real challenges to go around. The results of the most recent IBM CEO study, "Leading Through Connections" (www.ibm.com/services/us/en/c-suite/ceostudy2012/), highlighted that IT technology is the leading force for impacting immediate business, with people skills and market factors following closely behind. So how does a business deliver innovation and new systems when this study concluded that more and more of the relative proportion of IT budgets is eaten up maintaining existing systems? This same study noted that only one in five organizations allocate more than 50 percent of their IT budget to new projects—a sobering thought.

Supporting the conclusions of this IBM study, research from IDC's "Converged Systems: End User Survey Results" (September 2012, Doc# 236966) reported that the proportion of IT budgets that is allocated to server management and administration is a whopping 68 percent. Of more concern is what this proportion used to be in 1996—29 percent! Oddly enough, this change, which resulted in many IT shops being thought of as cost centers rather than critical business innovation centers, has occurred during a time when many

data center operations have been sent offshore to emerging economies such as Brazil, Russia, India, and China (often referred to as *BRIC* countries). This was supposed to reduce costs; however, administrative costs (which are people costs) are ironically going through the roof.

The rate of technological change and the realities of a Big Data world creates even more pressure on these precious IT resources. The mobile-social-cloud evolution completely changes the volume, veracity, velocity, and variety characteristics of the data you work with every day. From BYOD to the fact that 90 percent of mobile users keep their devices within arm's reach 100 percent of the time, it all means more data to manage, the extinction of batch windows, and continuous availability—all trends whose funding falls into the server spending and administration category of the IT shop.

Of course, all of this means that there's less money for innovative projects and new use cases, which brings us back to the results of the IBM study: More and more IT budgets are getting allocated to "keeping the lights on" instead of focusing on new projects and capabilities to drive growth.

With this in mind, it's no wonder that clients are looking for more efficient ways to work with their data. They are struggling to find new agile development methods, reduce management overhead, and fully optimize the use of their existing investments to free up capital for new investments. Although it's fair to note that investing in technologies like multicore servers and solid-state disks (SSDs), among others, can help solutions run faster and more efficiently, solely investing in these technologies is not enough! For example, are you fully exploiting your hardware systems? Have you noticed that multicore systems don't always make the software run faster? We've all had this experience at home: We get the latest quad-core machine and things don't run twice as fast as they did on our old dual-core laptop. After all, Intel, AMD, and Power processing architectures all have built-in massive parallelism capabilities. The real question is: "Was my software written to fully exploit these capabilities?"

These challenges served as the inspiration behind the last decade of DB2 innovations, and the latest DB2 10.5 release is no exception. IBM has made significant investments that won't just flatten the time to value for your solutions, but also will help you to derive the greatest benefit from the investments you've already made. For example, unlike other in-memory columnar offerings, DB2 isn't going to force you to toss out your existing hardware to run it;

in fact, the opposite is true. Your existing hardware will do even more for your solutions. For example, consider what Mohankumar Saraswatipura, the lead DBA at Reckitt Benckiser, had to say about DB2 10.5: "The performance of DB2 10.5 with BLU Acceleration is quite amazing. We ran our tests on a system that is about 16 times less powerful than our current production system. And yet BLU was able to outperform our production system in every aspect." (Reckitt Benckiser makes a large assortment of award-winning nameplate products that you see at the grocery store and likely use every day—Lysol, for example, and who can forget the memorable catchphrase "Calgon, take me away!")

DB2 10.5 incorporates leading technology from our research and development labs and delivers a groundbreaking DB2 release that includes the following enhancements:

- **BLU Acceleration** delivers incredible performance speed-up and compression ratios for analytical workloads, and it's delivered with a "Load and Go" operational paradigm that's more about what you *don't* have to do from an administration perspective than what you have to do. This technology comes from a number of innovations from IBM Research and includes ground-breaking techniques such as dynamic in-processing processing and more.
- **DB2 pureScale** enhancements include new disaster recovery options for even higher availability, removal of planned maintenance outages, miscellaneous performance enhancements, elastic scalability, platform and interconnect flexibility, and more.
- **NoSQL-style JSON Document Store** delivers support for DB2 operating as a document store for JavaScript Object Notation (JSON) documents. This support includes plug-and-play support for MongoDB-created applications. (Remember that DB2 10 introduced NoSQL support for graph databases and has long had a truly native XML document capabilities with its native pureXML store.)
- **Cloud extensions and features** make DB2 an optimal database for on- or off-premise cloud deployments. For example, both the DB2 NoSQL JSON Document Store and BLU Acceleration are cloud optimized.

- **Oracle compatibility** enhancements inflate the compatibility ratio of Oracle PL/SQL features that run natively in DB2 to such an extent that even well known Oracle pundits have taken notice, claiming their PL/SQL book samples can be used on DB2!
- **Miscellaneous enhancements,** such as various nips, tucks, and some neat new stuff makes DB2 more available, scalable, and consumable; these are the kind of features that never make it into the mainstream marketing messages, but you're glad they're in the product.

There is, of course, a lot more to the DB2 10.5 release than what we've listed here, and we cover a lot of it in this book. Don't let the point release naming fool you: DB2 10.5 is an inflection point release that's going to help you analyze your Big Data more quickly, compress your data to a fraction of what it was, keep you online, flatten your project's time to market curve, and help you take advantage of the social-mobile-cloud opportunity.

While this chapter can't give you a set of abs or an Olympic-sculpted build in 15 minutes, it can give you a quick recap of what was delivered in the DB2 10.1 release and a high-level tour of the DB2 10.5 release. After reading this chapter, you'll have a good idea of the new opportunities that are open to you if you build your application to the DB2 platform. Each of the aforementioned themes is detailed in the remainder of this book, where we frame the business challenge, the solution, and the DB2 approach. Sometimes we contrast how things are done in DB2 compared to other marketplace vendors, sometimes we contrast to the way things were done in previous releases of DB2, and, of course, we explain how the new features work.

## What Was Delivered: A Recap of DB2 10.1

The DB2 10.1 release included some special innovations to help you deliver solutions in a more agile, higher-performing, and more available manner than you ever could before. In this section, we highlight some of the more important features that were delivered in the DB2 10.1 release. We wrote a book about this release, which you can download for free (http://tinyurl.com/l2dtlbf) if you want all the details. By the way, if you're currently running DB2 9.7, you might want to consider migrating directly to DB2 10.5, because you'll get all of the great features mentioned in this section, as well

as the ones we cover in the remainder of this book, with only a single migration effort.

## DB2 10.1 Delivered Performance Improvements

DB2 10.1 was a fast release! A lot of work went into making sure that our clients received "out-of-the-box" performance improvements in the neighborhood of 35 percent (our lawyers want us to tell you that any results we mention in this book were observed in controlled environments and you might not get exactly the same results; we told our lawyers that some of our clients are experiencing even better results, so we invite you to try the technologies out for yourself and decide). New performance features debuting in DB2 10.1 included intraquery parallelism, new parallel algorithms, and a brand-new zigzag join operation that we've seen run commonplace queries for which it was designed three times faster than in previous releases! There are also performance enhancements to `RUNSTATS`, statistical views, index optimizations, hash joins, code path optimizations, range partitioning `ATTACH` operations, CPU workload allocation controls, and more.

## DB2 10.1 Delivered Even Lower Storage Costs

If you haven't already implemented DB2 compression, you're using too much disk space for your indexes, writing out larger logs and backups, and not using your server's memory allocations as effectively as you could; whereas implementing DB2 compression can not only help you save on disk space but, in many cases, your applications will also run much faster! DB2 has delivered market-leading innovations around compression for almost a decade, and DB2's compression capabilities represent one of the most popular feature sets in the history of DB2. After all, there's a reason why *Information Week* stated that "[DB2] row-level compression is a revolutionary development that will leave Oracle and Microsoft green with envy."

Clients that implemented DB2 compression before DB2 10.1 used a static table-wide dictionary and realized overall table compression savings of around 60 percent (typically, the largest tables would compress the most, at around 80 percent, but it all depends on the data). The DB2 9-point releases (DB2 9.5 and DB2 9.7) not only made compression autonomic, but also broadened the scope to database-wide compression with the addition of index, temporary-space (to the best of our knowledge, DB2 is still the only database

in the world that can do this), inline-LOB, and XML compression. This had the net effect of even further boosting overall database compression ratios.

In DB2 10.1, adaptive compression was added to the mix. Adaptive compression added page-level dictionaries to the existing repertoire of compression techniques, giving DB2 the added ability to compress data at the page level (dynamic compression). These page-level dictionaries are automatically updated as new data is added. With this synergy, we saw average table compression ratios grow to about 70 to 80 percent. Of course, this technology had a beneficial effect on overall database compression ratios too. DB2 10.1 also added log compression, which even further compresses the overall database size.

How effective has the nonstop investment in compression been for DB2 clients? One of our clients moved from Oracle to DB2 in April 2008. Before moving, they had almost a terabyte of data. After moving to DB2 and capturing a half-decade of additional production data, their database size hasn't changed! We think that this is a great example of how IBM's ongoing investment in storage optimization empowers our clients to do more with less. How will the new BLU Acceleration capabilities that are part of DB2 10.5 help out this producer of one of the world's most iconic brands? Their lead DBA told us, "Just when I thought things couldn't get any better, BLU Acceleration came along."

DB2 10.1 also introduced multitemperature storage management, which gave DBAs a management-by-intent framework whereby business policies could be applied for the most effective use of high-speed storage, such as SSD. Specifically, DB2 10.1 gave administrators the ability to create storage pools with specific devices assigned to them; for example, the fastest disks could be assigned to a `HOT` pool, typical disks to a `WARM` pool, and older disks relegated to unimportant or archive work in a `COLD` pool. New table spaces could simply point to these storage pools and inherit the performance characteristics of their devices.

## DB2 10.1 Delivered Improved Availability and Scalability

In DB2 10.1, the DB2 pureScale code base was merged with DB2, eliminating the need to install this industry-leading technology as a separate product (which is how it was done when it made its debut in DB2 9.8). This integration

gave clients support for range partitioning, table space–level backups, and Workload Manager (WLM) integration, among other notable features. Those clients who were reluctant to deploy InfiniBand in a pureScale configuration because it required extra work acquired the option to use the "secret sauce": DB2 pureScale communication protocols, as of DB2 10.1, support RDMA over Converged 10Gb Ethernet (or RoCE, pronounced "rocky") as an alternative interconnect.

Three significant features were also added for those clients who deploy DB2's turnkey high-availability and disaster recovery (HADR) solution. HADR in DB2 10.1 can support up to three standby servers and enables you to implement a delayed `APPLY` to help prevent errant transactions from being applied on the backup server or avoid human error (the number-one cause of down time). For high-throughput environments, DB2 10.1 HADR includes support for transaction spooling, which can be useful in avoiding backpressure on the production server.

Finally, a new high-speed continuous data ingestion (CDI) utility was delivered as part of DB2 10.1. The multithreaded `INGEST` utility is a high-speed, client-side utility that streams data from files or pipes in various data formats into DB2 tables. If you're familiar with DB2's `IMPORT` and `LOAD` utilities, you can think of the `INGEST` utility as a cross between them that aims to combine the best features of both, as well as add some brand-new capabilities. `INGEST` keeps tables ultra available, because it doesn't lock the target table; rather, it uses row locking to minimize its impact on concurrent activities against the same table. CDI is especially useful in situations where new files containing data are arriving all the time (in Big Data terms, we call this *velocity*). The data might be arriving continuously or trickling in from an extract, transform, and load (ETL) process. After experiencing just how beneficial this feature is, some clients have renamed their processes to extract, transform, and ingest (ETI)!

## DB2 10.1 Delivered More Security for Multitenancy

Row and column access control (RCAC) support was added to DB2 10.1; in combination with the existing label-based access control (LBAC), security administrators get an even more extended arsenal for separation of duty,

separation of concern, and principle of least privilege in their multitenancy database environments. Specifically, RCAC (or its common industry name, fine-grained access control, or FGAC) gives the security administrator (SECADM) new flexibility in defining what rows or columns users can access.

RCAC is based on a user's database role. A DB2 role (introduced in DB2 9.5) entitles a user to retrieve only certain records from the database. For example, a bank teller might only be able to see the mortgage papers that were issued by his branch. Records that are outside of an individual's role are never materialized (returned to the individual or part of interim processing answer sets). In addition, columns that are of a sensitive nature can be masked (think "*****") or set to a NULL value. RCAC is extremely flexible and considerably simpler to implement than LBAC, which is more suited to organizations that have rigid hierarchy-based security roles, such as those you might see in a nation's defense department.

## DB2 10.1 Delivered the Dawn of DB2 NoSQL Support

DB2 10.1 introduced support for one of the four NoSQL database genres (Key Value, Columnar, Graph, and Document) with its support for a graph store that's implemented using the World Wide Web Consortium's (W3C) Resource Description Framework (RDF) standard and fronted by the popular JENA application programming interface (API). The RDF W3C standard describes relationships and resources in the form of subject-predicate-object. In the NoSQL world, these types of data entities and relationships are called triples, and they are stored in specialized triplestore databases. DB2 10.1 can function as a true triplestore database, providing the ability to store and query RDF data.

In addition to supporting an RDF graph store, DB2 supports a W3C RDF query language called SPARQL, which is designed to retrieve data from an RDF graph store database and was devised by the W3C, the same people who developed many of the other standards on which the Web is based. SPARQL represents a strong foray by IBM into the NoSQL world. You'll find out later in this book how DB2 implements a JSON document store that is all the rage in today's mobile and cloud-dominated environment. BLU Acceleration includes columnar storage as one of its inspirations, and IBM

InfoSphere BigInsights includes a nonforked Hadoop distribution and HBase, among other NoSQL technologies, such as Flume, Oozi, Hive, and more.

### DB2 10.1 Delivered Even More Oracle Compatibility

Every DB2 release includes more Oracle compatibility features. We want to help you move to a better database! Included in DB2 10.1 were more flexible triggers that fire only once per SQL statement; locally defined data types, procedures, and functions in compound SQL statements; and a number of new scalar functions. With these enhancements, we've seen the ability for DB2 to *natively* process Oracle PL/SQL logic within single digits of 100 percent for the typical applications we see.

### DB2 10.1 Delivered Temporal Data Management

Temporal data management is all about managing and processing data in relation to time. DB2's support for system and business time (which we think is unique) enables you to automatically track and manage multiple versions of your data. When you define a system-period temporal table, you're instructing DB2 to automatically capture changes to the state of your table and to save "old" rows in a history table, a separate table with the same structure as your base table. Whenever a row is updated or deleted, the "before image" of the affected row is inserted into this history table.

Through simple declarative SQL statements, you can instruct DB2 10.1 to maintain a history of database changes or track effective business dates automatically, eliminating the need for such logic to be hand-coded into triggers, stored procedures, or application code.

## Introducing DB2 10.5: The Short Tour

No question about it, DB2 10.1 was an exciting release and has lots of great technology. If DB2 10.1 excites you, DB2 10.5 has the potential to blow you away, because beyond its enhancements of existing features, it introduces an inflection point technology known as BLU Acceleration.

In this section, we give you the short tour of the DB2 10.5 release. The purpose of this chapter is to introduce you to the key themes and technologies that debut in this release and that are described in the remainder of this book.

## DB2 with BLU Acceleration

BLU Acceleration is one of the most significant pieces of technology that's ever been delivered in DB2 and, we'd argue, in the database market in general. DB2 with BLU Acceleration delivers significant improvements in database compression and unparalleled performance improvements for analytic applications running on DB2. This technology is so important that Chapter 3 is dedicated to teaching you all about BLU, how it works, and what makes it so special.

There's a lot of talk about in-memory columnar data processing in the industry today, but BLU Acceleration is so much more. DB2 actually sees memory as the new disk when it comes to performance optimization, in that you only go to that tier if you have to; it also doesn't need to have all the data in memory. More about that later.

BLU Acceleration *is not* some new bolted-on storage engine that sits on top of DB2, or that requires you to upgrade your servers from among a small set of architectures. BLU Acceleration is part of the DB2 database engine's DNA, and every facet of DB2 is aware of it. It's not an afterthought, but a key component of the database. It uses the same pages, the same buffer pools, the same backup and recovery utilities, and more.

DB2 with BLU Acceleration includes a number of significant features that work together to make it an inflection point. We sometimes refer to this engineering as the "BLU Acceleration Seven Big Ideas."

### The First Big Idea: BLU Acceleration Is Simple (*"Just Load and Go"*)

BLU Acceleration is simple to implement and use. As we've said before, BLU Acceleration, from a management perspective, is more about what you no longer have to do than what you actually need to do. There's no need to create indexes, reorganize tables, select a partitioning key, define multidimensional clustering tables, build materialized views, and so on. All you have to do is *load* the data into a table and *go* start running your queries.

We see this technology as being well suited to a multitude of scenarios and purposes, from instantly speeding up your SAP Business Warehouse (BW) environments (without the risk of application migration or tricky hardware swaps), to accelerating your Cognos environments (we call it "*Fast on Fast*"), to

spinning off data marts and cubes for lightning-fast performance for agile campaign support, and more. Because DBAs don't have to spend time on performance tuning, BLU Acceleration frees up their time to work on new value projects instead of trying to get value out of existing projects. BLU effectively enables DBAs to support more analytic applications and bring to the business more opportunities to make a difference. We've seen clients experience up to quadruple-digit performance speed-ups on some of their queries, with double-digit average improvements in query response time overall! (Note that we said *average improvements in query response time*—this is very different from typical marketing claims by other vendors who typically quote "highlight reel" single queries as opposed to average query speedup.)

Handelsbanken (described by Bloomberg as one of the most successful and secure banks in the world) stated that they "were very impressed with the performance and simplicity of BLU. We found that some queries achieved an almost 100 times [faster] speed-up with literally no tuning!" That sums up this big idea: *load and go… go faster, that is*.

## The Second Big Idea: Extreme Compression and Computer-Friendly Encoding

If you thought that DB2 compression was great in DB2 10.1, then you're in for a surprise when you try DB2 with BLU Acceleration. In our observations, clients are seeing ten-fold compression rates when they move tables into a BLU Acceleration format. All of this compression is done automatically for you; you don't have to choose specific encoding mechanisms, or mix a myriad of high or low query and archive compression schemes to try to get it right. BLU Acceleration is about helping performance *and* compression.

BLU Acceleration uses a form of Huffman encoding to compress data, along with the more traditional adaptive dictionaries. By placing only values from an individual column on a page, DB2 is able to find more patterns and improve compression rates. As an added bonus, indexes, materialized query tables (MQTs), and other optimization objects aren't required, so you eliminate the space and overhead of creating these objects—after all, there's no point in compressing objects that you don't need! Some of our clients have seen savings up to 25-fold when they consider the compression rates alongside the objects that they were able to drop from the database schema.

After a value has been compressed, DB2 combines it with other compressed values to create a record that fits into a CPU register. By organizing data in this way, DB2 is able to load blocks of data into a processor core without any additional formatting or overhead. This attention to detail is what puts the acceleration into BLU Acceleration. Finally, in most cases, DB2 with BLU Acceleration doesn't need to uncompress the data when running a query.

Triton's head of DB2 Managed Services Team, Iqbal Goralwalla, told us that "when adaptive compression was introduced in DB2 10.1, having achieved storage savings of up to 70 percent, I was convinced this is as good as it gets. However, with DB2 10.5 and BLU Acceleration, I have been proven wrong! Converting my row-organized, uncompressed table to a column-organized table gave me a massive 93.5 percent storage saving!"

## The Third Big Idea: Deep Hardware Exploitation

The third big idea revolves around exploiting existing hardware. Today's modern processor architectures all have multiple registers on them and the ability to do multiple calculations with a single instruction. This technology is called *Single Instruction, Multiple Data* (SIMD). Software that's built to exploit SIMD (like DB2) can execute one instruction against all of the registers on the chip; in other words, SIMD multiplies the power of the CPU.

This built-in parallelism was never before exploited by DB2. Most vendors still don't exploit it today, and those that do, from what we can tell, don't do it to the extent that DB2 with BLU Acceleration does it. The DB2 value proposition here is that DB2 can further exploit the existing parallelism of your server instead of forcing you to buy new hardware to achieve similar results. You can take advantage of your existing hardware and get great performance from DB2.

To the best of our knowledge, DB2 with BLU Acceleration is the only technology of its kind that runs on IBM Power servers. Power servers are exceptionally good at SIMD exploitation because they typically have larger and wider registers in comparison to alternative architectures.

During the development of BLU Acceleration, we worked very closely with Intel to fully exploit their Advanced Vector Extensions (AVX) instruction set, which is available on Intel Xeon processor E5-based systems. Pauline Nist,

general manager of Intel's Enterprise Software Alliances, Datacenter & Connected Systems Group, summarized our joint engineering work by referencing an analytic test bed that Intel first ran on DB2 10.1 and then on DB2 10.5 with BLU Acceleration: "Intel is excited to see a 63x–133x (depending on the Intel processor) improvement in query processing performance using DB2 10.5 with BLU Acceleration over DB2 10.1."

## The Fourth Big Idea: Core-Friendly Parallelism

Modern chip architectures use a variety of cache levels to hold instructions and data. When looking for data, multilevel caches generally operate by checking the smallest Level 1 (L1) cache first, then the next larger cache (L2), followed by the shared cache (L3), before external memory (DRAM) is checked. The more data and instructions that you can keep in the CPU caches, the faster your workloads will complete.

BLU Acceleration, in combination with automated DB2 workload management, was designed to keep data in CPU caches as long as possible; in fact, its algorithms view spilling to DRAM in the same way as spilling to disk from a performance perspective. As a result, SQL requests that are running on a series of cores are prioritized so that one unit of work completes before another request is dispatched. This enables DB2 to achieve higher throughputs when workloads can be isolated to certain cores or sockets. By isolating workloads, DB2 can achieve higher cache coherency and eliminate low-level thrashing of the cache. The Reckitt Benckiser scenario referenced earlier in this chapter is a good example; even though they ran their tests in an environment that was *16 times less powerful* than their production system, the test system outperformed the production system.

Other database products aren't designed to exploit all of the cores on a server. A very common open-source relational database product is known for its scaling issues above a certain core threshold on a single server. This is exactly the pain point that BLU Acceleration addresses, because it was designed for today's multicore systems. Kent Collins, a Database Solutions Architect at BNSF Railway, sums it up this way: "During our testing, we couldn't help but notice that DB2 10.5 with BLU Acceleration is excellent at utilizing our hardware resources. The core-friendly parallelism that IBM talks about was clearly evident, and I didn't even have to partition the data across multiple servers."

## The Fifth Big Idea: Column Store

There are many benefits that a column-organized approach brings to analytic workloads. Chief among them is the fact that column-organized tables typically compress much more effectively. The probability of finding repeating patterns on a page is very high when the data on the page is from the same column. Row-organized tables, on the other hand, store data from columns in the same row, and the data of those columns can vary widely, thereby reducing the probability of finding repeating patterns on a page.

In DB2, column-organized tables can coexist with traditional row-organized tables; you're not required to commit to one or the other approach for your entire database. And, because BLU Acceleration is built right into the DB2 engine, the SQL, optimizer, utilities, and other functions are fully aware of it.

## The Sixth Big Idea: Scan-Friendly Memory Caching

Clients typically have a memory-to-disk ratio of 15 to 50 percent. What this means is that they can't possibly fit their entire table, or even the data that is required to execute a complex query, entirely into memory. What does this mean for customers using DB2 with BLU Acceleration? Although DB2 would make good use of it, there's no need to have excessive amounts of memory. BLU Acceleration was designed for the demands of a Big Data world where it is less and less likely that all of the data that queries need will fit into memory.

BLU Acceleration comes with a set of Big Data–aware algorithms for cleaning out memory pools that is more advanced than the typical "Least Recently Used" (LRU) algorithms that are associated with traditional technologies. These BLU Acceleration algorithms are designed from the ground up to detect "interesting" patterns and to hold those pages in the buffer pool as long as possible. The algorithms work side by side with DB2's traditional row-based algorithms. The net result is that you can get significant performance boosts from data already in memory, even though the memory might not be as big as your tables.

## The Seventh Big Idea: Data Skipping

DB2 keeps track of the minimum and maximum values that are found on a column-organized table's data pages. This information is updated dynamically (you don't have to manage anything—all automatic) and is used by the query optimizer to skip those data pages that don't contain values that are

needed to satisfy the query. If you're familiar with the Zone Maps used in the IBM PureData for Analytics offering (formerly known as Netezza), you can see where the inspiration for this feature came from; its net effect is a dramatic speed-up of query execution because a lot of unnecessary scanning is avoided.

### A Final Thought Before You Delve into the Wild BLU Yonder

As you can see, BLU Acceleration is a remarkable innovation that's part of the DB2 10.5 release. It gives you outstanding performance and extreme data compression without having to do complex tuning, or any tuning at all, for that matter. On average, the clients that we worked with have experienced 10-fold compression on their databases and found that their average query sets ran 10 to 25 times faster—and some even faster than that (of course, your results might vary). Mindray's Xu Chang is a world-renowned expert on multiple database technologies. We found him on LinkedIn talking about his experiences with DB2 10.5: "While expanding our initial DB2 tests with BLU Acceleration, we continued to see exceptional compression rates—our tables compressed at over 92 percent. But our greatest thrill wasn't the compression rates (though we really like those), but rather the improvement we found in query speed, which was more than 50 times faster than with row-organized tables."

## DB2 pureScale Goes Ultra HA, DR, and More…

DB2 10.5 contains a number of enhancements to pureScale: DB2's continuous-availability solution that was first released on distributed platforms in 2009. DB2 pureScale technology is based on the "gold standard" DB2 for z/OS coupling facility (CF) and is available on both the Linux and AIX platforms. This technology enables multiple DB2 members to access the same database with full locking and logging integrity because the DB2 pureScale central caching facility (which is a software implementation of the CF—so we refer to it as the CF in this book) is very efficient at messaging, locking, and data movement among cluster members. It does this with next to no operating system overhead or noticeable impact to the database.

A number of neat features have been added to the pureScale technology in the DB2 10.5 release. First, DB2 pureScale now includes high-availability disaster recovery (HADR) support. One of the areas HADR can be used in is when the primary data center goes down due to a complete power outage, flood, or other type of catastrophic event at the local site. HADR synchronizes data with a remote DB2 pureScale cluster, enabling that remote site to take over if your primary cluster goes down.

DB2 pureScale also gets a couple of new features that keep it available during planned maintenance windows. The first feature is the ability to add members online so that you can dynamically add more capacity to the system without shutting down the entire cluster. The second feature is online Fix Pack maintenance, which enables DB2 pureScale members to be brought down individually, have DB2 Fix Pack maintenance applied to them, and then be brought back online without affecting the availability of the application. After the entire cluster has been updated in this rolling fashion, it can be instructed to run the latest fix pack level. This rolling fix pack maintenance capability, along with the ability to perform rolling maintenance on the hardware, the operating system, the network, and other features, enhances DB2 pureScale's ability to remain continuously available without planned maintenance interruptions.

DB2 10.5 also introduces enhanced lock management techniques to reduce batch overhead, randomized index keys to reduce index page hot spots, member subsets to manage multitenancy environments which differ in service-level agreements. and the ability to back up a DB2 pureScale cluster and restore it to a DB2 server that isn't using the DB2 pureScale technology. Finally, the much-loved self-tuning memory manager (STMM) is now available on a per-member basis in a DB2 pureScale cluster.

## DB2 as a JavaScript Object Notation Document Store

If you're a die-hard relational DBA, we're guessing a year ago, it's likely you never heard about JavaScript Object Notation (JSON). Six months ago, you heard about it, but figured it would go away—just like your acid-washed jeans from the 1980s. Perhaps three months ago, you began hearing about it more and more, and the other day you woke up and said, "I'd better find out what JSON is all about, because his name keeps coming up

everywhere I look, and I've never even met the man." JSON isn't a person—it's a text-based open standard designed for data interchange that originated with JavaScript but has become ubiquitous. It's the new XML in the application development and NoSQL worlds.

In addition to the giant step the DB2 10.1 release took into the NoSQL world by supporting a Jena Resource Description Framework (RDF) graph triplestore, DB2 10.5 adds support for the persistence of JSON objects in a document-style database. DB2 is embracing this use of JSON by implementing a Java-based JSON API and the popular MongoDB-compliant API as a first class citizen in DB2. The JSON API provides services to JSON applications that developers aren't accustomed to in the NoSQL world; for example, atomicity and transactional control. But it still gives developers techniques like "fire and forget" to build Web 2.0 applications with speed and agility.

Quite simply, DB2 empowers you to use it as a NoSQL document JSON store such that you can continue to take advantage of the flexibility of JSON from an application development perspective and still benefit from the use of DB2 as the robust database back-end. You can use all of the data in DB2 facilities (such as replication, HADR, security, and other features) that aren't necessarily present in some of the NoSQL database environments. Your developers can use this new JSON object model while ensuring that it is kept in a robust, well-performing database engine like DB2.

## Oracle Compatibility

To top off the DB2 10.5 release, there are a number of new Oracle compatibility features that help clients port their database applications from Oracle to DB2. These features enable them to break free of high maintenance costs and complexity, and leverage some of the most advanced database technology in the world today. Along with the addition of a number of new Oracle SQL functions, DB2 10.5 includes three new types of indexes and the ability to create tables whose fields are larger than the default data page.

There are three index enhancements in DB2 10.5. The first one enables you to create an index on an expression rather than use generated columns in the table. This type of index facilitates more efficient queries against the database and eliminates the storage requirements of generated columns, along with their maintenance headaches—not to mention it simplifies application development too.

The second index enhancement enables unique indexes to include NULL values. NULL values in unique indexes were a challenge for many clients because you couldn't have more than one NULL value in a unique index. However, applications often require more than one unique value in a table (for example, `EMPLOYEE_NUM` and `SSN`). Although one value might be guaranteed to be available at insert time (`EMPLOYEE_NUM`), the second value might not be available yet. If more than one employee has forgotten to bring their `SSN` to work, the system cannot insert the record because that would duplicate the NULL value. As of DB2 10.5, multiple NULL values on a unique index are supported so that existing applications can be simplified and developers don't have to worry about multiple NULL values causing integrity problems.

The third index enhancement is support for random index keys. Applications that typically have a huge number of inserts and updates occurring against the same index page can now have their keys randomly spread across multiple pages. The use of hashing can reduce "hot spots" from occurring in the index. This reduction of hot spot behavior is useful for some DB2 pureScale environments or online transaction processing (OLTP) systems.

The final compatibility feature enables developers to create tables whose row definitions are bigger than the current page size. Many Web 2.0 developers create tables with inflated column sizes, an approach that we don't sanction for a variety of reasons that we cover later in this book. In DB2 10.5, if you insert a row that doesn't fit on a single page, DB2 will spill portions of the row into a long field space. Of course, as you'd expect, DB2 automatically manages the spillover, so there's nothing for a DBA to do here. Row management is done behind the scenes so that you'll never get an error if you insert more data than what fits on the page.

## Tools, Tools, Tools

DB2 10.5 has so many new features, you might wonder what additional tooling is available to help you administer the product. At the time that DB2 10.5 became generally available, a number of tools were updated to support the current release and some specific features:

- **Day one support for DB2 BLU Acceleration** IBM InfoSphere Optim Query Workload Tuner V4.1 for DB2 for Linux, UNIX, and Windows gives expert advice on what tables to convert to column organization based on workload analysis, estimated benefit, and what-if analysis.

- **DB2 pureScale Enhancements** The ability to perform rolling updates in pureScale has been incorporated into the Configuration Manager, along with improvements to the Task Assistant and the Script Scheduler.
- **Additional Improvements** Data Studio now supports the setup and management of multistandby HADR, as well as the configuration of multiple federation objects. Recovery Expert includes support for adaptive compression and multitemperature storage. Merge Backup, Recovery Expert, and High Performance Unload have all been enhanced to support DB2 10.5.

The following list shows the products and tools that support DB2 10.5:

- IBM InfoSphere Data Architect V9.1 is a collaborative data design solution. It enables you to discover, model, relate, standardize, and integrate diverse and distributed data assets throughout your enterprise.
- IBM Data Studio V4.1 provides an integrated, modular environment for database development and administration of IBM DB2 for Linux, UNIX, and Windows. This software is available at no charge.
- IBM InfoSphere Optim Query Workload Tuner V4.1 for DB2 for Linux, UNIX, and Windows provides expert recommendations to help you improve the performance of query workloads.
- IBM InfoSphere Optim Performance Manager V5.3 for DB2 for Linux, UNIX, and Windows provides DBAs and other IT staff with the information that they need to manage performance proactively and avoid problems before they impact the business.
- IBM InfoSphere Optim Configuration Manager V3.1 for DB2 for Linux, UNIX, and Windows provides an inventory of clients and servers, tracks changes to client/server properties, and compares client/server environments with best-practice configurations.
- IBM InfoSphere Optim pureQuery Runtime V3.3 for Linux, UNIX, and Windows provides a runtime environment and an application programming interface (API) that enhances the performance of existing in-house applications without having to modify them.
- IBM DB2 Merge Backup V2.1 for Linux, UNIX, and Windows minimizes the impact of backups and shortens recovery times on production servers.

- IBM DB2 Recovery Expert V4.1 recovers database objects safely, precisely, and quickly without having to resort to full database recovery.
- IBM InfoSphere Optim High Performance Unload V5.1 for DB2 for Linux, UNIX, and Windows helps DBAs work with very large quantities of data with less effort and faster results.
- IBM InfoSphere Optim Query Capture and Replay V1.1 for DB2 for Linux, UNIX, and Windows captures production workloads and replays them in nonproduction environments for more realistic tests.

Many of these tools are provided as part of DB2 Advanced Workgroup Server Edition and DB2 Advanced Enterprise Edition. They can be purchased separately for the other DB2 editions.

## Wrapping It Up...

In this chapter, we have given you a glimpse of the enhancements that we cover in this book. BLU Acceleration is one of the biggest innovations that DB2 has ever brought to market, if not the biggest. It's a game changer. Nevertheless, continuing to invest in infrastructure that ensures that DB2 keeps its marketplace edge around certain key tenets such as scalability, high availability, and performance guarantees that a number of features pertaining to these tenets will make it into every new DB2 release, and this release is no exception.

The world is changing, and in a mobile-social-cloud world, JSON is the new lingua franca. Mobile devices generate and consume data at unprecedented rates. As you've come to expect from DB2, the features that we deliver are consumable and comparatively light on the pocketbook; for example, BLU Acceleration is *free* for existing DB2 Advanced Enterprise Server Edition (DB2 AESE) customers, and we even introduced a brand-new Workgroup Advanced Edition to deliver this capability for small to medium-sized deployments with a price point that will make your jaw drop for what you get—in a good way.

We realize that your time is precious, and we want to thank you for at least getting to the end of Chapter 1. We hope that at this point you're intrigued enough to continue reading. We promise that in return for the minimal investment you must make to read this entire book, you will have a great grasp of the DB2 10.5 release and of some major marketplace trends as well. Enjoy!

# 2

# DB2 pureScale Reaches Even Higher

Perhaps next to the BLU Acceleration technology (which we cover in Chapter 3), the most significant enhancements in DB2 10.5 pertain to the DB2 pureScale enhancements. If you're not familiar with this technology, we give you a brief overview in this chapter. In DB2 10.5, the DB2 pureScale technology gets a major boost in availability from both a planned and unplanned availability perspective with support for DB2's High Availability Disaster Recovery (HADR) technology, an extension to pureScale's ultra-available planned maintenance operations to include DB2 Fix Packs, online member add without a backup, and in-place online table reorganization.

In addition, DB2 pureScale gets easier to manage. The multiple commands that are required to implement DB2 Fix Packs are now streamlined into a single command called `installFixPack`. There is support for cross-topology backup and restore operations, native multitenancy capabilities, extended support for self-tuning memory manager (STMM) from its previous integration into DB2 pureScale, explicit hierarchical locking (EHL), and more. In this chapter, we detail most of these new features, but we don't discuss the new DB2 10.5 pureScale support for STMM and online in-place table reorganization because these are long-standing features that are very familiar to DB2 users.

## In Case It's Your First Time… Recapping DB2 pureScale

Although it's outside the scope of this book to fully cover the DB2 pureScale technology—it's been in the marketplace for over four years now—we want you to understand what it does and how it does it so that you can fully appreciate the significance of the DB2 10.5 enhancements.

In a DB2 pureScale environment, DB2 runs on a cluster of *host computers*. The DB2 software that runs on a host computer is referred to as a *DB2 pureScale member,* or simply a *member*. A member is just the `db2sysc` process that you're accustomed to with a "vanilla" DB2 installation. As of DB2 9.5 and later, this process contains all of the agent threads because DB2 is fully threaded on all platforms. The only difference with DB2 pureScale is that you have multiple `db2sysc` processes (its threads include services for buffer pools, memory regions, log files, and so on), all accessing the same shared copy of the database over different host computers.

You can see an example of a pureScale cluster in Figure 2-1. This cluster has four members, which implies that each `db2sysc` process runs on its own host computer. The figure includes an "exploded view" of Member 3 to show you what *each* member contains.

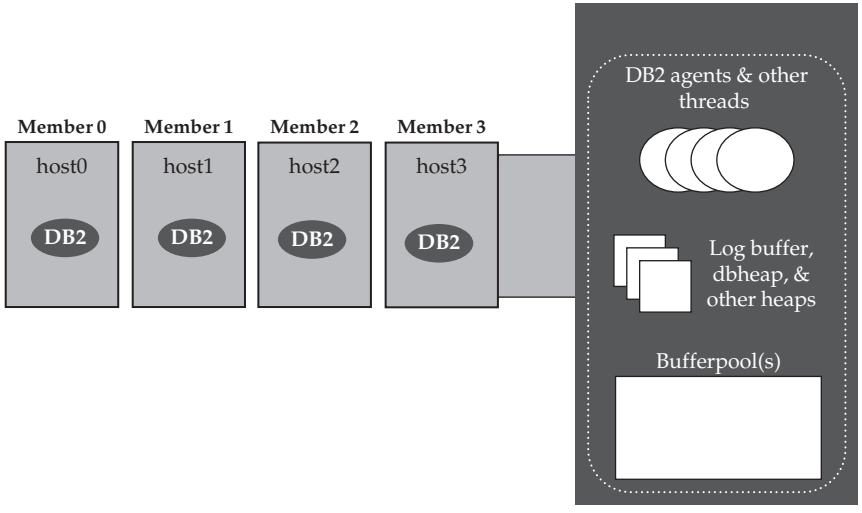


**Figure 2-1** *The DB2 pureScale member software running on four physical host computers in a DB2 pureScale cluster*

When people talk about the pureScale technology, they'll often use the term "host computer" to represent a *physical server* in a DB2 pureScale cluster. However, a host computer in a pureScale cluster could just as easily be a *virtualized operating system image;* for example, by using LPAR (Logical Partition) technology on an IBM Power server. In other words, from a DB2 pureScale perspective, a physical server could have two or more host computers on it.

As previously mentioned, DB2 pureScale runs on multiple host computers that can all access the same shared copy of the database. DB2 pureScale is, therefore referred to as a *data-sharing* architecture (just like DB2 for z/OS Sysplex), which is very well suited for scaling and keeping transactional workloads available. Each member in a DB2 pureScale cluster has equal access to a shared copy of the database over a storage area network (SAN). In addition, each member can write to its own log files as well as access another member's log files. This is very important for availability reasons. In Figure 2-2, we add this shared storage to our example DB2 pureScale environment.

DB2 pureScale's "nervous system" resides in the *cluster caching facility* (CF). The CF implements global locking services, global buffer management services, and more for a DB2 pureScale cluster; all of this is integral to DB2 pureScale's industry-leading scalability and availability characteristics.

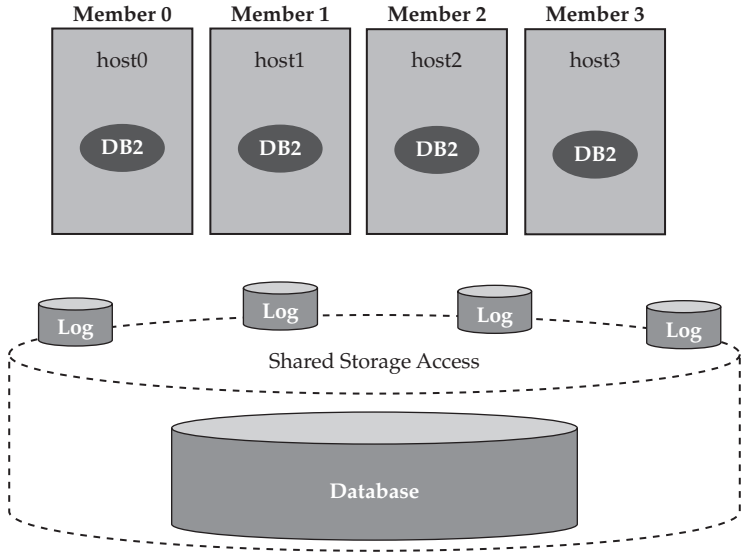


**Figure 2-2** *DB2 pureScale operates in a shared-everything architecture and, therefore, all members have equal access to a shared copy of the database over a SAN.*

We round out what a DB2 pureScale environment looks like in Figure 2-3, which shows two CF servers: a primary and a secondary. At a high level, the CFs use their memory to manage the state of the DB2 pureScale cluster by keeping track of global locks and global buffers, among other things, and most of this information is kept up to date synchronously at both the primary and secondary CF servers. If a DB2 pureScale cluster loses its primary cluster caching facility services, it can fail over to the secondary server within a matter of seconds. (Although it's not recommended for production environments, you can run a DB2 pureScale cluster with just a single CF; this might be appropriate for certain kinds of test environments.)

In Figure 2-3, you can also see that the primary and secondary CF servers are on their own host computers; however, it's important to note that these resources can be virtualized as well so that you don't have to dedicate a

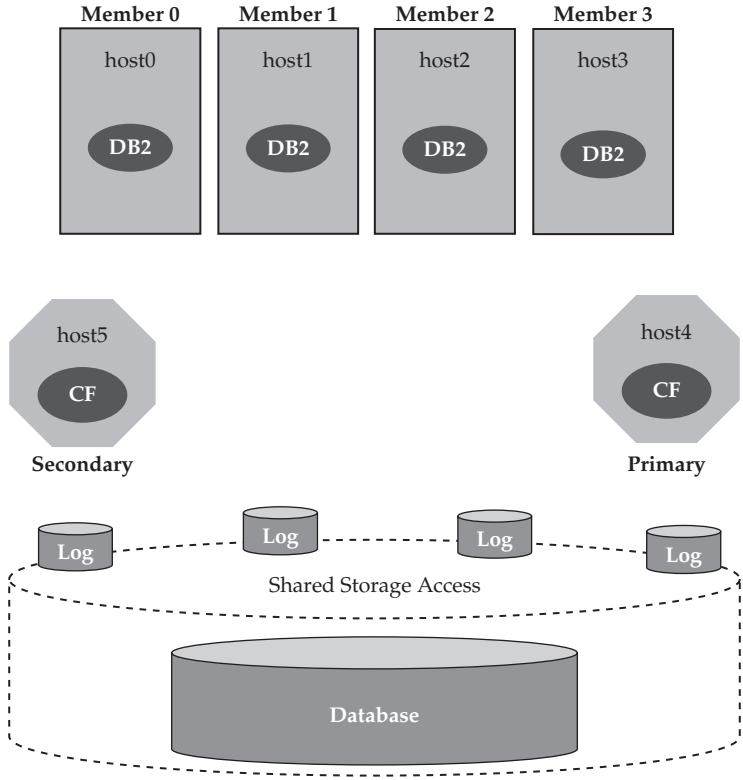


**Figure 2-3** *A typical DB2 pureScale solution includes host computers that are running the DB2 pureScale member software as well as primary and secondary CF servers.*

physical computer for these services. For example, you could have a DB2 pureScale cluster that consists of two physical computers, with each computer running one (or more) members and either the primary or secondary CF software. In short, to have a two-member DB2 pureScale cluster, you *don't need* to have four physically distinct host computers. (We get asked about licensing the CFs a lot, so we'll add here that you would never have to buy a license for a CF in a DB2 pureScale cluster.)

A DB2 pureScale environment uses a high-speed, low-latency, data-sharing interconnect for all performance-sensitive communications among the cluster's constituents. DB2 pureScale was engineered to be a different kind of transaction-optimized technology; therefore, it uses remote direct memory access (RDMA), which in itself is a differentiator compared to most database solutions in the marketplace. What's even more distinguishing is that DB2 pureScale uses *interrupt-free RDMA* (a differentiating term that we want you to remember when it comes to DB2 pureScale) when processing RDMA requests. You can use any RDMA-capable network; for example, InfiniBand and 10Gb Ethernet with a Mellanox RDMA–capable adapter both qualify. In Figure 2-4, we add a dedicated high-speed interconnect to our DB2 pureScale example environment.

The interrupt-free RDMA that pureScale leverages is the key ingredient in DB2 pureScale's "secret sauce": the ability to provide near-linear scaling for many workloads, even when no application affinity is configured. Competitors' data-sharing databases almost always require that the application be configured to target specific cluster members for different portions of the schema to minimize data-sharing traffic. When it comes to DB2 pureScale, our clients have proven that they see only about a 10 percent difference in scalability between completely affinitized and completely unaffinitized access to a DB2 pureScale cluster running a typical online transaction processing (OLTP) workload. This is a major accomplishment.

So, although it's always possible to squeeze out every last bit of performance by tying the application access patterns to a database cluster topology (typically required for meaningful scalability when using competing technologies), the difference between the most optimal and the least optimal application access pattern in DB2 pureScale is not large. This is why we say that the DB2 pureScale cluster topology is *transparent to applications*.

DB2 pureScale also includes an integrated component that we refer to as *DB2 Cluster Services*. DB2 Cluster Services is a synergistic and enhanced set of three well-proven IBM software components (and then some) that

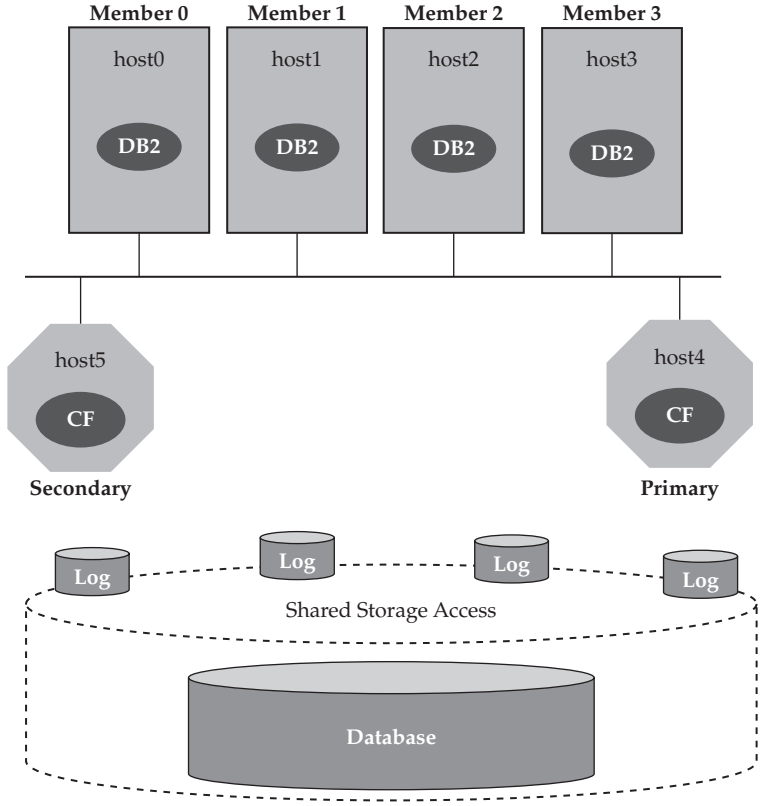


**Figure 2-4** *In a DB2 pureScale cluster, members and the CF servers communicate with each other over a high-speed interconnect with interrupt-free RDMA for all performance-sensitive communications.*

are fully integrated into DB2 pureScale. DB2 Cluster Services performs the following actions:

- Regulates "heartbeats" between members and the CF servers, which includes automatically detecting any component failures in the cluster
- Drives the automation of recovery events that occur when a failure is detected
- Handles the clustering and shared access to shared disk services by using a clustered file system that allows all members to access the shared storage through the clustered file system
- DB2 Cluster Services runs on all the constituents of a DB2 pureScale cluster, as shown in Figure 2-5.

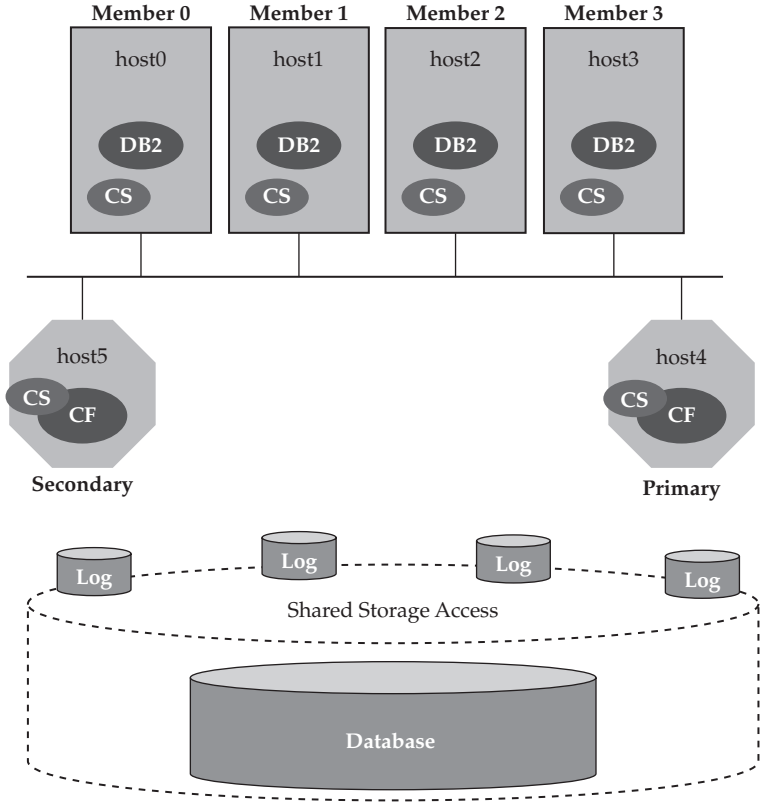


**Figure 2-5** *DB2 Cluster Services is part of the DB2 pureScale technology's DNA and provides multiple scalability and availability services to the cluster.*

The System z and DB2 for z/OS platforms played an enormous role in the creation of DB2 pureScale. The IBM System Technologies Group (STG) also played a key role in its development. DB2 Cluster Service's heart-beating capabilities come from IBM Reliable Scalable Cluster Technology (RSCT), which is used in PowerHA SystemMirror (formerly known as HACMP). DB2 pureScale's clustered file system is the IBM General Parallel File System (GPFS), a highly scalable and reliable file system that is used in many of today's computing clusters, including massive Hadoop clusters with its GPFS-FPO optimization. Of course, the CF technology itself has the System z Coupling Facility at its roots. Finally, Tivoli System Automation for Multi-platforms (SA for MP) is used for recovery automation and dependency management.

DB2 pureScale's recovery architecture is almost identical to that of DB2 on System z. A *force-at-commit protocol* is used so that before a transaction commit is acknowledged to the application, any pages that were updated by that transaction are sent to the CF, and any stale copies of those pages in the local buffer pools of other members are invalidated. Although this is but one example, every aspect of pureScale's recovery design is carefully optimized for recovery time. The result is a true data-sharing architecture, where surviving members of the cluster can continue to request locks, commit transactions, and read or write pages, *even* when a member's host computer fails.

What's more, pureScale shortens recovery times because it's able to release very rapidly any locks that are held by in-flight transactions on a failed member. Any transactions waiting for resources that are blocked by those locks can resume quickly. This is possible in a DB2 pureScale cluster because of one of its most differentiating features: *There is typically little to no database crash recovery to perform*. Folks that have taken the DB2 pureScale technology for a test drive *have found that database crash recovery completes in a fraction of the time that they are used to with competitive offerings*.

Although there are a lot of different IBM components built into it, DB2 pureScale is one fully integrated solution. When you install DB2 pureScale, all of the "stuff" that we've talked about so far in this chapter gets laid down automatically for you across all of the cluster's host computers. There are no failover scripts to write. And you patch a DB2 pureScale system with a DB2 pureScale Fix Pack, not by applying individual product maintenance.

To finish off our DB2 pureScale environment example, let's add the clients that connect to the database, as shown in Figure 2-6.

As you can see, clients can connect to any DB2 pureScale member in the cluster. By default, DB2 pureScale balances the workload across members of the cluster according to which member is least loaded. For example, an application might connect to Member 1 and issue a transaction against it, but the next transaction might get routed to Member 3 because Member 3 has a lower load than Member 1. If a member were to fail, the work running on it would be transparently rerouted to a surviving member, so the workload balancer algorithm needs to understand the availability of the cluster. DB2 pureScale can also handle the scenario in which you want to avoid a member that is scheduled for maintenance.

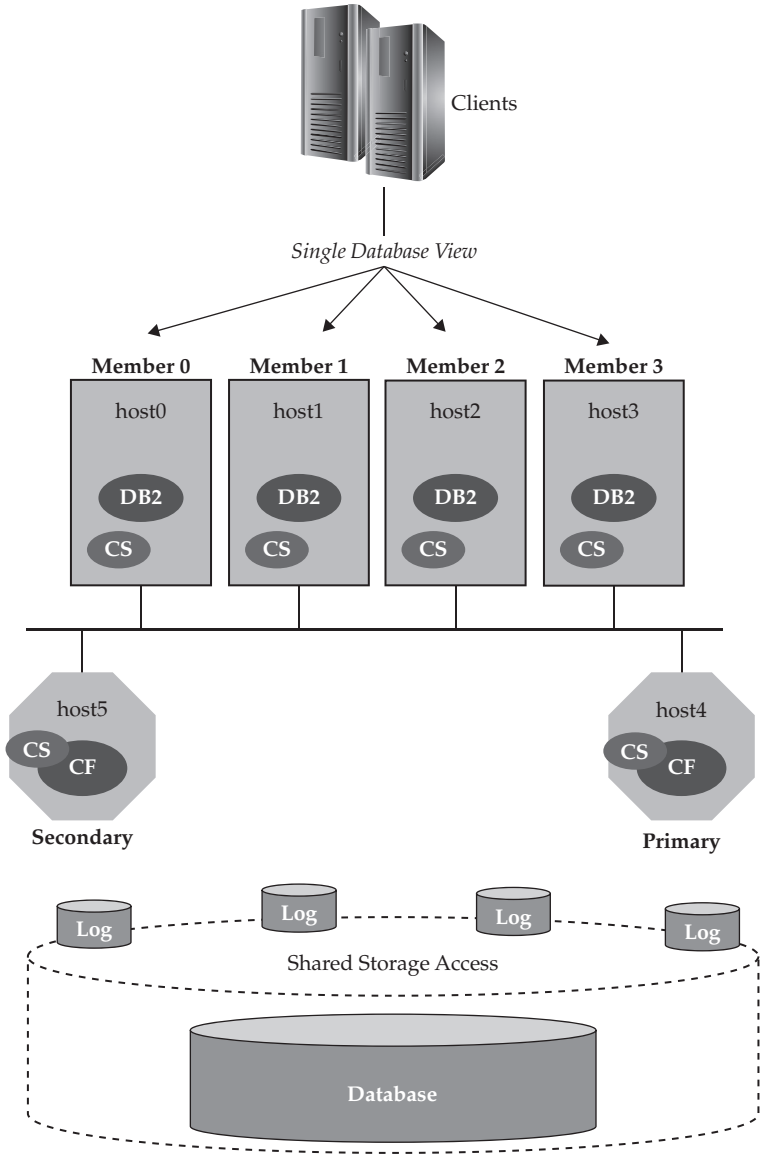


**Figure 2-6** *Clients connect to a DB2 pureScale cluster and are transparently workload-balanced across the cluster based on algorithms that understand the cluster's availability and utilization state.*

As you can see, the DB2 pureScale development teams partnered very closely with other IBM divisions to bring you this solution. The result is that industry-leading, data-sharing technology has become a first-class citizen in DB2 pureScale. You don't have to learn nondatabase interfaces to manage the cluster or create a clustered file system; we give you DBA-friendly commands and tooling for that. What's more, these contributing IBM technologies still exist on their own today, and each has been enhanced in its own right through the development of DB2 pureScale.

Now that you have a good understanding of what DB2 pureScale is, let's dive into the details of what is being offered in the DB2 10.5 release.

## You Can Put DB2 pureScale in More Places

When DB2 pureScale made its debut in the marketplace, it only supported Power AIX. Linux operating system support was added later, but only on IBM System x servers. In addition, only limited interconnect technologies were available (specific kinds of InfiniBand high-speed switches, among others). Although today we recommend that you adopt DB2 pureScale for dead simple application scaling and high availability through the IBM PureData System for Transactions offering, DB2 10.5 vastly opens up the landscape where you can deploy a "roll your own" (RYO) DB2 pureScale environment, and that's the point of this section.

As of DB2 10.5 Fix Pack 1 (and DB2 10.1 Fix Pack 2), DB2 pureScale is supported on any x86 Intel-compatible, rack-mounted server (including a non-IBM System x server) that supports any of the following InfiniBand QDR or Ethernet RoCE adapters: Mellanox ConnectX-2 EN 10Gb Ethernet Adapters with RoCE, Mellanox ConnectX-2 with Virtual Protocol Interconnect, MT27500 – Mellanox ConnectX-3 EN Dual-Port SFP+ 10Gb Ethernet Adapter, and MT27500 – Mellanox ConnectX-3 VPI QSFP Dual-Port InfiniBand card.

Of course, these requirements are always subject to change, and when they do, the aperture is broadened, so it's best to check the documentation. However, the point is clear: DB2 pureScale can be deployed in more and more RYO environments with enhancements that were added to this technology since it first became generally available.

## DB2 pureScale Gets Even *More* Available

We're not kidding when we tell you that we don't know of a single technology on distributed platforms that can provide better application-transparent scalability and availability for transactional applications.

In fact, we almost feel that we're doing the pureScale technology a disservice because we intentionally left out details about actual scaling experiences, and restricted ourselves to general statements about what folks are experiencing when they take DB2 pureScale for a test drive. For example, a systems integrator (SI) who provides professional services for a database vendor (that makes some of its clients see red) had the opportunity to test DB2 pureScale. This SI found that DB2 pureScale performed faster and that performance degraded significantly less when adding more nodes (members in DB2 pureScale-speak) to the cluster than was the case for the vendor's cluster technology. In fact, even when they tuned their application on their de facto technology stack for their "real" application cluster (they hard-coded logic in the application that directed it to a specific node in their cluster to avoid data-sharing performance overhead), DB2 pureScale *still* outperformed that technology *without* touching the application (it remained transparent).

Now consider the cost of always having to change your application to grow a cluster. Think about the risk. Now consider an optimal cloud-like environment where you want to leverage a utility-like computing model with elastic scale in a multi-tenant environment. You can't be changing your application in response to adding or removing computer capacity from the cluster—yet that is *exactly* what this other technology would have you do. That's the magic of the DB2 pureScale technology: *It yields tremendous performance results with application transparency.*

What's more, this SI found that DB2 pureScale could detect and recover from member failure much more quickly than the competitor's technology, *and* it didn't block I/O to disk during recovery operations—it just blocked access to rows locked by in-flight transactions. If you're not already familiar with the kind of results that are possible with DB2 pureScale, then this single example should pique your curiosity.

In this section, we detail the DB2 10.5 enhancements that make the pureScale technology even more available. DB2 pureScale has a solid reputation

for short recovery times from unplanned failures, but in this release, we've added a better disaster recovery story, minimization of planned downtimes, and more.

## DB2 pureScale Gets High Availability Disaster Recovery

DB2 High Availability Disaster Recovery (HADR) has been a decade-long staple in the DB2 repertoire that provides clients with near-effortless turnkey-like solution for HA or DR. The DB2 10.5 release extends HADR services to DB2 pureScale clusters.

In a non-pureScale clustered DB2 environment (think of a single server running DB2 Enterprise Server Edition, for example), HADR is used to ship database transaction logs at runtime between a primary database (the *primary*) and a standby database (the *standby*). The standby receives logs from the primary and replays those logs in real time to keep the standby up to date with the primary. It's outside the scope of this book to detail how HADR works, so if you need to learn more, check out the DB2 documentation on this topic at http://tinyurl.com/l6zkzto.

This amazing technology is priced right, unlike some other offerings that "guard" the data on a standby database. We often refer to HADR as "availability with clicks of a button." If you're familiar with HADR in a DB2 environment, there isn't much more to learn about it from a pureScale perspective, because HADR behaves in very much the same manner. It's still simple to set up and configure, and easy to manage. Because a pureScale environment itself is inherently highly available, when HADR is part of a pureScale discussion, it is always in the context of disaster recovery. Of course, the other methods for implementing a disaster recovery solution for DB2 pureScale still exist, namely Q Replication, Change Data Capture (CDC), storage-provided disk replication, and Geographically Dispersed pureScale Cluster (GDPC) among others; HADR adds to this repertoire of options. Although each disaster recovery approach has its advantages and disadvantages, one distinct advantage that HADR provides is simplicity of setup and maintenance.

HADR for pureScale is configured and managed much like HADR in non-clustered DB2 environments. When HADR is used in conjunction with DB2 pureScale, there is a *primary HADR cluster* (primary cluster) and a *secondary HADR standby cluster* (secondary cluster), where each cluster is made up of

multiple members and its own CFs. You set up an HADR pair in a DB2 pureScale environment in the same manner as in a nonclustered environment, by simply using a backup image from the primary HADR cluster to establish a standby database (by restoring it to the HADR standby cluster). When you configure HADR in a DB2 pureScale environment, the corresponding HADR configuration parameters are set on both the primary and standby pureScale clusters. When you use HADR in a DB2 pureScale environment, the standby must be a pureScale cluster (it can't just be DB2 Enterprise acting as an HADR standby to a DB2 pureScale cluster, for example) with the *same* topology (number of members) as the primary. That said, DB2 10.5 does support running more than one member on a physical machine; in other words, the standby cluster doesn't have to use the same number of physical machines, but it does need the same number of members.

When HADR is started on the standby pureScale cluster, the database is activated on only one member, which has the job of replaying the transaction logs sent from the primary cluster. Each member in the primary cluster ships its logs to the standby replay member through an HADR TCP connection, which handles communication between *each* member in the primary cluster and the *single* replay member in the standby cluster. The replay member on the HADR standby cluster is tasked with the job of merging and replaying the log streams.

Just as is the case with HADR in a non-pureScale environment, HADR support for pureScale includes nonforced (*role switch*) and forced (*failover*) takeover operations. Figure 2-7 shows a topological representation of HADR technology supporting a DB2 10.5 pureScale cluster. In this figure, the Beaverton DB2 pureScale cluster is operating as the primary cluster and the replay Member 0 on the Toronto-based pureScale HADR standby cluster handles the log replay activity.

## DB2 pureScale HADR Synchronization Modes

In DB2 10.5, HADR on DB2 pureScale clusters supports `ASYNC` and `SUPERASYNC` synchronization (sync) modes.

In `ASYNC` mode, log writes are considered successful only when the log records have been written to the log files on the primary database and delivered to the TCP layer of the primary system's host machine. In other words, under this mode, the transaction is considered to be complete after the log file has been scheduled for delivery to the standby and hardened on the primary

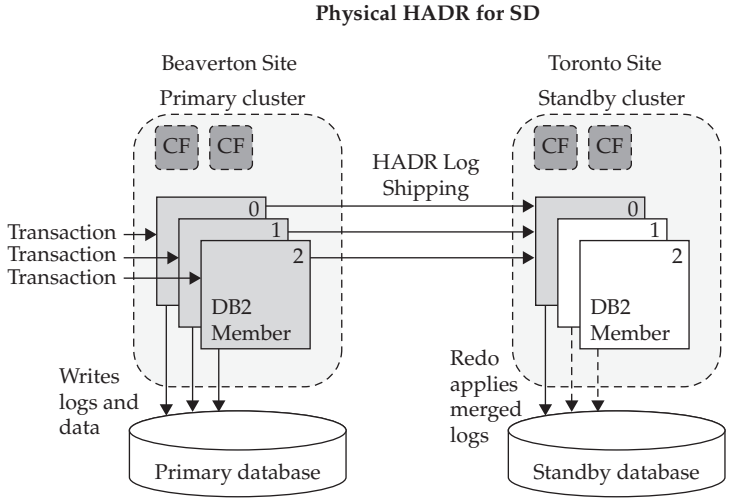


**Figure 2-7** *The DB2 HADR technology that is used in a DB2 pureScale environment, where Member 0 on the standby pureScale cluster is tasked with the log replay role required to keep the disaster recovery database up to date*

cluster. Because the primary system does not wait for acknowledgment from the standby system, if a failure occurs, transactions might be considered committed when they are still on their way to the standby database. A good analogy here has its roots in a classic spousal disagreement, where one yells from upstairs down to the other in the basement to bring something up. The one spouse knows they sent a message, but was the message even heard? In this mode of HADR, the spousal requester doesn't wait for an acknowledgment, such as "You bet honey, I'll get it for you, be right up." The requesting spouse (we won't say which one this is) just assumes the message was delivered. (For the record, we don't recommend this method of communication for marital harmony; we're more inclined to suggest the synchronous approach.)

In `SUPERASYNC` mode, log writes are considered successful after they've been successfully written to disk on the primary pureScale cluster. In this mode, the HADR pair can never be in `PEER` state as the transaction logs are shipped to the standby cluster via remote catch up. In our spousal communication analogy, it would be like the example request is considered sent to their spouse by just thinking about it. (We're certain most married readers have experience in this "What am I, a mind reader"? communication protocol too—also not recommended for marital communications.) This HADR mode

is useful if you don't want transactions to be blocked or to experience elongated response times because of network interruptions or congestion. If you think about it, this would be the fastest-performing mode for HADR, but also comes with the highest probability of transaction loss if the primary system fails because the primary cluster's database doesn't wait for acknowledgments from the standby database, as transactions are considered committed regardless of their replication state.

`ASYNC` and `SUPERASYNC` are typically what clients use when setting up HADR for disaster recovery environments. That's not the case for high-availability scenarios, which tend to use either `SYNC` or `NEARSYNC`, but remember that you're already getting the utmost in high availability with pureScale itself, so we're talking about disaster recovery specifically here. Why `ASYNC` and `SUPERASYNC` for disaster recovery then? To be prepared for all major types of disasters, you want a fair bit of distance between the two data centers. However, with any type of synchronized replication, the latency introduced due to extended distances between the primary site and the standby site is going to have an impact on the performance of the primary system. For this reason, use of synchronous HADR with "vanilla" DB2 today (`NEARSYNC` or `SYNC`) is typically limited to within 60 miles. This makes it a "metro" type of disaster recovery solution, which only protects clients from a limited class of disasters. Another constraint might be the cost of providing the necessary bandwidth and service levels for the pipe between the primary and disaster sites. This is needed for synchronous disaster recovery replication because delays in the data flow have an impact on the performance and responsiveness of the primary site. For these reasons and others, clients often choose an asynchronous disaster recovery solution, and pureScale's HADR capabilities are a perfect fit. If a client does have a particular need for a synchronous disaster recovery solution between two relatively close sites, other options exist, such as synchronous storage-based replication or Geographically Dispersed pureScale Cluster (GDPC).

## A DB2 pureScale HADR Example

Consider The Bogey Barn, an online golf retailer that has set up a DB2 pureScale environment to support their ordering and fulfillment system. They chose pureScale because it provides exceptional availability (the best that you can get on DB2 and, we believe, in the marketplace). They liked the

active-active no I/O-freeze to disk nature of DB2 pureScale technology, which differentiates itself from that other technology we are implicitly referring to in this chapter. Specifically, with DB2 pureScale, cluster member failures don't cause loss of access to the database.

That said, their DB2 pureScale cluster on its own lacked native disaster recovery capabilities. For example, if their database-hosting site were to encounter a disaster (fire, flood, a major power event, and so on), their database services would be lost. Worse yet, if their storage couldn't be recovered after the disaster either, massive data loss is possible, which could have serious financial implications to their business. With this in mind, the architectural planning team decided to harden their ordering and fulfillment system by implementing HADR when they migrated to the DB2 10.5 release.

The Bogey Barn's new disaster-protected pureScale environment includes a primary cluster (Site `A`, which houses a single member per physical machine, `p0` to `p3`) and a secondary cluster (Site `B`, which houses a single member per physical machine, `s0` to `s3`). Although there's a requirement that each cluster have the same number of members (not physical machines), we recommend that the clusters in such an environment mimic each other because in the event of a failure, users won't experience performance degradation. After all, if this system is backing your business, it needs to be as responsive after a disastrous event as it was before it. The `DB2` instance (whose instance owner is `db2inst`) houses the `myDb` database on all hosts (both primary and secondary) in the cluster. TCP port `4000` is used for HADR primary-to-standby communications on all of the hosts, and TCP port `8000` is used for SQL-based client/server communications.

The distance between the primary and secondary cluster is 500 miles, and the clusters are connected over a wide area network (WAN) that can support communications at 300Mbit/sec, with round-trip times for network packets of 80 microseconds. The Bogey Barn measured their site's transactional logging rates over a 24-hour period and observed a total of 1,728GB of log data generated on all log streams, which translates into 20MB/sec at the cluster level and 5MB/sec per stream. A 300Mbit/sec network can support up to 20MB/sec throughput rates, which enables peak logging rates to reach 37MB/sec.

Considering the network characteristics of the WAN and the implicit delay between the primary and secondary clusters, we recommended the `ASYNC` synchronization interval in this environment. If network throughput is just above the average logging rate, we would have recommended the

SUPERASYNC mode to enable the primary cluster to "get ahead" of the standby cluster during peak transaction times, while letting the secondary cluster "catch up" during nonpeak times; otherwise, the primary cluster's throughput will be capped at the network send rate during peak times.

Each cluster in this environment has a Gigabit Ethernet (GbE) network interface card (NIC) per member that's dedicated to handling HADR traffic (client/server traffic is handled by another NIC). This is a recommended setup for HADR on both pureScale and non-pureScale systems. The HADR NICs are connected to a router, which connects to the WAN. In this example, for the sake of simplicity, p0–p3 (for the primary cluster) and s0–s3 (for the secondary cluster) are assumed to be the host names that map to the IP addresses of these dedicated HADR NIC cards, even though the canonical names of these machines might be different.

**Configuring the Primary and Secondary pureScale Clusters** The first step in configuring a pureScale HADR environment is to take an online backup (no downtime is needed) on the primary cluster and to transport that image to the standby site. A subsequent restore operation sets up the HADR pair.

When you configure HADR in a DB2 pureScale environment, there are sets of corresponding HADR configuration parameters that need to be set on both the primary and standby pureScale clusters. Specifically, from a database configuration perspective, you set the following HADR configuration parameters to the *same value* on each member in the primary DB2 pureScale cluster:

```
HADR_TARGET_LIST      {s0:4000|s1:4000|s2:4000|s3:4000}
HADR_REMOTE_HOST      {s0:4000|s1:4000|s2:4000|s3:4000}
HADR_REMOTE_INST      db2inst
```

You also set the following parameters on each member (only member p0 is shown in the following example):

```
HADR_LOCAL_HOST       p0     # p1, p2, p3 for other members.
HADR_LOCAL_SVC        4000
```

And leave the following parameters at their default values:

```
HADR_SYNCMODE         ASYNC
HADR_TIMEOUT          120
HADR_SPOOL_LIMIT      AUTOMATIC
HADR_REPLAY_DELAY     0
HADR_PEER_WINDOW      0
```

Use the following command to set the HADR_LOCAL_HOST variable on all members:

```
db2_all 'db2 update db cfg for myDb member $DB2NODE using
  hadr_local_host hostname'
```

This command assumes that p0–p3 are the canonical names of the hosts (the names returned by the hostname command). If this isn't the case, you can use a custom script that returns the host name or the IP address that is used for HADR log shipping.

Next, set the local HADR service name (HADR_LOCAL_SVC); by not specifying a specific member number in this command, the configuration parameter will be updated globally for all members:

```
UPDATE DB CFG for myDb USING HADR_LOCAL_SVC 4000
```

On the standby cluster, set the same initial configuration parameters for each member, as shown in the following example:

```
HADR_TARGET_LIST      {p0:4000|p1:4000|p2:4000|p3:4000}
HADR_REMOTE_HOST      {p0:4000|p1:4000|p2:4000|p3:4000}
HADR_REMOTE_INST      db2inst
```

Set the HADR_LOCAL_HOST on the secondary cluster in the same way that you set it on the primary cluster, but use the s0–s3 range this time (one for each member). Moreover, HADR_LOCAL_SVC on the secondary cluster has to match its primary counterpart (set HADR_LOCAL_SVC=4000 on the secondary cluster).

Finally, on the secondary cluster, leave the same set of HADR configuration parameters at their default settings as you did for the primary cluster; namely, HADR_SYNCMODE, HADR_TIMEOUT, HADR_SPOOL_LIMIT, HADR_REPLAY_DELAY, and HADR_PEER_WINDOW.

**Start Your HADR Engines!** After your HADR environment is configured, you're ready to start up the HADR services and begin data synchronization. To start HADR in support of a pureScale cluster, you initially start it on the standby cluster member that you want to designate as the *preferred replay member,* using the following command:

```
START HADR ON DB myDb AS standby
```

Should there be an event that causes the standby cluster to become the primary, you may want to designate a preferred replay member of the current primary, which could find itself in the secondary role. If so, then that is

the member on which you want to run the following command, which will start HADR on the primary cluster:

```
START HADR ON DB myDb AS primary
```

The `START HADR` command requires no downtime, and you invoke it when the database is online.

***NOTE*** *Both primary and standby pureScale clusters will have a designated replay member. If you want to change the designated replay member on the current standby pureScale cluster, you need to deactivate the database on the standby (`DEACTIVATE DATABASE dbname`) and then restart HADR on the new preferred member on the standby cluster (`START HADR on DATABASE dbname`); the `START HADR` command must be issued on the member that you wish to be the preferred member. Similarly, to change the designated replay member on the current primary (future standby) pureScale cluster, you first have to stop the HADR processes on the primary cluster (`STOP HADR on DATABASE dbname`) and then start them up again as you did in the previous example, except this time on the new preferred replay member (on the primary cluster) using `START HADR on DATABASE dbname`; the command is to be issued on the member that is to become the new preferred member. While changing the preferred replay member on either the primary or secondary pureScale cluster, the primary database remains fully available and is processing transactions.*

After starting HADR to support disaster recovery for a DB2 pureScale cluster, we recommend that you immediately monitor the state of the cluster by issuing the following statement:

```
SELECT LOG_STREAM_ID, PRIMARY_MEMBER, STANDBY_MEMBER, HADR_STATE
 FROM <table> (mon_get_hadr(-2))
```

This query returns output that is similar to the following example:

```
LOG_STREAM_ID PRIMARY_MEMBER STANDBY_MEMBER HADR_STATE
------------- -------------- -------------- --------------
0             0              0              REMOTE_CATCHUP
1             1              0              REMOTE_CATCHUP
2             2              0              REMOTE_CATCHUP
3             3              0              REMOTE_CATCHUP
```

Note that the `STANDBY_MEMBER` is `0` on all streams; this is because all primary members connect to the designated standby replay member, which in this case is Member `0`.

The HADR streams are in `REMOTE_CATCHUP` state, which means that any logs generated since the online backup was taken are shipped from the primary cluster to the secondary cluster. After this shipping process reaches "end of log," the `HADR_STATE` changes to `PEER` state. When your HADR cluster is in `PEER` state, the primary database is fully protected and the standby is now a real-time copy of the primary.

## Recommendations for the Replay Member and Asymmetric Standby Clusters

Remember that only a single member on the standby cluster replays the logs. For obvious reasons, you might want to consider allocating more CPU and memory resources to the replay member. We've seen some "hardware budget challenged" clients configure nonreplay members as logical members that share host machines. The cost of the standby cluster is lower in this case, but in the event of a takeover, the performance of this cluster will be less than that of the original primary cluster. How you approach this is all about your applications and your service level agreement (SLA). Some environments can tolerate performance degradation after a disaster recovery event; but for those that can't, ensure that the designated replay member has enough resources to handle the task.

When you configure a secondary pureScale cluster as we suggest earlier, it is known as an *asymmetric HADR cluster.* A DBA might want to allocate only those resources (CPU, memory, and so on) that are required to run workloads on the secondary site *until* a failover occurs; in this case, the DBA is leveraging the fact that only a single member is used for replay. She might, therefore, virtualize the environment and assign only a fraction of the CPU cores that other inactive standby cluster members would normally have until a failover actually occurs.

## Highly Available Replay on a DB2 HADR pureScale Standby Cluster

When HADR is used in a DB2 pureScale environment, it preserves the core design philosophy of highly available database services for the standby cluster. For example, if a member in a standby cluster is performing replay and encounters an outage, DB2 automatically transfers the member's assigned

replay duties to another member on the standby cluster. As long as there is at least one online member in the standby, HADR replay services will continue. In fact, replay is so available, that if you want to completely stop this process, you have to explicitly deactivate it on the standby database.

Figure 2-8 shows a replay member failure and recovery on a standby system in an HADR pureScale cluster. On the left, you can see that only the replay member is active; it is busy replaying the logs that were shipped from the primary system. The lightly shaded fill on the other standby cluster members signifies that they are not active.

After replay moves away from the preferred replay member ("P" in Figure 2-8), DB2 does not automatically move replay duties back to the preferred member when that member comes back online. For example, perhaps the preferred replay member rebooted abnormally; when the preferred member is available again, it wouldn't assume its original role until you explicitly run the `DEACTIVATE DATABASE` command followed by the `ACTIVATE DATABASE` command (with appropriate options) on the standby database.

If a failure were to occur on the preferred member that's part of the *primary* HADR pureScale cluster in Figure 2-8, another member would step up and assume the role of shipping the transaction logs to the replay member on the standby. You can't control which standby member will assume the role of a failed replay member if such an event were to occur on the primary. DB2 pureScale simply looks at any members on which DB2 is running and randomly picks one.

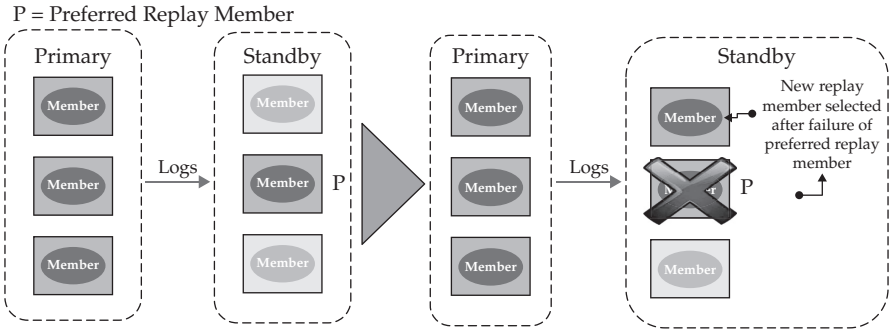


**Figure 2-8** *Failure of the replay member automatically causes one of the other members to take over and perform log replay on the standby HADR pureScale cluster.*

## Automatic Client Reroute in a DB2 pureScale HADR Cluster

Automatic Client Reroute (ACR) is the DB2 technology that is used to reroute DB2 clients from a failed primary server to the standby (which now becomes the primary) in the event of a planned or unplanned failover operation.

Setting up ACR is simple. In our example, you run the following commands on both the primary and secondary pureScale cluster:

```
db2 UPDATE ALTERNATE SERVER FOR DATABASE myDb USING HOSTNAME s0 PORT 8000
db2 UPDATE ALTERNATE SERVER FOR DATABASE myDb USING HOSTNAME p0 PORT 8000
```

The specified `port` is the SQL client/server communication port (the database manager configuration name, `SVCENAME`), *not* the HADR primary/standby communication port (the database manager configuration name, `HADR_LOCAL_SVC`).

In an ACR-enabled DB2 environment, the first time that a client connects to the primary server (the primary DB2 pureScale cluster in our example), the server returns the addresses of all primary members *plus* the alternative server address (`s0:8000`), which is the Member 0 address of the standby DB2 pureScale cluster. If a client can't connect to a member on the primary cluster, it tries another, and then another; if it can't establish a connection to any member of the primary cluster, it tries the standby cluster's Member 0.

For even better availability, we recommend leveraging a connection distributor (or multihome DNS entry) as an alternative server address, which is configured to include multiple members of the alternative server.

## Unplanned Failure: Roles Switch in a DB2 pureScale HADR Cluster

If the primary site in a DB2 pureScale HADR cluster fails, you have to explicitly perform a takeover (sometimes this is referred to as a *manual* takeover) by using the `TAKEOVER HADR` command, as shown in the following example:

```
db2 TAKEOVER HADR ON DATABASE myDB BY FORCE
```

As of the time that DB2 10.5 became generally available, the automated takeover that's provided by the deeply integrated Tivoli SA MP and that you're used to seeing in a "vanilla" HADR configuration isn't supported for HADR when used in a DB2 pureScale environment. For this reason, if you experience or suspect a failure of your primary pureScale cluster, we *strongly* recommend that you verify that the primary cluster is indeed experiencing an outage.

In our experience, clients typically don't want automated failover with their disaster recovery environments without their "consent." There are many reasons for this, and they are all associated with disaster recovery events from which you can recover (the data shop didn't burn down, but perhaps there was a serious power surge). For example, the disaster recovery site might not have the same "horsepower" as the primary, and certain activities need to be halted on the primary before the critical ones are moved over. Perhaps application servers that are part of the solution are close to the primary, and now that traffic is routed to the secondary (which is further away), performance might be impacted. Clients overwhelmingly told us that their disaster recovery plans are not automated and that they want to be proactively involved in such decisions.

**Reintegrating a Primary Server into a DB2 pureScale HADR Cluster** If an event takes out the primary pureScale cluster and work is failed over to the secondary (the new primary), at some point you're going to want to reintegrate the former primary cluster back into the HADR environment. You can do that by using the `START HADR` command, as shown in the following example:

```
START HADR ON DATABASE myDB AS STANDBY
```

If reintegration into the HADR cluster is successful, the cluster eventually reaches `PEER` state. Then, if you want to revert to the original role that each cluster played *before* the disaster occurred, you can issue the `TAKEOVER HADR` command, as shown in the following example:

```
TAKEOVER HADR ON myDB
```

If the reintegration attempt fails, the cluster that you intend to designate as the new standby cluster shuts down, at least initially, and must be re-created from a backup image.

## The Planned Failure: Roles Switch in a DB2 pureScale HADR Cluster

One of the great things about HADR is that it enables the seamless takeover of the primary's role by a standby cluster. This capability is the cornerstone of DB2's ability to minimize both planned and unplanned downtime. For example, consider a planned maintenance event, such as a weekend power-down of a data center that affects the entire primary cluster. HADR support for a planned noninvasive role switch avoids having to take an outage at the database level for such a planned maintenance event. Use the `TAKEOVER HADR` command for such a scenario after the HADR pair is in `PEER` state.

## Keep on Rolling, Rolling, Rolling: DB2 Fix Pack Updates Go Online

Since its debut in the DB2 9.8 release, DB2 pureScale has had the ability to support online maintenance; however, this support was limited to non-DB2 maintenance of the solution stack, such as firmware upgrades to the servers, LPARs, or general operating system (OS) fixes. For example, a DBA could identify a target member requiring some sort of maintenance (perhaps an OS patch), gracefully drain activity on that member, remove it from the cluster *without* impacting any running transactions, perform required maintenance actions, and transparently reintegrate it back into the cluster. When this work was complete, the DBA could move on to the next member and perform the same actions, essentially "rolling" through all of the members in this transparent manner until the maintenance was finished.

The DB2 10.5 release adds support for rolling updates that pertain to DB2 Fix Pack maintenance. This maintenance includes all of the DB2 pureScale software components, such as the DB2 server software itself, the CF software, and all of the integrated cluster services components (RSCT, Tivoli SA MP, and GPFS) that are delivered through the same maintenance pack, not separately.

What's more, the whole process of implementing a pureScale Fix Pack has been dramatically simplified. Before DB2 10.5, multiple steps were required to apply maintenance to a DB2 pureScale cluster. In DB2 10.1, you had to manually stop the target member, manually put the cluster manager and cluster file system services into maintenance mode, install the Fix Pack, run `db2iupdt`, exit maintenance mode, restart the pureScale servers on the member, and so on.

In DB2 10.5, all of these steps have been consolidated into a single command called `installFixPack`! This command replaces most of the manual steps that you used to perform for pureScale cluster maintenance.

### A DB2 pureScale Cluster Maintenance Example

Assume that you have a two-member DB2 pureScale cluster to which you want to apply DB2 10.5 Fix Pack 1. This cluster is shown in Figure 2-9. Note that `C` represents client applications that connect to the cluster (we're not showing the CF in this figure to keep things simple, but maintenance on the CF works in pretty much the same way—more on that in a bit).

The first step is to gracefully drain all active queries executing on Member 1. You aren't going to kill the running transactions—you're going to let them finish. You do this by invoking the `DB2STOP member1 QUIESCE` command.

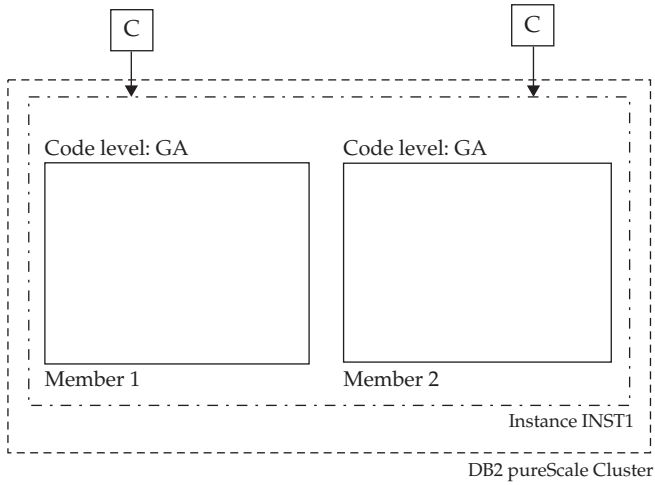


**Figure 2-9** *A DB2 10.5 pureScale cluster*

After command execution, DB2 won't route any new transactions to quiesced Member 1 (Figure 2-10).

It is worthy of note that while we show the use of the `DB2STOP QUIESCE` command here explicitly as a separate step, it's not required (if it isn't used without any options as is the case in this example) as `installFixPack` will automatically issue this command. Often users will want to issue a `DB2STOP QUIESCE` with a timeout option to ensure that only those applications that `COMMIT` or `ROLLBACK` during a reasonable period of time are waited on; the rest are forced.

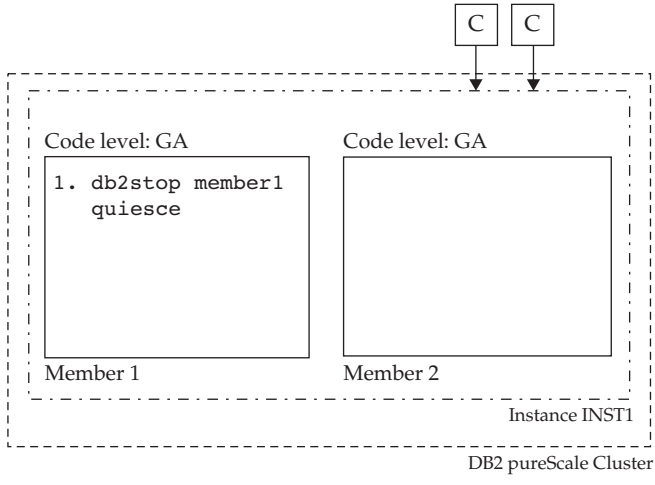


**Figure 2-10** *Member 1 is quiesced, and eventually all clients connect to Member 2, where all new incoming transactions on this cluster will execute.*

Now that Member 1 is "silent" and has no active work, you can invoke a single command to update the DB2-related code components on this member, as shown in Figure 2-11 and in the following example:

```
<new fixpack media path>/installFixPack -online member1 -I inst1
```

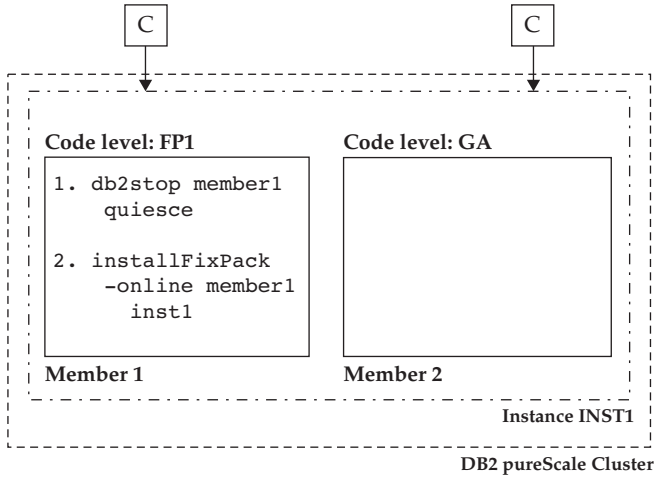


**Figure 2-11** *Maintenance is applied to quiesced Member 1 by using a single command to update all of the DB2-related pureScale software components.*

This command updates the binaries on Member 1 for the `inst1` instance to the new DB2 10.5 Fix Pack 1 level (FP1), which could contain maintenance that affects the DB2 engine, the GPFS file system, or the cluster services components (RSCT or Tivoli SA MP), among other components.

Member 1 will be automatically reintegrated into the cluster by the `installFixPack` command, which will issue a `DB2START member1` command under the covers. The pureScale load balancing recognizes that Member 1 has been started and starts to distribute work to it until the cluster is "balanced" from a workload perspective. At this time, because DB2 pureScale is heterogeneous from a DB2 code-level perspective, Member 1 actually runs in a special compatibility mode using the GA code until an explicit `COMMIT` option is specified on the `installFixPack` command (more about that later).

The same steps are repeated for Member 2 as you "roll" the maintenance through the cluster. During this time, all client connections (`C`) route to

Member 1 in the same way that they routed to Member 2 when maintenance was being applied to Member 1 (what we showed in Figure 2-10).

***NOTE*** *Although our running example is using a simple two-node DB2 pureScale cluster, you can extend this scenario to multiple members—up to the maximum 128 members supported by the DB2 pureScale technology. Just continue the rolling process that is outlined here until all members have the Fix Pack 1 binaries installed.*

Even though the entire cluster in Figure 2-12 has the Fix Pack 1 binaries running, each member is still executing transactions in compatibility mode with the old (GA) code level—you can't leverage any new capabilities delivered by the maintenance applied to all of the members until maintenance has been committed to the entire cluster. What's more, if for any reason a DBA doesn't want to stay at this code level, she can downgrade to the old (GA) code level, member by member, while online, using steps 1 to 3 in Figure 2-12, so long as the cluster hasn't been committed to the new maintenance level.

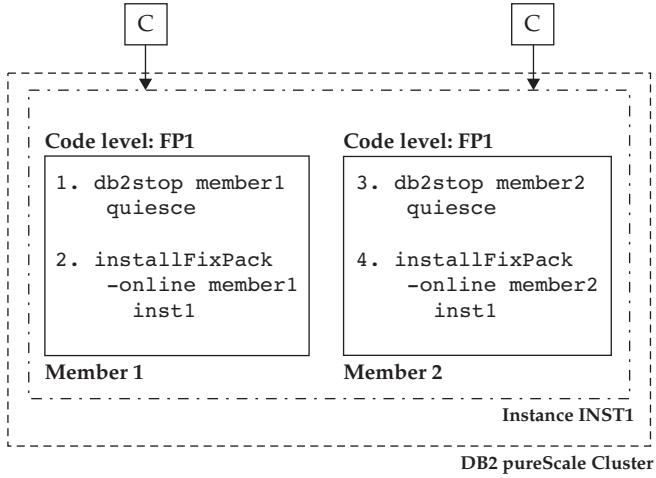


**Figure 2-12** *Maintenance has been applied (separately, in a rolling fashion) to both members in the cluster. Client connections and incoming work are balanced across the entire cluster. DB2 Fix Pack 1 binaries have been installed on each member.*

When you are satisfied with the new Fix Pack code, your final task is to commit the binaries by using the `installFixPack -COMMIT <instance_name>` command. This commit operation does exactly what its name implies: It commits the cluster to no longer run in compatibility mode.

After you've committed the new level to the cluster, you can't downgrade to the previous binaries (GA level in this example) in a rolling online fashion.

The scenario for applying maintenance to the CFs in your cluster is identical; although not explicit in our simplified example, we want to note that you need to upgrade both members and CFs to the same binary level before committing the binaries to the cluster.

### So You Want to Roll and Update? Our Best Practices

If you're going to take advantage of this incredible new feature, we want you to know about some hints, tips, and best practices that we've learned along the way:

- You should upgrade only a single machine, virtual session (an LPAR, for example), or host at a time.
- Apply maintenance to the cluster members *before* you apply it to the CFs.
- When it's time to apply maintenance to the CFs, apply it to the standby CF before the primary CF. Furthermore, we recommend you apply maintenance to the primary CF during a period of lower activity (if possible) to minimize the impact of primary CF failover, which occurs when DB2 services are stopped on the primary CF. Applications typically won't notice this slight pause; if the cluster is experiencing overall lower activity during this time, this pause will be even less noticeable.

## Adding Members Online

In the spirit of being an elastic, highly available system, DB2 10.5 lets you add new members to a DB2 pureScale cluster *without* the need to stop other members in the instance or to deactivate the database. Transactions can continue to run on other members while a new member is added to the instance (that is, while a new host is added to the cluster). There is no perceived impact to existing members that are running workloads. Furthermore, the new member can be added to an existing or new member subset (described later in this chapter).

If workload balancing is enabled when the new member starts, clients see this additional cluster capacity and will consider this member in their routing algorithm for new connections or transactions.

Finally, and perhaps most importantly, before DB2 10.5, an offline database backup was required before a new member could be added to ensure DB2 had a recoverability point that included the new member in the backup image. This was required before DB2 10.5 because DB2 backup and recovery services couldn't roll forward through the add member event, and that's why you needed a backup image right after you added a new member (and before you ran any new transactions). This way, in the event of a disaster, you'd have a backup to recover from. In DB2 10.5, this requirement has been eliminated.

## Cross-Topology Backup and Restore

DB2 10.5 supports cross-topology backup and restore operations that enable you to restore a non-DB2 pureScale backup image (for example, from a nonclustered DB2 Enterprise Server Edition database) to a DB2 pureScale cluster, and the reverse. If you want to restore an image from a non-pureScale environment to a pureScale one (or the reverse), you need to take an offline database backup.

You can also take a DB2 pureScale backup of a subset of members and restore that to a cluster with a superset of members *without* an offline backup. For example, you could take a backup image from a DB2 pureScale cluster with three members and restore it to a cluster with five members, as shown in Figure 2-13. Note that if you restore an image from a larger cluster to a smaller cluster, you need to take an offline backup … for now.

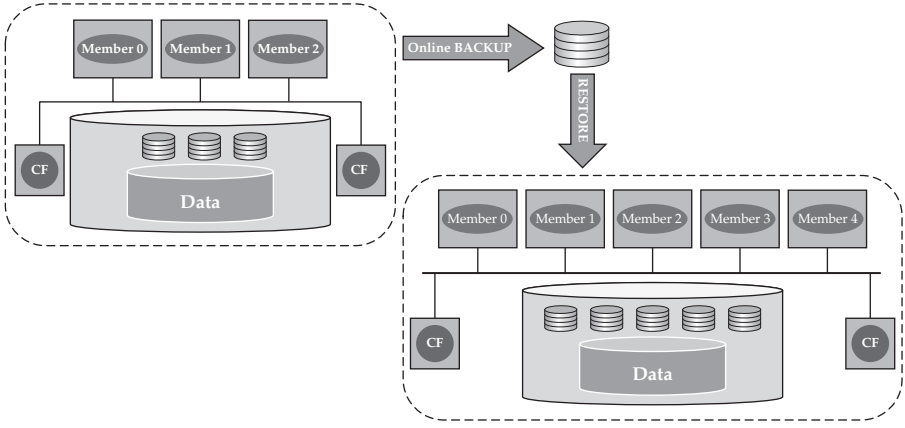


**Figure 2-13** *Backing up a smaller DB2 pureScale cluster and restoring the image to a larger cluster while online is a nice new feature in the DB2 10.5 release.*

There are two common scenarios where we feel that this enhancement is going to be very useful for our clients. The first one is obvious. If you want to migrate your existing DB2 server to a DB2 pureScale cluster, you can do this via backup and restore operations that make short work of this task. Keep in mind that your database has to meet DB2 pureScale requirements for this process to work. For example, DB2 pureScale requires that all table spaces—including those used for the catalog—use automatic storage. (Automatic storage is a requirement for the vast majority of new DB2 features going forward.)

This enhancement is also useful when you want to create a nonproduction version of a DB2 pureScale database in a nonclustered environment for quality assurance or test purposes; after all, in these environments, you might not need a highly scalable active-active environment. For example, some clients choose to mimic a pureScale cluster on a single test machine where they test and practice disaster recovery, while a number of other machines host the same database to support application development. Because one of the virtues of the DB2 pureScale technology is not having to worry about application awareness, moving that code to the pureScale production server is just going to result in faster performance.

The second area where we feel that these new capabilities are going to help our clients is this: You can now restore an *online* backup image from a point in time *before* a new member was added to the instance, and then roll forward through the logs to the addition of the new member and beyond—this is a logical extension of DB2 10.5's support for adding members online in a DB2 pureScale environment.

## Workload Consolidation in a DB2 pureScale Environment

DB2 10.5 can provision native multitenancy services to a DB2 pureScale cluster with its new member subset functionality. Before DB2 10.5, it was only possible to configure a workload to be directed to all members (via workload balancing) or to one specific member. The new member subset feature in DB2 10.5 enables the formation of a group (subset) of members to which applications can be directed, and corresponding workloads are balanced only across that defined subset. Figure 2-14 shows a scenario in which different workloads (one is transaction-based and the other is batch-oriented) are balanced

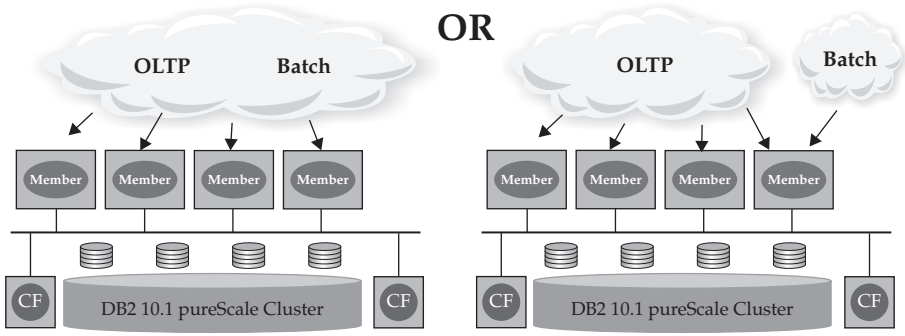


**Figure 2-14** *Distributing workloads across a DB2 pureScale cluster in the DB2 10.1 release; workloads are directed either to all members or to one member.*

among all members in a pureScale instance, the way things were done by default prior to DB2 10.5.

With the multitenancy feature in DB2 10.5, you can implement a DB2 pureScale cluster with characteristics such as the one shown in Figure 2-15. A subset (majority) of the cluster is dedicated to handling the transaction-processing workloads (OLTP), and another subset (minority) of the cluster handles batch operations.

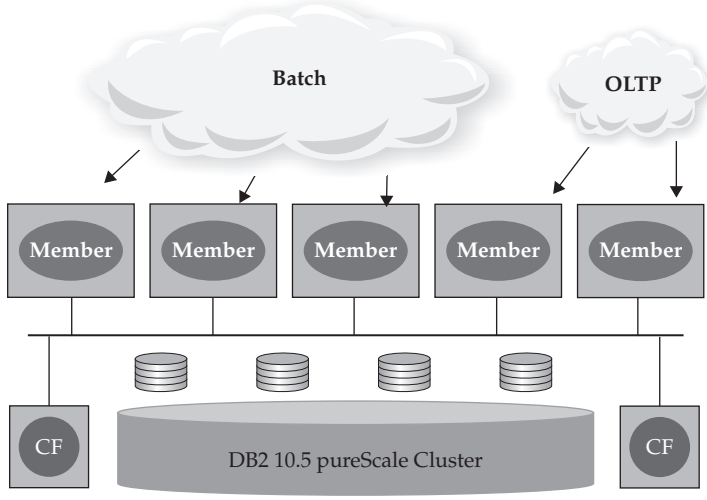


**Figure 2-15** *Exploiting the multitenancy capabilities of DB2 pureScale 10.5*

Creating and expanding subsets across a DB2 pureScale cluster is actually a lot easier than you might think; in fact, the development teams put a lot of work into this to make it a straightforward process. Consider the cluster shown in Figure 2-16.

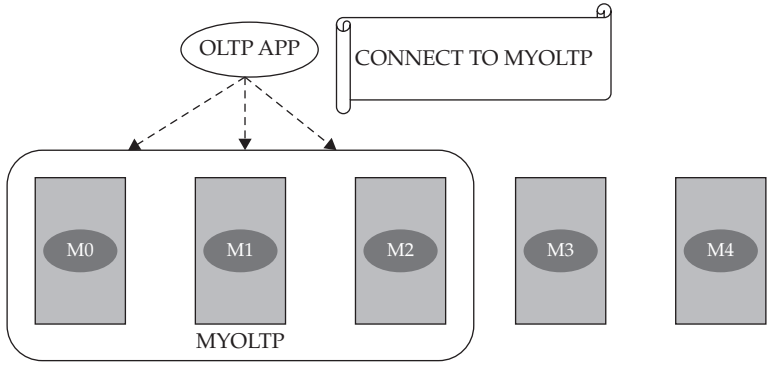


**Figure 2-16** *Creating a cluster subset to implement a multitenancy strategy in a DB2 pureScale 10.5 environment*

In this DB2 pureScale environment, the `OLTP` application connects to a member subset called `MYOLTP`; this subset was created by calling the `SYSPROC.WLM_CREATE_MEMBER_SUBSET` routine, as shown in the following example:

```
CALL SYSPROC.WLM_CREATE_MEMBER_SUBSET('oltp_db',
'<databaseAlias>MYOLTP</databaseAlias>', ' (0,1,2)')
```

If you want to dynamically add more capacity to this subset online so that the `OLTP` application's work is spread across four members instead of the three that you see in Figure 2-16, call the `SYSPROC.WLM_ALTER_MEMBER_SUBSET` routine, as shown in the following example:

```
CALL SYSPROC.WLM_ALTER_MEMBER_SUBSET('oltp_app', NULL, '( ADD 3 )')
```

## Explicit Hierarchical Locking

Explicit hierarchical locking (EHL) is a feature that's designed to remove any data-sharing performance overhead for DB2 pureScale tables or table range partitions that are only accessed from a single member. For example, a batch workload that is running on a single member may be updating tables or table range partitions that are not being accessed at the same time by the transactional workload that is running on other members. Another example is where

the pureScale cluster is being used as a consolidation platform, hosting multiple databases that are being accessed on different members by their own dedicated applications. If a particular database is only being accessed by its applications through a single member, all of the data access qualifies for EHL. You can enable this EHL feature by setting the new DB2 pureScale `OPT_DIRECT_WRKLD` configuration parameter to `YES`. When this feature is enabled, DB2 automatically monitors and detects whether data sharing is occurring across the cluster's tables or table range partitions.

If DB2 determines that a table or table range partition is being accessed from only a single member (as could be the case in some workload consolidation scenarios), the table or table range partition is placed into EHL mode, which has the effect of suspending the communication of data-sharing information (lock information, page information, and so on) with the CF. If an application spawns activity that causes access to this same table or table range partition on another member in the cluster, the table or table range partition automatically exits EHL mode when this multiple access pattern is detected by DB2.

EHL can deliver a performance boost for certain types of applications that don't access the broader cluster; however, there is a trade-off: Recovery times are a bit longer because recently committed updates aren't forced to the CF for tables or table range partitions in EHL mode, as they would be during normal DB2 pureScale operations. In addition, if you're updating a table or table range partition in EHL mode and the member handling the connection fails, the entire table or table range partition is considered locked during member restart because the CF doesn't have the fine-grained information that describes which individual record locks were held by in-flight transactions on the failed member.

## Wrapping It Up...

As you can see, DB2 10.5 includes major updates to the industry-leading DB2 pureScale software. Although the focus is on even higher levels of availability (HADR and rolling Fix Pack updates are the "eye candy" features), there are also some features that make DB2 pureScale even easier to use: a single command that bundles a set of previous commands for applying maintenance, multitenancy capabilities, and automated locking optimizations for specific workloads.

IBM PureData System for Transactions is the member of the IBM PureSystems family that includes the DB2 pureScale software in a tightly packed modality that provides a highly reliable and scalable database platform designed to help flatten complexity, accelerate time to value, and lower ongoing data management costs. This expertly integrated system enables IT departments to easily deploy, optimize, and manage transactional database workloads, and sets a new standard for workload-optimized systems.

IBM PureData System for Transactions is industry recognized for its four virtues: speed, simplicity, scalability, and smarts. Specifically, the industry-leading DB2 pureScale software provides this solution with database recovery times that are measured in seconds. It's considered simple because it's more than just an appliance. It's an entire ecosystem with an A–Z management toolset that enables you to deploy expert-built patterned databases in minutes, as opposed to weeks, months, or more. IBM PureData System for Transactions is highly scalable because it lets you grow the system from small to medium to large deployments without any planned downtime. Finally, it's smart because it supports Oracle database applications with minimal changes (it also supports DB2 applications without any changes too) in addition to a host of other smart features.

The built-in database management expertise in IBM PureData System for Transactions enables the system to do many of the ongoing administration tasks automatically, freeing up database staff from routine work. The fact that it's integrated by design results in factory-optimized systems that are designed for high reliability and scalability out of the box, streamlining system integration efforts. These systems provide a simplified experience from design to purchase to maintenance, which helps reduce total cost of operations. IBM PureData System for Transactions represents the future of data services delivery and management. It combines the simplicity of a workload-optimized appliance with the flexibility of tailor-made systems, providing both fast time to value and customization to meet specific business needs.

Some of the features that are described in this chapter have already made their way into IBM PureData System for Transactions, others are on the way. This great example of flattening the time to success is brought to you by the world's most sophisticated, available, and scalable distributed transaction system in today's marketplace.

# 3

# BLU Acceleration: Next-Generation Analytics Technology Will Leave Others "BLU" with Envy

BLU Acceleration technology finds its DB2 debut in the DB2 10.5 release. This market-disruptive technology is inspired by some pretty hot trending topics in today's IT marketplace, including in-memory analytics and columnar storage. BLU Acceleration uses innovative and leading-edge technology to optimize the underlying hardware stack of your data server and squarely addresses many of the shortcomings of existing columnar and in-memory solutions.

The first question we always get asked about BLU Acceleration is "What does 'BLU' stand for?" The answer: nothing. We're sure it's somewhat related to IBM's "Big Blue" nickname, and we like that, because it's suggestive of big ideas and leading-edge solutions. The IBM research project behind it was called Blink Ultra, so perhaps that's it, but don't let trying to figure out the name as an acronym keep you up at night; this technology is going to let you sleep like a baby.

Now let's dive into BLU Acceleration and answer the rest of the questions that we know you have and that we know how to answer. In this chapter, we introduce you to the BLU Acceleration technology. We discuss it from a

business value perspective, covering the benefits our clients are seeing and also what we've personally observed. That said, we don't skip over how we got here. There are a number of ideas—really big ideas—that you need to be aware of to fully appreciate just how groundbreaking BLU Acceleration really is, so we detail those from a technical perspective as well.

# What Is BLU Acceleration?

BLU Acceleration is all about capturing unrealized (or deriving new) value to your business from your existing (and future) analytics-focused hardware, software, and your human capital investments. BLU Acceleration is also about letting you ask questions and do stuff you've not been able to do in the past. If you asked us to sum up all of the BLU Acceleration value propositions that we cover in this chapter by placing them into just three buckets (that we will call *pillars*), they'd be *Next-Generation Database for Analytics, Seamlessly Integrated,* and *Hardware Optimized.*

## Next-Generation Database for Analytics

The value that's associated with this first pillar was identified by a self-challenge for our organization to define what technologies or attributes would be worthy of a bucket called Next-Generation Database for Analytics (NextGen). For example, you hear a lot these days about in-memory analytics, a proven mechanism to drive higher levels of performance. BLU Acceleration is about *dynamic in-memory analytics*—its optimizations extend well beyond memory so that, unlike some alternative technologies, all of the data required by the query doesn't have to fit into memory to avoid massive performance issues. Some alternative technologies return query errors if the data can't fit into memory (which isn't the case with BLU Acceleration); others are worse. BLU Acceleration uses in-memory processing to provide support for the dynamic movement of data from storage *with* intelligent prefetching. After all, we're in a Big Data era; if the cost of memory drops by `X%` every year, but the amount of data we're trying to store and analyze increases by `Y%(>X%)`, a nondynamic memory-only solution isn't going to cut it—you're going to come up short. This is why a NextGen technology needs to avoid the rigid requirement that all of the query's data has to fit into memory to experience super-easy, super-fast analytics. Of course, better compression can help to get

more data into your memory heaps, but NextGen analytics can't be bound by memory alone; it needs to understand I/O. For this reason, we like to think of BLU Acceleration as *in-memory optimized, not main memory constrained.*

Columnar processing has been around for a while and is attracting renewed interest, but it has some drawbacks that a NextGen platform must address. BLU Acceleration is about enhanced columnar storage techniques, including supporting the coexistence of row-organized and column-organized tables in the same database, and even in the same workload and SQL. In fact, if both an in-memory and traditional database could be contained in the same database technology's process model, using the same application programming interfaces (APIs), skills, backup and recovery (BaR) protocols, and more, that would also be a NextGen technology list item. BLU Acceleration does all of this.

A NextGen database should have what IBM refers to as *actionable compression.* BLU Acceleration has patented compression techniques that preserve order such that the DB2 engine can (in most cases) work with the compressed data without having to decompress it first. In fact, BLU Acceleration has a very broad range of operations that can be run on compressed data without first having to decompress it. These include, but are not limited to, equality processing (=, <>), range processing (`BETWEEN`, <=, <, =>, >), grouping, joins, and more.

A NextGen database that's geared toward analytics must find new and innovative ways to optimize storage that not only finds disk space savings, but optimizes the I/O pipe, and more. Although it's fair to note that the power and cooling costs associated with database storage have experienced double-digit compounded annual growth rates over the last few years, and that mitigation of these costs is an imperative in our Big Data world, that's not the whole story. Finding higher and higher compression ratios enables you to get more data into memory—effectively driving up in-memory yields since more data is in the memory-optimized part of the database. BLU Acceleration does this too.

Finally, making things easy to use and consume is a requirement for any NextGen technology. In our Big Data world, companies are facing steep learning curves to adopt new technologies (something that IBM is aggressively addressing with its toolkits and accelerators, but that's outside of the scope of this book). You want to get all of the benefits of a NextGen analytics

engine with minimum disruption, and that's going to mean ensuring that impressive NextGen technology is consumable and simple to use, but at the same time delivers amazing performance and incredible compression ratios. We cover this topic in the next section.

To sum up, a NextGen database needs to deliver out-of-the-box high performance for complex queries, deliver groundbreaking storage savings, and ultimately flatten the cost of the analytics curve with an unapologetic focus on consumability. We think a NextGen database has to be *disruptive in quantum benefits without interfering with your operations.* We're sure by the time you are done reading this chapter, you'll conclude that DB2 10.5 with BLU Acceleration is a NextGen analytics database.

## Seamlessly Integrated

BLU Acceleration isn't an add-on to DB2; it's *part of* DB2. There are lots of in-memory database technologies in the marketplace—some risky newcomers, some requiring new skills—but BLU Acceleration is seamlessly built into DB2. It's in its DNA, and this is not only going to give you administrative efficiencies and economies of scale, but risk mitigation as well. Seamless integration means that the SQL language interfaces surfaced to your applications are the same no matter how the table is organized. It means that backup and restore strategies and utilities like `LOAD` and `EXPORT` are consistent, and so on.

Think about it: BLU Acceleration is exposed to you as a simple table object in DB2 10.5. It's not a new engine; it's not bolted on; it's simply a new format for storing table data. Don't overlook this fact: We could have brought this technology to market sooner, but we intentionally chose to make BLU Acceleration part of DB2, not a layer on top of DB2. DB2 with BLU Acceleration looks and feels just like the DB2 you've known for years, except that a lot of the complexity around tuning your analytic workloads has disappeared; in addition, performance takes a quantum leap forward and the storage footprint of your data is slashed to a fraction. If you're new to DB2, after you start to use it, you'll get a sense of how easy it is to use and why it doesn't leave DBAs "seeing red," as does one of our competitors.

Another compelling fact that resonates with clients and analysts (and makes our competitors jealous) is that you don't have to rip and replace hardware to get all the benefits of BLU Acceleration.

## Hardware Optimized

The degree to which BLU Acceleration technology is optimized for the entire hardware stack is a very strong, industry-leading value proposition. BLU Acceleration takes advantage of the latest processing technologies, such as *parallel vector processing*, which we detail later in this chapter. If you think about the traditional approaches to performance tuning, you're likely to focus on three areas: memory, CPU, and I/O. We don't know of any other vendor who has put as much engineering effort into optimizing all three hardware-provisioned computing resources for their in-memory database technology as IBM.

Sure, there are other products in the marketplace that have an in-memory analytics solution, but our experience (and that of the clients we've been talking to) is that as soon as the data exceeds the memory allocation by a few bytes, those products keel over from a performance perspective, some "error out," and others do even worse! We've seen other database systems that claim they are I/O optimized; however, as more and more data is placed into memory, their advantage is mitigated. If your solution is focused on optimizing a single performance factor, you're going to end up with limitations and trade-offs.

BLU Acceleration optimizes the entire hardware stack, and it seeks every opportunity (memory, CPU, and I/O) to squeeze all that it can out of your hardware investment. As an analogy, think about your home computer, laptop, or even your smart phone. They all have multicore processing capabilities. How much of the software you've paid for has been developed from the ground up to fully exploit those cores? BLU Acceleration is designed to fully exploit all the computing resources provisioned to the DB2 server (we detail this in the section "How BLU Acceleration Came to Be: The Seven Big Ideas" later in the chapter).

# Convince Me to Take BLU Acceleration for a Test Drive

In this section, we share some of our findings and some of our customers' experiences, which serve to illustrate just how disruptive (in a good way) the DB2 10.5 BLU Acceleration technology really is.

## Pedal to the Floor: How Fast Is BLU Acceleration?

It's fast. Remember this motto: *Super analytics, super easy.* In both our internal and client performance tests, we've seen anywhere from single-digit to quadruple-digit performance speed-ups. Although your results are subject to a lot of variance (depending on your workload, your data, what server you're running on, and other things you can imagine our lawyers want us to write here), you *are* going to hear about—and likely experience for yourself—head-shaking performance speed-ups with minimal effort.

We asked a highly regarded DBA that we know, Andrew Juarez (Lead SAP Basis and DBA at Coca-Cola Bottling Consolidated), to try BLU Acceleration, and he found that "it makes our analytical queries run 4–15x faster."

Intel publicly shared their experience with BLU Acceleration's exploitation of leading chipset technologies, like their Advanced Vector Extensions (ACX) instruction set on their E5-based processing cores. Not only did Pauline Nist (GM, Enterprise Software Alliances, Datacenter, and Connected Systems Group) conclude that "customers running this hardware can now immediately realize dramatically greater performance boosts at lower cost per query," her team quantified that statement by noting that they're "excited to see a 88x improvement in query processing performance using DB2 10.5 with BLU Acceleration over DB2 10.1."

As a final reference point (we're just giving you samples here because there are a lot more), we invited another highly regarded DBA, Kent Collins (Database Solution Architect, BNSF Railway), to put BLU Acceleration through its paces. Kent told us that he was going to bring some of his most troublesome queries to BLU Acceleration and then tell us what he thought. When he finished the tests, Kent noted that "it was amazing to see the faster query times compared to the performance results with our row-organized tables. The performance of four of our queries improved by over 100x, with the best outcome being a query that finished 137x faster! On average, our analytic queries were running 74x faster when using BLU Acceleration."

It just gets better from there. At Taikang Life, workloads improved by 30x; Triton saw a 45x performance boost, and Mindray 50x! With that in mind, we're pretty comfortable suggesting that if you use BLU Acceleration for the right analytical workloads, you should expect to see, on average, an 8- to 25-fold performance—or better—improvement for your queries. (Ensure you

realize what we mean by *average*—some clients see triple-digit and beyond performance speed-up for single queries (often the most troublesome) but it's important to appreciate the average speed-up of the query set, which is what we are referring to here.)

## From Minimized to Minuscule: BLU Acceleration Compression Ratios

You're going to hear a lot about the NextGen columnar technology being used in BLU Acceleration, and you'll likely correlate that with storage savings. Indeed, column organization can greatly reduce the database footprint for the right kinds of tables, but there's more to it than that.

BLU Acceleration is as much about *what you don't get* as *what you do get*. With BLU Acceleration, you don't get to create indexes or create aggregate tables—quite simply, all the storage that would have been required for secondary indexes and aggregates is no longer needed. In fact, if your trusted extract, transform, load (ETL) processes have validated the uniqueness of your data, you don't even need to use storage to enforce it because we've extended the informational constraints capabilities introduced in DB2 9 to include uniqueness; this allows you to use this enhancement to inform the DB2 optimizer about the uniqueness of a row without persisting a secondary table object to strictly enforce that uniqueness. The bottom line is that all of these objects take up space, and because you're not going to need these objects when you use column-organized tables (and if you're not enforcing uniqueness because a reliable ETL process has cleansed and validated the data), you're going to save lots of space. This all said, if you want the database to enforce uniqueness, it does that too! You can still create uniqueness constraints and primary keys as you always have in the past and they'll be enforced with BLU Acceleration tables as they are with regular DB2 tables today—nothing is different.

We asked three of our customers (a famous investment brokerage house, a popular independent software vendor [ISV], and a well-known manufacturer whose products you likely see everytime you go for a country drive) to take tables in their database schemas that support query workloads and load them three times for comparison: once in DB2 10.1 without any compression, once in DB2 10.1 with automatic adaptive compression, and once using DB2 10.5 with BLU Acceleration. Their results are shown in Figure 3-1.

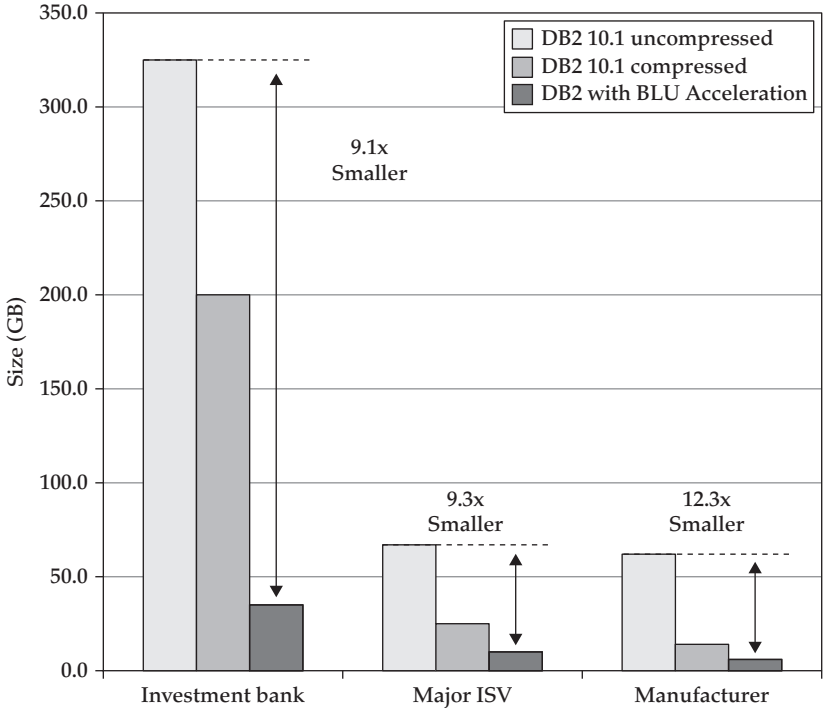


**Figure 3-1** *A sample of BLU Acceleration experiences from a table compression perspective*

Many vendors communicate storage measurements that are based solely on data compression ratios, but we've taken a much broader view. Figure 3-1 reflects the total number of pages that are being used for a table, including all the data, indexes, compression dictionaries, and table metadata—in short, the entire footprint of the schema.

In Figure 3-1, the lightly shaded bar on the far left of each data set shows the result of loading the schema into a regular DB2 10.1 database without any compression. The darker filled bar in the middle of each data set shows the result of loading the *same* schema on the *same* server (everything is the same here; that's important to remember) using the automatic adaptive compression capabilities first introduced in the DB2 10.1 release. Finally, the darkest shaded bar on the far right of each data set shows the schema's footprint using DB2 with BLU Acceleration.

Don't forget: DB2 has a wide spectrum of runtime optimizations for your analytics environment, and it has a holistic approach that we've yet to see anyone else adopt. For example, the more concurrency you need, the more temp space you're going to need. If a table spills into temp storage, DB2 may choose to *automatically* compress it *if* the database manager deems it beneficial to the query; if DB2 hypothesizes that it'll need to reference a large temporary table again, it might compress it as well. So, although temp space is a *definite* requirement in an analytics environment, we want you to know that those unique benefits aren't even shown in Figure 3-1.

Andrew Juarez, Lead SAP Basis and DBA at Coca-Cola Bottling Consolidated, told us that "in our mixed environment *(it includes both row- and column-organized tables in the same database),* we realized an amazing 10–25x reduction in the storage requirements for the database when taking into account the compression ratios, along with all the things I no longer need to worry about: indexes, aggregates, and so on." And Kent Collins, Database Solution Architect for BNSF Railway, stated that when "using DB2 10.5 with BLU Acceleration, our storage consumption went down by about 10x compared to our storage requirements for uncompressed tables and indexes. In fact, I was surprised to find a 3x increase in storage savings compared to the great compression that we already observed with DB2 Adaptive Compression."

## Where Will I Use BLU Acceleration?

Considering the value proposition that we shared with you in the last couple of sections, we're sure that you're eager to see what BLU Acceleration can do for you. As with any technology, there are areas in which you'll want to use BLU Acceleration and areas in which you won't.

As enterprises begin to see that the limits of their current enterprise data warehouse (EDW) and data mart technologies affect applications that are used to drive decision making, they're desperately looking for solutions. Typical challenges include latency, synchronization, or concurrency problems, and scalability or availability issues. The off-loading of data that supports online analytical processing (OLAP) applications from an EDW into a data mart environment is an excellent opportunity to remedy these types of problems through the power of BLU Acceleration and your off-the-shelf analytics toolset.

BLU Acceleration simplifies and accelerates the analysis of data in support of business decisions because it empowers DBAs to effortlessly transform poorly performing analytic databases into super-performing ones, while at the same time insulating the business from front-end application changes and toolset changes. From a DBA perspective, it's an instant performance boost—just load up the data in BLU Acceleration and go…analyze. One of our clients at a DB2 10.5 announcement event told the press that "…I thought my database had abended because a multimillion row query was processed so fast." Think of that impact on your end users.

BLU Acceleration makes your business agile too. Typical data marts require architecture changes, capacity planning, storage choices, tooling decisions, and optimization and index tuning; with BLU Acceleration, the simplicity of create, load, and go becomes a reality—it's not a dream.

We've done a lot of integration work with Cognos Business Intelligence (Cognos BI) and BLU Acceleration. For example, deploying a Cognos-based front end is done by simply modeling business and dimensional characteristics of the database and then deploying them for consumption using Cognos BI's interactive exploration, dashboards, and managed reporting. BLU Acceleration technology flattens the time-to-value curve for Cognos BI (or any analytics toolsets for that matter) by decreasing the complexity of loading, massaging, and managing the data at the data mart level. We think perhaps one of the most attractive features in what we've done with BLU Acceleration and Cognos is that the Cognos engine looks at a DB2 column-organized table just like it does a row-organized one. Since they are both just DB2 tables to Cognos, a Cognos power user can convert underlying row-organized tables to BLU Acceleration *without changing anything* in the Cognos definitions (all the steps we just outlined); that's very cool! Finally, DB2 10.5 includes five user licenses of Cognos BI, so you can get started right away and experience this integration for yourself.

Almost all lines of business find that although transactional data systems are sufficient to support the business, the data from these online transaction processing (OLTP) or enterprise resource planning (ERP) systems isn't surfaced to their units as actionable information; the data is "mysterious" because it isn't organized in a way that would suit an analytical workload. This quandary gives way to a second popular use case: create data marts directly off transactional databases for fast line-of-business reporting.

Because we've made BLU Acceleration so simple, DBAs can effortlessly spin up line-of-business–oriented data marts to rapidly react to business requirements. For example, consider a division's CMO who's sponsoring a certain marketing promotion. She wants to know how it's progressing and to analyze the information in a timely manner. DB2 with BLU Acceleration empowers this use case.

Given the columnar nature of BLU Acceleration, data doesn't have to be indexed and organized to support business queries. As well, the data mart can now contain and handle the historical data that's continuously being spawned out of a system of record, such as a transactional database.

As you can see, BLU Acceleration is designed for data mart–like analytic workloads that are characterized by activities such as grouping, aggregation, range scans, and so on. These workloads are typically processing more than 1 percent of the active data and accessing less than 25 percent of the table's columns in a single query. Although not required, star and snowflake schemas are going to be commonplace in leading BLU Acceleration candidates' schema design. DB2 10.5 with BLU Acceleration is also cloud-ready, so if you're provisioning analytic services in this manner, it's a nice deployment fit as well. You will even find DB2 with BLU Acceleration offered via various cloud vendors through their respective Software-as-a-Service (SaaS) procurement channels.

Finally, although it feels like magic, BLU Acceleration isn't for all workloads and use cases. For example, operational queries that access a single row or a few rows (likely by using an index) aren't the right fit for BLU Acceleration. If your environment continually processes fewer than 100 rows per commit, BLU Acceleration isn't the best choice because the commit scope compromises the analysis that can be done on the data for optimal compression rates. There are some other reasons not to use BLU Acceleration, but they tend to be related to shorter-term restrictions so they're not worth mentioning because by the time you are reading this book, they will likely be gone.

## How BLU Acceleration Came to Be: The Seven Big Ideas

The number seven is pretty special in almost any domain. If you're a numbers person, it's a base and double Mersenne prime, a factorial prime, and a

safe prime. If you study the Christian bible, you'll find the seven days of Genesis, the seven Seals of Revelation, and more. Artist? The Group of Seven means something to you. Mythologists have their seven heavens, and if you're a music lover and fan of Queen, you've undoubtedly heard their "Seven Seas of Rye" hit song. There also happen to be seven "big ideas" that served as guiding principles behind the BLU Acceleration design points, which we cover in this section.

## Big Idea Number 1: KISS It

Although we're sure that you're going to want to kiss the development team behind BLU Acceleration when you get a chance to take it for a test drive, the core idea behind BLU Acceleration was to ensure our development teams gave it a KISS—"Kept it simple, Sweetie." (We may have used a different word than Sweetie in our decree, but we'll leave it at that.) The point was to keep it super easy *for you*—we didn't want dozens of new parameter controls, additional software to install, make you change your application, forklift migrate your hardware, and so on. After all, if you look at traditional relational database management systems (RDBMS) and the kinds of complexity that they've introduced, you can begin to see why much of the proportional cost of administration and management of servers and software has risen relative to total IT spending in the last decade.

Keeping it simple was the key behind our *Super Analytics, Super Easy* BLU Acceleration tagline. In fact, the simplicity of DB2 with BLU Acceleration is one of its key value propositions. When we talk about administration of the BLU Acceleration technology, we tell you more about what you *don't have to do* than what you have to do. How cool is that?

First and foremost, there's no physical design tuning to be done. In addition, operational memory and storage attributes are automatically configured for you right out of the box (more on that in a bit). Think about the kinds of things that reside in a DB2 DBA's toolkit when it comes to performance tuning and maintenance: these are the things *you're not going to do* with BLU Acceleration. You're not going to spend time creating indexes, creating materialized query tables (MQTs), dimensioning data using multidimensional clustering (MDC) tables, distributing data with a partitioning key, reclaiming space, collecting statistics, and more. *None* of this is needed to derive instant value from BLU Acceleration technology. From a DBA's perspective, you just

*load and go* and instantly start enjoying performance gains. Now compare that to some of our competitors who make you face design decisions like the ones we've just outlined.

One of the reasons that BLU Acceleration is easy to use is related to the fact that it uses the same DB2 process model that you're used to, the same storage constructs, the same buffer pools, the same SQL language interfaces, the same administration tools, and so on—but with seriously enhanced algorithms inside. It's *not* bolted-on DB2 technology; *it is* DB2 technology. The beautiful thing here is that you can have your column-organized tables in the same table space—and even in the same buffer pool for that matter—as your row-organized tables, *and* the same query can access all of that data. Don't overlook this point; it's *really* important and unique, and deserves repeating in this book—which we do. We delve into the details of how DB2 BLU Acceleration is really just a part of DB2 in the section "Under the Hood: A Quick Peek Behind the Scenes of BLU Acceleration" later in this chapter.

Looking back at this authoring team's collective century-plus experience in the database world, we can't help but take note of all the tuning papers we've read (or written) and courses we've attended (or taught). We've allocated memory and tuned it. We've built covering indexes, used various kinds of compression options (thankfully even for row-organized tables, DB2 automates almost all of this with a single switch), tuned I/O, invoked specialized buffer pool algorithms, and more. You'll find all of this work pretty much eliminated in DB2 with BLU Acceleration, replaced with a management-by-intent policy in which you tell DB2 to run in analytics mode; and once set, the process model, workload management, memory allocations, almost everything is pretty much automatically configured for DB2 to operate with an analytics persona.

Have you ever cooked microwavable popcorn, followed the instructions (cook for 3 to 4 minutes), and then found that some of the kernels were perfectly popped, some were kind of burnt, and others didn't pop at all? We have a microwave oven in our development lab that has a "Popcorn" button. We're not really sure how it does what it does, but if we put a bag of popcorn in the oven and press this button, we end up with a late-night "hackathon" snack that would make Orville Redenbacher proud. Although DB2 with BLU Acceleration won't make popcorn for you, it's essentially the same approach. You "press the button," load the data, and run queries. It's hard to imagine

things getting any simpler than that, and that's why we call BLU Acceleration *Super Analytics, Super Easy.*

## Big Idea Number 2: Actionable Compression and Computer-Friendly Encoding

The second big idea behind BLU Acceleration pertains to how DB2 encodes and compresses data and the way that it "packs" the CPU registers. DB2 with BLU Acceleration also has the ability to operate on the data *while it's still compressed* (yes, you read that right).

Now, why would you want to do this, and more so, why is this such a big idea? First, if the data is smaller, you can obviously put more of it into memory. However, if you can save the most space on the data that's repeated the most (more on this in a bit), you're going to expend even less computing resource to work on the data that you use the most. Now, if DB2 can operate on the data while it's still compressed, not only will the engine realize more memory savings, but it'll also save on all that CPU that would have gone into decompressing the data just to evaluate predicates, for example. (We're sure you've figured this out by now, but DB2 with BLU Acceleration does both.)

Let's illustrate this benefit by thinking about a typical analytic query that might have to mass scan over several of a table's columns, decompressing and evaluating all the values for the predicate columns, even though perhaps only 1 to 10 percent of that data satisfies the predicates in the query. In a BLU Acceleration environment, if only 5 percent of the data actually satisfies the query, the database engine needs to decompress only about 5 percent of the data in order to return it to the client application, effectively avoiding 95 percent of the decompression cycles that would have been consumed by typical compression technologies.

For example, DB2 with BLU Acceleration technology can apply equality and inequality predicates on compressed data *without* spending CPU cycles to decompress that data; as you can imagine, this is *very* valuable for range queries (for example, "Give me the run rate percentage and year-over-year dollar growth of revenue for area A between year X and year Y").

The way that BLU Acceleration is able to work on compressed data comes from the way it's encoded—both on disk and in memory. BLU Acceleration's fundamental approach to compression is a variation of Huffman encoding. *Huffman encoding* looks for the symbols that occur most frequently and gives

them a shorter representation. Think of it this way: Something that appears many times should be compressed more than other things that do not appear as often. For example, if you're compressing the complete works of Shakespeare and the most frequent symbol is the letter "e," Huffman encoding would compress it with just one bit. The letter "z" occurs much less frequently, and therefore a Huffman encoding scheme might compress it with seven bits. This encoding method yields very impressive compression ratios. Using approximate Huffman encoding as its base, BLU Acceleration adds several other compression methods to eke out a super-tight form factor for its data.

Now let's show you how this actually works because BLU Acceleration doesn't really encode on a per-letter basis. In fact, the scope of BLU Acceleration encoding is at the column-value level. Consider a compressed `Last_Name` column in a table that tracks complaints for a certain airline carrier. The name of a frequent complainer (with four registered complaints), `Zikopoulos`, has ten characters in it, but as you can see on the left side of Figure 3-2, it's using a smaller encoding scheme than `Huras`, which has only five characters and two registered complaints. Notice that `Lightstone` is the same length as `Zikopoulos`, but he has only one complaint and therefore has the largest bit-encoded representation.

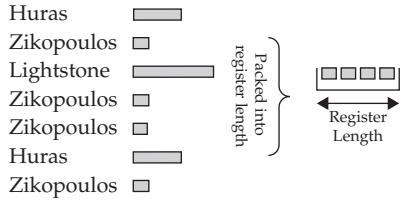


**Figure 3-2** *A representation of how BLU Acceleration encodes and packs the CPU register for optimal performance and compression*

This encoding is dynamic in nature. This is a really important point: Consumability is in the DNA of all our technologies. For example, in DB2 9.7 we introduced index compression, which was designed to enable the DB2 engine to automatically detect, define, and dynamically adapt the prefix that it compresses as one of its three auto-selected and possible combined index compression techniques. This is something that a major competitor's technology can't do; that technology requires DBAs to drop and manually redefine the

compression prefix to adapt to new data. The key point is that BLU Acceleration can dynamically alter the compression bias using adaptive compression for new data loads that changes the distribution of common symbols, thereby ensuring continuous optimization.

*Offset coding* is another compression optimization technique that's used in BLU Acceleration. Offset coding is very useful with numeric data, the kind of data you'd typically see in financial applications. Think about an options chain for a traded stock and its corresponding open interest on contracts for sale. `IBM` might show `100` interested contracts for a certain price, `101` for a call in the following month, and `102` of a call that is even further out, and so on. With offset coding, instead of trying to compress `100`, `101`, and `102`, DB2 stores a single value (`100`) and then stores the offsets to that value (`1`, `2`, …). This technique is actually very similar to how DB2 compresses index record IDs (RIDs), one of the three autonomic index compression algorithms that DB2 can dynamically apply to indexes. There are other compression optimization techniques used in BLU Acceleration, such as prefix compression, and other things that we're not allowed to talk about, but they all combine to provide multifaceted, complete, and impactfully obvious compression capabilities.

Values in BLU Acceleration are compressed to be *order preserving*, which means that they can be compared to each other while they are still compressed. For example, let's assume the value `50000` compresses to a symbol like `10` (for simplicity's sake) and `20000` compresses to `01`. These two values are ordered (`01` is less than `10`, as `20000` is less than `50000`). As a result, DB2 can perform a lot of predicate evaluations without decompressing the values. A query such as `select * … where C1 < 50000` using this example becomes `select * … where C1 < 10`. In this example, DB2 could filter out the values that are greater than `50000` without having to decompress (materialize) the data. Quite simply, the fact that DB2 only has to decompress qualifying rows is a tremendous performance boost. Notice how instead of decompressing all the data to see if it matches the predicate (`< 50000`), which could require decompressing billions of values, BLU Acceleration will simply compress the predicate into the encoding space of the column. That's just cool!

The final tenet of this big idea pertains to the actual encoding: BLU Acceleration takes the symbols' bits and packs them as tightly as possible into

*vectors:* collections of bits that match (as closely as possible) the width of the CPU register; this is what you see on the right side of Figure 3-2 (although the figure is intentionally oversimplified and intended to not reveal all of our secrets, we think you'll get the gist of what we're doing). This is a big deal because it enables DB2 to flow the data (in its compressed form) into the CPU with *maximum* efficiency.

To sum up, all of the big idea components in this section will yield a synergistic combination of effects: better I/O because the data is smaller, which leads to more density in the RAM, optimized storage, and more efficient CPU because we're operating on the data without decompressing it and packing that data in the CPU "registered aligned." This is all going to result in much better (we'll call it blistering) performance gains for the right workloads.

## Big Idea Number 3: Multiplying the Power of the CPU

The third big idea has its genesis in the exploitation of a leading-edge CPU technology found in today's modern processors: *Single Instruction Multiple Data* (SIMD). SIMD instructions are low-level CPU instructions that enable you to perform the same operation on multiple data points at the same time. If you've got a relatively new processor in your laptop or home computer, chances are it's SIMD-enabled, because this technology was an inflection point for driving rich multimedia applications. For example, you'll find SIMD exploitation for tasks such as adjusting the saturation, contrast, or brightness in your photo-editing applications; adjusting the volume in digital audio applications; and so on. DB2 10.5 with BLU Acceleration will auto-detect whether it's running on an SIMD-enabled CPU (for example, a qualifying Power or Intel chip) and *automatically* exploit SIMD to effectively multiply the power of the CPU. Automation is a recurring theme in DB2; it automatically detects and senses its environment and dynamically optimizes the runtime accordingly. For example, DB2 pureScale will detect whether your storage subsystem is enabled with SCSI3 PR and use an alternative time-to-recovery technology with faster fencing of disks if it detects this kind of storage at engine start time, DB2 can tell that it's running in a Power environment and exploit decimal floating-point arithmetic on the processor core for specific math functions; and during data prefetch, DB2 can detect the

kind of prefetching that's happening and adjust its prefetch accordingly. These are just a handful of the many examples where DB2 operates as a sensory mechanism where it can dynamically detect and react to its environment—without DBA intervention—for optimal results.

Now, you might be thinking that SIMD exploitation is like hyper-threading, in which the server is "tricked" into thinking that it has more cores, but that's not it at all. Multithreading actually weaves work items between pauses, so although it's definitely an optimization, SIMD is something completely different. Because SIMD instructions are very low-level specific CPU instructions, DB2 can use a single SIMD instruction to get results from multiple data elements (perform equality predicate processing, for example) as long as they are in the same register. DB2 can put 128 bits into an SIMD register and evaluate all of that data with a single instruction, and this is why this big idea is so important: Being able to compress data as much as possible, compress the most commonly occurring data, and then optimally pack it at register widths reveals the synergy that lies at the core of this big idea. You can also run DB2 with BLU Acceleration on a server that doesn't include SIMD-enabled CPUs; you just won't realize the full benefits of what we've done. That said, even if your server isn't SIMD enabled, DB2 can "fake out" the CPU to provide some of the SIMD benefits. For example, BLU Acceleration is supported on Power 6, but SIMD isn't supported on that platform; instead, when BLU Acceleration is used on Power 6, it will emulate hardware SIMD with software SIMD (using bitmasking to achieve some parallelism) to deliver some of the SIMD benefits, even when the CPU does naturally support it.

We'll use Figure 3-3 to illustrate the power of SIMD using a scan operation that involves a predicate evaluation. (Although Figure 3-3 shows a predicate evaluation, we want you to be aware that SIMD exploitation can be used for join operations, arithmetic, and more—it's not just a scan optimization.) On the right side of this figure, we show four column values being processed at one time—this is only for illustration purposes, as it's quite possible to have more than four data elements processed by a single instruction with this technology.

Contrast the right side of Figure 3-3 with the left, which shows an example of how predicate evaluation processing would work if we didn't take the time to engineer BLU Acceleration to automatically detect, exploit, and optimize

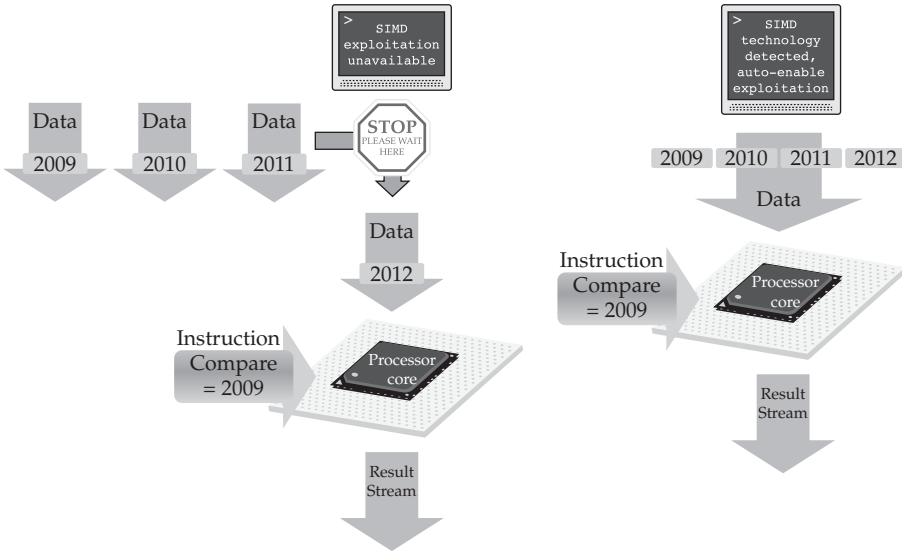


**Figure 3-3** *Comparing predicate evaluations with and without SIMD in a CPU packing optimized processing environment like DB2 10.5 with BLU Acceleration*

SIMD technology, or implement big idea number 2 to optimally encode and pack the CPU register with data. In such an environment, instead of optimally packing the register width, things just happen the way they do in so many competing technologies. Each value is loaded one at a time into its own register for predicate evaluation. As you can see on the left side of Figure 3-3, other data elements queue up for predicate evaluation, each requiring distinct processing cycles.

When the same operation is executed in a DB2 with BLU Acceleration environment (the right side of Figure 3-3), the efficiency is obvious. We like to think of it as a four-for-one sale: DB2 can get multiple results from multiple data elements with a single instruction, thereby multiplying the performance of these operations. Of course, in a DB2 environment, there's also no wasted runtime cycles spent aligning the data to pack the CPU register.

Now imagine that your query is matching a primary key to a foreign key—it could use SIMD for grouping or mathematical operations, and more. These kinds of operations are critical in exactly the kinds of workloads for which BLU Acceleration was designed: scanning, joining, grouping, and so on.

In summary, big idea number 3 is about multiplying the power of the CPU for the key operations typically associated with analytic query processing, such as scanning, joining, grouping, and arithmetic. By exploiting low-level instructions available on modern CPUs and matching that with optimizations for how the data is encoded and packed on the register (big idea 2), DB2 literally multiplies the power of the CPU: A single instruction can get results on multiple data elements with relatively little runtime processing.

At this point, our big ideas compress and encode the data, then pack it into a set of vectors that matches the width of the CPU register as closely as possible. This literally gives you the biggest "bang" for every CPU "buck" (cycles that the server consumes); we literally squeeze out every last drop of CPU power. Have you heard the expression "squeezing blood from a stone"? In most cases, that's not a good thing, but with your hardware and software investments, it's a very good thing.

## Big Idea Number 4: Parallel Vector Processing

Multicore parallelism and operating on vectors of data with extreme efficiencies when it comes to memory access is a very powerful idea about actually delivering on something in which you've invested for a long time: exploiting the growing number of processing cores on a server's socket.

Central to this big idea is that we want DB2 to be extraordinarily diligent in leveraging the processing cores that are available on a server. Where should data be placed in a cache that will be revisited? How should work be shipped across sockets? Now we know what you're thinking: "What's the big deal here? Every vendor says that they've been exploiting multicore parallelism for years." Our response to such a question? Sure. Everyone *says* they do it, but do they really? Consider your own personal computing environment. For all the cores that you have in your system, don't you find it odd that performance seems to be getting worse? And as applications get more feature rich and require more computing resources, the question that you really should be asking yourself is: "Was my software specifically written to parallelize operations?" Hint: This is not an easy thing to do, especially well, so ensure you look past the veneer of typical vendor claims.

In our experience, almost every vendor that we can think of has had only a moderate amount of success using core-friendly parallelism. For analytics parallelism, one approach has been to create a massively parallel processing

(MPP) shared-nothing database cluster. The PureData System for Analytics (powered by Netezza technology) leverages this topology; so does the DB2 Database Partitioning Feature (DPF)—the technology behind the PureData System for Operational Analytics—as does Teradata's architecture (which strongly resembles DB2 DPF). In such environments, you're able to distribute a query across a cluster of machines. That's been the prescription for driving parallelism in very large databases (VLDBs), and it will *continue* to be so.

But think about smaller warehouses and marts that contain raw data in the 10TB or less range, which could reach 30TB in a fully loaded warehouse when you include indexes, aggregates, and so on (even higher if that data isn't compressed, perhaps over 100TB). Why can't they fit into a single system? Well, in a shared-everything, single-server system (and for increased efficiencies in an MPP cluster where more and more CPU cores are available on an individual node), processing occurs in a single slice of memory. The entire system is sharing the same heaps and RAM buffer allocations. What you typically find is that when you scale within the system from one to two threads, the results are generally pretty good; with two to four threads, they are not too shabby. Above that, you have diminishing returns—in fact, in our client testing observations of commonplace in-market databases, we found that parallelism trails off pretty badly.

Why? Because of the physics of the system, which suggests that as you scale across sockets, and as your data starts to exceed what's sitting in the level 2 (L2) cache, you start to incur more and more latency. That means it's taking more time for the CPU to access data in memory. Servers try to minimize this latency through CPU advancements. For example, prefetching the data and trying to keep it in the L2 cache versus L3 cache, which is an order of magnitude (or more) faster to access than data in RAM. But the hardware can only guess at what data to prefetch and what data to keep in these fast caches.

There's also the concept of *false sharing,* which occurs when memory is updated in one L1 cache, causing the corresponding L1 cache of another CPU to be invalidated. Once the L1 cacheline is invalidated, all of the data in that cacheline needs to be refreshed, causing further delays for that CPU. Only extremely careful programming techniques can help avoid this problem. The more sockets and memory channels on the server, the worse this problem gets.

These are just some of the reasons why single systems traditionally don't scale well for analytics beyond four- to eight-way parallelism, and that's part of the reason why almost every BI vendor has a shared-nothing architecture. You may never have heard of some of these problems. Indeed, they're pretty esoteric. That's exactly why they are hard to solve. This is why we call memory the new disk. (Traditionally, while you performance tuned your database, if a query spilled to disk, it would typically slow down performance. However, in BLU Acceleration, we kind of raise it up a level and view "query spills to in-line memory out of the CPU caches" as a performance inhibitor and that's what we mean when we say "memory is the new disk.")

So, what if your software could do better? We talked about making analytics super-easy in our first big idea. What if BLU Acceleration could change the inflection point where you need to leverage MPP technologies? Would that keep costs down and make it easier? If you could fit your marts on a single-server solution and not have to manage a cluster, wouldn't that make things easier and you can still leverage the power of MPP scale-out when needed? We believe that your existing server has the power to shift the "when to scale out or scale up" inflection point because the processing power is already there. The problem is that most analytic database engines haven't been written to fully exploit those resources—we mean to *really* exploit those resources.

In DB2 with BLU Acceleration, we've done a lot of hard work so that you can truly "Kill It with Iron" (KIWI). KIWI starts with the premise that we know that the CPU core counts on today's (and tomorrow's) servers are going to continue to increase. We wanted to ensure that BLU Acceleration is available where it can be used the most and to keep things very easy. (Remember our *Super Analytics, Super Easy* tagline?) The secret for us has been to realize that memory is the new disk. For BLU Acceleration, main memory access is too slow. It is something to be avoided. BLU Acceleration is designed to access data on disk very rarely, access RAM only occasionally, and do the overwhelming bulk of its processing from data and instructions that reside in a CPU cache. Super easy from an administrative perspective means within the confines of one server, which means we aren't going to ask DBAs to divide their data into logical partitions. To do this, we had to engineer DB2 to pay very careful attention to CPU cache affinity and memory management so that the majority of memory access occurs in a CPU cache and not by accessing data from RAM over and over again. By operating almost exclusively on data in a CPU cache

and not in RAM, BLU Acceleration minimizes the "latency" and is able to keep your CPUs busy. We've seen a lot of benefit from the work we've done in this area, and have personally seen excellent scalability between 32- and 64-way parallelism—where even a single query can exploit 64 cores in shared memory with almost perfect scalability. It just gets easier as workload concurrency increases with more queries running at once.

In summary, big idea number 4 recognizes that servers have an increasingly larger numbers of cores. DB2 with BLU Acceleration is designed from the ground up to take advantage of the cores that you have and to always drive multicore parallelism for your queries. This is all done in shared memory—it is not DPF parallelism. Our focus on efficient parallel processing with memory-conscious algorithms enables us to fully utilize the power of multiple CPU cores.

## Big Idea Number 5: Get Organized…by Column

Big idea number 5 revolves around something that's become pretty trendy (despite being around for a long time) and commonplace in the last couple of years: *column store.* Essentially, the idea is to bring all of the typical benefits attributed to columnar stores, such as I/O minimization through elimination, improved memory density, scan-based optimizations, compelling compression ratios, and so on, to DB2. Again, we ask you to consider that what makes BLU Acceleration so special isn't just that it's an in-memory column store; rather, it's how we implemented it as a *dynamic* in-memory column store with the other big ideas.

### What Is a Column Store?

We'll assume you've got the gist of what a column-organized table is because there are so many technologies using this approach in today's marketplace, so we won't spend too much time describing it. To ensure that we are all on the same page, however, Figure 3-4 shows a simplification of a row-organized table (on the left) and a column-organized table (on the right). As you can see, a column-organized table orients and stores its data by column instead of by row. This technique is well suited to specific warehousing scenarios.

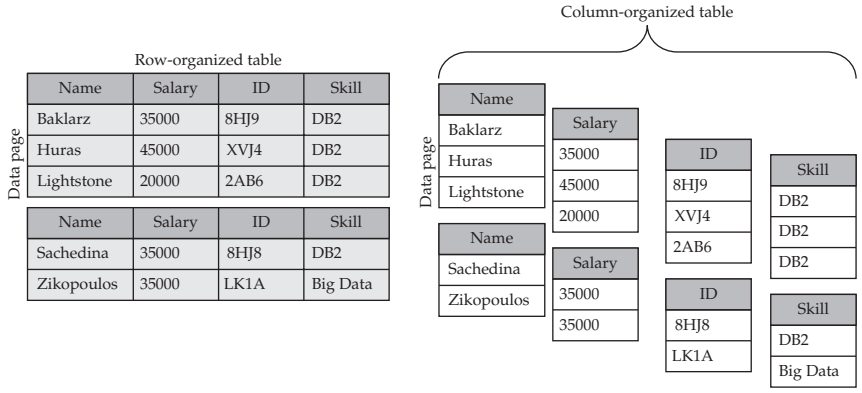


**Figure 3-4** *Comparing row-organized and column-organized tables*

With a columnar format, a single page stores the values of just a single column, which means that when the database engine performs I/O to retrieve data, it just performs I/O for only the columns that satisfy the query. This can save a lot of resources when processing certain kinds of queries.

On disk, you see a single page dedicated to a particular column's data. Because in an analytics environment data (because there's a high probability of repeated data on a page—like the `Skill` column) within a column is more self-similar, we've observed better compression ratios with column stores than with traditional row-store compression technology. That said, the DB2 compression technology that's available for row organized tables has been seen to achieve levels as good as columnar for many environments, and unlike columnar, is very well suited for transactional environments too. This is what makes DB2 so special: All this stuff is used side by side, shared in many cases, complementary, and integrated.

## Columnar Is Good—BLU Acceleration Engineering Makes It Great

BLU Acceleration uses columnar technology to store a table on disk and in memory. However, by combining that with all the big ideas detailed so far, we think it gives DB2 10.5 big advantages over our competitors.

By using a column store with encoding, DB2 is able to get an additional level of compression that leads to even more downstream I/O minimization. In addition, recall that DB2 is able to perform predicate evaluation (among other operations) against compressed data by using its actionable compres-

sion capabilities, and that even further reduces the amount of I/O and processing that needs to be performed.

Of course, overall I/O is reduced with columnar technology because you only read what the query needs. This can often make 95 percent of the I/O go away because most analytic workloads access only a subset of the columns. For example, if you're only accessing 20 columns of a 50-column table in a traditional row store, you end up having to do I/O and consume server memory even for data in columns that are of no interest to the task of satisfying the query.

With BLU Acceleration column-organized tables, the rows are not accessed or even "stitched" back together until absolutely necessary—and at the latest possible moment in a query's processing, for that matter. In most cases, this will occur when the answer set is returned to the end user, or when you join a column-organized table with a row-organized table. This process of combining the columns as late as possible is called *late materialization.*

Let's use a simple example to explain why this helps. Consider a query on a 20-column table that has 1 billion rows; the query has predicates on columns A, B and C, and 1 percent of each column's data satisfies these predicates. For example:

```
select A, B, C from REGIONSALES where A < 10 and B < 400 and C = USA
```

In a row store, this kind of query would only run well if the user had designed system indexes on columns A, B, and C. Failing that, all 1 billion rows would need to be read. In a column store that has late materialization, such as BLU Acceleration, the database starts by scanning only the column that's likely to be the most selective (has the fewest qualifying values). Let's assume that after scanning the first column, only 1 percent of the data remains. (Remember that BLU Acceleration can scan and operate on compressed data without needing to decompress it.) Next, DB2 accesses the second column, but only those qualifying 1 percent of records that survived the first column scan are looked at, leaving 1 percent of 1 percent (0.01 percent) of data that passes through the engine; finally, the third column is processed and that leaves just 1 percent of 1 percent of 1 percent (0.0001 percent) of the original data set.

Now consider just how much data was really accessed to satisfy this query. You can see that the total data accessed is dominated by the initial column scan, which is just 1/20th of the table's data (1 of the 20 columns), and it didn't even require decompression, and this is why it pays to access the columnar data as late in the processing as possible—as the query progresses, the data gets increasingly filtered.

To sum up, column store is *a part of* BLU Acceleration. DB2 with BLU Acceleration leverages columnar technology to help deliver minimal I/O (performing I/O only on those columns and values that satisfy the query), better memory density (column data is kept compressed in memory), and extreme compression. However, when it's mixed with other big ideas, such as packing the data into scan-friendly structures, and more, BLU Acceleration really sets itself apart from the rest of the industry.

## Big Idea Number 6: Dynamic In-Memory Processing

Dynamic in-memory processing covers a set of innovative technologies to ensure that BLU Acceleration is truly optimized for memory access, but not limited to the size of RAM—just in case your active data is (or may one day become) bigger than the buffer pool. We like to think of BLU Acceleration as being better than in-memory processing. Two major technologies are at play in this concept: *scan-friendly memory caching* and *dynamic list prefetching*.

Scan-friendly memory caching is a powerful idea, which, to the best of our knowledge, is unique to BLU Acceleration. Effectively caching data in memory has historically been difficult for systems that have more data than memory. Many database systems have memory caches, but they tend to work great for transaction processing and not very well for analytic workloads. That's because their algorithm's access patterns are radically different. Transaction-processing systems process a lot of random access patterns on highly skewed data patterns because in these environments, some data is much hotter than other data. For those workloads, databases have traditionally used variations of the Least Recently Used (LRU) and Most Recently Used (MRU) paging algorithms, which had the effect of keeping the data used most recently in memory (the MRU algorithm) while evicting the old data (the LRU algorithm).

Now it's fair to note that analytic workloads have some elements of skew as well. For example, recent data (within the past month) is usually hotter than older data (often referred to as cold—perhaps in your environment, that's five years old or more). That said, for analytic workloads, the pattern of data access is much more egalitarian (it's likely not to favor a set of pages, thereby creating a "hot" page) within the active data set, as queries are bound

to touch a lot of data while performing scans, aggregations, and joins. Egalitarian access presents quite the challenge for a database system; after all, how can you optimally decide what to keep in memory when all pages in a range are more or less equal? This presents a conundrum: The database manager wants to have some elements of the traditional caching approach that favors hot data, but also requires a new approach that recognizes that even the hot data may be larger than RAM and have fairly even access patterns. With BLU Acceleration, not only can the data be larger than allocated RAM, what's more, DB2 is *very* effective at keeping the data in memory, which allows for efficient reuse and minimizing I/O; this is true even in cases where the working set and individual scans are larger than the available memory. DB2's scan-friendly memory caching is an automatically triggered cache-replacement algorithm that provides egalitarian access, and it's something new and powerful for analytics. Our tagline for this is "DB2 performance doesn't fall off a cliff when the query's data requirements are bigger than the RAM."

Consider what happens when a database tries to use a transactional algorithm to decide what to keep in memory with heavy scan-based analytic workloads. An analytic query in such a workload is likely to start at the top of the table and scan its way to the bottom. However, with a transactional optimized page-cleaning algorithm, by the time the scanner gets to the bottom of the table, the least recently used pages that were read at the start of the scan become the victims. In other words, because the table scan is reading so many pages into the buffer pool, some of the required pages are going to get flushed out. The next time a scan begins on this table, the incoming query is going to start scanning the pages at the beginning of the table and find that the target pages are no longer in memory; what a shame, because this will not only trigger I/O, but lots of I/O. This scenario plays itself out continually because such a database is trying to use an algorithm for cache replacement that was designed fundamentally for transaction processing and use it for analytics.

The *analytics-optimized page replacement* algorithm that's associated with BLU Acceleration assumes that the data is going to be highly compressed and will require columnar access, and that it's likely the case that all of this active data (or at least 70 to 80 percent of it) is going to be put into memory.

When surveying our clients, we found that the most common memory-to-disk ratio was about 15 to 50 percent. Assuming a conservative 10-fold

compression rate, there's still a high probability that you're going to be able to fit most (if not all) of your active data in memory when you use DB2 10.5 with BLU Acceleration. But while we expect most, if not all, of the active data to fit in memory for the majority of environments, *we don't require it*. When DB2 accesses column-organized data, it'll automatically use its scan-friendly memory-caching algorithm to decide which pages should stay in memory in order to minimize I/O, as opposed to using an algorithm based on LRU, which is good for OLTP and not as optimized for analytics.

What makes this big idea so unique is that DB2 automatically adapts the way it operates based on the organization of the table (row-organized or column-organized) being accessed. Remember the DBA doesn't have to do anything here; there are no optimization hints to give, configuration parameters to set; it just happens automatically.

The second aspect of dynamic in-memory processing is the new prefetch technology called dynamic list prefetching. The prefetching algorithms for BLU Acceleration have been completely redesigned for its columnar parallel vector processing engine. These algorithms take a very different approach because BLU Acceleration doesn't have indexes to tell it what pages are interesting to the query (list prefetching), which would be a common case with a row-organized table. Of course, DB2 could simply prefetch every page of every column that appears in a query, but that would be wasteful, as we saw with the previous late materialization example where each predicate filtered 99 percent of the remaining data. With 0.0001 percent of the data qualifying, prefetching 100 percent of the data for all columns would be a massive waste of I/O. BLU Acceleration addresses this challenge with an innovative strategy to prefetch only a subset of pages that are interesting (from a query perspective), *without* the ability to know far in advance what they are. We call this *dynamic list prefetching* because the specific list of pages can't be known in advance via an index.

To sum up, remember that one of the special benefits of BLU Acceleration in comparison to traditional in-memory columnar technologies is that performance doesn't "fall off the cliff" if your data sets are so large that they won't fit into memory. If all of your data *does* fit into memory, it's going to benefit you, but in a Big Data world, that isn't always going to be the case, and this important BLU Acceleration benefit should not be overlooked.

## Big Idea Number 7: Data Skipping

The seventh (but not final! …we don't like saying final because more are coming as we enhance this technology in subsequent releases) big idea that inspired BLU Acceleration is *data skipping*. The idea is very simple: We wanted DB2 to be able to skip over data that's of no interest to the active query workload. For example, if a query was to calculate the sum of last month's sales, there's no need to look at any data that's older than last month that's sitting in the warehouse. With data-skipping technology, DB2 10.5 can *automatically* skip over the nonqualifying data because it keeps metadata that describes the minimum and maximum range of data values on "chunks" of the data. This enables DB2 to automatically detect large sections of data that don't qualify for a query and to effectively ignore them. Data skipping can deliver an order of magnitude in savings across compute resources (CPU, RAM, and I/O). Of course, in keeping with the "*Super Easy*" part of the BLU Acceleration mantra we've been chanting throughout this book, this metadata is automatically maintained during `INSERT`, `UPDATE`, and `DELETE` activity, so you don't have to worry about defining or maintaining it.

BLU Acceleration's data skipping is conceptually similar to the Zone Map technology found in the PureData System for Analytics (formerly known as Netezza) family. We say similar because unlike Zone Maps, this metadata isn't tied to any particular page or extent boundary—it's tied to a certain "chunk" of data records (about 1,000). Because data skipping allows a query to skip over ranges of uninteresting data, DB2 is able to avoid touching this data, whether it's on disk or in memory, during query execution.

DB2's data-skipping metadata is actually kept in a tiny column-organized table called a *synopsis table*, which is automatically maintained, loaded, and persisted into memory when needed. This metadata is extremely valuable because it empowers DB2 to save precious compute resources by enabling the database engine to intelligently and accurately skip over data that's not needed; this saves I/O and keeps the system's memory filled with useful active data rather than filling it up with data that the query doesn't actually need. Of course, all this means that more memory is available for the important data you really want to run analytics on.

## Seven Big Ideas Optimize the Hardware Stack

If you've spent any amount of time performance tuning a database, you know that there are three things you always care about: memory, CPU, and I/O. You iterate through various approaches to remove bottlenecks in any of these hardware attributes to find higher and higher levels of performance.

The driving force behind many of the amazing results coming from BLU Acceleration technology was a relentless focus on hardware utilization. (BLU Acceleration aside, DB2 10.5 contains lots of technologies for enhanced exploitation of memory, processors, and I/O, which leads to lower total cost of ownership, better resource utilization, better performance, and so on.)

Figure 3-5 presents a summary of the seven big ideas from the perspective of squeezing every last ounce of compute resources from your hardware stack.

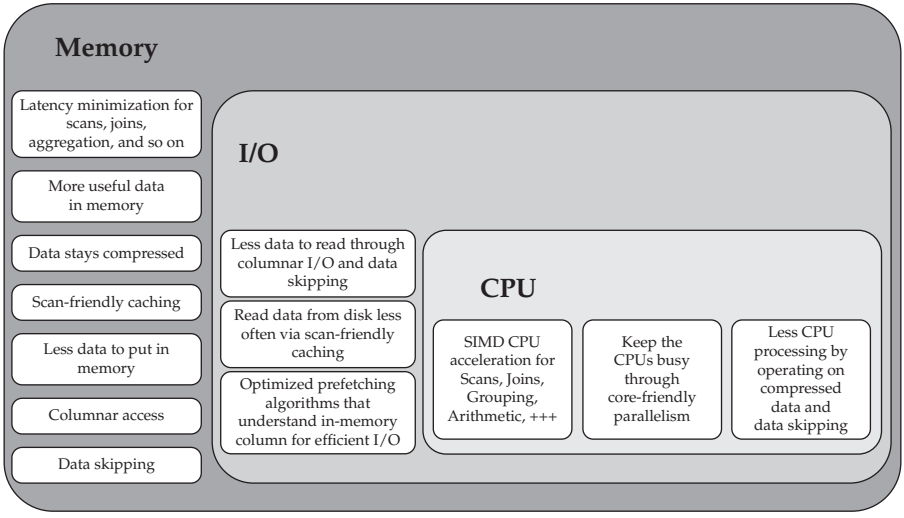


**Figure 3-5** *Looking at BLU Acceleration's seven big ideas from a compute resource perspective: the hardware stack*

# When Seven Big Ideas Deliver One Incredible Opportunity

Let's see how these diverse ideas have been integrated into a single offering by working through a typical example in analytics processing. As always, "your mileage will vary," depending on the kind of data you're storing, your server, the queries you're running, and so on.

With that said, let's start with the assumption that you have a 32-core server and you've got 10 years of data (2004 to 2013) that's ready to be loaded into a relatively wide (100-column) table. Before loading the data, the on-disk *raw* footprint of this data is 10TB. An analyst wants to run a simple query that counts all of the new customers that were added to your company's loyalty program through the various campaigns (Web, mailings, and so on) run in a specific year. Using a favorite BI tool, such as Cognos BI, and without any idea what BLU Acceleration is, the analyst composes the following query: `select count(*) from LOYALTYCLIENTS where year = '2012'`. Our goal? To provide the analyst with subsecond response times from this single nonpartitioned, 32-core server *without* creating any indexes or aggregates, partitioning the data, and so on.

When we tabled this scenario with our testing team (without mentioning the BLU Acceleration technology), they laughed at us. We did the same thing in front of a seasoned DBA with one of our biggest clients, and she told us, "Impossible, not without an index!" Figure 3-6 shows how the seven big ideas worked together to take an incredible opportunity and turn it into something truly special.

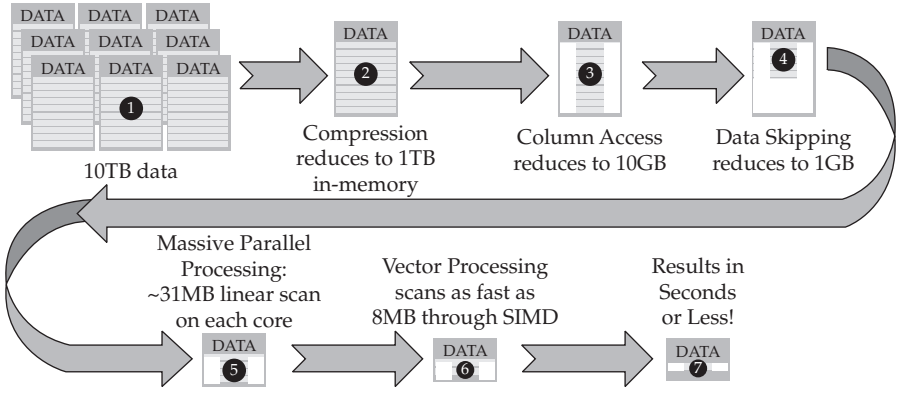


**Figure 3-6** *Watching how some of the seven big ideas that went into BLU Acceleration manifest into incredible performance opportunities*

The example starts with 10TB of data (❶) that's sitting on a file system waiting to be loaded into a DB2 database with BLU Acceleration. Although we've seen much higher compression ratios, in many cases, we saw, on average, an order of magnitude (10x) reduction in the raw data storage requirements

when just the BLU Acceleration encoding and compression techniques were taken into account, so now we have 1TB (❷) of data. Note that there aren't any indexes or summary tables here. In a typical data warehouse, 10TB of raw data is going to turn into a 15- to 30-TB footprint by the time these traditional kinds of performance optimizations are taken into account. In this example, the 10TB of data is *raw* data. It's the size before the data is aggregated, indexed, materialized, and so on. When it's loaded into a BLU Acceleration table, that 10TB becomes 1TB.

The analyst's query is looking only for loyalty members acquired in `2012`. YEAR is just one column in the 100-column LOYALTYCLIENTS table. Since DB2 needs to only access a single column in this column-organized table, you can therefore divide the 1TB of loaded data by 100. Now we're down to 10GB (❸) of data that needs to be processed. However, BLU Acceleration isn't finished applying its seven big ideas yet!

While DB2 is accessing this 10GB of data by using its columnar algorithms, it also applies the data-skipping big idea to skip over the other nine years of data in the YEAR column. At this point, *DB2 skips over the nonqualifying data without any decompression or evaluation processing.* Now we're down to 1GB (❹): DB2 is now left solely looking at a single column of data, and within that column, a single discrete interval. Thanks to scan-friendly memory caching, it's likely all of that data can be accessed at main memory speeds. Now DB2 with BLU Acceleration takes that 1GB and parallelizes it across the 32 cores on the server (❺), with incredible results because of the work that was done to implement the fourth big idea: parallel vector processing. This means that each server core only has to work on about 31MB of data (1GB = 1000MB/32 cores = 31.25MB). *DB2 is still operating on compressed data at this point, and nothing has been materialized.* It's really important to remember this because all of the CPU, memory density, and I/O benefits still apply.

DB2 now applies the other big ideas, namely actionable compression (operating on the compressed data, carefully organized as vectors that match the register width of the CPU) and leveraging SIMD optimization (❻). These big ideas take the required scanning activity to be performed on the remaining 32MB of data and make it run several times faster than on traditional systems. How fast? We think you already know the answer to that one—*it depends.* The combined benefits of actionable compression and SIMD can be very profound. For the sake of this illustration, let's say the speed-up over traditional

row-based systems is 4 times faster per byte (we think it's often much higher, but we are being conservative—or rather being told to be). With this in mind, the DB2 server, from a performance perspective, only has to scan about 8MB (~32MB/4 speedup factor = 8MB) of data compared to a traditional system. Think about that for a moment. Eight megabytes is about the size of a high-quality digital image that you can capture with your camera. A modern CPU can chew through that amount of data in less than a second…no problem. The end result? We accepted a seemingly impossible challenge on 10TB of raw data and were able to run a typical analytic query on it in less than a second using the application of seven big ideas (❼).

## Under the Hood: A Quick Peek Behind the Scenes of BLU Acceleration

BLU Acceleration comes with a number of DB2 optimizations that provide a platform for the incredible results we've detailed in this chapter. There's a "knob" to run DB2 in analytics mode, and automatic workload management provides unlimited concurrency without your having to worry about resource pressures. There's a new capability around uniqueness that lets you avoid storage and overhead for unique indexes when data is trusted, a utility to easily convert row-organized tables to column-organized tables, a query workload advisor that suggests how to organize a table based on a query set, and much more.

In this section, we discuss some of the things that go on "behind the scenes," as well as some higher-level details on how things work in a BLU Acceleration environment. (If you're not a propeller head, you can stop reading here—the remainder of the chapter may not interest you; that said, by all means, feel free to read on.)

### BLU Acceleration Is a New Format for the Bytes in a Table, NOT a New Database Engine

Because BLU Acceleration is part of the DB2 DNA, its developmental context is not so much that of a new product, but one that draws benefits from the maturity and stability of a decades-long robust, proven, and familiar DB2 kernel. In fact, considerably less than 1 percent of the DB2 engine code

was changed to bring BLU Acceleration to market because the process model stayed the same, the buffer pools stayed the same, as did the locking structures, package cache, recovery, and much more. And because BLU Acceleration is seamlessly built into the DB2 kernel, from an external perspective, the technology all looks the same. But what does this have to do with stability and maturity? Over the last 20 years, our labs have built up a mind-boggling collection of test cases that we run against every new release of DB2. We didn't have to write a lot of new QA for BLU Acceleration; we just ran the thousands of existing test cases and bucket regression tests against tables using BLU Acceleration with the result that this is, by far, the most test cases and coverage that we've ever had for a new feature. Quite simply, a BLU Accelerated table is just a new database object that interfaces with the rest of DB2 in more or less the same manner as a conventional row-organized table did before, and this mitigates risk for those folks who might be tempted to view this as completely new technology.

How much a part of DB2 is BLU Acceleration? Well, relatively early in the project, one of us was writing code for BLU Acceleration's integration into the DB2 kernel. He wondered whether he could get `BACKUP` to work. After all, why wouldn't it work? These tables are just using regular DB2 pages. Although this was very early code, it ran successfully. Looking at the backup image, sure enough, all of the data pages were there. Of course, the most important backup image is the one you can restore (always, always test recovery), so he decided to restore the data. That, too, worked! Even a query against the restored tables worked. He didn't have to change a single line of code to get this scenario working in the prototype, one of the tangible benefits of building BLU Acceleration right into the DB2 kernel.

## The Day In and Day Out of BLU Acceleration

From a Data Definition Language (DDL) perspective, when you create a column-organized table, you just specify `ORGANIZE BY COLUMN` instead of `ORGANIZE BY ROW`. (You can use the `DFT_TABLE_ORG` database configuration parameter to set the default table organization in lieu of specifying the `ORGANIZE BY COLUMN` clause on the `CREATE TABLE` statement.) If `DB2_WORKLOAD=ANALYTICS`, the `DFT_TABLE_ORG` value is set to `COLUMN`. Uniqueness and nullability are enforced, just as they are in a row-organized

DB2 table, but there aren't options for compression with BLU Acceleration because compression is always on. You still access the `SYSCAT.TABLES` view to look at table metadata; we just added a new column (`TABLEORG`) with `R` (row organized) and `C` (column organized) descriptors. As you can see, working with BLU Acceleration objects is pretty much the same as other objects and so we cut this section short.

## Informational Constraints, Uniqueness, and BLU Acceleration

Informational constraints, first introduced in DB2 9, provide DBAs with a mechanism to inform the DB2 optimizer about certain data characteristics without having to enforce them in the database. For example, if your data set has two discrete values for gender (`M|F`) *and the data has come from a trusted source,* you could define this as a `NOT ENFORCED` informational constraint and bypass the database manager overhead and on-disk footprint associated with enforcing such a business rule. However, the DB2 optimizer could still take advantage of understanding the data distribution during query rewrite or access plan generation.

*Informational uniqueness* is the enhancement to informational constraints that debuts in the DB2 10.5 release in support of BLU Acceleration. Imagine if you could tell the database manager that a row is unique without requiring enforcement. DB2 would save compute resources but still be able to generate superior query execution plans. This enhancement was delivered in support of BLU Acceleration because unique indexes (needed to enforce uniqueness) don't really compress that well (compression is about the processing of repeating patterns, not unique ones), they occupy considerable space, and they must be maintained. Informational uniqueness is an optional optimization—the *default* in DB2 10.5 is to *still enforce* uniqueness. If you're going to use this optimization, and there's no reason not to—in fact, we recommend it if applicable—just *be sure that your data is truly unique.*

The following code snippets show you how to define informational uniqueness (or primary key) constraints on a new or existing table:

```
CREATE TABLE t1 (c1 INTEGER NOT NULL,
  c2, INTEGER, PRIMARY KEY (c1) NOT ENFORCED);
ALTER TABLE t1 ADD CONSTRAINT unique1 UNIQUE (c2) NOT ENFORCED;
```

## Getting the Data to BLU: Ingestion in a BLU Acceleration Environment

Data ingest is an important concept in today's analytics landscape: We can't tell you how many times we hear about companies wanting to flatten the latency of their analytics. BLU Acceleration dramatically reduces the time it takes to go from raw data sitting in some kind of flat file (such as an ASCII file) on the system to making that data queryable, a process that includes the time that's required to load the data, build covering indexes, collect and plot data statistics, compress the data, and apply other optimizations.

BLU Acceleration supports the typical ways in which you get data in and out of a DB2 table, namely `LOAD`, `INGEST`, `IMPORT`, `EXPORT`, and Data Manipulation Language (DML) operations such as `INSERT`, `UPDATE`, and `DELETE`. BLU Acceleration is heavily optimized (it comes with highly effective multicore parallelism for both `INSERT` and `LOAD` processing) to achieve rates that are similar (not identical) to the performance of these operations against row-organized tables. That said, you're likely to find that the time to load from file to query is usually faster than loading a comparable row-organized table; we estimate anywhere from 1.3x to as high as 3.7x faster when you take into consideration all the activities that are performed to make raw data performance queryable when all is said and done. Of course, your mileage is going to vary, depending on the specific characteristics of your environment. How's all of this possible? After all, aren't column stores more computationally complex to format during load processing? While it's true that getting the data into a column-organized table can require a bit more processing (mostly due to the advanced compression algorithms), the extra time spent compressing the data is more than saved by the avoidance that a row store spends building and maintaining indexes, collecting statistics, and more.

## Automated Workload Management That Is BLU-Aware

If you studied computer science, you've likely been exposed to the "movie theater" problem (and if you've ever gone shopping for Black Friday or Boxing Day Christmas-time clear-out sales, you'll have your own analogy to draw from). Think about it: A movie theater has a big entrance door that

could probably accommodate groups, four abreast, at any one time. If you had a movie theater filled with thousands of people and everybody tried to leave all at once, you'd end up with a big stampede for the door that would effectively "jam" the exit because all of the moviegoers are competing for the same resource (the door that leads out of the theater).

Now imagine a scenario in which you had the ability to instruct those moviegoers to act in a civilized manner (which would definitely be helpful during the Christmas shopping season) such that they leave the theater in an orderly fashion…four at a time. That would provide a great form of workload management for this problem domain. While bargain hunters and moviegoers likely won't listen to you, one cool thing about computers is that they do exactly what you tell them to do! The movie theater problem is the idea behind the automated workload management that's part of BLU Acceleration. The inspiration behind workload management for BLU Acceleration is you've got a server with some set number of processing cores and a certain amount of memory; let's not let everyone's queries try to fight each other for those resources all at once because we know they are going to run really fast, like really fast, so we just need to keep order and everyone will be happy.

DB2 10.5 with BLU Acceleration includes built-in and automated query resource consumption controls to deliver even higher performance to your analytic database environments. Analytic queries are typically resource-intensive and compete for CPU cycles and memory. Workload management for BLU Acceleration is yet another optimization that's automatically enabled when DB2 is running in `ANALYTICS` mode. When in this mode, the DB2 engine automatically enforces a workload management policy that regulates the number of queries that are going to be consuming resources at any given time. Specifically, DB2 10.5 automatically allows a high level of analytic query concurrency but limits the number of queries that consume resources simultaneously. When the database server is optimized for analytics concurrency, other workloads can still connect to the database server and they can still issue queries, but a finite number are going to be allowed to execute at any one time to keep the entire process running at peak efficiency.

In DB2 10.5, workload management for BLU Acceleration is implemented by creating a new default workload manager (WLM) threshold object on the database server. When this policy is created, DB2 actually fine-tunes it for the underlying hardware architecture. This auto-created policy is the same kind of

policy you could create manually using the `CREATE THRESHOLD` command in DB2. In DB2 10.5, this auto-created policy exists even if you're not using BLU Acceleration. If you set `DB2_WORKLOAD=ANALYTICS`, this policy will be enabled by default; if the database isn't in this mode, while this policy still exists, it won't be enabled and you'll have to explicitly enable it to use it. We want to note that while there are default WLM objects that you're likely aware of in previous versions of DB2 (for example, service classes), this throughput policy marks the first time we've included default concurrency control in DB2.

Before DB2 10.5, unless you explicitly created a WLM policy, by default, all queries ran in the `SYSDEFAULTSUBCLASS`. In DB2 10.5, read-oriented SQL that's over an estimated cost will be mapped to run in the new `SYSDEFAULTMANAGEDSUBCLASS`, which has a default BLU Acceleration–tuned concurrency limit that's auto-configured in consideration of the underlying hardware stack. Of course, you can augment these prebuilt workload management policies for BLU Acceleration with your own custom policies, or build all of your policies from scratch.

## Querying Column-Organized and Row-Organized Tables

One of the design points for BLU Acceleration was the seamless mixing and matching of table types (row-organized or column-organized) in the same database, table spaces, buffer pools, and queries. Mixing table types is key to some workloads and their underlying schemas. For example, point queries with highly selective index access favor row-organized tables, as do small, frequent write operations.

You should be aware that analytic queries are definitely going to run better if all the tables being accessed are column organized, because internal casting operations have to be done when you join row-organized and column-organized tables.

## The Maintenance-Free Life of a BLU Acceleration Table

When we told you that BLU Acceleration tables are virtually maintenance free, we weren't kidding; there's a lot of "automatic" stuff in here, such as automatic configuration and tuning, automatic statistics collection, automatic

workload management, and automatic space reclamation. In the area of space reclamation, there's simply no need for typical (and costly) DBA space management tasks or `REORG` operations—space is freed up while work continues.

BLU Acceleration supports automatic space reclamation. Extents with no active values are freed and subsequently returned for reuse by any table (row-organized or column-organized) in its associated table space. This is a significant benefit. If you're familiar with the pseudo delete mechanisms introduced for MDC tables in the DB2 9 release, you'll understand this process very well.

Let's work through a simple example. Consider a table with a number of key columns that stores data from different years. Over time, you want to roll out the older data and roll in new data; some folks have two-year rolling ranges, some use days or weeks, and some as much as a decade—it all depends on your business. Figure 3-7 illustrates a typical scenario in which you want to roll out a year of data from a BLU-Accelerated table named `SALES`. To do so, you would issue a SQL statement, such as: `DELETE * FROM sales WHERE year = '2012'`.

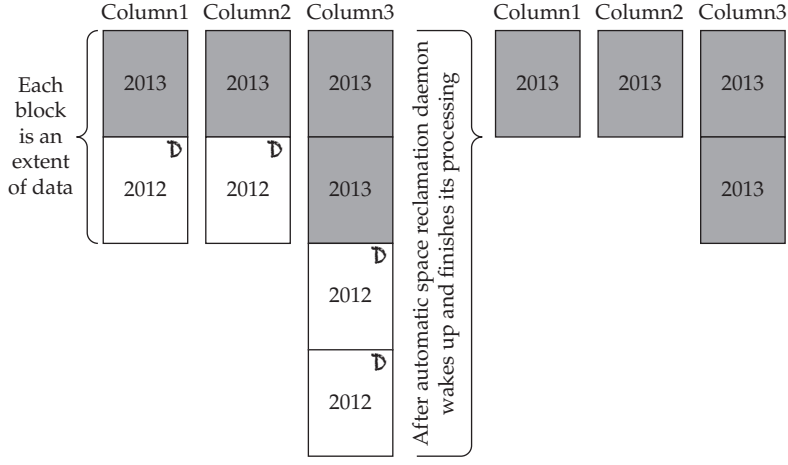


**Figure 3-7** *Space reclamation of empty extents in a BLU-Accelerated table*

In Figure 3-7, each box represents a data extent for a specific year. From a storage footprint perspective, after you run this statement, all of the data still exists on disk and is allocated to the table. But to database scanners, data associated with `2012` appears to have been deleted (these extents are tagged

with a pseudo-deletion marker, 𝔇 in Figure 3-7), and from an application perspective, this data no longer exists.

BLU Acceleration has a daemon that wakes up from time to time to search for those pseudo-deleted extents (𝔇) using DB2's automatic table maintenance (ATM), first introduced in DB2 8.2 and now extended with this capability for column-organized tables. When the daemon has done its work, those extents will have been removed from the table and returned to the table space so that they're available for reuse by any table in that table space. In our tests, the overhead from this daemon was miniscule (~1 percent), so we don't think you're going to find it disruptive; in fact, we don't even think you're going to notice it, but you'll appreciate what it does for you.

## Getting to BLU-Accelerated Tables

You'll find BLU Acceleration in any of the advanced DB2 editions, the newly announced DB2 Advanced Workgroup Server Edition (DB2 AWSE), and DB2 Advanced Enterprise Server Edition (DB2 AESE). Assuming that you've got entitlement to either of these editions (there are several nuances if you are upgrading from a previous DB2 version, but that's outside the scope of this book), and you want to keep things simple and effective, you really just need DB2 running in `ANALYTICS` mode (`DB2_WORKLOAD=ANALYTICS`) before you *Load and Go!*

It's a great idea to have DB2 set to run in `ANALYTICS` mode *before* creating the database, because DB2 will configure itself for analytic processing. For example, when a database is created in this persona, it makes column-organized tables the default table type; puts in place an in-memory, columnar-cognizant prefetching and cleaning algorithm; enables the BLU-optimized automatic workload management and automatic space reclamation features that are detailed in the last sections; configures page and extent sizes for analytics; and automatically initializes memory for caching, sorting, hashing, and utilities based on the server's detected compute resources.

We understand that it's not going to be practical for everyone to create a new database in `ANALYTICS` mode, so if you can't (or don't want to)—not to worry! All of the automation that you get when creating a new DB2 database in `ANALYTICS` mode can be enabled with a few commands in just a few

minutes. We cover this in the upcoming section "Ready, Set, Go! Hints and Tips We Learned Along the Way."

If you have some existing tables that you want to convert to BLU Acceleration tables, there's a utility (`db2convert`) to help you get there. You can run this operation online (it's based on the online table move features that first appeared in the DB2 9 time frame). You can access this utility through the command line or the Optim Data Studio management toolset (select the **Migrate to Column Storage** option when right-clicking a row-organized table) as shown in Figure 3-8.

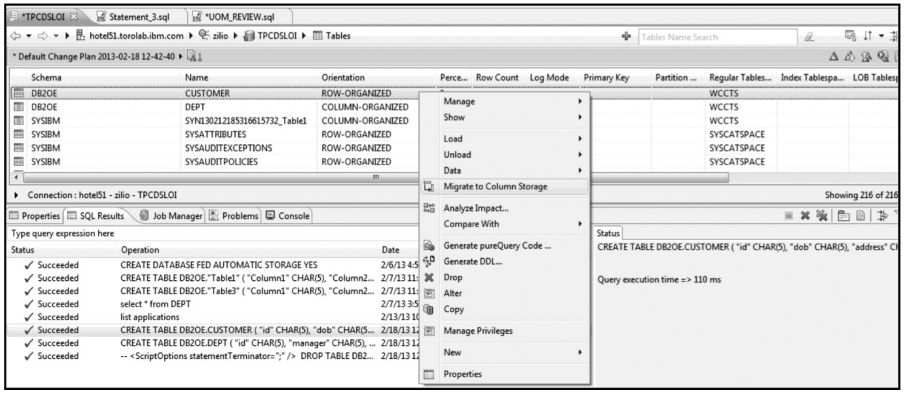


**Figure 3-8** *Space reclamation of empty extents in a BLU-Accelerated table*

Note that there are no utilities or tools that support the conversion of column-organized tables to row-organized tables. If you want to manually convert a table in this manner, you need to unload the column-organized data and then reload it into a new row-organized table.

Optim Query Workload Tuner has been extended such that it can be used to examine your workload and suggest whether a column-organized table is likely to enhance performance. An example is shown in Figure 3-9. Notice how if you select recommendations for Table Organization the options for additional optimizations aren't available. That's because these technologies aren't needed if the table was recommended as column-organized. If the advisor suggests organizing the table by row, you could then rerun the advisor and select these other options and deselect the Table Organization option.

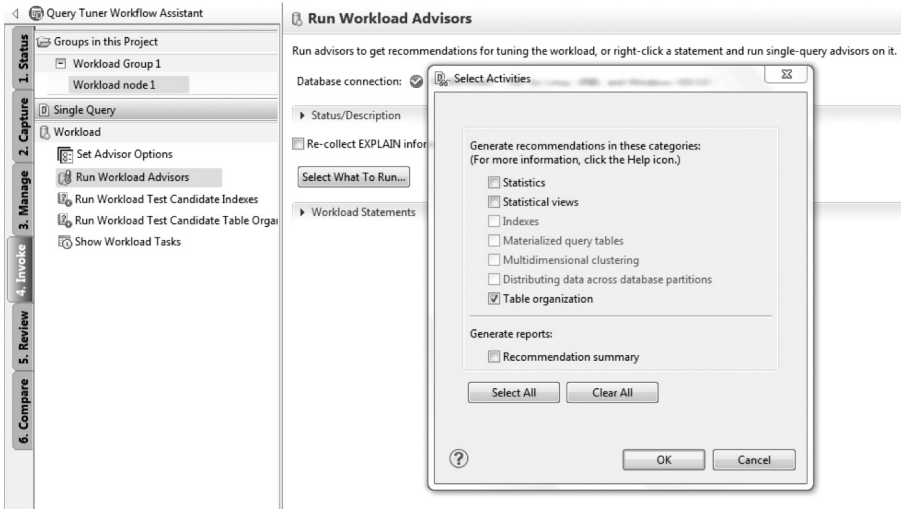


**Figure 3-9** *Optim Query Workload Tuner includes the ability to evaluate captured query sets and recommend BLU Acceleration tables for performance enhancements.*

# Ready, Set, Go! Hints and Tips We Learned Along the Way

We believe that BLU Acceleration is such an incredible market-leading move forward, we strongly recommend trying it out (assuming that your environment meets the criteria that we've outlined in this chapter) before trying other partitioning strategies or approaches to get your analytical marts to scale. In this final section, we leave you with some of the tricks and tips that we've learned along the way. The hints and tips in this section come from some of our own observations while we were writing this book, and working with customers. The important BLU Acceleration-related bookmark you'll make for the next year is going to be to the "Best Practice: Optimizing Analytics Workloads using DB2 10.5 with BLU Acceleration" paper at https://ibm.biz/BdDrnq.

## First Things First: Do This!

`db2set DB2_WORKLOAD=ANALYTICS`. A simple setting, but it's the key to all the magic. If your database will be used purely for analytic workloads,

this registry setting takes care of most of the system configuration for you. Because it configures some characteristics that are defined at database creation time (ensure that your database is enabled for `AUTOCONFIGURE`), for best results, you should set `DB2_WORKLOAD=ANALYTICS` *prior to* creating your database. This setting makes column-organized tables the default table type and configures page size, extent size, memory (buffer pools, sort, lock list, application heap, and so on), automatic workload management, new page cleaning and prefetching algorithms specifically designed for BLU Acceleration technology, automatic space reclamation, automatic statistics collection, and more. We repeated this here because it's so important. We also want to note that if you access a row-organized table sitting in the same database schema, DB2 can use its traditional page-cleaning and prefetching algorithms too.

If you can't set `DB2_WORKLOAD=ANALYTICS` before creating your database, you can still set it and run the `AUTOCONFIGURE` command to achieve many of the tuning benefits (it configures optimizations such as automatic workload management, space reclamation, page size, and extent size).

## Unable to Set DB2_WORKLOAD=ANALYTICS?

You might not be able to set `DB2_WORKLOAD=ANALYTICS` prior to creating the database for various reasons. Perhaps you're migrating an existing database from an earlier version of DB2, or perhaps your database supports mixed workloads and will include combinations of column-organized and row-organized tables. In a mixed environment where you have a significant amount of transaction processing occurring, you probably don't want to configure the database as though it is running an entirely analytic workload. Whatever the reason, if you can't create a new database in `ANALYTICS` mode, here's what setting `DB2_WORKLOAD=ANALYTICS` does, so you can pick and choose which optimizations best suit your environment:

- The page size (in KB) is set to `32`.
- Extent size is set to `4`.
- The default table organization (`DFT_TABLE_ORG`) database configuration parameter is set to `COLUMN`.
- Memory is divided (roughly) equally between the buffer pool and shared sort heap threshold (`SHEAPTHRESH_SHR`).

- The sort heap (SORTHEAP) is set to a moderate fraction (for example, 5 percent) of SHEAPTHRESH_SHR. (See the "Automated Memory Tuning" section that follows.)
- Intrapartition parallelism is enabled.
- Sets the default degree of parallelism (DFT_DEGREE) to ANY.
- Automatic statistics collection is enabled.
- Sets UTIL_HEAP_SZ to a sizeable value. (This gives the load utility elbow room to generate high-quality compression dictionaries. We have some specific recommendations in the later section "For Optimal Compression Results….")
- Self-tuning memory manager (STMM), buffer pools, the lock list, and package cache are all enabled.

Finally, as a general rule of thumb, if you are using DB2's sophisticated workload management capabilities, we recommend a concurrency threshold no larger than the number of cores available on the system and a timeron threshold of 75,000.

## Automated Memory Tuning

DB2's self-tuning memory management (STMM) is an incredible piece of technology. However, in the first release of DB2 with BLU Acceleration, sort memory is not self-tuning for column-organized tables, so you have to watch out for this. Specifically, both the short heap (SORTHEAP) and shared sort heap threshold (SHEAPTHRESH_SHR) should be set to static values; if DB2_WORKLOAD=ANALYTICS, this will have been done for you.

In addition, you should know that BLU Acceleration makes heavy use of sort memory because it's used for grouping, joins, and vector memory processing; therefore, the requirements are higher than for row-organized processing. Of course, as is the case with traditional row-organized tables, when sort memory gets constrained, the database will start spilling to disk. We recommend setting SHEAPTHRESH_SHR to a value that's similar to your buffer pool memory. This is *considerably* more generous than what's usually recommended for row-organized tables, but the results are worth it. Trust us, because SORTHEAP is usually configured as a fraction of SHEAPTHRESH_SHR; 10 percent is common. Remember, BLU Acceleration is a dynamic

in-memory database, and as its name would suggest, it's going to rely on memory; you're not going to have constraints if all your data won't fit into memory like some of the alternative technologies available in today's marketplace, but be prepared for more memory requirements.

## For Optimal Compression Results…

Compression is a hallmark of BLU Acceleration technology, not just because of the storage savings delivered, but because its vector-processing engine operates on compressed values. Better compression often results in better query performance.

In a BLU Acceleration environment, you should expect to see temporary increases in DB2's utility heap requirements. This is because the `LOAD` utility is designed to optimize compression encoding by doing a first pass over the data in what's known as the *analyze phase;* this phase is used to scan the data, construct histograms, and build the compression dictionaries. Therefore, the more memory that you make available to `LOAD`, the deeper this analysis can be, which will result in better compression ratios. For this reason, consider increasing the `UTIL_HEAP_SZ` while the `LOAD` utility is running (you can reduce it again when it's finished). For sizeable tables, several gigabytes of utility heap are likely suitable. If you're loading terabyte-class tables, tens of gigabytes of utility heap are likely more appropriate. For these reasons, we also recommend that you avoid a large number of concurrent `LOAD` operations.

For optimal compression results, you should be thinking about initially loading as much data as possible—a large initial load operation is healthy. Column compression dictionaries are built by analyzing the data during the first load operation against a table, and therefore, it's extremely valuable for that initial load operation to contain a representative and reasonably large amount of data. Following the initial load operation, new data values not found in the dictionary are compressed using page-level dictionaries.

## Data Statistics? Don't Bother, We've Got It Covered

Many of you are accustomed to performing periodic `RUNSTATS` jobs to collect statistics on tables and indexes, especially after tasks such as `LOAD`,

IMPORT, or INGEST. With BLU Acceleration, it's all automated! You can still execute those tasks, but there's a lot less need to. The LOAD utility collects statistics automatically, and automatic table statistics are enabled by default on all new databases. At this point, you might be wondering if there's really any need to issue a RUNSTATS job. Yes, if you've populated a table with a process that uses INSERT (such as IMPORT or INGEST) and you want to start running queries immediately after the task, then you've got a good reason to invoke RUNSTATS because automatic statistics collection will not likely have been initiated by the time you want to immediately run your query after ingesting the new data into the target table (but it will be initiated when it wakes up at its next interval).

## INSERT Performance

We think you'll be pleased with INSERT performance for column-organized tables. That said, keep in mind that BLU Acceleration is optimized for *bulk* processing. For this reason, we recommend *commit rates of 10,000 rows or more.* If you perform tasks using INSERT (including IMPORT or INGEST) and commit more frequently (such as every 1,000 rows or less), your INSERT performance might suffer.

## How to Skip More: Get the Most Out of Data Skipping

Every column-organized table maintains a child *synopsis table,* which is also column organized and stores encoded vectors of data. The synopsis tables store metadata about the columns of their parent tables and are used at query runtime to perform data skipping. They work best when data in the parent tables is clustered on predicate columns.

Now, there's *no need* to sort your data, as is the case with many competing technologies. However, if you're looking for a little extra "boost," sorting your data before a LOAD can give a nice jolt to the database engine. The sorting doesn't need to be exact. For example, consider a table with predicates on a DATE column; sorting the entire table by DATE might be prohibitive. Sorting by month or quarter will usually be more than sufficient to achieve great data-skipping benefits.

## A NextGen Database Does Best with the Latest Hardware

BLU Acceleration is designed to leverage the latest microprocessor architectures in a very intimate way. Although our algorithms can run on readily available microprocessors, we're constantly on the lookout for ways to optimize newer instruction sets and topologies, which provide facilities to maximize memory bandwidth, advanced SIMD instructions, larger CPU caches, and more. If you have a choice, go with the latest processors available.

## Memory, Memory, and More Memory

By now you can fully appreciate how DB2 with BLU Acceleration is better than a typical in-memory database because you don't have to size a buffer pool with enough memory for all of the active data. What's more, any I/O will be performed with high efficiency by using the new *dynamic list prefetching* technique. After data is fetched into memory, it's processed with all of DB2's advanced processing power. Nevertheless, it certainly helps to keep the most active data in RAM, which begs the question: "How much RAM is ideal for my data?"

When sizing your server for memory, consider what fraction of the columns in your largest tables are active; this should give you a good rule of thumb for sizing your buffer pools. For example, consider a 40-column table with ten years of data that BLU Acceleration technology has reduced from 10TB to 1TB. When examining this table from an "activity" perspective, only the most recent five years are considered "warm" or "hot," and only 8 of the 40 columns are active. In this example, we'd recommend an ideal buffer pool size of 100GB: 1TB * (5/10) * (8/40). And remember, if you're implementing in-memory technology, it's nice to have more memory. It seems obvious, but you are going to want more memory than you would typically allocate to your database.

## Converting Your Row-Organized Tables into Column-Organized Tables

If you're converting tables from a row-organized format into a column-organized format with BLU Acceleration using the `db2convert` utility, we want you to know that this conversion process requires temporary space for both

the source and target tables—so make any adjustments necessary. In addition, because there's no online process to convert a column-organized table back into a row-organized table, we strongly recommend that you run `BACKUP` before converting any tables.

## Wrapping It Up...

In this chapter we introduced you to the "crown jewel" in the DB2 10.5 release: BLU Acceleration. We talked about seven big ideas that are the technical inspirations behind BLU Acceleration's four pillars: dynamic in-memory processing, actionable compression, parallel vector processing, and data skipping. The pillars combine to greatly accelerate, simplify, and conserve your environment's resources.

A plethora of clients have noticed. For example, LIS.TEC's Joachim Klassen observed one of the key benefits of BLU Acceleration when he noted: "Even if your data does not completely fit into memory, you still have great performance gains. In the tests we ran we were seeing queries run up to 100x faster with BLU Acceleration." It's a very different approach than what's been taken by some other vendors. After all, if the price of memory drops by about 30 percent every 18 months, yet the amount of data grows by 50 percent, and in a Big Data world data is being used to move from transactions to interactions, you're not going to be able to fit all your data in memory—and that's why BLU Acceleration is so different.

It should be clear by now that simplicity and "Load and Go" rule supreme when it comes to BLU Acceleration descriptors. Indeed, we commented that BLU Acceleration was more about what you no longer do than what you need to do. Randy Wilson, a top-notch DBA who works for Blue Cross/Blue Shield of Tennessee, sums up this point best when he tried his workload running on a partitioned database environment and ran it on a single DB2 server with BLU Acceleration: "We've tested DB2 10.5 with BLU Acceleration and found that it can be up to 43x faster with an analytic workload than our existing multiserver partitioned database environment. Without having to build indexes, we can just load and go. The performance out of the box is outstanding."

We could brag some more, but don't take our word for it, don't take our customers' world for it (but be jealous if you're not using BLU Acceleration): We invite you to try it for yourself. It won't take long; you just Load and Go!

# 4

# Miscellaneous Performance and SQL Compatibility Enhancements

There's no question that the most noticeable features of the DB2 10.5 release fall under the BLU Acceleration and DB2 pureScale enhancements category; that said, DB2 10.5 has a number of other enhancements that make it an even richer platform from an operational perspective. For example, there are DB2 10.5 enhancements aimed at the continual DB2 drumbeat of making it as seamless as possible to migrate to DB2 from the Oracle database (who's kidding who: this is what we mean when we use marketing-speak and say "compatibility"); there are some features that have been driven into the product to make way for DB2's new JavaScript Object Notation (JSON) document store capabilities (think MongoDB-styled applications); some general performance enhancements; and more. We decided to put all the "stuff" that isn't a chapter on its own, but still pretty big from a SQL compatibility, availability, and performance perspective, into a set of Miscellaneous chapters.

## Expression-Based Indexes

DB2 10.5 introduced the concept of expression-based indexes, which as its name suggests, allows you to define indexes on general expressions (this is used heavily in the DB2 JSON Document Store, which we cover in Chapter 6). Expression-based indexes are going to make applications that rely on the

type of functional searches that this new feature empowers, to perform much faster (since they will no longer be forced to perform full table or index scans), as well as make the lives of application developers easier.

We think you'll be better able to appreciate how expression-based indexes will help your DB2 environment with a simple example, so let's start with the following table in mind:

```
CREATE TABLE employee (
     id         INTEGER NOT NULL,
     lastname   VARCHAR(20) NOT NULL,
     salary     DECIMAL(10,2) NOT NULL,
     bonus      DECIMAL(10,2) )
```

To speed up query performance, a DBA typically would want to create indexes on important application lookups, which may include complex expressions such as a total compensation (equal to your salary plus your yearly bonus combined) and case-insensitive names (it's faster to search for a name without having to worry about byte-level comparisons of uppercase or lowercase letters, and so on). For example, a ubiquitous set of indexes that illustrate this could be created as follows:

```
CREATE INDEX emp_upper ON employee (UPPER(lastname))

CREATE INDEX comp ON employee (salary + bonus)

CREATE INDEX emp_comp ON employee
               (UPPER(lastname), (salary + bonus))
```

Of course, if you tried to create any of these indexes *before* DB2 10.5, they would fail, since index on expression support didn't exist (you can certainly create generated columns in a base table like the ones shown here, just not indexes).

## Get Faster: Query Processing Before and After DB2 10.5 with Support for Index Expressions

Keeping with our example to help you better understand the benefits that DB2 10.5 brings with its new index expression support, first let's consider how things worked in DB2 10 (or earlier versions). Take for example the following ubiquitous SQL query that's looking for an employee by last name without case sensitivity:

```
SELECT * FROM employee WHERE UPPER(lastname) = 'RICK'
```

With the previous query (before expression-based indexes existed in DB2), DB2 would have to perform a full table (or index) scan to satisfy this query, *even if* there was an index on `LASTNAME`. In other words, to find any employee whose last name is `RICK`, all rows in the table (or all keys in the index) would need to be examined and compared against the value `RICK`.

Now consider that DB2 10.5 lets you define an index on `UPPER(lastname)`; this is going to yield a number of benefits. First, in consideration of our example, this enhancement means that this popular query can now be executed with an index range scan that accesses the fraction of the index that corresponds to the actual values in the predicate. As you've likely inferred by the words "full table scan" in the description of how things worked in previous versions of DB2, there's going to be a performance boost here. We typically see (and, of course, it depends on the data distribution and the query) a dramatic reduction in the number of pages that need to be accessed to address queries like the one in our example and this results in a dramatic improvement in performance.

## Expression-Based Indexes Make Application Development Easier

Another benefit of the new DB2 10.5 index on expression capability is that it makes the whole process of application development much simpler. Before this feature was supported in DB2, developers would often resort to adding application-side code to approximate the behavior of a function-based index. Of course, the downside to this approach is that it required extra development effort, pushed data performance techniques out of the data layer, and required subsequent application deployment for new searches. DBAs could try and work around this missing feature with generated columns, but that's more than likely to require additional storage; it's a work-around, but it's not as clean or efficient, and it has its limitations too.

For example, one approach that's been used in the past to compensate for this missing feature in DB2 of old was to add a column to a table, where the column value is automatically generated based on a defined expression. The following is a typical workaround for this missing feature in previous versions of DB2:

```
ALTER TABLE employee ADD COLUMN emp_upper GENERATED ALWAYS AS
          (UPPER(lastname))
```

Of course, the extra column consumes more space and adds complexity to the application in that it needs to deal with this new column.

### Expression-Based Indexes Have Views

When you start to implement this handy new DB2 feature in your environment, we wanted to point out that you may see additional views being created when you use expression-based indexes; don't worry about it—this is normal. DB2 will automatically create and manage (there's nothing for you to do here) a set of statistical views to help facilitate statistics collection on your expression-based indexes. For example, when the `EMP_UPPER` index was created earlier in this section, DB2 would have automatically created a statistical view with a system-generated name—for example, it would have generated Data Definition Language (DDL) such as: `CREATE VIEW emp_upper_v(K00) AS SELECT UPPER(lastname) FROM employee`. If you were to drop the `EMP_UPPER` index, DB2 would automatically drop its associated statistical view.

As a DBA, when you collect statistics on a table with an expression-based index and choose to also include the collection on index statistics, DB2 will use this statistical view to collect the statistics that correspond to the specified expression. Again, all this occurs automatically, and the net effect is that there's no need for you to collect statistics separately on the view.

## Excluding NULL Keys from Indexes

Another index-related capability that's new in DB3 10.5 in addition to expression-based indexes is the ability to exclude NULL keys from the indexes themselves. Quite simply, this new DB2 10.5 feature allows you to define indexes that ignore NULL keys. We bet you're wondering at this point how ignoring NULL keys in an index could be valuable. As we're sure you're aware (and likely experienced), "more from less" has always been a key thrust of each new DB2 release, and that certainly applies with this feature when it comes to performance; you get more by excluding NULL keys from indexes and less resource consumption.

We'll illustrate the benefits of NULL exclusions from indexes using an example. Consider the following ORDERS table from an online ordering system:

```
CREATE TABLE orders (
           shopping_cart_id   INTEGER,
           order_num          INTEGER,
           custname           VARCHAR(30) NOT NULL,
           item_num           INTEGER NOT NULL,
           price              DECIMAL(10,2) )
```

Let's assume in this example that when a new order is submitted, the order number is not immediately known and, therefore, the order's corresponding record gets inserted into the ORDERS table with a NULL value for the ORDER_NUM key.

This is pretty typical in a mobile world, where applications are often broken down into browsing and buying modules. When potential clients are browsing and adding items to their online shopping carts, that stuff is persisted (for abandoned cart analysis, availability in case of a browser's session crash, and so on); however, a true order number isn't generated until a potential client's order has passed a payment authorization check and the item is verified to be in-stock (the application is likely using a very forgiving isolation level during the browsing and addition to cart phases of an order, but needs tighter semantics at purchase time). If the payment and availability checks pass, and the order is verified, a subsequent transaction in the ordering system assigns an order number to the corresponding row in the ORDERS table, which is updated with this order number. Let's further assume that there's a frequent need to look up past orders by order number, and this leads the DBA who supports this ordering application to create an index that looks like this:

```
CREATE INDEX order_numi ON orders (order_num)
```

If you were to eavesdrop on a DBA's soliloquy while they created this supporting index structure (yes, we know a couple who think and talk to themselves just like this), you'd hear them say, "It's a shame that this index needs to include orders that are in-process and don't yet have order numbers. Wouldn't it reduce index maintenance costs by only including orders that actually have order numbers? After all, when I look up orders, I'm only going to be interested in orders that have finished processing, those that are real and have been assigned order numbers."

Our answer to such a thought-provoking and privately pontificating DBA is a resounding "Yes!" However, before DB2 10.5, there was no way to accomplish this, and now you get a good idea for the inspiration behind this index enhancement. Specifically, before DB2 10.5, all indexes had to index all the rows of a table, including those with NULL keys. The DB2 10.5 release adds the ability to exclude NULL keys from an index using the aptly named `EXCLUDE NULL KEYS` option of the `CREATE INDEX` statement; for example:

```
CREATE INDEX order_numi ON orders (order_num) EXCLUDE NULL KEYS
```

The index created with the previous DDL will instruct DB2 10.5 (or later) to ignore rows with NULL index keys during index maintenance. Quite simply, when a row with a NULL order number is inserted into our example index's corresponding base table, no key will be inserted into the `ORDER_NUMI` index. What's more, if (or when) that same row's order number is updated from NULL to an actual non-NULL value (the order is hardened and becomes real), at that point in time, the correct value will be inserted into the index as a new index key.

This new feature not only avoids the processing overhead of unnecessarily maintaining NULL keys in an index, but also has the additional benefit in that there are savings associated with storage and buffer pool memory resources because the NULLs aren't being stored. Of course, just like in the new expression-based indexes feature we talked about in the previous section, this is yet another feature that fits into the "do more with less" DB2 theme that repeats itself across every release—excluding NULLs from indexes can yield more performance with less resource consumption.

## Index NULL Exclusion Simplifies Application Development

Excluding NULLs from indexes has another advantage beyond performance, and less resource consumption: it can foster a more simplified application development environment, particularly with respect to unique constraints. Continuing with our online ordering system example, it would surely make sense for each order number to be unique, and you could imagine the benefit if this business rule could be pushed into the database and DB2 could be instructed to enforce this constraint by defining the index as `UNIQUE`. Obviously, before DB2 10.5, where there was a potential for multiple order

numbers that inserted as NULL values into the index at any given time, this option simply wasn't possible. The old work-around? The application development team would need to manually code uniqueness checking and prevention into their application. Of course, as was the case with expression-based indexes, that's more effort, code to maintain, potential for errors, a reduction in agility, an increase in deployment costs, and more. The goal is to push this into the database system (where it belongs), and with the DB2 10.5 support for NULL exclusion from indexes, this all becomes possible.

Of course, with the new enhancements for NULL exclusion from indexes in mind, you could create an index for our online ordering application as follows:

```
CREATE UNIQUE INDEX order_numi ON orders (order_num)
   EXCLUDE NULL KEYS
```

Now that this logic is pushed into the database, it doesn't need to be implemented as application-side code. The application development teams could remove the code and simplify everyone's lives. What's more, it promotes reusability because different applications wouldn't have to import the corresponding code module (or even worse, each write their own in a large company, and that's not uncommon at all). They could just rely on the database to enforce this logic—the way it should be for data-related logic.

## When 2 + 2 = 4: Getting Even Richer Semantics by Combining Features

At this point, you're likely thinking that you should be able to mix the ability for DB2 10.5 to create expression-based indexes with its ability to exclude NULLS from the indexes themselves; what's more, using these two features together will undoubtedly provide some synergistic benefits in the form of advanced capabilities. If this is what you're thinking, then you're bang on! As usual, we'll use an example to illustrate this statement.

Let's continue our example with the assumption that you added an `ORDER_STATUS` column to the `ORDERS` table we defined earlier in this chapter. As orders come into the system, the `ORDER_STATUS` column takes on a value of `NEW`. When the order is in the middle of processing, the `ORDER_STATUS` is creatively changed to `PROCESSING`. Finally, with an even greater dash of creativity, when the `ORDER_STATUS` is finished, its status is updated

with the keyword COMPLETE. Let's further assume that any completed order numbers are kept online in the system for several months for typical reporting purposes. With this in mind, your application's order number generation logic allows for completed order numbers to be reused.

In this scenario we outlined earlier, you couldn't apply a unique constraint directly to the ORDER_NUMI index because of the potential for order number reuse. However, in DB2 10.5, you can combine expression-based indexes and the ability to exclude NULL keys from an index to apply a unique constraint to the ORDER_NUMBER column, and furthermore, only apply this business rule if the status of the order is *not* COMPLETE, as follows:

```
CREATE UNIQUE INDEX order_numi ON orders
           (DECODE (order_status,  'new', order_num,
                                     'processing', order_num, NULL))
            EXCLUDE NULL KEYS
```

If the DECODE option (this option is similar to the existing CASE expression; it was added in DB2 9.5 as part of the Oracle compatibility feature set) was a person talking to DB2, it would basically tell DB2, "If the ORDER_STATUS is NEW or PROCESSING, use ORDER_STATUS as the index key; otherwise, use NULL (if the ORDER_STATUS is COMPLETE)."

These features combined in the previous example would result in DB2 enforcing unique ORDER_NUMS so long as the ORDER_STATUS was NEW or PROCESSING; its net synergistic effect means that DB2 will enforce unique ORDER_NUMS so long as ORDER_STATUS is in a NEW or PROCESSING state, which provides an advanced semantic that precisely matches the needs of the application.

## Random Ordering for Index Columns

Another interesting indexing enhancement in the DB2 10.5 release is the ability to randomize the order of index columns when they are stored in the index, which can help optimize performance in some particular scenarios (in case you are wondering why someone would want to do this). Keeping with the online ordering system example used in this chapter, let's create the ORDER_NUMI unique index on the ORDER_NUM column in the ORDERS table as follows (this is a simpler form of this index than the one we created earlier):

```
CREATE UNIQUE INDEX order_numi ON orders(order_num)
```

Let's assume that in our application, order numbers are generated by incrementing the last order number used (a typical algorithm for this kind of application). Now take a moment and consider what physically happens within the `ORDER_NUMI` index as new order numbers are inserted into the index through high-throughput workloads, where many thousands or more concurrent transactions may be generating order numbers. If region thrashing comes to mind, you're spot-on. If you're not a DBA, think about the last time you were at a conference and they opened the buffet luncheon doors and everyone went to the closest two buffet lines—things got pretty inefficient (and dare we say rowdy at some of the database conferences we've attended).

What's happening is that all of these transactions (people trying to eat) are trying to insert their order numbers (get some food on their plate) into the same region of the index (from the same buffet line). Specifically, dropping the analogy because it's making us hungry just writing it, the transactions are trying to insert order numbers into the *same* page of the index's B-tree because indexes physically store columns in their ordered sequence. For example, in an ascending index, if order number `1000` is stored on index leaf page `5`, DB2 will also try to store order number `1001` on leaf page `5`.

In our running example, it's conceivable that some individual index pages can be competed for by multiple concurrent transactions, leading to less-than-optimal performance, and this is where the DB2 10.5 random index feature comes in. As of DB2 10.5, you could define the `ORDER_NUMI` index using the `RANDOM` keyword in the following manner:

```
CREATE UNIQUE INDEX order_numi ON orders (order_num RANDOM)
```

When the `RANDOM` keyword is used, an order number is randomized before it's stored in the index (you can kind of think of it as a hashing algorithm, in the same manner that a row is hashed to a specific DB2 database partition when DB2 is clustered using the Database Partitioning Facility). This means that if order number `1000` is stored on index leaf page `5`, order number `1001` will very likely be stored on some other page (something *other* than page `5`), which, of course, has the downstream benefit of more evenly spreading out index page targets for agents with the intent of writing out a sequenced order number, alleviating the competition for individual index pages and leading to improved performance.

### To Random Order an Index Column or Not to Random Order an Index Column… That Is the Question

We're sure that random ordering an index column sounds great (and logical); however, as with most things in the computing world (and life, for that matter), there are some trade-offs to consider with the new DB2 10.5 random ordering on index capabilities feature so we thought we'd comment on them here. So why don't you just use this new feature for all your indexes? As you might have concluded, if you instruct DB2 to store the index columns in random order, you won't be able to use such an index to satisfy queries that have ordering requirements. For example, the random index created in the previous section could not be used by DB2 to satisfy the following query:

```
SELECT * FROM orders ORDER BY order_num
```

If your application is laden with a high dependency on ordering, a traditional ascending index on the `ORDER_NUM` column might just be your best bet. Of course, this guidance is going to change from application to application; it's going to depend on the query makeup, the application's concurrency, read to write ratio characteristics, and more. This all said, while not a panacea, random indexes are a great feature that can help improve application performance in some situations.

## More Data on a Row: Extended Row Size

Before DB2 10.5, the maximum row size of a table was limited by the page size of the table space where the table was created. This means that before DB2 10.5, a 4KB page size could have a maximum row size of 4,005 bytes; an 8KB page was limited to 8,101 bytes; 16KB to 16,293 bytes; and a 32KB page was limited to 32,677 bytes.

These row limits, especially considering the effects of DB2 compression on them, have worked just fine for most applications. That said, we've come across occasional cases where an application is coded to require a row size that exceeds the 32,677 limit (which is dependent on a 32KB page size). Where we've seen this requirement is in applications where the team's coding practice is to use arbitrarily large column sizes (popular in web-based

applications where developers want a sort of "bit bucket" to dump character data into), as opposed to the true maximum size required by the application semantic, such as the DDL we had from one client show here:

```
CREATE TABLE customers ( firstname VARCHAR(32000),
                         lastname VARCHAR(32000)... )
```

If you were to try and create this `CUSTOMERS` table in a previous version of DB2, the `CREATE TABLE` statement would fail because the row size is too large. Our first recommendation to these kinds of application development houses is to use the more realistically sized (and efficient) `VARCHAR` data type for columns such as `FIRSTNAME` and `LASTNAME`; after all, we've yet to come across names this long. That said, we also understand that changing legacy or existing application code is not always practical, and so DB2 10.5 includes a new capability to extend the row size limit of a traditional DB2 table.

As of DB2 10.5, rows can exceed the table space page size. When a row exceeds the page size limits detailed earlier in this section, under the covers (and seamless to the DBA), DB2 will split up the row into separate pieces and place the portion that doesn't fit on a single page into a separate "special" large object (LOB) that won't be seen by applications as a column.

We want to note that the design point of DB2's new extended row feature isn't for use cases where you want to access all of the table's columns and most of the rows in that table exceed the table's page size. In this kind of workload pattern, the performance overhead of accessing the separate large object may become significant. In contrast, this new extended row support is focused on the far more typical, where very large rows are rare and/or only subsets of columns are typically referenced. In such cases, the overhead of accessing the separate large object will typically be insignificant.

As you're probably aware, or rather come to expect from our track record in the previous decade of DB2 releases, our development teams are maniacally focused on delivering new capabilities that are simple to use and implement. You've seen this in a number of features; compression is a great example, with its management by intent policy. For example, in DB2, you simply specify a table attribute that tells DB2 you want it to optimize the ecosystem for storage savings and performance. When a table is created using this attribute, it will automatically enable row and temporary table compression; it will automatically enable index compression and pick and choose which of DB2's three index compression algorithms has the most

positive effect on your environment (it can choose on, all, or a subset). The self-tuning memory management (STMM) infrastructure in DB2 is designed to learn and profile your running application and adapt heap allocations for best performance. Of course, BLU Acceleration's "Load and Go" proposition is the latest example of our incessant focus on consumability. When you consider that we keep talking about how DB2 automates the great things it does for you where it makes sense, it should be no surprise that extended rows are enabled by default for any new database you create in DB2 10.5.

With this in mind, we understand behavior consistency and predictability for existing workloads is just as important for our clients when they move to a new release of DB2. For this reason, if you have any existing databases in your environment and then upgrade to DB2 10.5, the new extended row support will be disabled. If you want to enable it, it's simple—just update the `EXTENDED_ROW_SZ` database configuration parameter and set it to `ENABLE`; it's an online operation.

# 5

# Miscellaneous Availability Enhancements

DB2 10.5 delivers a number of availability enhancements in addition to the DB2 pureScale enhancements that we cover in Chapter 2, including extended capabilities and new features in the areas of space reclamation for insert time clustering tables (for example, extents don't have to be completely free to take advantage of its special space reclamation characteristics); reorganization enhancements that affect tables with overflow and pointer records; adaptive compression support for online table reorganization; and some useful online table management features for tables with referential constraints that are no longer blocked. In this chapter, we outline these "not in the highlights reel" availability enhancements because they're bound to be well appreciated by any DBA who manages a DB2 ecosystem.

## Better Online Space Reclamation for Insert Time Clustering Tables

Insert time clustering (ITC) tables were first introduced in the DB2 10.1 release. ITC tables provide an effective way of maintaining data clustering and easier management of space utilization. ITC tables have similar characteristics to multidimensional clustering (MDC) tables. For example, both of these table types use block-based allocation and block indexes. From a clustering perspective, you can think of ITC tables as clustering data using a virtual column which clusters rows based on row insertion time. You create

an ITC table by specifying the `ORGANIZE BY INSERT TIME` clause on the `CREATE TABLE` statement. You can use the `ADMIN_MOVE_TABLE` routine to convert existing regular tables into ITC tables online, or use `EXPORT`, `IMPORT`, or `LOAD` to move data from a source table to a target table. DB2 10.5 simplifies the management of these tables.

## Understanding Insert Time Clustering Tables—A Primer

The best way to understand how ITC tables work is to compare them with regular tables, and we'll start by taking a look at how a regular table manages its space. When rows are inserted into a regular table, they're placed on an arbitrary page; more accurately, DB2 has an internal space search algorithm that selects a page with enough space for the row, but from a user's point of view, the page is seemingly selected at random.

Consequently, as rows are deleted over time, a table can have quite a bit of free space that is spread out (or "fragmented") in small quantities over many pages. Although this space can be consumed by newly inserted rows in the same table, other tables can't easily use this space. If you want to enable other tables to use this newly available space, a reorganization (`REORG`) operation must be performed, which has the effect of tightly packing the existing rows together on a data page, starting with the table's first page. The end result is numerous empty pages at the end of the table, and a set of empty pages that are subsequently made available to other tables in the table space, as shown in Figure 5-1.

A reorganization operation such as the one shown in Figure 5-1 can be performed in place in an online manner. This method can take a fair amount of time to complete, due to the amount of data movement taking place; however, it's important to note that this is by design: Online DB2 table reorganization is designed to make the table ultra-available during the operation. You can throttle or pause and resume such an operation in DB2, and its other benefit is that it bypasses the doubling of storage requirements that is typically required by other vendors' "shadow" approaches to online table reorganization.

We can tell you that sets of rows that are inserted at about the same time are often deleted at about the same time. For example, consider an invoice

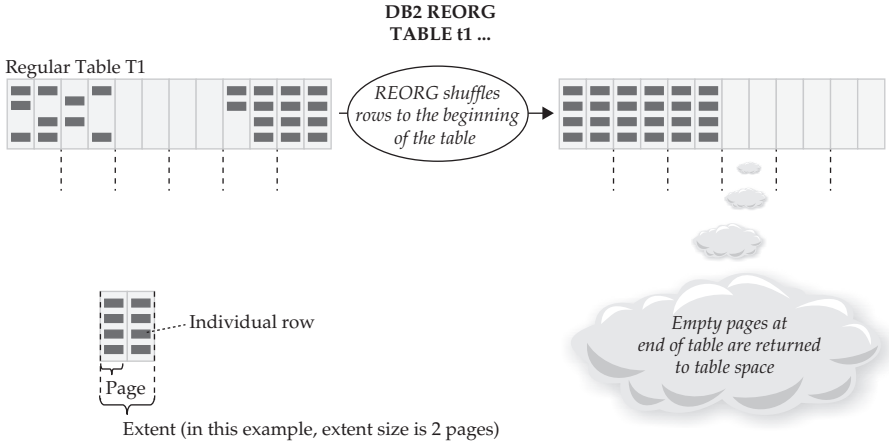


**Figure 5-1** *How DB2 performs a typical table reorganization*

tracking system that regularly runs a batch job to delete invoices that are more than 90 days old. When this batch job runs, any rows that were inserted 90 days ago are deleted. If these rows happen to be located on the same data pages, those pages are now empty and things are simple. Unfortunately, as with most things in life, it's never quite that simple. In the case of a regular table, the rows that you want to delete are not likely to be on the same page. Wouldn't it be nice if all the rows that are targeted for deletion based on time were initially inserted on the same pages? And if those pages, when empty, could easily be released back to the table space without a traditional reorganization operation, wouldn't that be terrific? This is where ITC tables help because that's *exactly* what they were designed to do.

An ITC table clusters rows according to the time at which they were inserted; rows that are inserted at about the same time are placed on the same page, or on a page in the same extent (an extent is just a contiguous group of pages). If these same rows are deleted during a batch operation, entire pages (or entire extents) become empty. Moreover, ITC tables allow such empty extents to be returned to the table space quickly (and automatically) through DB2's automated table maintenance daemon, or manually by specifying the `RECLAIM EXTENTS` option on the `REORG TABLE` command, as in the following example:

```
REORG TABLE t1 RECLAIM EXTENTS ALLOW WRITE ACCESS
```

We want to emphasize that this is a *very* quick *and* online operation. Unlike a traditional reorganization operation, space reclamation from ITC tables doesn't involve the movement of rows, as was the case in Figure 5-1. In fact, all that happens is that the empty extents are returned to the table space, as shown in Figure 5-2.

ITC table reorganizations are fast for two reasons. First, the empty extents are easy to find because they're marked empty after the last row is deleted. Second, there's no row or data movement—the operation just returns empty extents to the table space, in place and online.

## What's New for Insert Time Clustered Tables in DB2 10.5

In DB2 10.1, extents had to be completely empty for them to be returned to the table space. DB2 10.5 enhances the value proposition for ITC tables by enabling even partially empty extents to be reclaimed. To help explain why this enhancement to ITC tables could be useful, let's return to the invoice system scenario. In a more true-to-life example, the batch delete job in our example might delete all rows that are more than 90 days old *and that have been paid*. This means that there might be the odd case in which one or two rows are left behind because the associated invoices haven't been paid, thereby preventing all of this space from being reclaimed.

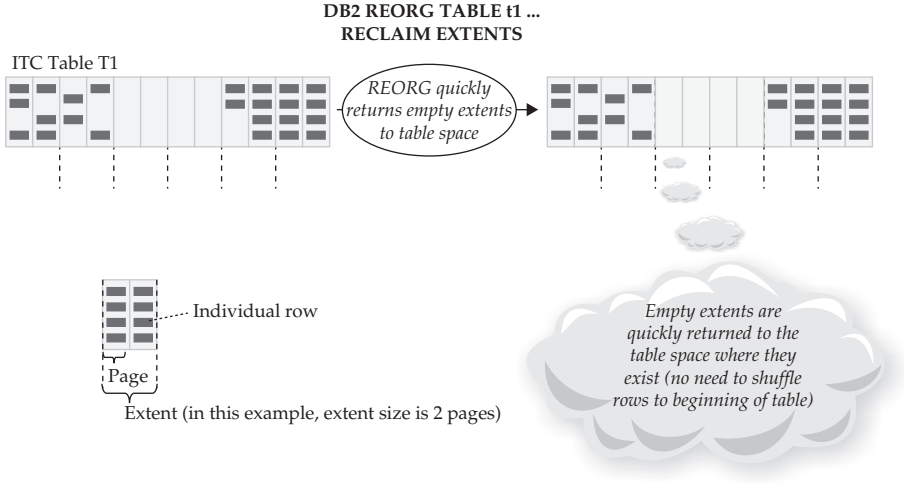


**Figure 5-2** *How DB2 performs an ITC table reorganization*

Consider the benefit if those rows representing unpaid invoices that are holding up the full reclamation of space in an almost empty extent could be consolidated on a small number of pages. If you think that's a great idea, you'll be pleased to know that's exactly what the DB2 10.5 enhancement for ITC tables does: A `REORG RECLAIM EXTENTS` operation automatically moves rows from almost empty extents to other extents so that the resulting empty extents can be returned to the table space, as shown in Figure 5-3. Again, and as you've come to expect from DB2, all of this occurs automatically whether you are using DB2's automated table maintenance daemon or you are using the `REORG RECLAIM EXTENTS` command to reclaim the space.

This improved `RECLAIM EXTENTS` operation is still very fast in DB2 10.5 because the row movement is limited to nearly empty extents.

Note that you can use the `ADMIN_GET_TAB_INFO` stored procedure to monitor the amount of space that is available for reclamation from an ITC table. For example, the following query returns the total amount of space that is available for reclamation from table T1:

```
SELECT RECLAIMABLE_SPACE FROM TABLE(SYSPROC.ADMIN_GET_TAB_INFO('myschema','t1'))
```

DB2 10.5 updates the output from this command to accurately show the space that can be reclaimed by freeing up nearly empty extents.

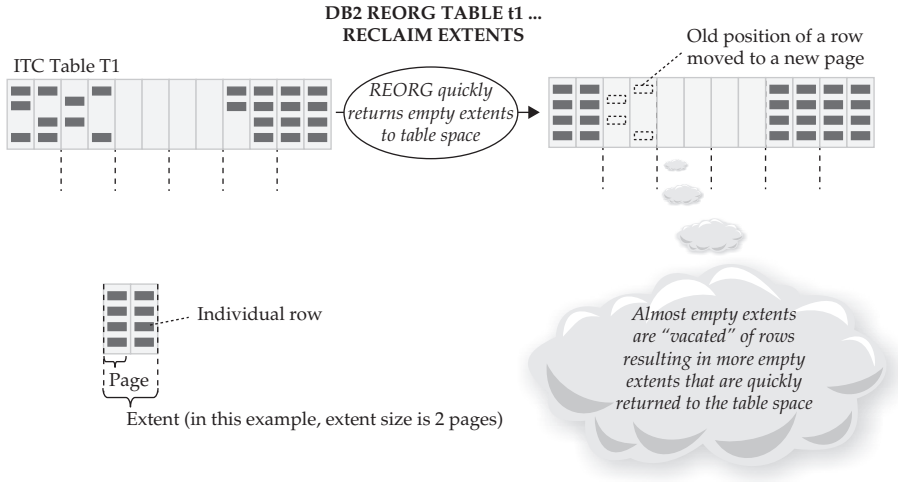


**Figure 5-3** *How DB2 10.5 performs an ITC table reorganization; it's smarter, because it can pack nearly empty extents to automatically return more free space to the table space.*

# Other DB2 10.5 Reorganization Enhancements

DB2 10.5 includes a few more enhancements for table reorganization that are designed to increase the availability of a DB2 server. Enhancements include a "fastpath" option for collapsing overflow and pointer records; support for adaptive compression with online, in-place reorganization; and support for tables with referential constraints that are referenced in the online `ADMIN_MOVE_TABLE` stored procedure.

## Fastpath! Collapsing Overflow and Pointer Records

Let's start this section by ensuring that you understand what we mean by overflow and pointer records. Let's assume that you've got an employee in Florida who moves from Tampa to St. Petersburg. This personnel change requires an update to the employee's record in the `EMPLOYEE` table. Changing the `CITY` column value (defined as `VARCHAR`) for this employee from "Tampa" to "St. Petersburg" increases the length of the row. Suppose that there isn't enough free space on the page to contain this new value. In this case, DB2 will insert the new version of the row on a different page that has enough space and consider the old version of the row to be its internal location, and use an associated record identifier (RID) for the newly moved row. The *new* version of the row is called an *overflow record,* and the *old* version, which now contains an RID, is called a *pointer record* because it "points" to the location of the actual row.

When an overflow record and pointer are used to support an update, any index references to this row still refer to the old location of the row. Although this operation is seamless, if very frequent row-enlarging updates spawn the creation of many overflow and pointer records, performance-sensitive applications could see some impact because index access to these rows requires access to two pages instead of one: one to read the pointer record, and the other to follow the pointer record to the page that actually contains the row data. Of course, a reorganization operation will convert the overflow/pointer pair to a single normal record and update indexes; however, as we mentioned

earlier, DB2's online reorganization protocol is designed to be ultra-available and nonblocking, and can take some time to complete.

Suppose that you've been noticing a small performance degradation in some of your queries since a large number of updates occurred. After reviewing the built-in DB2 `OVERFLOW_ACCESSES` monitor element, you realize that your table has some overflow and pointer records that are affecting the performance of your mission-critical applications. You obviously want to collapse them, but you don't want to perform a complete table reorganization. The DB2 10.5 fastpath option directly addresses this scenario with the new `CLEANUP OVERFLOWS` option on the `REORG TABLE` command. For example:

```
REORG TABLE t1 INPLACE CLEANUP OVERFLOWS
```

This reorganization operation simply scans the table for overflow/pointer pairs and converts each pair into a normal record. Because this operation is limited to cleaning up overflow records alone, it executes much more quickly than a full reorganization operation.

## Support for Adaptive Compression with Online, In-place Reorganization

Have you ever tried to perform an online, in-place table reorganization on a table that is using the adaptive compression feature that was introduced in the DB2 10.1 release? If you did, you likely weren't pleased with the results: a "not supported" SQL error code. To be honest, this was simply a piece of work that didn't get done in time for the general availability of the DB2 10.1 release, and for one reason or another (we were really going all in on the stuff we were coding up for DB2 10.5) it didn't make it into a Fix Pack, and thus it makes its debut in DB2 10.5.

## Support for ADMIN_MOVE_TABLE Routine with Referential Constraints

Have you ever tried to use the `ADMIN_MOVE_TABLE` routine to move a table to a new table space, or make some general schema changes while maintaining online access to the table? If you had referential constraints defined on this table, you'd find out that you couldn't do that. In DB2 10.5, you can.

## Wrapping It Up…

In this chapter, we talked about some of the high-availability enhancements that get overshadowed when you look at all the availability work that's gone into the DB2 pureScale 10.5 feature set. We also discussed how the ability to reclaim space from insert time clustering tables becomes easier in DB2 10.5, as does the management of tables with overflow and pointer records. Finally, we covered new support for some things that you couldn't do before, such as invoking the `ADMIN_MOVE_TABLE` routine on a table with referential constraints, or performing online reorganization operations on tables using adaptive compression. All in all, DB2 10.5 "raises the bar" for availability.

# 6

# DB2 10.5: New Era Applications

It seems that everywhere you turn, people talk about "NoSQL" databases and how they will ultimately replace traditional databases. In fact, many people believe that there is no more innovation in the relational world and that Big Data projects can only work with often creatively named "NoSQL" databases. DB2 10.5 is here to prove them wrong and help their applications scale and stay up and running! It *is* true that the flexibility associated with the NoSQL model is extremely well suited for certain kinds of applications, especially in the mobile-social-cloud world. After all, getting an accurate count of the "likes" on your Facebook page at any one time or the number of Twitter followers you have isn't something that is typically considered mission critical. At the same time, relational database management systems (RDBMS) *are* critical for other kinds of transactional operations. (Are you getting the feeling early on that modern era information management architectures will have a mix of NoSQL and SQL style applications, depending on the data need?)

A good data management foundation requires both approaches, depending on the tasks at hand. Here is how we see it: NoSQL and SQL databases both have characteristics that are well suited to specific application styles, but availability, scalability, and governance are relevant to both. The great thing about DB2 is that release after release, it's folding a NoSQL agile style of data management into its rock-solid and proven foundation. The DB2 10.5 release continues to evolve its NoSQL capabilities with new JSON-document

store services that are designed in the same style as MongoDB; in fact, you can even run MongoDB applications in DB2 by simply embedding a driver and pointing it to a DB2 10.5 server! And this is the point: DB2 is becoming agile. In baseball, a player doesn't use the same hand to catch and throw a ball—unless they only have one hand—because it isn't efficient or agile. To the contrary, a baseball player uses each arm in a highly optimized and coordinated fashion: one for the task of throwing and the other for catching. You can use DB2 in this matter from a database persistence perspective. If you can use its services that are fine tuned for agility (NoSQL) and those that are fine tuned for traditional database needs (SQL)—you're not trying to do everything with a single limb. The end result is a more agile platform that is robust, governed, available, and reliable.

## What's in the NoSQL Name?

So what exactly is a "NoSQL" database? NoSQL doesn't mean "no SQL"; it means "not only SQL." In fact, it's kind of ironic that the biggest movement during the last year or so in the world of NoSQL is the push to get SQL there.

NoSQL developed out of the necessity to store and manipulate a variety of data that doesn't fit rigorous RDBMS requirements and would often rather trade-off traditional benefits associated with databases for a new one: extreme flexibility. Sometimes folks refer to NoSQL as "schema later" or "schema on read"; in contrast, the classic RDBMS model could aptly be dubbed "schema first" or "schema on write." Some folks say that NoSQL databases are created to avoid the need for DBAs. After all, it's fair to assume that in most cases, when developers think performance, they think "How fast can I build my application?" whereas DBAs think "Am I meeting my service-level agreements?"

For example, if a developer on a whim wanted to model a new data entity and add new attributes, that's dead simple in JavaScript Object Notation (JSON); just use one of the base data types and instantly evolve the application. In the RDBMS world, the developer goes to the DBA team, which then has to evolve the schema, and the whole "performance means how fast can I build an application" idea hits a roadblock. It's not that application developers hate DBAs—this isn't an example of Mets and Yankees fans in Major League Baseball; rather, it's a value proposition that each group relies upon to do its job. The good news is that DB2 10.5 is here to give these communities the best of both worlds! In the tech community, this is often referred to as *polyglot persistence.*

There are many different ways to classify NoSQL technology, and we think that the following makes the most sense out of the NoSQL style taxonomies we've seen:

- **Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents. The most popular document databases in today's marketplace are JSON-based; MongoDB, Couchbase, CouchDB, and Apache Jackrabbit are among the more popular ones—oh ya, and soon to be DB2!
- **Graph databases** store information about networks, such as social connections, by using directed or undirected relationships, mixes, and multigraphs. The focus for these databases is on the *connections* between data. Some of the more popular ones include Jena, InfiniteGraph, and Neo4j. DB2 10 implemented a Jena graph store that provides users with a first-class graph database and support for the native Resource Description Framework (RDF) query language, SPARQL. If you are thinking that the following people seem to be connected to this influential person via a set of Facebook interactions (often called leaders and followers), you've got a feel for this style.
- **Key-value stores** contain data whereby every single item in the database is stored as an attribute name, or key, together with its value. It's kind of like a hash table, or your file system, where the file is the content and the path is the key to getting to that content. This design is very popular for applications where performance is the name of the game, as well as for web sessionization applications, among many others. It's the second most popular NoSQL design after document databases. With that said, more and more NoSQL styles are being combined. Some popular names in this style include Dynamo, Riak, the 600 pound yellow elephant in the room … Hadoop, and Voldemort (we aren't afraid to say that word out loud).
- **Column stores** (sometimes called *wide-column stores*) place columns of data together, instead of rows, and are optimized for queries over large data sets. This technology is starting to get blurred with both traditional RDBMS and NoSQL style data stores; for example, many key-value stores have columnar store services. Column stores are very well suited

to storing sparse data sets, which are very popular in a social-mobile-cloud world, because there is no incurred storage cost for NULL values. Column stores are also great at supporting applications where only a subset of the data is needed. In the NoSQL world, HBase is perhaps the most popular, with Accumulo (think HBase with cell-level security) and Cassandra also seeing their share of popularity. Of course, DB2 with BLU Acceleration has column store capabilities, as do other RDBMS players—although admittedly we don't think they are as good and you can find out why in Chapter 3.

The development of these database styles helps us to answer two questions: "What are we trying to accomplish?" and "What kind of data do we need to address the business problems that we are trying to solve?" Many business problems involve a mixture of data, and we ultimately need to combine a number of different approaches to tackle them.

In the SQL world, consistency is of utmost importance; it's the foundation of relational database technology. With NoSQL databases, consistency is more often than not an after-the-fact consideration, and the ACID (atomic, consistent, isolated, and durable) properties of the relational world are not found at the top of requirements list. This difference shows why *both* styles are needed and why neither is going away. If it's a transaction order, statement, or record, consistency matters, but if you're trolling through log data trying to discover why someone abandoned her online shopping cart, consistency isn't as important.

Organizations that use DB2 can gain flexibility and the agility benefits of NoSQL deployment while retaining (if needed) the ACID properties, availability, and performance advantages of the well-proven DB2 relational database technology. Think about those DB2 enterprise-class attributes such as transaction control, or governance controls such as separation of duty or concern, label- or row-based access control, and more; these are the kinds of benefits that open-source projects typically don't provide (or provide with the level of sophistication found in the DB2 implementation), and this is why we know that some of the technologies found in DB2 have a place in the NoSQL database world.

## DB2 pureXML: Where DB2 First Dipped Its "Toe" in the NoSQL Pool

If years ago we asked you to think about the term Big Data, before it became the hot catch phrase that it's become, Extensible Markup Language (XML) would likely come to mind. DB2 9 delivered a truly native XML engine called DB2 pureXML. This really caught the industry by storm because, unlike its counterparts at the time, pureXML was a genuinely pure and native XML framework. DB2 didn't shred the XML or stuff it into a large object (LOB) under the guise of a "native" data type that made its XML services appear native, solely to application developers; DB2 pureXML kept the XML services purely inside the database as well. Specifically, IBM built from the ground up a pure, native XML storage capability and reworked the DB2 engine into a hybrid engine that understands and processes both XML and relational data.

The DB2 pureXML technology, which is freely available in all DB2 editions, treats XML data as a tree of XML elements and attributes and stores it that way (with no fidelity loss), thereby enabling faster insertion and retrieval of this semistructured format. This approach differs from those taken by most other database vendors, who under the covers typically store the XML as a string of characters stuffed into a character large object (CLOB), or shred away the XML tags and store it relationally. When most other vendors refer to their XML support as *native,* they are talking from the application programming interface (API) point of view. In practice, they shred or stuff the XML into the relational model, or provide you with a dizzying array of options. DB2 actually has a native compiler for XML—it's why we called it *pure.*

Of course, when you don't store your XML in a pure format like the one used by DB2, there's always a trade-off. You typically need to choose one of two evils: performance loss or reduced flexibility. The good news is that with DB2's pureXML, you never have to make such trade-offs.

DB2 pureXML technology has enabled DB2 customers to extend DB2 functionality well beyond traditional transaction processing and data warehousing tasks. DB2 was dealing with Big Data long before many of the NoSQL databases were all the rage. For more details on DB2 pureXML, refer to the DB2 documentation on this topic (http://tinyurl.com/mag5bfp).

## DB2 as a Graph Database: RDF, SPARQL, and Other Shiny Things

The Resource Description Framework (RDF) is a World Wide Web Consortium (W3C) standard for describing web resources. The RDF describes web resources in the form of *subject-predicate-object*. For example, a simple web page might have a title, an author, creation and modification dates, and attribution information, all of which can be of significant value if machines could be enabled to search for and discover "connected" resources on the Web. The ability of computers to use such metadata to understand information about web resources is the idea behind the Semantic Web.

DB2 10.1 included the ability to function as a native triplestore database that could be used to store and query RDF data. DB2's triplestore services are implemented as a set of five DB2 tables and includes an access API (based on Jena, the open-source Semantic Web framework for Java) that's entirely client-based and provides commands and interfaces for working with RDF data.

A *triplestore,* as its name implies, is a database that is optimized for the storage and retrieval of triples. A *triple* is a data entity that is composed of the aforementioned subject-predicate-object: for example, "Paul works at IBM" or "George knows Matt." Structuring data in compliance with RDF enables software programs to understand how disparate pieces of information fit together.

Because a triplestore is implemented inside of DB2, many of the features within the DB2 engine can be used to provide reliability and scalability services to the triplestore. This NoSQL technology in DB2 opened up new opportunities for organizations to seize some of the benefits of a NoSQL approach above and beyond the XML document store capabilities first introduced in DB2 9.

After an RDF triplestore has been implemented in DB2, the data can be retrieved using SPARQL, the RDF query language that is much like SQL. SPARQL is a W3C standard that is not tied to any particular RDF data store. To use SPARQL, you typically write a Java application and leverage the Jena API. Jena's support for SPARQL is contained in a module called *ARQ* (ARQ is the query engine for Jena). These are all of the pieces that you need to work with an RDF data store in DB2.

With triple graph store support in DB2 10.1, you don't have to lose the quality assurance of SQL or make expensive and risky architectural changes

to gain NoSQL graph style capabilities. Instead, use NoSQL APIs to extend DB2 and implement a paradigm that many of the NoSQL proponents are promoting. We covered this topic extensively in *Warp Speed, Time Travel, Big Data, and More: DB2 10 New Features* (McGraw-Hill Education, 2012), available at http://tinyurl.com/l2dtlbf.

## DB2 as a JSON Document Store

With the rise of social media and mobile applications, there are new opportunities for businesses to engage with their customers. Applications need to be developed quickly to respond to business problems or opportunities. There are many different technologies being assembled to rapidly deliver solutions, and JavaScript with JSON is common across most.

Applications and database schemas might need to change frequently, and some applications require flexible data interchange. JSON is an open standard whose specification can be found in the IETF RFC 4627. You can also find a wealth of information at www.JSON.org. JSON, a subset of JavaScript, is designed to be minimal, portable, and textual. It has only six kinds of data values (which keeps it simple), it's easy to implement and easy to use, and it's language independent (most programming languages today have features that map easily to JSON). JSON is commonly used to exchange data between programs that are written in all modern programming languages, and it comes in a human-readable text format, just like XML.

JSON is all the rage these days, and it's not because XML is gone. Hardly: There are many standards that are based on XML. But in the social-mobile-cloud world, JSON seems to get all the attention, mainly because it's less verbose and results in smaller document sizes; it's tightly integrated with JavaScript, which has a lot of focus in this space; and most new development tools support JSON.

### What Does JSON Look Like? JSON Document Structure

In JSON, there are four primitive types (string, number, boolean, and NULL) and two structured types (object and array) that are pretty common to most programming languages.

*Objects* are the primary concept in JSON (think "row" if you are an RDBMS aficionado): an unordered collection of name/value pairs in which the value can be any JSON value; objects can also contain other objects or arrays. *Arrays* are ordered sequences of any primitive type and can also contain an object.

JSON objects can be nested, but nesting isn't normally more than a few levels deep. The following code is an example of a JSON document:

```
{
  "firstName": "John",
  "lastName" : "Smith",
  "age"      : 25,
  "address"  :
  {
      "streetAddress": "21 2nd Street",
      "city"         : "New York",
      "state"        : "NY",
      "postalCode"   : "10021"
  },
 "phoneNumber":
 [
    {
      "type"  : "home",
      "number": "212 555-1234"
    },
    {
      "type"  : "fax",
      "number": "646 555-4567"
    }
 ]
}
```

Every JSON document starts with an open brace ({) and ends with a close brace (}). The document contains a series of name/value pairs, each pair separated by a comma:

```
    "firstName": "John",
    "lastName" : "Smith",
```

The name of each field is enclosed by double quotation marks (for example, `"firstname"`). Names and values are separated by a colon (`:`). String values must be enclosed by double quotation marks (for example, `"John"`). A JSON object doesn't place any restrictions on what is contained in your name/value pairs. That's up to the application. Remember, it's all about

simplicity here. Contrast this with XML, which has all sorts of rules for tags, data types, and positional parameters.

Other JSON objects can be embedded within a JSON object. An address field is a perfect example of an embedded object within the document:

```
"address"   :
        {
            "streetAddress": "21 2nd Street",
            "city"          : "New York",
            "state"         : "NY",
            "postalCode"    : "10021"
        }
```

As you can see, the `address` name/value pair is made up of the container name `"address"` followed by another JSON object (`"streetAddress"`) rather than a primitive data type. Within this embedded object you will find primitive types that help to make up a complete address object.

The phone number field is made up of an array type. An array starts with an open bracket (`[`) and ends with a close bracket (`]`). For example:

```
"phoneNumber":
        [
           {
             "type"   : "home",
             "number": "212 555-1234"
           },
           {
             "type"   : "fax",
             "number": "646 555-4567"
           }
        ]
```

With nesting and arrays, a lot of information about an entity can be contained within a JSON document.

If you're a DBA, you might start to get slightly uncomfortable about these arrays and objects. Are you wondering whether this approach leads to a higher level of complexity and overhead when dealing with such documents? Well… you might be right, because in a relational model, multiple phone numbers would probably have been handled as a separate telephone table, rather than multiple columns in one main table. (Besides, how can you possibly know how many telephone columns that you will eventually need in a table?) A JSON document would contain any number of phone numbers

and retrieving the document would return all phone numbers (whether you needed them or not!). So while JSON documents may look complex, implementing multi-valued fields is extremely easy. Contrast that with SQL, where you would need to split the table into two and then use join statements to retrieve the phone numbers you need.

All of this highlights one of the fundamental differences between document stores and relational databases when it comes to retrieving records: With document stores, there is no notion of a "join." A JSON developer needs to write code to join records between two or more collections ("tables" for you relational people), and that's why JSON developers might stuff everything into an object. The idea behind this is that you're dealing with an entire entity (you never know what you might need—or need to add—from a document) and that two applications trying to update the same object simultaneously would be extremely rare.

Is this inability to join documents in NoSQL stores a real limitation? It really comes down to the complexity of the application. Many JSON applications are written with a focus on "customer" or "user," with almost all (if not all) interactions being handled through the application. In other words, there's rarely a need to access this data through anything but a customer view. Contrast this with an inventory application, which has multiple sources of input (warehouse feeds, returns, deliveries, and so on) and output. There are many ways to look at and update the data, so the complexity of the relationships becomes much greater. Think about it: From a document perspective, do you really want to place the name and address of a product supplier in every product document?

As JSON applications become more complex, developers will place nested structures (or arrays) in their own collections. An additional key field would be added to the new collections to identify where these documents belong. Doesn't sound so different from a relational table, does it? This restructuring will help to manage the document collections, but it still places the burden of joining multiple collections on the shoulders of the application developer. That's the price of flexibility!

## Frequent Application Changes

Relational databases require that schemas be defined before you can add data. For example, you might want to store customer data such as first and

last name, phone numbers, e-mail address, and so on. In the SQL world, the database needs to know the data fields and the data format in advance.

Having to predefine data types doesn't make for agile development! For example, let's assume that you suddenly decide to store the "Ship To" address along with the "Bill To" address. You'll need to add that column to an RDBMS database (which, as a developer, means getting some time with the DBA) and then migrate the entire database to the new schema. If you frequently change your application in response to new requirements or during prototyping, for example, this slow paradigm ultimately becomes unacceptable. There's no easy and efficient way to use a relational database with data that is completely unstructured or unknown in advance.

NoSQL databases are designed to allow the insertion of data without any predefined schema. Application changes can be made without worrying about back-end database evolution, which means that development is faster and less database administration is required.

## Flexible Data Interchange

One of the benefits of JSON is that it's text-based and position-independent, exactly like XML. XML has been in the marketplace for a long time, but JSON is winning greater mindshare among developers when it comes to a data interchange construct. It all comes down to simplicity. JSON is simpler to use because it is focused on working with objects only. XML is a document markup language in addition to being a data exchange language. This duality of purpose makes it much harder to work with XML. JSON also tends to be more compact, requiring fewer bytes to store and flow across the network—for example, `<Mytag>value</Mytag>` versus `Mytag:value`.

Another important factor in JSON's favor is its relationship to JavaScript. JavaScript works in all browsers, is available on smart phones, and is perhaps becoming most popular because of its ubiquity around server-side applications that use JavaScript frameworks like `node.js`; remember, developers love anything that makes their lives easier and that lets them program faster, and JavaScript is all that. JSON also gives developers greater synergy with their programming language of choice. The majority of application development languages support JSON because most languages have features that map easily to JSON concepts like object, structure, and record.

Given the popularity of JSON, it's not surprising that JSON-based document store databases like MongoDB began to emerge. MongoDB (from "humongous") is a scalable, high-performance, open-source database. It is a document store; each document is written as a unit with its own individual structure. MongoDB's API is a native mixture of JSON objects and JavaScript functions.

To support customers in their use of NoSQL document stores, DB2 10.5 introduces the DB2 JSON Document Store by implementing a popular JSON API, MongoDB API, into the DB2 product. JSON support in DB2 10.5 is surfaced to application developers through a set of tools and API interfaces, as shown in Figure 6-1, and can be be summed up by the following components:

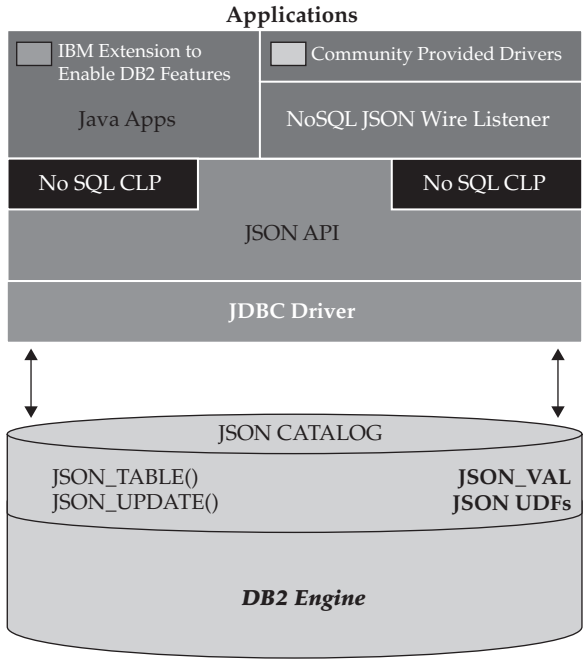


**Figure 6-1** *How DB2 10.5 supports JSON with both a Java-based API and a MongoDB-compliant wire listener; the choice is yours!*

- **A command-line shell to administer and query JSON data** The DB2 JSON command-line shell provides an interactive way to work with JSON objects. The shell enables a user to create, modify, and drop collections, as well as manipulate records within the collections themselves (we detail this feature in the next section).
- **A Java API for application development** The DB2 NoSQL JSON Java API provides a set of methods for storing, retrieving, and manipulating JSON documents. These methods can be called by native Java applications directly through the API to work with JSON documents stored in the DB2 database's JSON services. Developers can work with data that was selected from collections by using `INSERT`, `UPDATE`, or `DELETE` operations, and so on. However, when you write JSON applications to the DB2-provided JSON API, you also boost your applications with a host of features that made DB2 what it is, thereby reaping all of the benefits of this JSON integration: joins, multistatement transaction boundary support, security, high availability, management/operations, and more.
- **A NoSQL (MongoDB-compliant) wire listener to accept and respond to requests that are sent over the network** The DB2 NoSQL for JSON wire listener is a server application that intercepts the MongoDB Wire Protocol over Binary JSON (BSON). BSON is a binary-encoded serialization of JSON documents; it contains extensions that enable the representation of data types of the JSON specification. The wire listener leverages the DB2 NoSQL for JSON API to interface with DB2 as the data store. You can execute a MongoDB application written in the preferred application programming language (Java, NodeJS, PHP, Ruby, and so on), or use the MongoDB command-line interface (CLI) to communicate directly with DB2.

The following sections provide an introduction to JSON document stores, along with a glimpse into the DB2 JSON command-line shell and how you can use it to manipulate JSON objects.

## Document Store Databases

As much as a few NoSQL pundits would like to distance themselves from relational databases, they can't help but acknowledge the many similarities between the two database technologies. For example:

- A MongoDB instance contains zero or more databases.
- A database can have zero or more collections; you can think of a collection as a table in the relational world.
- A document sits inside of a collection and is composed of multiple fields; you can think of a document as a row in the relational world.
- Fields contain name/value pairs and are analogous to columns.
- Indexes are used to improve query performance.

If there are so many similarities between these two technologies, why bother using JSON? Because the objects are not manipulated in the same way. A relational database stores a value at the intersection of a particular table column and row, whereas a document-oriented database stores its values at the document level. Each document in the collection can have its own unique set of fields, and trying to model this in a relational database would require that a column be created to represent every possible field. This approach would result in a waste of space because many of these fields might be absent from a particular document; this challenge is often referred to as *sparse data sets* and is common in a social-mobile-cloud world.

Another significant drawback is that a developer needs to know all possible column values at design time; otherwise, the table would need to be modified every time the design changes. This doesn't fit well into an agile development environment where a developer might choose to add or drop attributes (columns) on the fly or evolve them over time. Indeed, understanding the nature of a JSON document and how it fits into today's modern-era applications helps to clarify why it's so popular with developers. But application developers must also understand the DBA's needs—DB2 helps these communities live in harmony!

## Manipulating JSON Documents with DB2

This section takes you through the `insert`, `update`, `delete`, and `find` commands that are part of the JSON support in DB2 10.5. There are certainly

more features than what we can cover in this short chapter, so we include some good references at the end if you want more details about how DB2 and JSON work together.

## Creating a Database

If you want to create JSON objects inside a DB2 database, you need a Java Runtime Environment (JRE) of 1.5+ (or a JDK), the DB2 JDBC drivers (`db2jcc.jar` or `db2jcc4.jar`) in the `CLASSPATH`, and a suitable DB2 10.5 database.

***TIP*** *Although DB2 10.5 lets you mix column-organized and row-organized tables in the same database, we've found it best to set* `DB2_WORKLOAD` *to the NULL value (*`db2set DB2_WORKLOAD=`*) because JSON objects need to be in row format.*

Let's start our example by creating the following database:

```
CREATE DATABASE DB2JSON
   AUTOMATIC STORAGE YES
   USING CODESET UTF-8 TERRITORY US
   COLLATE USING SYSTEM
   PAGESIZE 32 K;
```

***TIP*** *You might need to specify the database server host name or IP address and the port number if they are different from the default (*`localhost:50000`*).*

After you've created the `DB2JSON` database, you can start the command-line JSON processor by using the `db2nosql.sh` script that is located in the `/sqllib/home/json/bin` directory. The `db2nosql` command has a number of options, some of which are described in the following list:

- `-DB database_name` (if none is provided, DB2 prompts you for a value)
- `-user username` (default is the connected user)
- `-hostname host_URL` (default is `localhost`)
- `-port db_port_no` (default is `50000`)
- `-password password` for database and user (if none is provided, DB2 prompts you for the proper authentication values)
- `-setup enable/disable` (`enable` creates JSON artifacts; `disable` removes them)

For example, running the `db2nosql.sh` script without any parameters elicits a prompt for the database name (entering a NULL value causes the command line to exit):

```
db2nosql.sh
JSON Command Shell Setup and Launcher.
This batch script assumes your JRE is 1.5 and higher. 1.6 will mask your password.
Type db2nosql.sh -help to see options
Enter DB:
```

If this is the first time that the `db2nosql` command processor has connected to this database, it will prompt you to issue the `enable` command. This command updates the database with specific functions and tables that support the NoSQL JSON implementation in DB2, some of which are shown in Figure 6-1.

```
nosql>Type your JSON query and end it with ;<ENTER>
nosql>Type help() or help for usage information
nosql>
nosql>Setup Tables and Functions seems to have not been created or have been
created incorrectly. Please type enable(true) and enter to set them up. You must
have the correct admin privileges.
If you do not, type enable(false) and enter to see the SQL that will be used.
nosql>enable(true);
Executing SQL...
Database Artifacts created successfully
```

A DB2 NoSQL JSON document store allows the definition of multiple JSON namespaces by using DB2 schemas as qualifiers. When you connect to an enabled database for the first time, the default namespace is the currently connected user and will remain as such for the duration of the active session unless you explicitly change the JSON namespace by invoking the `use` command; for example:

```
nosql>use customer
Switched to schema CUSTOMER
```

The schema name is case-insensitive. You can check your current connection by using the `db` command, as shown in the following example:

```
nosql>db
Database: jdbc:db2:DB2JSON Schema: CUSTOMER
```

The `db` qualifier represents the currently connected database (`DB2JSON`) and the schema (`CUSTOMER`), and any JSON documents that are created by using this connection are organized in collections that exist under this namespace.

Note that although NoSQL collections do not enforce any document structure, documents in a collection generally share common characteristics.

## Inserting Documents

You can insert a document into a collection by using the `insert` command. For example:

```
db.customers.insert({"firstname":"Fred", "lastname":"Flints"});
db.customers.insert({"firstname":"Barney", "lastname":"Ruby"});
```

In this example, `CUSTOMERS` is the name of the collection, and it's found under the `CUSTOMER` schema. If this collection didn't already exist, DB2 would automatically create it, which is behavior that is consistent with MongoDB in the NoSQL world.

Documents typically have a unique identifier associated with them; after all, you want to find the record again, don't you? If a field that is tagged with the attribute name `_id` is found, that field is used as a unique identifier, and it's expected that all new documents will contain an `_id` of the same data type. If no `_id` is present, the system generates unique object identifiers for each document.

As you might expect at this point, deleting a collection is pretty simple too.

```
db.customers.drop();
```

You can use the `find` command to list all of the records in the `CUSTOMERS` collection.

```
nosql>db.customers.find();
nosql>Row 1:
nosql> {
nosql> "_id":{"$oid":"51fe9395986c38bd67da614d"},
nosql> "firstname":"Fred",
nosql> "lastname":"Flints",
nosql> "age":45
nosql> }
nosql>Row 2:
nosql> {
nosql> "_id":{"$oid":"51fe939c986c38bd67da614e"},
nosql> "firstname":"Barney",
nosql> "lastname":"Ruby",
nosql> "age":43
nosql> }
nosql>2 rows returned in 6 milliseconds.
```

The records contain the `_id` identifier to uniquely identify each record. Although automatic `_id` generation is useful, it's not particularly easy to remember. As an alternative, you could create another field that uniquely identifies the customer, or replace the `_id` with the real customer ID. To use a different unique identifier, the collection has to be explicitly created by using the `createCollection` command:

```
db.customers.drop();
db.createCollection("customers",{_id : "$long"});
db.customers.insert({"_id":1, "firstname":"Fred", "lastname":"Flints", "age":45});
db.customers.insert({"_id":2, "firstname":"Barney", "lastname":"Ruby", "age":43});
```

The `createCollection` command lets you specify other settings to enable DB2 features such as compression, time travel query, and table space settings, among others.

The previous sample document contains only three different data types: string (`"firstname":"Fred"`), number (`"age": 45`), and long integer (`"_id": 1`). Keep in mind, however, that DB2 *does* support additional data types, such as date, timestamp, binary, and integer, so you're not stuck with just these three.

## Importing Documents

Inserting documents into a JSON store can be really inefficient if you do it one record at a time. Luckily, DB2 also allows records to be imported from a file. Of course, the imported file must be in a valid JSON notation, because any records that aren't in the proper JSON notation are rejected by DB2 during this operation. You can improve throughput by specifying how often the utility should commit records during processing, a familiar concept for DBAs who manage relational data in DB2.

The following example shows you how to import a valid input file into the DB2 JSON document store:

```
{"_id": 1,"firstname":"Fred","lastname":"Flints","age" : 45}
{"_id": 2,"firstname":"Barney","lastname":"Ruby","age" : 43}
nosql>db.customers.importFile("customers.js");
2 objects were imported in 14 milliseconds.
```

You can use the `sampleSchema` command to retrieve information about the structure of a collection that's been imported. This command analyzes a

subset of documents and returns a list of attributes and their frequency. For example:

```
nosql>db.customers.sampleSchema();
  {
  "._id":"2;type:Long",
  ".age":"2;type:Integer",
  ".firstname":"2;type:String",
  ".lastname":"2;type:String"
  }
```

This information is similar to the DB2 `DESCRIBE` command, and can be useful in determining what name/value pairs are in the schema. You can also use this information to check whether any name/value pairs are very rarely used, perhaps because of a spelling mistake in one of the fields.

## Updating and Deleting Documents

You can use the `update` command to update one or more documents. Several optional arguments let you specify the scope of the update. Here is the command syntax:

```
update(<condition>, <fields to update>, <upsertFlag>, <multiRowFlag>)
```

The `upsertFlag` and `multiRowFlag` options can be set to either `true` or `false`; the default is `false`. When `upsertFlag` is set to `true`, the system inserts a record into the collection even if an existing record is not found. When `multiRowFlag` is set to `false`, only the first matched record is updated. When `multiRowFlag` is set to `true`, the update applies to all matching records.

If you want to replace the value of an individual field, you must use the `$set` function. For example, the following command updates Fred's age to `49`:

```
nosql>db.customers.update({_id:1},{$set:{age:49}});
Updated 1 rows.
nosql>db.customers.find({_id:1});
nosql>Row 1:
nosql> {
nosql> "_id":1,
nosql> "firstname":"Fred",
nosql> "lastname":"Flints",
nosql> "age":49
nosql> }
nosql>1 row returned in 7 milliseconds.
```

What happens if you try to update the record without using the `$set` function? Your entire record is replaced!

```
nosql>db.customers.update({_id:1},{age:49});
Updated 1 rows.
nosql>db.customers.find({_id:1});
nosql>Row 1:
nosql> {
nosql> "_id":1,
nosql> "age":49
nosql> }
nosql>1 row returned in 3 milliseconds.
```

## Selecting Documents

So far, the `find` command has been used to return all records in a collection. A find operation can be much more selective by using conditions to limit the records that are returned and a projection list to return only specific name/value pairs—a list of the most popular comparison and logical operators that you can use in DB2, when working with JSON documents is shown in Table 6-1.

The following `find` command returns only the names of customers called `Barney` and their current ages. A *projection list* contains the names of attributes and, for each attribute, a value of `1` (to include it) or `0` (to exclude it). The only attribute that you can exclude when using a projection list is the record identifier (`_id`).

| Operator | Usage | Example |
|---|---|---|
| `$eq` | Equal to | `age:{$eq:49}` (same as `age:49`) |
| `$le` | Less than or equal to | `bonus:{$le:4521}` |
| `$lt` | Less than | `width:{$lt:4.25}` |
| `$ge` | Greater than or equal to | `grade:{$ge:"C"}` |
| `$gt` | Greater than | `weight:{$gt:127}` |
| `$ne` | Not equal | `firstname:{$ne:"Fred"}` |
| `$in` | In list | `age:{$in:[43,46,49]}` |
| `$nin` | Not in list | `name:{$nin:["Fred","Barney"]}` |
| `$and` | And (both true) | `$and:[{"grade":"C"},{"age":43}]` |
| `$or` | Or (either true) | `$or:[{"name":"Fred"},{"bonus":234}]` |
| `$nor` | Neither true | `$nor:[{weight:{$gt:7}},{age:{$lt:200}}]` |
| `$not` | Logical not | `$not:{"rating":2}` |

**Table 6-1** *Comparison and Logic Operators*

```
nosql>db.customers.find({age:{$eq:43}},{_id:0, firstname:1, age:1});
nosql>Row 1:
nosql> {
nosql> "firstname":"Barney",
nosql> "age":43
nosql> }
nosql>1 row returned in 5 milliseconds.
```

The `find` command also has options to limit result sets, find the first matching document, aggregate data, count documents, find distinct values, find distinct values within groupings, calculate averages, sort values in ascending or descending order, and select a substring of an attribute. The `find` command offers plenty of other features to keep even the most skeptical NoSQL developers happy. In fact, there are features that can even improve query performance!

Indexes serve NoSQL and relational engines well for certain kinds of applications. To help speed up queries, you can create indexes on frequently searched attributes. In DB2, you can create indexes on single or multiple JSON attributes, and these indexes that can be sorted in ascending or descending order. Note that an index is always created on the `_id` column, which is used to ensure uniqueness and fast retrieval times for individual records.

You can use the `ensureIndex` command to create an index on a field in DB2. For example, the following command creates an index on "lastname" in ascending order:

```
db.customers.ensureIndex({"lastname":1});
```

JSON indexes are automatically maintained by DB2 and provide a faster way of searching for this field.

## Wrapping It Up…

So do you still think that you need a separate document-store database? Truth is, developers have been choosing their document stores based on agility and the whole "performance is speed to project delivery" bit. Now DBAs can offer that value proposition to them while applying the enterprise-grade principles that they require. And if they don't want to exploit some amazing "above and beyond" capabilities that DB2 provides, they don't even have to change their application.

The DB2 NoSQL JSON solution provides you with the flexibility of the JSON data representation within the proven DB2 database—the best of both worlds. DB2 10.5 helps you to develop modern-era applications by using a JSON programming paradigm that is patterned after the MongoDB data model and query language … without sacrificing agility.

You can administer and interactively query JSON data by using a command-line shell, develop Java applications by using an IBM-provided Java driver for JSON, and use any driver that implements the MongoDB protocol. This last feature gives you access to the DB2 NoSQL JSON store from a variety of modern languages, including `node.js`, PHP, Python, and Ruby, as well as more traditional languages such as C, C++, and Perl.

And that's why DB2 is a great relational *and* NoSQL database!

For more details about DB2 NoSQL JSON support, visit the IBM developerWorks JSON zone (www.ibm.com/developerworks/topics/json/) or see the DB2 JSON documentation.

# Resources to Build Your DB2 Skills

Rely on the wide range of IBM experts, programs, and services that are available to help you take your Information Management skills to the next level. Participate in our online community through developerWorks. Find tutorials, articles, whitepapers, videos, demos, best practices, DB2 Express downloads, and more. Visit **ibm.com/**developerworks/data/.

## IBM Certification Exams

Find industry-leading professional certification exams, including new certifications for both DB2 10.1 and DB2 10.5 with BLU Acceleration:

- DB2 10 Fundamentals (Exam 610)
- DB2 10 Database Administrator for LUW (Exam 611)
- DB2 10.5 Database Administration Upgrade Exam (Exam 311)
- DB2 10 Advanced Database Administrator for LUW (Exam 614)

Visit **ibm.com**/certify for more information and exam availability.

## IBM Training

IBM is committed to helping our clients achieve the skills and expertise to take their careers to the next level. We offer a comprehensive portfolio of technical training and education services designed for individuals, companies, and public organizations to acquire, maintain, and optimize their IT skills in IBM Software and IBM Systems. Visit **ibm.com**/software/data/education for details and course availability.

## BigData University

Learn about DB2 and various big data technologies at your pace and at your place. Big Data University offers helpful online courses with instructional videos and exercises to help you master new concepts. Course completion is marked with a final exam and a certificate. Visit **bigdatauniversity.com**.

## Information Management Bookstore

Find the electronic version of this book, links to the most informative DB2 books on the market, along with valuable links and offers to save you money and enhance your skills. Visit **http://bit.ly/DB2_Books**.

## IBM Support for DB2

Access the IBM Support Portal to find technical support information for DB2 10.5, including downloads, notifications, technical documents, flashes, and more. Visit **ibm.com**/support. Stay current on the latest DB2 10.5 support information through our blog, available at **ibm.com**/developerworks/mydeveloperworks/blogs/IM.

## IBM Data Magazine

The magazine's mission is to deliver substantive, high-quality content on the latest data management developments and IBM advances, as well as create a strong community of the world's top information management professionals. IBM Data magazine vividly demonstrates how the smart use of data and information advances broad business success, providing the context that enables data management professionals at all levels to make more informed choices and create innovative, synchronized, agile solutions.

See more at: ibmdatamag.com/.

## International DB2 User Group (IDUG)

IDUG is all about community. Through education events, technical resources, unique access to fellow users, product developers and solution providers, they offer an expansive, dynamic technical support community. IDUG delivers quality education, timely information and peer-driven product training and utilization that enable DB2 users to achieve organizational business objectives, and to drive personal career advancement. Visit idug-db2.com and idug.org.

## Join the Conversation

Stay current as DB2 10 evolves by using social media sites to connect to experts and to contribute your voice to the conversation. Visit one or more of the following:

- https://twitter.com/IBM_DB2
- https://facebook.com/DB2community
- http://bit.ly/BLUVideos
- http://linkd.in/DB2Professional
- https://twitter.com/IBMdatamag
- Blogs: http://www.planetdb2.com/
- Slideshare: http://www.slideshare.net/IBMDB2
- Google+ Page: DB2 with BLU Acceleration